<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>irf rename</td>
<td>Rename an IRF result in an IRF file</td>
<td>338</td>
</tr>
<tr>
<td>irf set</td>
<td>Set the active IRF file</td>
<td>340</td>
</tr>
<tr>
<td>irf table</td>
<td>Tables of IRFs, dynamic-multiplier functions, and FEVDs</td>
<td>343</td>
</tr>
<tr>
<td>mgarch</td>
<td>Multivariate GARCH models</td>
<td>349</td>
</tr>
<tr>
<td>mgarch ccc</td>
<td>Constant conditional correlation multivariate GARCH models</td>
<td>355</td>
</tr>
<tr>
<td>mgarch ccc postestimation</td>
<td>Postestimation tools for mgarch ccc</td>
<td>370</td>
</tr>
<tr>
<td>mgarch dcc</td>
<td>Dynamic conditional correlation multivariate GARCH models</td>
<td>375</td>
</tr>
<tr>
<td>mgarch dcc postestimation</td>
<td>Postestimation tools for mgarch dcc</td>
<td>390</td>
</tr>
<tr>
<td>mgarch dvech</td>
<td>Diagonal vech multivariate GARCH models</td>
<td>395</td>
</tr>
<tr>
<td>mgarch dvech postestimation</td>
<td>Postestimation tools for mgarch dvech</td>
<td>408</td>
</tr>
<tr>
<td>mgarch vcc</td>
<td>Varying conditional correlation multivariate GARCH models</td>
<td>414</td>
</tr>
<tr>
<td>mgarch vcc postestimation</td>
<td>Postestimation tools for mgarch vcc</td>
<td>429</td>
</tr>
<tr>
<td>mswitch</td>
<td>Markov-switching regression models</td>
<td>434</td>
</tr>
<tr>
<td>mswitch postestimation</td>
<td>Postestimation tools for mswitch</td>
<td>460</td>
</tr>
<tr>
<td>newey</td>
<td>Regression with Newey–West standard errors</td>
<td>470</td>
</tr>
<tr>
<td>newey postestimation</td>
<td>Postestimation tools for newey</td>
<td>475</td>
</tr>
<tr>
<td>pergram</td>
<td>Periodogram</td>
<td>481</td>
</tr>
<tr>
<td>perron</td>
<td>Phillips–Perron unit-root test</td>
<td>489</td>
</tr>
<tr>
<td>prais</td>
<td>Prais–Winsten and Cochrane–Orcutt regression</td>
<td>494</td>
</tr>
<tr>
<td>prais postestimation</td>
<td>Postestimation tools for prais</td>
<td>506</td>
</tr>
<tr>
<td>psdensity</td>
<td>Parametric spectral density estimation after arima, arfima, and ucm</td>
<td>509</td>
</tr>
<tr>
<td>rolling</td>
<td>Rolling-window and recursive estimation</td>
<td>520</td>
</tr>
<tr>
<td>sspace</td>
<td>State-space models</td>
<td>528</td>
</tr>
<tr>
<td>sspace postestimation</td>
<td>Postestimation tools for sspace</td>
<td>552</td>
</tr>
<tr>
<td>threshold</td>
<td>Threshold regression</td>
<td>560</td>
</tr>
<tr>
<td>threshold postestimation</td>
<td>Postestimation tools for threshold</td>
<td>571</td>
</tr>
<tr>
<td>tsappend</td>
<td>Add observations to a time-series dataset</td>
<td>575</td>
</tr>
<tr>
<td>tsfill</td>
<td>Fill in gaps in time variable</td>
<td>582</td>
</tr>
<tr>
<td>tsfilter</td>
<td>Filter a time series for cyclical components</td>
<td>587</td>
</tr>
<tr>
<td>tsfilter bk</td>
<td>Baxter–King time-series filter</td>
<td>606</td>
</tr>
<tr>
<td>tsfilter bw</td>
<td>Butterworth time-series filter</td>
<td>615</td>
</tr>
<tr>
<td>tsfilter cf</td>
<td>Christiano–Fitzgerald time-series filter</td>
<td>624</td>
</tr>
<tr>
<td>tsfilter hp</td>
<td>Hodrick–Prescott time-series filter</td>
<td>633</td>
</tr>
<tr>
<td>tsline</td>
<td>Time-series line plots</td>
<td>640</td>
</tr>
<tr>
<td>tsreport</td>
<td>Report time-series aspects of a dataset or estimation sample</td>
<td>646</td>
</tr>
<tr>
<td>tsrevar</td>
<td>Time-series operator programming command</td>
<td>653</td>
</tr>
<tr>
<td>tsset</td>
<td>Declare data to be time-series data</td>
<td>656</td>
</tr>
<tr>
<td>tssmooth</td>
<td>Smooth and forecast univariate time-series data</td>
<td>673</td>
</tr>
<tr>
<td>tssmooth dexp</td>
<td>Double-exponential smoothing</td>
<td>675</td>
</tr>
<tr>
<td>tssmooth exp</td>
<td>Single-exponential smoothing</td>
<td>682</td>
</tr>
<tr>
<td>tssmooth hwinters</td>
<td>Holt–Winters nonseasonal smoothing</td>
<td>691</td>
</tr>
<tr>
<td>tssmooth ma</td>
<td>Moving-average filter</td>
<td>699</td>
</tr>
<tr>
<td>tssmooth nl</td>
<td>Nonlinear filter</td>
<td>705</td>
</tr>
<tr>
<td>tssmooth shwinters</td>
<td>Holt–Winters seasonal smoothing</td>
<td>708</td>
</tr>
<tr>
<td>ucm</td>
<td>Unobserved-components model</td>
<td>718</td>
</tr>
<tr>
<td>ucm postestimation</td>
<td>Postestimation tools for ucm</td>
<td>747</td>
</tr>
<tr>
<td>var intro</td>
<td>Introduction to vector autoregressive models</td>
<td>754</td>
</tr>
</tbody>
</table>
Contents

var ............................................................ Vector autoregressive models 761
var postestimation ........................................... Postestimation tools for var 774
var svar ....................................................... Structural vector autoregressive models 778
var svar postestimation ....................................... Postestimation tools for svar 797
varbasic ...................................................... Fit a simple VAR and graph IRFs or FEVDs 800
varbasic postestimation ...................................... Postestimation tools for varbasic 805
vargranger .................................................. Pairwise Granger causality tests after var or svar 809
varlmar ....................................................... LM test for residual autocorrelation after var or svar 815
varnorm ...................................................... Test for normally distributed disturbances after var or svar 819
varsoc ......................................................... Obtain lag-order selection statistics for VARs and VECMs 825
varstable ..................................................... Check the stability condition of VAR or SVAR estimates 832
varwle ......................................................... Obtain Wald lag-exclusion statistics after var or svar 838
vec intro ..................................................... Introduction to vector error-correction models 844
vec ........................................................... Vector error-correction models 863
vec postestimation ........................................... Postestimation tools for vec 888
veclmar ....................................................... LM test for residual autocorrelation after vec 892
vecnorm ...................................................... Test for normally distributed disturbances after vec 896
vecrank ....................................................... Estimate the cointegrating rank of a VECM 900
vecstable ...................................................... Check the stability condition of VECM estimates 909
wntestb ....................................................... Bartlett’s periodogram-based test for white noise 914
wntestq ....................................................... Portmanteau (Q) test for white noise 919
xcorr .......................................................... Cross-correlogram for bivariate time series 922
Glossary .......................................................... 927
Subject and author index ........................................ 936
Cross-referencing the documentation

When reading this manual, you will find references to other Stata manuals, for example, [U] 27 Overview of Stata estimation commands; [R] regress; and [D] reshape. The first example is a reference to chapter 27, Overview of Stata estimation commands, in the User’s Guide; the second is a reference to the regress entry in the Base Reference Manual; and the third is a reference to the reshape entry in the Data Management Reference Manual.

All the manuals in the Stata Documentation have a shorthand notation:

- [GSM] Getting Started with Stata for Mac
- [GSU] Getting Started with Stata for Unix
- [GSW] Getting Started with Stata for Windows
- [U] Stata User’s Guide
- [R] Stata Base Reference Manual
- [BAYES] Stata Bayesian Analysis Reference Manual
- [D] Stata Data Management Reference Manual
- [FN] Stata Functions Reference Manual
- [LASSO] Stata Lasso Reference Manual
- [XT] Stata Longitudinal-Data/Panel-Data Reference Manual
- [MI] Stata Multiple-Imputation Reference Manual
- [RPT] Stata Reporting Reference Manual
- [SVY] Stata Survey Data Reference Manual
- [I] Stata Glossary and Index
Description

This manual documents Stata’s time-series commands and is referred to as [TS] in cross-references.

After this entry, [TS] Time series provides an overview of the `ts` commands. The other parts of this manual are arranged alphabetically. If you are new to Stata’s time-series features, we recommend that you read the following sections first:

- [TS] Time series Introduction to time-series commands
- [TS] tsset Declare a dataset to be time-series data

Stata is continually being updated, and Stata users are always writing new commands. To ensure that you have the latest features, you should install the most recent official update; see [R] update.

Also see

[U] 1.3 What’s new
[R] Intro — Introduction to base reference manual
Description

The Time-Series Reference Manual organizes the commands alphabetically, making it easy to find individual command entries if you know the name of the command. This overview organizes and presents the commands conceptually, that is, according to the similarities in the functions that they perform. The table below lists the manual entries that you should see for additional information.

Data management tools and time-series operators.
These commands help you prepare your data for further analysis.

Univariate time series.
These commands are grouped together because they are either estimators or filters designed for univariate time series or preestimation or postestimation commands that are conceptually related to one or more univariate time-series estimators.

Multivariate time series.
These commands are similarly grouped together because they are either estimators designed for use with multivariate time series or preestimation or postestimation commands conceptually related to one or more multivariate time-series estimators.

Forecasting models.
These commands work as a group to provide the tools you need to create models by combining estimation results, identities, and other objects and to solve those models to obtain forecasts.

Within these three broad categories, similar commands have been grouped together.

Data management tools and time-series operators

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tsset</code></td>
<td>Declare data to be time-series data</td>
</tr>
<tr>
<td><code>tsfill</code></td>
<td>Fill in gaps in time variable</td>
</tr>
<tr>
<td><code>tsappend</code></td>
<td>Add observations to a time-series dataset</td>
</tr>
<tr>
<td><code>tsreport</code></td>
<td>Report time-series aspects of a dataset or estimation sample</td>
</tr>
<tr>
<td><code>tsrevar</code></td>
<td>Time-series operator programming command</td>
</tr>
<tr>
<td><code>rolling</code></td>
<td>Rolling-window and recursive estimation</td>
</tr>
<tr>
<td><code>[D] fred</code></td>
<td>User-definable business calendars</td>
</tr>
<tr>
<td><code>import</code></td>
<td>Import data from Federal Reserve Economic Data</td>
</tr>
</tbody>
</table>
Univariate time series

Estimators

- **arima**
  - Autoregressive fractionally integrated moving-average (ARFIMA) models

- **arfima**
  - Postestimation tools for arfima

- **arima**
  - ARIMA, ARMAX, and other dynamic regression models

- **arfima postestimation**
  - Postestimation tools for arfima

- **arch**
  - Autoregressive conditional heteroskedasticity (ARCH) family of estimators

- **arch postestimation**
  - Postestimation tools for arch

- **mswitch**
  - Markov-switching regression models

- **mswitch postestimation**
  - Postestimation tools for mswitch

- **newey**
  - Regression with Newey–West standard errors

- **newey postestimation**
  - Postestimation tools for newey

- **prais**
  - Prais–Winsten and Cochrane–Orcutt regression

- **prais postestimation**
  - Postestimation tools for prais

- **ucm**
  - Unobserved-components model

- **ucm postestimation**
  - Postestimation tools for ucm

- **threshold**
  - Threshold regression

- **threshold postestimation**
  - Postestimation tools for threshold

Time-series smoothers and filters

- **tsfilter bk**
  - Baxter–King time-series filter

- **tsfilter bw**
  - Butterworth time-series filter

- **tsfilter cf**
  - Cristiano–Fitzgerald time-series filter

- **tsfilter hp**
  - Hodrick–Prescott time-series filter

- **tssmooth ma**
  - Moving-average filter

- **tssmooth dexpontial**
  - Double-exponential smoothing

- **tssmooth exponential**
  - Single-exponential smoothing

- **tssmooth hwinters**
  - Holt–Winters nonseasonal smoothing

- **tssmooth shwinters**
  - Holt–Winters seasonal smoothing

- **tssmooth nl**
  - Nonlinear filter
Diagnostic tools

- **corrgram**
  - Tabulate and graph autocorrelations
- **xcorr**
  - Cross-correlogram for bivariate time series
- **cump**
  - Graph cumulative spectral distribution
- **pergram**
  - Periodogram
- **psdensity**
  - Parametric spectral density estimation
- **estacplot**
  - Plot parametric autocorrelation and autocovariance functions
- **estaroots**
  - Check the stability condition of ARIMA estimates
- **dfgls**
  - DF-GLS unit-root test
- **dfuller**
  - Augmented Dickey–Fuller unit-root test
- **pperron**
  - Phillips–Perron unit-root test
- **estbsknown**
  - Test for a structural break with a known break date
- **estbsingle**
  - Test for a structural break with an unknown break date
- **estbcusum**
  - Cumulative sum test for parameter stability
- **regress postestimation time series**
  - Postestimation tools for regress with time series
- **mswitch postestimation**
  - Postestimation tools for mswitch
- **wntestb**
  - Bartlett’s periodogram-based test for white noise
- **wntestq**
  - Portmanteau (Q) test for white noise

Multivariate time series

**Estimators**

- **dfactor**
  - Dynamic-factor models
- **dfactor postestimation**
  - Postestimation tools for dfactor
- **mgarch ccc**
  - Constant conditional correlation multivariate GARCH models
- **mgarch ccc postestimation**
  - Postestimation tools for mgarch ccc
- **mgarch dcc**
  - Dynamic conditional correlation multivariate GARCH models
- **mgarch dcc postestimation**
  - Postestimation tools for mgarch dcc
- **mgarch dvech**
  - Diagonal vech multivariate GARCH models
- **mgarch dvech postestimation**
  - Postestimation tools for mgarch dvech
- **mgarch vcc**
  - Varying conditional correlation multivariate GARCH models
- **mgarch vcc postestimation**
  - Postestimation tools for mgarch vcc
- **sspace**
  - State-space models
- **sspace postestimation**
  - Postestimation tools for sspace
- **var**
  - Vector autoregressive models
- **var postestimation**
  - Postestimation tools for var
- **var svar**
  - Structural vector autoregressive models
- **var svar postestimation**
  - Postestimation tools for svar
- **varbasic**
  - Fit a simple VAR and graph IRFs or FEVDs
- **varbasic postestimation**
  - Postestimation tools for varbasic
- **vec**
  - Vector error-correction models
- **vec postestimation**
  - Postestimation tools for vec

Also see *Stata Dynamic Stochastic General Equilibrium Models Reference Manual*. 
Diagnostic tools

[TS] varlmar  Perform LM test for residual autocorrelation
[TS] varnorm  Test for normally distributed disturbances
[TS] varsoc    Obtain lag-order selection statistics for VARs and VECMs
[TS] varstable Check the stability condition of VAR or SVAR estimates
[TS] varwle    Obtain Wald lag-exclusion statistics
[TS] veclmar   Perform LM test for residual autocorrelation
[TS] vecnorm   Test for normally distributed disturbances
[TS] vecrank   Estimate the cointegrating rank of a VECM
[TS] vecstable Check the stability condition of VECM estimates

Forecasting, inference, and interpretation

[TS] irf create Obtain IRFs, dynamic-multiplier functions, and FEVDs
[TS] fcast compute Compute dynamic forecasts after var, svar, or vec
[TS] vargranger Perform pairwise Granger causality tests

Graphs and tables

[TS] corrgram Tabulate and graph autocorrelations
[TS] xcorr      Cross-correlogram for bivariate time series
[TS] pergram    Periodogram
[TS] irf graph  Graphs of IRFs, dynamic-multiplier functions, and FEVDs
[TS] irf cgraph Combined graphs of IRFs, dynamic-multiplier functions, and FEVDs
[TS] irf cgraph  Overlaid graphs of IRFs, dynamic-multiplier functions, and FEVDs
[TS] irf table  Tables of IRFs, dynamic-multiplier functions, and FEVDs
[TS] irf ctable Combined tables of IRFs, dynamic-multiplier functions, and FEVDs
[TS] fcast graph Graph forecasts after fcast compute
[TS] tsline     Plot time-series data
[TS] varstable  Check the stability condition of VAR or SVAR estimates
[TS] vecstable  Check the stability condition of VECM estimates
[TS] wntestb    Bartlett’s periodogram-based test for white noise

Results management tools

[TS] irf add    Add results from an IRF file to the active IRF file
[TS] irf describe Describe an IRF file
[TS] irf drop    Drop IRF results from the active IRF file
[TS] irf rename Rename an IRF result in an IRF file
[TS] irf set     Set the active IRF file
Forecasting models

- **[TS] forecast**: Econometric model forecasting
- **[TS] forecast adjust**: Adjust a variable by add factoring, replacing, etc.
- **[TS] forecast clear**: Clear current model from memory
- **[TS] forecast coefvector**: Specify an equation via a coefficient vector
- **[TS] forecast create**: Create a new forecast model
- **[TS] forecast describe**: Describe features of the forecast model
- **[TS] forecast drop**: Drop forecast variables
- **[TS] forecast estimates**: Add estimation results to a forecast model
- **[TS] forecast exogenous**: Declare exogenous variables
- **[TS] forecast identity**: Add an identity to a forecast model
- **[TS] forecast list**: List forecast commands composing current model
- **[TS] forecast query**: Check whether a forecast model has been started
- **[TS] forecast solve**: Obtain static and dynamic forecasts

---

**Remarks and examples**

Remarks are presented under the following headings:

- Data management tools and time-series operators
- Univariate time series: Estimators
- Univariate time series: Time-series smoothers and filters
- Univariate time series: Diagnostic tools
- Multivariate time series: Estimators
- Multivariate time series: Diagnostic tools
- Forecasting models
- Additional resources

**Data management tools and time-series operators**

Because time-series estimators are, by definition, a function of the temporal ordering of the observations in the estimation sample, Stata’s time-series commands require the data to be sorted and indexed by time, using the `tsset` command, before they can be used. `tsset` is simply a way for you to tell Stata which variable in your dataset represents time; `tsset` then sorts and indexes the data appropriately for use with the time-series commands. Once your dataset has been `tsset`, you can use Stata’s time-series operators in data manipulation or programming using that dataset and when specifying the syntax for most time-series commands. Stata has time-series operators for representing the lags, leads, differences, and seasonal differences of a variable. The time-series operators are documented in [TS] `tsset`.

You can also define a business-day calendar so that Stata’s time-series operators respect the structure of missing observations in your data. The most common example is having Monday come after Friday in market data. [D] Datetime business calendars provides a discussion and examples.

`tsset` can also be used to declare that your dataset contains cross-sectional time-series data, often referred to as panel data. When you use `tsset` to declare your dataset to contain panel data, you specify a variable that identifies the panels and a variable that identifies the time periods. Once your dataset has been `tsset` as panel data, the time-series operators work appropriately for the data.

`tsfill`, which is documented in [TS] `tsfill`, can be used after `tsset` to fill in missing times with missing observations. `tsset` will report any gaps in your data, and `tsreport` will provide more details about the gaps. `tsappend` adds observations to a time-series dataset by using the information set by `tsset`. This function can be particularly useful when you wish to predict out of sample after
fitting a model with a time-series estimator. `tsrevar` is a programmer’s command that provides a way to use `varlists` that contain time-series operators with commands that do not otherwise support time-series operators.

`rolling` performs rolling regressions, recursive regressions, and reverse recursive regressions. Any command that stores results in `e()` or `r()` can be used with `rolling`.

`import fred` can be used to import economic and financial time-series data from the Federal Reserve Economic Data (FRED) into Stata. See [D] `import fred` for details on the `import fred` command and the FRED interface, which allows you to easily search available series and select which ones you want to import.

**Univariate time series: Estimators**

The seven univariate time-series estimators currently available in Stata are `arfima`, `arima`, `arch`, `mswitch`, `newey`, `prais`, and `ucm`. `newey` and `prais` are really just extensions to ordinary linear regression. When you fit a linear regression on time-series data via ordinary least squares (OLS), if the disturbances are autocorrelated, the parameter estimates are usually consistent, but the estimated standard errors tend to be underestimated. Several estimators have been developed to deal with this problem. One strategy is to use OLS for estimating the regression parameters and use a different estimator for the variances, one that is consistent in the presence of autocorrelated disturbances, such as the Newey–West estimator implemented in `newey`. Another strategy is to model the dynamics of the disturbances. The estimators found in `prais`, `arima`, `arch`, `arfima`, and `ucm` are based on such a strategy.

`prais` implements two such estimators: the Prais–Winsten and the Cochrane–Orcutt generalized least-squares (GLS) estimators. These estimators are GLS estimators, but they are fairly restrictive in that they permit only first-order autocorrelation in the disturbances. Although they have certain pedagogical and historical value, they are somewhat obsolete. Faster computers with more memory have made it possible to implement full information maximum likelihood (FIML) estimators, such as Stata’s `arima` command. These estimators permit much greater flexibility when modeling the disturbances and are more efficient estimators.

`arima` provides the means to fit linear models with autoregressive moving-average (ARMA) disturbances, or in the absence of linear predictors, autoregressive integrated moving-average (ARIMA) models. This means that, whether you think that your data are best represented as a distributed-lag model, a transfer-function model, or a stochastic difference equation, or you simply wish to apply a Box–Jenkins filter to your data, the model can be fit using `arima`. `arch`, a conditional maximum likelihood estimator, has similar modeling capabilities for the mean of the time series but can also model autoregressive conditional heteroskedasticity in the disturbances with a wide variety of specifications for the variance equation.

`arfima` estimates the parameters of autoregressive fractionally integrated moving-average (ARFIMA) models, which handle higher degrees of dependence than ARIMA models. ARFIMA models allow the autocorrelations to decay at the slower hyperbolic rate, whereas ARIMA models handle processes whose autocorrelations decay at an exponential rate.

Markov-switching models are used for series that are believed to transition over a finite set of unobserved states, allowing the process to evolve differently in each state. The transitions occur according to a Markov process. `mswitch` estimates the state-dependent parameters of Markov-switching dynamic regression models and Markov-switching autoregression models.

Unobserved-components models (UCMs) decompose a time series into trend, seasonal, cyclical, and idiosyncratic components and allow for exogenous variables. `ucm` estimates the parameters of
UCMs by maximum likelihood. UCMs can also model the stationary cyclical component using the stochastic-cycle parameterization that has an intuitive frequency-domain interpretation.

_threshold fits a linear model that allows coefficients to differ across regions determined by threshold values. Threshold models can be used, for instance, to capture breaks or asymmetries that are observed in many time series. The number of thresholds may be known, or _threshold can choose the optimal number of thresholds. See [TS] _threshold.

**Univariate time series: Time-series smoothers and filters**

In addition to the estimators mentioned above, Stata also provides time-series filters and smoothers. The Baxter–King and Christiano–Fitzgerald band-pass filters and the Butterworth and Hodrick–Prescott high-pass filters are implemented in _tfsfilter; see [TS] _tfsfilter for an overview.

Also included are a simple, uniformly weighted, moving-average filter with unit weights; a weighted moving-average filter in which you can specify the weights; single- and double-exponential smoothers; Holt–Winters seasonal and nonseasonal smoothers; and a nonlinear smoother. Most of these smoothers were originally developed as ad hoc procedures and are used for reducing the noise in a time series (smoothing) or forecasting. Although they have limited application for signal extraction, these smoothers have all been found to be optimal for some underlying modern time-series models; see [TS] _tssmooth.

**Univariate time series: Diagnostic tools**

Stata’s time-series commands also include several preestimation and postestimation diagnostic and interpretation commands. _corrgram estimates the autocorrelation function and partial autocorrelation function of a univariate time series, as well as _Q statistics. These functions and statistics are often used to determine the appropriate model specification before fitting ARIMA models. _corrgram can also be used with _wntestb and _wntestq to examine the residuals after fitting a model for evidence of model misspecification. Stata’s time-series commands also include the commands _pergram and _cump, which provide the log-standardized periodogram and the cumulative-sample spectral distribution, respectively, for time-series analysts who prefer to estimate in the frequency domain rather than the time domain.

_psdensity computes the spectral density implied by the parameters estimated by _arfima, _arima, or _ucm. The estimated spectral density shows the relative importance of components at different frequencies. _estat_acplot computes the autocorrelation and autocovariance functions implied by the parameters estimated by _arima. These functions provide a measure of the dependence structure in the time domain.

_xcorr estimates the cross-correlogram for bivariate time series and can similarly be used for both preestimation and postestimation. For example, the cross-correlogram can be used before fitting a transfer-function model to produce initial estimates of the IRF. This estimate can then be used to determine the optimal lag length of the input series to include in the model specification. It can also be used as a postestimation tool after fitting a transfer function. The cross-correlogram between the residual from a transfer-function model and the prewhitened input series of the model can be examined for evidence of model misspecification.

When you fit ARMA or ARIMA models, the dependent variable being modeled must be covariance stationary (ARMA models), or the order of integration must be known (ARIMA models). Stata has three commands that can test for the presence of a unit root in a time-series variable: _dfuller performs the augmented Dickey–Fuller test, _pperron performs the Phillips–Perron test, and _dfgls performs a modified Dickey–Fuller test. _arfima can also be used to investigate the order of integration. After estimation, you can use _estat_aroots to check the stationarity of an ARMA process.
After using `mswitch` to fit a Markov-switching model, two postestimation commands help interpret the results. `estat transition` reports the transition probabilities and the corresponding standard errors in a tabular form. `estat duration` computes the expected duration of being in a given state and displays. See [TS] `mswitch postestimation`.

After fitting a model with `regress` or `ivregress`, `estat sbknown` and `estat sbsingle` test for structural breaks. `estat sbknown` tests for breaks at known break dates, and `estat sbsingle` tests for a break at an unknown break date; see [TS] `estat sbknown` and [TS] `estat sbsingle`. After `regress`, `estat sbcusum` tests for stability of regression coefficients. This can be a test for structural breaks due to changes in regression coefficients over time; see [TS] `estat sbcusum`.

The remaining diagnostic tools for univariate time series are for use after fitting a linear model via OLS with Stata’s `regress` command. They are documented collectively in [R] `regress postestimation time series`. They include `estat dwatson`, `estat durbinalt`, `estat bgodfrey`, and `estat archlm`. `estat dwatson` computes the Durbin–Watson $d$ statistic to test for the presence of first-order autocorrelation in the OLS residuals. `estat durbinalt` likewise tests for the presence of autocorrelation in the residuals. By comparison, however, Durbin’s alternative test is more general and easier to use than the Durbin–Watson test. With `estat durbinalt`, you can test for higher orders of autocorrelation, the assumption that the covariates in the model are strictly exogenous is relaxed, and there is no need to consult tables to compute rejection regions, as you must with the Durbin–Watson test. `estat bgodfrey` computes the Breusch–Godfrey test for autocorrelation in the residuals, and although the computations are different, the test in `estat bgodfrey` is asymptotically equivalent to the test in `estat durbinalt`. Finally, `estat archlm` performs Engle’s LM test for the presence of autoregressive conditional heteroskedasticity.

**Multivariate time series: Estimators**

Stata provides commands for fitting the most widely applied multivariate time-series models. `var` and `svar` fit vector autoregressive and structural vector autoregressive models to stationary data. `vec` fits cointegrating vector error-correction models. `dfactor` fits dynamic-factor models. `mgarch ccc`, `mgarch dcc`, `mgarch dvech`, and `mgarch vcc` fit multivariate GARCH models. `sspace` fits state-space models. Many linear time-series models, including vector autoregressive moving-average (VARMA) models and structural time-series models, can be cast as state-space models and fit by `sspace`. `dsge` fits linear dynamic stochastic general equilibrium (DSGE) models, and `dsgenl` fits nonlinear DSGE models. DSGE models are unique in that equations in the system allow current values of variables to depend not only on past values but also on expectations of future values.

**Multivariate time series: Diagnostic tools**

Before fitting a multivariate time-series model, you must specify the number of lags of the dependent variable to include. `varsoc` produces statistics for determining the order of a VAR or VECM.

Several postestimation commands perform the most common specification analysis on a previously fitted VAR or SVAR. You can use `varlmar` to check for serial correlation in the residuals, `varnorm` to test the null hypothesis that the disturbances come from a multivariate normal distribution, and `varstable` to see if the fitted VAR or SVAR is stable. Two common types of inference about VAR models are whether one variable Granger-causes another and whether a set of lags can be excluded from the model. `vargranger` reports Wald tests of Granger causation, and `varwle` reports Wald lag exclusion tests.

Similarly, several postestimation commands perform the most common specification analysis on a previously fitted VECM. You can use `veclmar` to check for serial correlation in the residuals, `vecnorm`
to test the null hypothesis that the disturbances come from a multivariate normal distribution, and `vecstable` to analyze the stability of the previously fitted VECM.

VARs and VECMs are often fit to produce baseline forecasts. `fcast` produces dynamic forecasts from previously fitted VARs and VECMs. See [TS] `fcast compute` and [TS] `fcast graph`.

Many researchers fit VARs, SVARs, VECMs, and DSGEs because they want to analyze how unexpected shocks affect the dynamic paths of the variables. Stata has a suite of `irf` commands for estimating IRF functions and interpreting, presenting, and managing these estimates; see [TS] `irf`.

### Forecasting models

Stata provides a set of commands for obtaining forecasts by solving models, collections of equations that jointly determine the outcomes of one or more variables. You use Stata estimation commands such as `regress`, `reg3`, `var`, and `vec` to fit stochastic equations and store the results using `estimates store`. Then you create a forecast model using `forecast create` and use commands, including `forecast estimates` and `forecast identity`, to build models consisting of estimation results, nonstochastic relationships (identities), and other model features. Models can be as simple as a single linear regression for which you want to obtain dynamic forecasts, or they can be complicated systems consisting of dozens of estimation results and identities representing a complete macroeconometric model.

The `forecast solve` command allows you to obtain both stochastic and dynamic forecasts. Confidence intervals for forecasts can be obtained via stochastic simulation incorporating both parameter uncertainty and additive random shocks. By using `forecast adjust`, you can incorporate outside information and specify different paths for some of the model’s variables to obtain forecasts under alternative scenarios. See [TS] `forecast`.

### Additional resources

In addition to the manual, there are several other resources available for learning about Stata’s time-series features.

- **On the web:** You can watch the Time series in Stata playlist on our YouTube Channel. You can read our posts about time-series analysis at The Stata Blog: Not Elsewhere Classified. You can search for blog posts from within Stata by typing `search time series blog`.

- **Training:** Stata’s NetCourse 461 provides a comprehensive introduction to univariate time-series analysis. See [https://www.stata.com/netcourse/univariate-time-series-intro-nc461](https://www.stata.com/netcourse/univariate-time-series-intro-nc461) for more information. We also offer a classroom and web-based training course, Time-Series Analysis Using Stata.

- **Books:** We recommend the Stata Press book Introduction to Time Series Using Stata by Sean Becketti. See [https://www.stata-press.com/books/introduction-to-time-series-using-stata](https://www.stata-press.com/books/introduction-to-time-series-using-stata) for full comments from our technical group.

References


Becketti, S. 2013. Introduction to Time Series Using Stata. College Station, TX: Stata Press.


Also see

[U] 1.3 What’s new

[R] Intro — Introduction to base reference manual
### Description

The `arch` command fits regression models in which the volatility of a series varies through time. Usually, periods of high and low volatility are grouped together. ARCH models estimate future volatility as a function of prior volatility. To accomplish this, `arch` fits models of autoregressive conditional heteroskedasticity (ARCH) by using conditional maximum likelihood. In addition to ARCH terms, models may include multiplicative heteroskedasticity. Gaussian (normal), Student’s $t$, and generalized error distributions are supported.

Concerning the regression equation itself, models may also contain ARCH-in-mean and ARMA terms.

### Quick start

**ARCH model of $y$ with first- and second-order ARCH components and regressor $x$ using `tsset` data**

```plaintext
arch y x, arch(1,2)
```

Add a second-order GARCH component

```plaintext
arch y x, arch(1,2) garch(2)
```

Add an autoregressive component of order 2 and a moving-average component of order 3

```plaintext
arch y x, arch(1,2) garch(2) ar(2) ma(3)
```

As above, but with the conditional variance included in the mean equation

```plaintext
arch y x, arch(1,2) garch(2) ar(2) ma(3) archm
```

**EGARCH model of order 2 for $y$ with an autoregressive component of order 1**

```plaintext
arch y, earch(2) egarch(2) ar(1)
```
Menu

ARCH/GARCH
Statistics > Time series > ARCH/GARCH > ARCH and GARCH models

EARCH/EGARCH
Statistics > Time series > ARCH/GARCH > Nelson’s EGARCH model

ABARCH/ATARCH/SDGARCH
Statistics > Time series > ARCH/GARCH > Threshold ARCH model

ARCH/TARCH/GARCH
Statistics > Time series > ARCH/GARCH > GJR form of threshold ARCH model

ARCH/SAARCH/GARCH
Statistics > Time series > ARCH/GARCH > Simple asymmetric ARCH model

PARCH/PGARCH
Statistics > Time series > ARCH/GARCH > Power ARCH model

NARCH/GARCH
Statistics > Time series > ARCH/GARCH > Nonlinear ARCH model

NARCHK/GARCH
Statistics > Time series > ARCH/GARCH > Nonlinear ARCH model with one shift

APARCH/PGARCH
Statistics > Time series > ARCH/GARCH > Asymmetric power ARCH model

NPARCH/PGARCH
Statistics > Time series > ARCH/GARCH > Nonlinear power ARCH model
## Syntax

```
arch depvar [ indepvars ] [ if ] [ in ] [ weight ] [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>noconstant</code></td>
<td>suppress constant term</td>
</tr>
<tr>
<td><code>arch(numlist)</code></td>
<td>ARCH terms</td>
</tr>
<tr>
<td><code>garch(numlist)</code></td>
<td>GARCH terms</td>
</tr>
<tr>
<td><code>saarch(numlist)</code></td>
<td>simple asymmetric ARCH terms</td>
</tr>
<tr>
<td><code>tarch(numlist)</code></td>
<td>threshold ARCH terms</td>
</tr>
<tr>
<td><code>aarch(numlist)</code></td>
<td>asymmetric ARCH terms</td>
</tr>
<tr>
<td><code>narch(numlist)</code></td>
<td>nonlinear ARCH terms</td>
</tr>
<tr>
<td><code>narchk(numlist)</code></td>
<td>nonlinear ARCH terms with single shift</td>
</tr>
<tr>
<td><code>abarch(numlist)</code></td>
<td>absolute value ARCH terms</td>
</tr>
<tr>
<td><code>atarch(numlist)</code></td>
<td>absolute threshold ARCH terms</td>
</tr>
<tr>
<td><code>sdgarch(numlist)</code></td>
<td>lags of $\sigma_t$</td>
</tr>
<tr>
<td><code>earch(numlist)</code></td>
<td>news terms in Nelson’s (1991) EGARCH model</td>
</tr>
<tr>
<td><code>egarch(numlist)</code></td>
<td>lags of $\ln(\sigma_t^2)$</td>
</tr>
<tr>
<td><code>parc(numlist)</code></td>
<td>power ARCH terms</td>
</tr>
<tr>
<td><code>tparch(numlist)</code></td>
<td>threshold power ARCH terms</td>
</tr>
<tr>
<td><code>aparch(numlist)</code></td>
<td>asymmetric power ARCH terms</td>
</tr>
<tr>
<td><code>nparch(numlist)</code></td>
<td>nonlinear power ARCH terms</td>
</tr>
<tr>
<td><code>nparchk(numlist)</code></td>
<td>nonlinear power ARCH terms with single shift</td>
</tr>
<tr>
<td><code>pgarch(numlist)</code></td>
<td>power GARCH terms</td>
</tr>
<tr>
<td><code>constraints(constraints)</code></td>
<td>apply specified linear constraints</td>
</tr>
</tbody>
</table>

### Model 2

- `archm` include ARCH-in-mean term in the mean-equation specification
- `archmlags(numlist)` include specified lags of conditional variance in mean equation
- `archmexp(exp)` apply transformation in `exp` to any ARCH-in-mean terms
- `arima(#p,#d,#q)` specify ARIMA($p, d, q$) model for dependent variable
- `ar(numlist)` autoregressive terms of the structural model disturbance
- `ma(numlist)` moving-average terms of the structural model disturbances

### Model 3

- `distribution(dist [#])` use `dist` distribution for errors (may be `gaussian`, `normal`, `t`, `or` `ged`; default is `gaussian`)
- `het(varlist)` include `varlist` in the specification of the conditional variance
- `savesspace` conserve memory during estimation
Priming

- `arch0(xb)` compute priming values on the basis of the expected unconditional variance; the default
- `arch0(xb0)` compute priming values on the basis of the estimated variance of the residuals from OLS
- `arch0(xbwt)` compute priming values on the basis of the weighted sum of squares from OLS residuals
- `arch0(xb0wt)` compute priming values on the basis of the weighted sum of squares from OLS residuals, with more weight at earlier times
- `arch0(zero)` set priming values of ARCH terms to zero
- `arch0(#)` set priming values of ARCH terms to `#`
- `arma0(zero)` set all priming values of ARMA terms to zero; the default
- `arma0(p)` begin estimation after observation `p`, where `p` is the maximum AR lag in model
- `arma0(q)` begin estimation after observation `q`, where `q` is the maximum MA lag in model
- `arma0(pq)` begin estimation after observation `(p + q)`
- `arma0(#)` set priming values of ARMA terms to `#`
- `condobs(#)` set conditioning observations at the start of the sample to `#`

SE/Robust

- `vce(vcetype)` `vcetype` may be `opg`, `robust`, or `oim`

Reporting

- `level(#)` set confidence level; default is `level(95)`
- `detail` report list of gaps in time series
- `nocons` do not display constraints
- `display_options` control columns and column formats, row spacing, and line width

Maximization

- `maximize_options` control the maximization process; seldom used
- `collinear` keep collinear variables
- `coeflegend` display legend instead of statistics

You must `tsset` your data before using `arch`; see `[TS] tsset.`

`depvar` and `varlist` may contain time-series operators; see `[U] 11.4.4 Time-series varlists.
by, `fp`, `rolling`, `statsby`, and `xi` are allowed; see `[U] 11.1.10 Prefix commands.
iweights are allowed; see `[U] 11.1.6 weight.
collinear and `coeflegend` do not appear in the dialog box.
See `[U] 20 Estimation and postestimation commands` for more capabilities of estimation commands.

To fit an ARCH(#_m) model with Gaussian errors, type

```
    . arch depvar ..., arch(1/#_m)
```

To fit a GARCH(#_m,#_k) model assuming that the errors follow Student’s t distribution with 7 degrees of freedom, type

```
    . arch depvar ..., arch(1/#_m) garch(1/#_k) distribution(t 7)
```

You can also fit many other models.
Details of syntax

The basic model \texttt{arch} fits is

\begin{equation}
    y_t = x_t \beta + \epsilon_t
\end{equation}

\begin{equation}
    \text{Var}(\epsilon_t) = \sigma_t^2 = \gamma_0 + A(\sigma, \epsilon) + B(\sigma, \epsilon)^2
\end{equation}

The $y_t$ equation may optionally include ARCH-in-mean and ARMA terms:

\begin{equation}
    y_t = x_t \beta + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t
\end{equation}

If no options are specified, $A() = B() = 0$, and the model collapses to linear regression. The following options add to $A()$ (\(\alpha, \gamma, \) and \(\kappa\) represent parameters to be estimated):

<table>
<thead>
<tr>
<th>Option</th>
<th>Terms added to $A()$</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{arch()}</td>
<td>$A() = A() + \alpha_{1,1} \epsilon_{t-1}^2 + \alpha_{1,2} \epsilon_{t-2}^2 + \cdots$</td>
</tr>
<tr>
<td>\texttt{garch()}</td>
<td>$A() = A() + \alpha_{2,1} \sigma_{t-1}^2 + \alpha_{2,2} \sigma_{t-2}^2 + \cdots$</td>
</tr>
<tr>
<td>\texttt{saarch()}</td>
<td>$A() = A() + \alpha_{3,1} \epsilon_{t-1} + \alpha_{3,2} \epsilon_{t-2} + \cdots$</td>
</tr>
<tr>
<td>\texttt{tarch()}</td>
<td>$A() = A() + \alpha_{4,1} \epsilon_{t-1}^2 (\epsilon_{t-1} &gt; 0) + \alpha_{4,2} \epsilon_{t-2}^2 (\epsilon_{t-2} &gt; 0) + \cdots$</td>
</tr>
<tr>
<td>\texttt{aarch()}</td>
<td>$A() = A() + \alpha_{5,1} (</td>
</tr>
<tr>
<td>\texttt{narch()}</td>
<td>$A() = A() + \alpha_{6,1} (\epsilon_{t-1} - \kappa_{6,1})^2 + \alpha_{6,2} (\epsilon_{t-2} - \kappa_{6,2})^2 + \cdots$</td>
</tr>
<tr>
<td>\texttt{narchk()}</td>
<td>$A() = A() + \alpha_{7,1} (\epsilon_{t-1} - \kappa\gamma)^2 + \alpha_{7,2} (\epsilon_{t-2} - \kappa\gamma)^2 + \cdots$</td>
</tr>
</tbody>
</table>

The following options add to $B()$:

<table>
<thead>
<tr>
<th>Option</th>
<th>Terms added to $B()$</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{abarch()}</td>
<td>$B() = B() + \alpha_{8,1}</td>
</tr>
<tr>
<td>\texttt{atarch()}</td>
<td>$B() = B() + \alpha_{9,1} (</td>
</tr>
<tr>
<td>\texttt{sdgarch()}</td>
<td>$B() = B() + \alpha_{10,1} \sigma_{t-1} + \alpha_{10,2} \sigma_{t-2} + \cdots$</td>
</tr>
</tbody>
</table>

Each option requires a \texttt{numlist} argument (see \cite{U} \texttt{11.1.8 numlist}), which determines the lagged terms included. \texttt{arch(1)} specifies $\alpha_{1,1} \epsilon_{t-1}^2$, \texttt{arch(2)} specifies $\alpha_{1,2} \epsilon_{t-2}^2$, \texttt{arch(1,2)} specifies $\alpha_{1,1} \epsilon_{t-1}^2 + \alpha_{1,2} \epsilon_{t-2}^2$, \texttt{arch(1/3)} specifies $\alpha_{1,1} \epsilon_{t-1}^2 + \alpha_{1,2} \epsilon_{t-2}^2 + \alpha_{1,3} \epsilon_{t-3}^2$, etc.

If the \texttt{earch()} or \texttt{egarch()} option is specified, the basic model fit is

\begin{equation}
    y_t = x_t \beta + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t
\end{equation}

\begin{equation}
    \ln \text{Var}(\epsilon_t) = \ln \sigma_t^2 = \gamma_0 + C(\ln \sigma, z) + A(\sigma, \epsilon) + B(\sigma, \epsilon)^2
\end{equation}

where $z_t = \epsilon_t / \sigma_t$. $A()$ and $B()$ are given as above, but $A()$ and $B()$ now add to $\ln \sigma_t^2$ rather than $\sigma_t^2$. (The options corresponding to $A()$ and $B()$ are rarely specified here.) $C()$ is given by
Common terms

<table>
<thead>
<tr>
<th>Option</th>
<th>Terms added to ( C() )</th>
</tr>
</thead>
<tbody>
<tr>
<td>earch()</td>
<td>( C() = C() + \alpha_{11,1} z_{t-1} + \gamma_{11,1} (</td>
</tr>
<tr>
<td>egarch()</td>
<td>( C() = C() + \alpha_{12,1} \ln \sigma_{t-1}^2 + \alpha_{12,2} \ln \sigma_{t-2}^2 + \cdots )</td>
</tr>
</tbody>
</table>

Instead, if the \( \text{parc}() \), \( \text{tparch}() \), \( \text{aparch}() \), \( \text{nparch}() \), \( \text{nparchk}() \), or \( \text{pgarch}() \) options are specified, the basic model fit is

\[
y_t = x_t \beta + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t
\]

\[
\{\text{Var}(\epsilon_t)\}^{\varphi/2} = \sigma_t^{\varphi} = \gamma_0 + D(\sigma, \epsilon) + A(\sigma, \epsilon) + B(\sigma, \epsilon)^2
\]

where \( \varphi \) is a parameter to be estimated. \( A() \) and \( B() \) are given as above, but \( A() \) and \( B() \) now add to \( \sigma_t^{\varphi} \). (The options corresponding to \( A() \) and \( B() \) are rarely specified here.) \( D() \) is given by

<table>
<thead>
<tr>
<th>Option</th>
<th>Terms added to ( D() )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parc()</td>
<td>( D() = D() + \alpha_{13,1} \epsilon_{t-1} + \alpha_{13,2} \epsilon_{t-2} + \cdots )</td>
</tr>
<tr>
<td>tparch()</td>
<td>( D() = D() + \alpha_{14,1} \epsilon_{t-1}(\epsilon_{t-1} &gt; 0) + \alpha_{14,2} \epsilon_{t-2}(\epsilon_{t-2} &gt; 0) + \cdots )</td>
</tr>
<tr>
<td>aparch()</td>
<td>( D() = D() + \alpha_{15,1} (</td>
</tr>
<tr>
<td>nparch()</td>
<td>( D() = D() + \alpha_{16,1}</td>
</tr>
<tr>
<td>nparchk()</td>
<td>( D() = D() + \alpha_{17,1}</td>
</tr>
<tr>
<td>pgarch()</td>
<td>( D() = D() + \alpha_{18,1} \sigma_{t-1} + \alpha_{18,2} \sigma_{t-2} + \cdots )</td>
</tr>
</tbody>
</table>

Common models

<table>
<thead>
<tr>
<th>Common term</th>
<th>Options to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCH (Engle 1982)</td>
<td>arch()</td>
</tr>
<tr>
<td>GARCH (Bollerslev 1986)</td>
<td>arch() garch()</td>
</tr>
<tr>
<td>ARCH-in-mean (Engle, Lilien, and Robins 1987)</td>
<td>archm ( \text{arch}() ) [garch()]</td>
</tr>
<tr>
<td>GARCH with ARMA terms</td>
<td>arch() garch() ar() ma()</td>
</tr>
<tr>
<td>EGARCH (Nelson 1991)</td>
<td>earch() egarch()</td>
</tr>
<tr>
<td>TARCH, threshold ARCH (Zakoian 1994)</td>
<td>abarch() atarch() sdgarch()</td>
</tr>
<tr>
<td>GJR, form of threshold ARCH (Glosten, Jagannathan, and Runkle 1993)</td>
<td>arch() tarch() [garch()]</td>
</tr>
<tr>
<td>SAARCH, simple asymmetric ARCH (Engle 1990)</td>
<td>arch() saarch() [garch()]</td>
</tr>
<tr>
<td>PARCH, power ARCH (Higgins and Bera 1992)</td>
<td>parch() [pgarch()]</td>
</tr>
<tr>
<td>NARCH, nonlinear ARCH</td>
<td>narch() [garch()]</td>
</tr>
<tr>
<td>NARCHK, nonlinear ARCH with one shift</td>
<td>narchk() [garch()]</td>
</tr>
<tr>
<td>A-PARCH, asymmetric power ARCH (Ding, Granger, and Engle 1993)</td>
<td>aparch() [pgarch()]</td>
</tr>
<tr>
<td>NPARCH, nonlinear power ARCH</td>
<td>nparch() [pgarch()]</td>
</tr>
</tbody>
</table>
In all cases, you type

```
arch depvar [indepvars], options
```

where `options` are chosen from the table above. Each option requires that you specify as its argument a `numlist` that specifies the lags to be included. For most ARCH models, that value will be 1. For instance, to fit the classic first-order GARCH model on `cpi`, you would type

```
. arch cpi, arch(1) garch(1)
```

If you wanted to fit a first-order GARCH model of `cpi` on `wage`, you would type

```
. arch cpi wage, arch(1) garch(1)
```

If, for any of the options, you want first- and second-order terms, specify `optionname(1/2)`. Specifying `garch(1) arch(1/2)` would fit a GARCH model with first- and second-order ARCH terms. If you specified `arch(2)`, only the lag 2 term would be included.
Reading arch output

The regression table reported by arch when using the normal distribution for the errors will appear as

| op.depvar | Coef. | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|------------|-------|-----------|-----|------|----------------------|
| depvar     |       |           |     |      |                      |
| x1         | # ... |           |     |      |                      |
| x2         |       |           |     |      |                      |
| L1.        | # ... |           |     |      |                      |
| L2.        | # ... |           |     |      |                      |
| _cons      | # ... |           |     |      |                      |
| ARCHM      |       |           |     |      |                      |
| sigma2     | # ... |           |     |      |                      |
| ARMA       |       |           |     |      |                      |
| ar         | # ... |           |     |      |                      |
| L1.        |       |           |     |      |                      |
| ma         | # ... |           |     |      |                      |
| L1.        |       |           |     |      |                      |
| HET        |       |           |     |      |                      |
| z1         | # ... |           |     |      |                      |
| z2         |       |           |     |      |                      |
| L1.        | # ... |           |     |      |                      |
| L2.        | # ... |           |     |      |                      |
| ARCH       |       |           |     |      |                      |
| arch       | # ... |           |     |      |                      |
| L1.        |       |           |     |      |                      |
| garch      | # ... |           |     |      |                      |
| L1.        |       |           |     |      |                      |
| aparch     | # ... |           |     |      |                      |
| L1.        |       |           |     |      |                      |
| etc.       |       |           |     |      |                      |
| _cons      | # ... |           |     |      |                      |
| POWER      |       |           |     |      |                      |
| power      | # ... |           |     |      |                      |

Dividing lines separate “equations”.

The first one, two, or three equations report the mean model:

\[ y_t = x_t \beta + \sum \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t \]

The first equation reports \( \beta \), and the equation will be named \([depvar]\); if you fit a model on \( d.cpi \), the first equation would be named \([cpi]\). In Stata, the coefficient on \( x1 \) in the above example could be referred to as \([depvar]_b[x1]\). The coefficient on the lag 2 value of \( x2 \) would be referred to as \([depvar]_b[L2.x2]\). Such notation would be used, for instance, in a later \text{test} command; see \[R\] test.
The [ARCHM] equation reports the $\psi$ coefficients if your model includes ARCH-in-mean terms; see options discussed under the Model 2 tab below. Most ARCH-in-mean models include only a contemporaneous variance term, so the term $\sum_i \psi_i g(\sigma^2_{t-i})$ becomes $\psi \sigma^2_t$. The coefficient $\psi$ will be [ARCHM]_b[ sigma2]. If your model includes lags of $\sigma^2_t$, the additional coefficients will be [ARCHM]_b[L1.sigma2], and so on. If you specify a transformation $g()$ (option archmexp()), the coefficients will be [ARCHM]_b[ sigma2ex], [ARCHM]_b[L1.sigma2ex], and so on. sigma2ex refers to $g(\sigma^2_t)$, the transformed value of the conditional variance.

The [ARMA] equation reports the ARMA coefficients if your model includes them; see options discussed under the Model 2 tab below. This equation includes one or two “variables” named ar and ma. In later test statements, you could refer to the coefficient on the first lag of the autoregressive term by typing [ARMA]_b[L1.ar] or simply [ARMA]_b[L.ar] (the L operator is assumed to be lag 1 if you do not specify otherwise). The second lag on the moving-average term, if there were one, could be referred to by typing [ARMA]_b[L2.ma].

The next one, two, or three equations report the variance model.

The [HET] equation reports the multiplicative heteroskedasticity if the model includes it. When you fit such a model, you specify the variables (and their lags), determining the multiplicative heteroskedasticity; after estimation, their coefficients are simply [HET]_b[op.varname].

The [ARCH] equation reports the ARCH, GARCH, etc., terms by referring to “variables” arch, garch, and so on. For instance, if you specified arch(1) garch(1) when you fit the model, the conditional variance is given by $\sigma^2_t = \gamma_0 + \alpha_{1,1} \epsilon^2_{t-1} + \alpha_{2,1} \sigma^2_{t-1}$. The coefficients would be named [ARCH]_b[_cons] ($\gamma_0$), [ARCH]_b[L.arch] ($\alpha_{1,1}$), and [ARCH]_b[L.garch] ($\alpha_{2,1}$).

The [POWER] equation appears only if you are fitting a variance model in the form of (3) above; the estimated $\varphi$ is the coefficient [POWER]_b[ power].

Also, if you use the distribution() option and specify either Student’s $t$ or the generalized error distribution but do not specify the degree-of-freedom or shape parameter, then you will see two additional rows in the table. The final row contains the estimated degree-of-freedom or shape parameter. Immediately preceding the final row is a transformed version of the parameter that arch used during estimation to ensure that the degree-of-freedom parameter is greater than two or that the shape parameter is positive.

The naming convention for estimated ARCH, GARCH, etc., parameters is as follows (definitions for parameters $\alpha_i$, $\gamma_i$, and $\kappa_i$ can be found in the tables for $A()$, $B()$, $C()$, and $D()$ above):
Options

arch(numlist) specifies the ARCH terms (lags of $\epsilon_t^2$).

Specify arch(1) to include first-order terms, arch(1/2) to specify first- and second-order terms, arch(1/3) to specify first-, second-, and third-order terms, etc. Terms may be omitted. Specify arch(1/3 5) to specify terms with lags 1, 2, 3, and 5. All the options work this way.

arch() may not be specified with aarch(), narch(), narchk(), nparchk(), or nparch(), as this would result in collinear terms.

garch(numlist) specifies the GARCH terms (lags of $\sigma_t^2$).

saarch(numlist) specifies the simple asymmetric ARCH terms. Adding these terms is one way to make the standard ARCH and GARCH models respond asymmetrically to positive and negative innovations. Specifying saarch() with arch() and garch() corresponds to the SAARCH model of Engle (1990).

saarch() may not be specified with narch(), narchk(), nparchk(), or nparch(), as this would result in collinear terms.

tarch(numlist) specifies the threshold ARCH terms. Adding these is another way to make the standard ARCH and GARCH models respond asymmetrically to positive and negative innovations. Specifying tarch() with arch() and garch() corresponds to one form of the GJR model (Glosten, Jagannathan, and Runkle 1993).

tarch() may not be specified with tparch() or aarch(), as this would result in collinear terms.

aarch(numlist) specifies the lags of the two-parameter term $\alpha_i (|\epsilon_t| + \gamma_i \epsilon_t)^2$. This term provides the same underlying form of asymmetry as including arch() and tarch(), but it is expressed in a different way.

aarch() may not be specified with arch() or tarch(), as this would result in collinear terms.
narch(numlist) specifies the lags of the two-parameter term $\alpha_i(\epsilon_t - \kappa)^2$. This term allows the minimum conditional variance to occur at a value of lagged innovations other than zero. For any term specified at lag $L$, the minimum contribution to conditional variance of that lag occurs when $\epsilon_{t-L}^2 = \kappa_L$—the squared innovations at that lag are equal to the estimated constant $\kappa_L$.

narch() may not be specified with arch(), saarch(), narchk(), nparchk(), or nparch(), as this would result in collinear terms.

narchk(numlist) specifies the lags of the two-parameter term $\alpha_i(\epsilon_t - \kappa)^2$; this is a variation of narch() with $\kappa$ held constant for all lags.

narchk() may not be specified with arch(), saarch(), narch(), nparchk(), or nparch(), as this would result in collinear terms.

abarch(numlist) specifies lags of the term $|\epsilon_t|$.

atarch(numlist) specifies lags of $|\epsilon_t| (\epsilon_t > 0)$, where $(\epsilon_t > 0)$ represents the indicator function returning 1 when true and 0 when false. Like the TARCH terms, these ATARCH terms allow the effect of unanticipated innovations to be asymmetric about zero.

sdgarch(numlist) specifies lags of $\sigma_t$. Combining atarch(), abarch(), and sdgarch() produces the model by Zakoian (1994) that the author called the TARCH model. The acronym TARCH, however, refers to any model using thresholding to obtain asymmetry.

earch(numlist) specifies lags of the two-parameter term $\alpha z_t + \gamma (|z_t| - \sqrt{2/\pi})$. These terms represent the influence of news—lagged innovations—in Nelson’s (1991) EGARCH model. For these terms, $z_t = \epsilon_t / \sigma_t$, and arch assumes $z_t \sim N(0,1)$. Nelson derived the general form of an EGARCH model for any assumed distribution and performed estimation assuming a generalized error distribution (GED). See Hamilton (1994) for a derivation where $z_t$ is assumed normal. The $z_t$ terms can be parameterized in either of these two equivalent ways. arch uses Nelson’s original parameterization; see Hamilton (1994) for an equivalent alternative.

egarch(numlist) specifies lags of $\ln(\sigma_t^2)$.

For the following options, the model is parameterized in terms of $h(\epsilon_t)^\varphi$ and $\sigma_t^\varphi$. One $\varphi$ is estimated, even when more than one option is specified.

parch(numlist) specifies lags of $|\epsilon_t|^\varphi$. parch() combined with pgarch() corresponds to the class of nonlinear models of conditional variance suggested by Higgins and Bera (1992).

tparch(numlist) specifies lags of $(\epsilon_t > 0)|\epsilon_t|^\varphi$, where $(\epsilon_t > 0)$ represents the indicator function returning 1 when true and 0 when false. As with tarch(), tparech() specifies terms that allow for a differential impact of “good” (positive innovations) and “bad” (negative innovations) news for lags specified by numlist.

tarch() may not be specified with tarch(), as this would result in collinear terms.

aparch(numlist) specifies lags of the two-parameter term $\alpha (|\epsilon_t| + \gamma \epsilon_t)^\varphi$. This asymmetric power ARCH model, A-PARCH, was proposed by Ding, Granger, and Engle (1993) and corresponds to a Box–Cox function in the lagged innovations. The authors fit the original A-PARCH model on more than 16,000 daily observations of the Standard and Poor’s 500, and for good reason. As the number of parameters and the flexibility of the specification increase, more data are required to estimate the parameters of the conditional heteroskedasticity. See Ding, Granger, and Engle (1993) for a discussion of how seven popular ARCH models nest within the A-PARCH model.

When $\gamma$ goes to 1, the full term goes to zero for many observations and can then be numerically unstable.
nparch() specifies lags of the two-parameter term $\alpha |\epsilon_t - \kappa|^2$.

nparch() may not be specified with arch(), saarch(), narch(), narchk(), or nparchk(), as this would result in collinear terms.

nparchk() specifies lags of the two-parameter term $\alpha |\epsilon_t - \kappa|^2$; this is a variation of nparch() with $\kappa$ held constant for all lags. This is the direct analog of narchk(), except for the power of $\varphi$. nparchk() corresponds to an extended form of the model of Higgins and Bera (1992) as presented by Bollerslev, Engle, and Nelson (1994). nparchk() would typically be combined with the pgarch() option.

nparchk() may not be specified with arch(), saarch(), narch(), narchk(), or nparch(), as this would result in collinear terms.

pgarch() specifies lags of $\sigma_t^2$.

constraints(); see [R] Estimation options.

archm specifies that an ARCH-in-mean term be included in the specification of the mean equation. This term allows the expected value of depvar to depend on the conditional variance. ARCH-in-mean is most commonly used in evaluating financial time series when a theory supports a tradeoff between asset risk and return. By default, no ARCH-in-mean terms are included in the model.

archm specifies that the contemporaneous expected conditional variance be included in the mean equation. For example, typing

```
. arch y x, archm arch(1)
```

specifies the model

$$y_t = \beta_0 + \beta_1 x_t + \psi \sigma_t^2 + \epsilon_t$$

$$\sigma_t^2 = \gamma_0 + \gamma \epsilon_{t-1}^2$$

archmlags() is an expansion of archm that includes lags of the conditional variance $\sigma_t^2$ in the mean equation. To specify a contemporaneous and once-lagged variance, specify either archm archmlags(1) or archmlags(0/1).

archmexp() applies the transformation in exp to any ARCH-in-mean terms in the model. The expression should contain an X wherever a value of the conditional variance is to enter the expression. This option can be used to produce the commonly used ARCH-in-mean of the conditional standard deviation. With the example from archm, typing

```
. arch y x, archm arch(1) archmexp(sqrt(X))
```

specifies the mean equation $y_t = \beta_0 + \beta_1 x_t + \psi \sigma_t + \epsilon_t$. Alternatively, typing

```
. arch y x, archm arch(1) archmexp(1/sqrt(X))
```

specifies $y_t = \beta_0 + \beta_1 x_t + \psi / \sigma_t + \epsilon_t$.

arima(#) is an alternative, shorthand notation for specifying autoregressive models in the dependent variable. The dependent variable and any independent variables are differenced # times, 1 through #p lags of autocorrelations are included, and 1 through #q lags of moving averages are included. For example, the specification

```
. arch y, arima(2,1,3)
```

is equivalent to

```
. arch D.y, ar(1/2) ma(1/3)
```
The former is easier to write for classic ARIMA models of the mean equation, but it is not nearly as expressive as the latter. If gaps in the AR or MA lags are to be modeled, or if different operators are to be applied to independent variables, the latter syntax is required.

`ar(numlist)` specifies the autoregressive terms of the structural model disturbance to be included in the model. For example, `ar(1/3)` specifies that lags 1, 2, and 3 of the structural disturbance be included in the model. `ar(1,4)` specifies that lags 1 and 4 be included, possibly to account for quarterly effects.

If the model does not contain regressors, these terms can also be considered autoregressive terms for the dependent variable; see [TS] `arima`.

`ma(numlist)` specifies the moving-average terms to be included in the model. These are the terms for the lagged innovations or white-noise disturbances.

`distribution(dist [#])` specifies the distribution to assume for the error term. `dist` may be `gaussian`, `normal`, `t`, or `ged`. `gaussian` and `normal` are synonyms, and `#` cannot be specified with them.

If `distribution(t)` is specified, `arch` assumes that the errors follow Student’s t distribution, and the degree-of-freedom parameter is estimated along with the other parameters of the model. If `distribution(t #)` is specified, then `arch` uses Student’s t distribution with `#` degrees of freedom. `#` must be greater than 2.

If `distribution(ged)` is specified, `arch` assumes that the errors have a generalized error distribution, and the shape parameter is estimated along with the other parameters of the model. If `distribution(ged #)` is specified, then `arch` uses the generalized error distribution with shape parameter `#`. `#` must be positive. The generalized error distribution is identical to the normal distribution when the shape parameter equals 2.

`het(varlist)` specifies that `varlist` be included in the specification of the conditional variance. `varlist` may contain time-series operators. This `varlist` enters the variance specification collectively as multiplicative heteroskedasticity; see Judge et al. (1985). If `het()` is not specified, the model will not contain multiplicative heteroskedasticity.

Assume that the conditional variance depends on variables `x` and `w` and has an ARCH(1) component. We request this specification by using the `het(x w) arch(1)` options, and this corresponds to the conditional-variance model

\[
\sigma_t^2 = \exp(\lambda_0 + \lambda_1 x_t + \lambda_2 w_t) + \alpha \epsilon_{t-1}^2
\]

Multiplicative heteroskedasticity enters differently with an EGARCH model because the variance is already specified in logs. For the `het(x w) earch(1) egarch(1)` options, the variance model is

\[
\ln(\sigma_t^2) = \lambda_0 + \lambda_1 x_t + \lambda_2 w_t + \alpha z_{t-1} + \gamma(|z_{t-1}| - \sqrt{2/\pi}) + \delta \ln(\sigma_{t-1}^2)
\]

`savespace` conserves memory by retaining only those variables required for estimation. The original dataset is restored after estimation. This option is rarely used and should be specified only if there is insufficient memory to fit a model without the option. `arch` requires considerably more temporary storage during estimation than most estimation commands in Stata.
arch — Autoregressive conditional heteroskedasticity (ARCH) family of estimators 25

Priming

arch0(cond_method) is a rarely used option that specifies how to compute the conditioning (presample or priming) values for $\sigma^2_t$ and $\epsilon^2_t$. In the presample period, it is assumed that $\sigma^2_t = \epsilon^2_t$ and that this value is constant. If arch0() is not specified, the priming values are computed as the expected unconditional variance given the current estimates of the $\beta$ coefficients and any ARMA parameters.

arch0(xb), the default, specifies that the priming values are the expected unconditional variance of the model, which is $\sum_{1}^{T} \hat{\epsilon}^2_t / T$, where $\hat{\epsilon}_t$ is computed from the mean equation and any ARMA terms.

arch0(xb0) specifies that the priming values are the estimated variance of the residuals from an OLS estimate of the mean equation.

arch0(xbwt) specifies that the priming values are the weighted sum of the $\hat{\epsilon}^2_t$ from the current conditional mean equation (and ARMA terms) that places more weight on estimates of $\epsilon^2_t$ at the beginning of the sample.

arch0(xb0wt) specifies that the priming values are the weighted sum of the $\hat{\epsilon}^2_t$ from an OLS estimate of the mean equation (and ARMA terms) that places more weight on estimates of $\epsilon^2_t$ at the beginning of the sample.

arch0(zero) specifies that the priming values are 0. Unlike the priming values for ARIMA models, 0 is generally not a consistent estimate of the presample conditional variance or squared innovations.

arch0(#) specifies that $\sigma^2_t = \epsilon^2_t = # \text{ for any specified nonnegative #.}$ Thus arch0(0) is equivalent to arch0(zero).

arma0(cond_method) is a rarely used option that specifies how the $\epsilon_t$ values are initialized at the beginning of the sample for the ARMA component, if the model has one. This option has an effect only when AR or MA terms are included in the model (the ar(), ma(), or arima() options specified).

arma0(zero), the default, specifies that all priming values of $\epsilon_t$ be taken as 0. This fits the model over the entire requested sample and takes $\epsilon_t$ as its expected value of 0 for all lags required by the ARMA terms; see Judge et al. (1985).

arma0(p), arma0(q), and arma0(pq) specify that estimation begin after priming the recursions for a certain number of observations. $p$ specifies that estimation begin after the $p$th observation in the sample, where $p$ is the maximum AR lag in the model; $q$ specifies that estimation begin after the $q$th observation in the sample, where $q$ is the maximum MA lag in the model; and $pq$ specifies that estimation begin after the $(p + q)$th observation in the sample.

During the priming period, the recursions necessary to generate predicted disturbances are performed, but results are used only to initialize preestimation values of $\epsilon_t$. To understand the definition of preestimation, say that you fit a model in 10/100. If the model is specified with ar1(1,2), preestimation refers to observations 10 and 11.

The ARCH terms $\sigma^2_t$ and $\epsilon^2_t$ are also updated over these observations. Any required lags of $\epsilon_t$ before the priming period are taken to be their expected value of 0, and $\epsilon^2_t$ and $\sigma^2_t$ take the values specified in arch0().

arma0(#) specifies that the presample values of $\epsilon_t$ are to be taken as # for all lags required by the ARMA terms. Thus arma0(0) is equivalent to arma0(zero).

condobs(#) is a rarely used option that specifies a fixed number of conditioning observations at the start of the sample. Over these priming observations, the recursions necessary to generate predicted disturbances are performed, but only to initialize preestimation values of $\epsilon_t$, $\epsilon^2_t$, and $\sigma^2_t$. 
Any required lags of $\epsilon_t$ before the initialization period are taken to be their expected value of 0 (or the value specified in \texttt{arma0()}). and required values of $\epsilon_t^2$ and $\sigma_t^2$ assume the values specified by \texttt{arch0()}. \texttt{condobs()} can be used if conditioning observations are desired for the lags in the ARCH terms of the model. If \texttt{arma()} is also specified, the maximum number of conditioning observations required by \texttt{arma()} and \texttt{condobs(#)} is used.

\texttt{SE/Robust}

\texttt{vce(vcetype)} specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (\texttt{robust}) and that are derived from asymptotic theory (\texttt{oim}, \texttt{opg}); see \texttt{[R] vce option}.

For ARCH models, the robust or quasi–maximum likelihood estimates (QMLE) of variance are robust to symmetric nonnormality in the disturbances. The robust variance estimates generally are not robust to functional misspecification of the mean equation; see Bollerslev and Wooldridge (1992).

The robust variance estimates computed by arch are based on the full Huber/White/sandwich formulation, as discussed in \texttt{[P] _robust}. Many other software packages report robust estimates that set some terms to their expectations of zero (Bollerslev and Wooldridge 1992), which saves them from calculating second derivatives of the log-likelihood function.

\texttt{Reporting}

\texttt{level(#)}; see \texttt{[R] Estimation options}.

\texttt{detail} specifies that a detailed list of any gaps in the series be reported, including gaps due to missing observations or missing data for the dependent variable or independent variables. \texttt{nocnsreport}; see \texttt{[R] Estimation options}.

\texttt{display_options: noci, nopvalues, vsquish, cformat(\texttt{\%fmt}), pformat(\texttt{\%fmt}), sformat(\texttt{\%fmt}), and nolstretch}; see \texttt{[R] Estimation options}.

\texttt{Maximization}

\texttt{maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), gtolerance(#), nrtolerance(#), nonrtolerance, and from(init_specs)}; see \texttt{[R] Maximize} for all options except \texttt{gtolerance()}, and see below for information on \texttt{gtolerance()}.

These options are often more important for ARCH models than for other maximum likelihood models because of convergence problems associated with ARCH models—ARCH model likelihoods are notoriously difficult to maximize.

Setting \texttt{technique()} to something other than the default or BHHH changes the \texttt{vcetype} to \texttt{vce(oim)}.

The following options are all related to maximization and are either particularly important in fitting ARCH models or not available for most other estimators.

\texttt{gtolerance(#)} specifies the tolerance for the gradient relative to the coefficients. When $|g_i| \leq \texttt{gtolerance()}$ for all parameters $b_i$ and the corresponding elements of the gradient $g_i$, the gradient tolerance criterion is met. The default gradient tolerance for arch is $\texttt{gtolerance(.05)}$.

\texttt{gtolerance(999)} may be specified to disable the gradient criterion. If the optimizer becomes stuck with repeated “(backed up)” messages, the gradient probably still contains substantial values, but an uphill direction cannot be found for the likelihood. With this option, results can often be obtained, but whether the global maximum likelihood has been found is unclear.
When the maximization is not going well, it is also possible to set the maximum number of iterations (see [R] Maximize) to the point where the optimizer appears to be stuck and to inspect the estimation results at that point.

*from(init_specs)* specifies the initial values of the coefficients. ARCH models may be sensitive to initial values and may have coefficient values that correspond to local maximums. The default starting values are obtained via a series of regressions, producing results that, on the basis of asymptotic theory, are consistent for the \( \beta \) and ARMA parameters and generally reasonable for the rest. Nevertheless, these values may not always be feasible in that the likelihood function cannot be evaluated at the initial values *arch* first chooses. In such cases, the estimation is restarted with ARCH and ARMA parameters initialized to zero. It is possible, but unlikely, that even these values will be infeasible and that you will have to supply initial values yourself.

The standard syntax for *from()* accepts a matrix, a list of values, or coefficient name value pairs; see [R] Maximize. *arch* also allows the following:

- *from(archb0)* sets the starting value for all the ARCH/GARCH/... parameters in the conditional-variance equation to 0.
- *from(armab0)* sets the starting value for all ARMA parameters in the model to 0.
- *from(archb0 armab0)* sets the starting value for all ARCH/GARCH/... and ARMA parameters to 0.

The following options are available with *arch* but are not shown in the dialog box:

*collinear, coeflegend*; see [R] Estimation options.

**Remarks and examples**

The volatility of a series is not constant through time; periods of relatively low volatility and periods of relatively high volatility tend to be grouped together. This is a commonly observed characteristic of economic time series and is even more pronounced in many frequently sampled financial series. ARCH models seek to estimate this time-dependent volatility as a function of observed prior volatility. Sometimes the model of volatility is of more interest than the model of the conditional mean. As implemented in *arch*, the volatility model may also include regressors to account for a structural component in the volatility—usually referred to as multiplicative heteroskedasticity.

ARCH models were introduced by Engle (1982) in a study of inflation rates, and there has since been a barrage of proposed parametric and nonparametric specifications of autoregressive conditional heteroskedasticity. Overviews of the literature can found in Bollerslev, Engle, and Nelson (1994) and Bollerslev, Chou, and Kroner (1992). Introductions to basic ARCH models appear in many general econometrics texts, including Davidson and MacKinnon (1993), Kmenta (1997), Stock and Watson (2019), and Wooldridge (2020). Harvey (1989) and Enders (2004) provide introductions to ARCH in the larger context of econometric time-series modeling, and Hamilton (1994) gives considerably more detail in the same context. Becketti (2013, chap. 8) provides a simple introduction to ARCH modeling with an emphasis on how to use Stata's *arch* command.

*arch* fits models of autoregressive conditional heteroskedasticity (ARCH, GARCH, etc.) using conditional maximum likelihood. By “conditional”, we mean that the likelihood is computed based on an assumed or estimated set of priming values for the squared innovations \( \epsilon_t^2 \) and variances \( \sigma_t^2 \) prior to the estimation sample; see Hamilton (1994) or Bollerslev (1986). Sometimes more conditioning is done on the first \( a \), \( g \), or \( a + g \) observations in the sample, where \( a \) is the maximum ARCH term lag and \( g \) is the maximum GARCH term lag (or the maximum lags from the other ARCH family terms).
The original ARCH model proposed by Engle (1982) modeled the variance of a regression model’s disturbances as a linear function of lagged values of the squared regression disturbances. We can write an ARCH($m$) model as

$$y_t = x_t \beta + \epsilon_t$$  \hspace{1cm} \text{(conditional mean)}

$$\sigma_t^2 = \gamma_0 + \gamma_1 \epsilon_{t-1}^2 + \gamma_2 \epsilon_{t-2}^2 + \cdots + \gamma_m \epsilon_{t-m}^2$$  \hspace{1cm} \text{(conditional variance)}

where

$\epsilon_t^2$ is the squared residuals (or innovations)

$\gamma_i$ are the ARCH parameters

The ARCH model has a specification for both the conditional mean and the conditional variance, and the variance is a function of the size of prior unanticipated innovations—$\epsilon_t^2$. This model was generalized by Bollerslev (1986) to include lagged values of the conditional variance—a GARCH model. The GARCH($m, k$) model is written as

$$y_t = x_t \beta + \epsilon_t$$

$$\sigma_t^2 = \gamma_0 + \gamma_1 \epsilon_{t-1}^2 + \gamma_2 \epsilon_{t-2}^2 + \cdots + \gamma_m \epsilon_{t-m}^2 + \delta_1 \sigma_{t-1}^2 + \delta_2 \sigma_{t-2}^2 + \cdots + \delta_k \sigma_{t-k}^2$$

where

$\gamma_i$ are the ARCH parameters

$\delta_i$ are the GARCH parameters

In his pioneering work, Engle (1982) assumed that the error term, $\epsilon_t$, followed a Gaussian (normal) distribution: $\epsilon_t \sim \mathcal{N}(0, \sigma_t^2)$. However, as Mandelbrot (1963) and many others have noted, the distribution of stock returns appears to be leptokurtotic, meaning that extreme stock returns are more frequent than would be expected if the returns were normally distributed. Researchers have therefore assumed other distributions that can have fatter tails than the normal distribution; arch allows you to fit models assuming the errors follow Student’s $t$ distribution or the generalized error distribution. The $t$ distribution has fatter tails than the normal distribution; as the degree-of-freedom parameter approaches infinity, the $t$ distribution converges to the normal distribution. The generalized error distribution’s tails are fatter than the normal distribution’s when the shape parameter is less than two and are thinner than the normal distribution’s when the shape parameter is greater than two.

The GARCH model of conditional variance can be considered an ARMA process in the squared innovations, although not in the variances as the equations might seem to suggest; see Hamilton (1994). Specifically, the standard GARCH model implies that the squared innovations result from

$$\epsilon_t^2 = \gamma_0 + (\gamma_1 + \delta_1) \epsilon_{t-1}^2 + (\gamma_2 + \delta_2) \epsilon_{t-2}^2 + \cdots + (\gamma_k + \delta_k) \epsilon_{t-k}^2 + w_t - \delta_1 w_{t-1} - \delta_2 w_{t-2} - \delta_3 w_{t-3}$$

where

$w_t = \epsilon_t^2 - \sigma_t^2$

$w_t$ is a white-noise process that is fundamental for $\epsilon_t^2$

One of the primary benefits of the GARCH specification is its parsimony in identifying the conditional variance. As with ARIMA models, the ARMA specification in GARCH allows the conditional variance to be modeled with fewer parameters than with an ARCH specification alone. Empirically, many series with a conditionally heteroskedastic disturbance have been adequately modeled with a GARCH(1,1) specification.
An ARMA process in the disturbances can easily be added to the mean equation. For example, the mean equation can be written with an ARMA(1, 1) disturbance as

\[ y_t = x_t \beta + \rho (y_{t-1} - x_{t-1} \beta) + \theta \epsilon_{t-1} + \epsilon_t \]

with an obvious generalization to ARMA(p, q) by adding terms; see [TS] arima for more discussion of this specification. This change affects only the conditional-variance specification in that \( \epsilon_t^2 \) now results from a different specification of the conditional mean.

Much of the literature on ARCH models focuses on alternative specifications of the variance equation. arch allows many of these specifications to be requested using the saarch() through pgarch() options, which imply that one or more terms may be changed or added to the specification of the variance equation.

These alternative specifications also address asymmetry. Both the ARCH and GARCH specifications imply a symmetric impact of innovations. Whether an innovation \( \epsilon_t^2 \) is positive or negative makes no difference to the expected variance \( \sigma_t^2 \) in the ensuing periods; only the size of the innovation matters—good news and bad news have the same effect. Many theories, however, suggest that positive and negative innovations should vary in their impact. For risk-averse investors, a large unanticipated drop in the market is more likely to lead to higher volatility than a large unanticipated increase (see Black [1976], Nelson [1991]). saarch(), tarch(), aarch(), abarch(), earch(), aparch(), and tparch() allow various specifications of asymmetric effects.

narch(), narchk(), nparch(), and nparchk() imply an asymmetric impact of a specific form. All the models considered so far have a minimum conditional variance when the lagged innovations are all zero. “No news is good news” when it comes to keeping the conditional variance small. narch(), narchk(), nparch(), and nparchk() also have a symmetric response to innovations, but they are not centered at zero. The entire news-response function (response to innovations) is shifted horizontally so that minimum variance lies at some specific positive or negative value for prior innovations.

ARCH-in-mean models allow the conditional variance of the series to influence the conditional mean. This is particularly convenient for modeling the risk–return relationship in financial series; the riskier an investment, with all else equal, the lower its expected return. ARCH-in-mean models modify the specification of the conditional mean equation to be

\[ y_t = x_t \beta + \psi \sigma_t^2 + \epsilon_t \] (ARCH-in-mean)

Although this linear form in the current conditional variance has dominated the literature, arch allows the conditional variance to enter the mean equation through a nonlinear transformation \( g() \) and for this transformed term to be included contemporaneously or lagged.

\[ y_t = x_t \beta + \psi_0 g(\sigma_t^2) + \psi_1 g(\sigma_{t-1}^2) + \psi_2 g(\sigma_{t-2}^2) + \cdots + \epsilon_t \]

Square root is the most commonly used \( g() \) transformation because researchers want to include a linear term for the conditional standard deviation, but any transform \( g() \) is allowed.

Example 1: ARCH model

Consider a simple model of the U.S. Wholesale Price Index (WPI) (Enders 2004, 87–93), which we also consider in [TS] arima. The data are quarterly over the period 1960q1 through 1990q4.

In [TS] arima, we fit a model of the continuously compounded rate of change in the WPI, \( \ln(WPI_t) - \ln(WPI_{t-1}) \). The graph of the differenced series—see [TS] arima—clearly shows periods of high volatility and other periods of relative tranquility. This makes the series a good candidate for ARCH modeling. Indeed, price indices have been a common target of ARCH models. Engle (1982) presented the original ARCH formulation in an analysis of U.K. inflation rates.
First, we fit a constant-only model by OLS and test ARCH effects by using Engle’s Lagrange multiplier test (estat archlm).

```
. use https://www.stata-press.com/data/r16/wpi1
. regress D.ln_wpi
```

```
Source | SS df MS Number of obs = 123
-------+---------------------------------------------------------------------
Model   | 0.000206697 0 . Prob > F = .
Residual | .02521709 122 .02521709 122 R-squared = .0000
Total   | .02521709 122 .02521709 122 Adj R-squared = 0.0000
```

```
D.ln_wpi | Coef. Std. Err. t P>|t| [95% Conf. Interval]
_ -cons   | 0.0108215 .0012963 8.35 0.000 .0082553 .0133878
```

```
. estat archlm, lags(1)
LM test for autoregressive conditional heteroskedasticity (ARCH)

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.366</td>
<td>1</td>
<td>0.0038</td>
</tr>
</tbody>
</table>
```

Because the LM test shows a p-value of 0.0038, which is well below 0.05, we reject the null hypothesis of no ARCH(1) effects. Thus we can further estimate the ARCH(1) parameter by specifying arch(1). See [R] regress postestimation time series for more information on Engle’s LM test.

The first-order generalized ARCH model (GARCH, Bollerslev 1986) is the most commonly used specification for the conditional variance in empirical work and is typically written GARCH(1, 1). We can estimate a GARCH(1, 1) process for the log-differenced series by typing

```
. arch D.ln_wpi, arch(1) garch(1)
```

(Setting optimization to BHHH)
We have estimated the ARCH(1) parameter to be 0.436 and the GARCH(1) parameter to be 0.454, so our fitted GARCH(1,1) model is

\[ y_t = 0.0061 + \epsilon_t \]
\[ \sigma^2_t = 0.436 \epsilon^2_{t-1} + 0.454 \sigma^2_{t-1} \]

where \( y_t = \ln(w_{pi_t}) - \ln(w_{pi_{t-1}}) \).

The model Wald test and probability are both reported as missing (.). By convention, Stata reports the model test for the mean equation. Here and fairly often for ARCH models, the mean equation consists only of a constant, and there is nothing to test.

Example 2: ARCH model with ARMA process

We can retain the GARCH(1,1) specification for the conditional variance and model the mean as an ARMA process with AR(1) and MA(1) terms as well as a fourth-lag MA term to control for quarterly seasonal effects by typing
To clarify exactly what we have estimated, we could write our model as

\[ y_t = 0.007 + 0.792 (y_{t-1} - 0.007) - 0.342 \epsilon_{t-1} + 0.245 \epsilon_{t-4} + \epsilon_t \]

\[ \sigma_t^2 = 0.204 \epsilon_{t-1}^2 + 0.695 \sigma_{t-1}^2 \]

where \( y_t = \ln(wpi_t) - \ln(wpi_{t-1}) \).

The ARCH(1) coefficient, 0.204, is not significantly different from zero, but the ARCH(1) and GARCH(1) coefficients are significant collectively. If you doubt this, you can check with `test`.
Technical note

The rather ugly iteration log on the previous result is typical, as difficulty in converging is common in ARCH models. This is actually a fairly well-behaved likelihood for an ARCH model. The “switching optimization to …” messages are standard messages from the default optimization method for arch. The “backed up” messages are typical of BFGS stepping as the BFGS Hessian is often overoptimistic, particularly during early iterations. These messages are nothing to be concerned about.

Nevertheless, watch out for the messages “BFGS stepping has contracted, resetting BFGS Hessian” and “backed up”, which can flag problems that may result in an iteration log that goes on and on. Stata will never report convergence and will never report final results. The question is, when do you give up and press Break, and if you do, what then?

If the “BFGS stepping has contracted” message occurs repeatedly (more than, say, five times), it often indicates that convergence will never be achieved. Literally, it means that the BFGS algorithm was stuck and reset its Hessian and take a steepest-descent step.

The “backed up” message, if it occurs repeatedly, also indicates problems, but only if the likelihood value is simultaneously not changing. If the message occurs repeatedly but the likelihood value is changing, as it did above, all is going well; it is just going slowly.

If you have convergence problems, you can specify options to assist the current maximization method or try a different method. Or, your model specification and data may simply lead to a likelihood that is not concave in the allowable region and thus cannot be maximized.

If you see the “backed up” message with no change in the likelihood, you can reset the gradient tolerance to a larger value. Specifying the gtolerance(999) option disables gradient checking, allowing convergence to be declared more easily. This does not guarantee that convergence will be declared, and even if it is, the global maximum likelihood may not have been found.

You can also try to specify initial values.

Finally, you can try a different maximization method; see options discussed under the Maximization tab above.

ARCH models are notorious for having convergence difficulties. Unlike in most estimators in Stata, it is common for convergence to require many steps or even to fail. This is particularly true of the explicitly nonlinear terms such as aarch(), narch(), aparch(), or archm (ARCH-in-mean), and of any model with several lags in the ARCH terms. There is not always a solution. You can try other maximization methods or different starting values, but if your data do not support your assumed ARCH structure, convergence simply may not be possible.

ARCH models can be susceptible to irrelevant regressors or unnecessary lags, whether in the specification of the conditional mean or in the conditional variance. In these situations, arch will often continue to iterate, making little to no improvement in the likelihood. We view this conservative approach as better than declaring convergence prematurely when the likelihood has not been fully
arch is estimating the conditional form of second sample moments, often with flexible functions, and that is asking much of the data.

### Technical note

if `exp` and in `range` are interpreted differently with commands accepting time-series operators. The time-series operators are resolved before the conditions are tested, which may lead to some confusion. Note the results of the following `list` commands:

```
. use https://www.stata-press.com/data/r16/archxmpl
. list t y l.y in 5/10
    | t     | y     | L.y   |
  -------------|-------------|-------|
        5. | 1961q1 | 30.8   | 30.7  |
        6. | 1961q2 | 30.5   | 30.8  |
        7. | 1961q3 | 30.5   | 30.5  |
        8. | 1961q4 | 30.6   | 30.5  |
        9. | 1962q1 | 30.7   | 30.6  |
       10. | 1962q2 | 30.6   | 30.7  |
. keep in 5/10
(118 observations deleted)
. list t y l.y
    | t     | y     | L.y   |
  -------------|-------------|-------|
        1. | 1961q1 | 30.8   |     . |
        2. | 1961q2 | 30.5   | 30.8  |
        3. | 1961q3 | 30.5   | 30.5  |
        4. | 1961q4 | 30.6   | 30.5  |
        5. | 1962q1 | 30.7   | 30.6  |
        6. | 1962q2 | 30.6   | 30.7  |
```

We have one more lagged observation for `y` in the first case: `l.y` was resolved before the in restriction was applied. In the second case, the dataset no longer contains the value of `y` to compute the first lag. This means that

```
. use https://www.stata-press.com/data/r16/archxmpl, clear
. arch y l.x if twithin(1962q2, 1990q3), arch(1)
```

is not the same as

```
. keep if twithin(1962q2, 1990q3)
. arch y l.x, arch(1)
```
> Example 3: Asymmetric effects—EGARCH model

Continuing with the WPI data, we might be concerned that the economy as a whole responds differently to unanticipated increases in wholesale prices than it does to unanticipated decreases. Perhaps unanticipated increases lead to cash flow issues that affect inventories and lead to more volatility. We can see if the data support this supposition by specifying an ARCH model that allows an asymmetric effect of “news”—innovations or unanticipated changes. One of the most popular such models is EGARCH (Nelson 1991). The full first-order EGARCH model for the WPI can be specified as follows:

```
use https://www.stata-press.com/data/r16/wpi1, clear
arch D.ln_wpi, ar(1) ma(1 4) earch(1) egarch(1)
(setting optimization to BHHH)
Iteration 0:  log likelihood = 227.5251
Iteration 1:  log likelihood = 381.68412
(output omitted)
Iteration 23: log likelihood = 405.31453
```

ARCH family regression -- ARMA disturbances

<table>
<thead>
<tr>
<th>D.ln_wpi</th>
<th>OPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln_wpi _cons</td>
<td>0.0087342</td>
</tr>
<tr>
<td></td>
<td>0.0034004</td>
</tr>
<tr>
<td></td>
<td>2.57</td>
</tr>
<tr>
<td></td>
<td>0.010</td>
</tr>
<tr>
<td></td>
<td>0.0020695</td>
</tr>
<tr>
<td></td>
<td>0.0153989</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARMA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ar L1.</td>
<td>0.7692139</td>
</tr>
<tr>
<td></td>
<td>0.0968393</td>
</tr>
<tr>
<td></td>
<td>7.94</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>0.5794124</td>
</tr>
<tr>
<td></td>
<td>0.9590154</td>
</tr>
<tr>
<td>ma L1.</td>
<td>-0.3554623</td>
</tr>
<tr>
<td></td>
<td>0.1265721</td>
</tr>
<tr>
<td></td>
<td>-2.81</td>
</tr>
<tr>
<td></td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>-0.6035391</td>
</tr>
<tr>
<td></td>
<td>-0.1073855</td>
</tr>
<tr>
<td>L4.</td>
<td>0.2414626</td>
</tr>
<tr>
<td></td>
<td>0.0863834</td>
</tr>
<tr>
<td></td>
<td>2.80</td>
</tr>
<tr>
<td></td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>-0.0721543</td>
</tr>
<tr>
<td></td>
<td>0.4107709</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARCH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>earch L1.</td>
<td>0.4063939</td>
</tr>
<tr>
<td></td>
<td>0.11635</td>
</tr>
<tr>
<td></td>
<td>3.49</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>0.1783521</td>
</tr>
<tr>
<td></td>
<td>0.6344358</td>
</tr>
<tr>
<td>earch_a L1.</td>
<td>0.2467327</td>
</tr>
<tr>
<td></td>
<td>0.1233357</td>
</tr>
<tr>
<td></td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>0.045</td>
</tr>
<tr>
<td></td>
<td>0.0049993</td>
</tr>
<tr>
<td></td>
<td>0.4884662</td>
</tr>
<tr>
<td>egarch L1.</td>
<td>0.8417332</td>
</tr>
<tr>
<td></td>
<td>0.0704074</td>
</tr>
<tr>
<td></td>
<td>11.96</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>0.7037372</td>
</tr>
<tr>
<td></td>
<td>0.9797291</td>
</tr>
<tr>
<td>_cons</td>
<td>-1.488366</td>
</tr>
<tr>
<td></td>
<td>0.6604354</td>
</tr>
<tr>
<td></td>
<td>-2.25</td>
</tr>
<tr>
<td></td>
<td>0.024</td>
</tr>
<tr>
<td></td>
<td>-2.782795</td>
</tr>
<tr>
<td></td>
<td>-1.1939363</td>
</tr>
</tbody>
</table>

Our result for the variance is

\[
\ln(\sigma_t^2) = -1.49 + 0.406 z_{t-1} + 0.247 (|z_{t-1}| - \sqrt{2/\pi}) + 0.842 \ln(\sigma_{t-1}^2)
\]

where \(z_t = \epsilon_t/\sigma_t\), which is distributed as \(N(0, 1)\).

This is a strong indication for a leverage effect. The positive \(L1.earch\) coefficient implies that positive innovations (unanticipated price increases) are more destabilizing than negative innovations. The effect appears strong (0.406) and is substantially larger than the symmetric effect (0.247). In fact, the relative scales of the two coefficients imply that the positive leverage completely dominates the symmetric effect.
This can readily be seen if we plot what is often referred to as the news-response or news-impact function. This curve shows the resulting conditional variance as a function of unanticipated news, in the form of innovations, that is, the conditional variance $\sigma^2_t$ as a function of $\epsilon_t$. Thus we must evaluate $\sigma^2_t$ for various values of $\epsilon_t$—say, $-4$ to $4$—and then graph the result.

Example 4: Asymmetric power ARCH model

As an example of a frequently sampled, long-run series, consider the daily closing indices of the Dow Jones Industrial Average, variable `dowclose`. To avoid the first half of the century, when the New York Stock Exchange was open for Saturday trading, only data after `1jan1953` are used. The compound return of the series is used as the dependent variable and is graphed below.

We formed this difference by referring to `D.ln_dow`, but only after playing a trick. The series is daily, and each observation represents the Dow closing index for the day. Our data included a time variable recorded as a daily date. We wanted, however, to model the log differences in the series, and we wanted the span from Friday to Monday to appear as a single-period difference. That is, the day before Monday is Friday. Because our dataset was `tsset` with `date`, the span from Friday to Monday was 3 days. The solution was to create a second variable that sequentially numbered the observations. By `tsset`ing the data with this new variable, we obtained the desired differences.

```
. generate t = _n
. tsset t
```
Now our data look like this:

```
. use https://www.stata-press.com/data/r16/dow1, clear
. generate dayofwk = dow(date)
. list date dayofwk t ln_dow D.ln_dow in 1/8

D.
<table>
<thead>
<tr>
<th>date</th>
<th>dayofwk</th>
<th>t</th>
<th>ln_dow</th>
<th>D.ln_dow</th>
</tr>
</thead>
<tbody>
<tr>
<td>02jan1953</td>
<td>5</td>
<td>1</td>
<td>5.677096</td>
<td>.</td>
</tr>
<tr>
<td>05jan1953</td>
<td>1</td>
<td>2</td>
<td>5.682899</td>
<td>.0058026</td>
</tr>
<tr>
<td>06jan1953</td>
<td>2</td>
<td>3</td>
<td>5.677439</td>
<td>-.0054603</td>
</tr>
<tr>
<td>07jan1953</td>
<td>3</td>
<td>4</td>
<td>5.672636</td>
<td>-.0048032</td>
</tr>
<tr>
<td>08jan1953</td>
<td>4</td>
<td>5</td>
<td>5.671259</td>
<td>-.0013762</td>
</tr>
<tr>
<td>09jan1953</td>
<td>5</td>
<td>6</td>
<td>5.661223</td>
<td>-.0100365</td>
</tr>
<tr>
<td>12jan1953</td>
<td>1</td>
<td>7</td>
<td>5.653191</td>
<td>-.0080323</td>
</tr>
<tr>
<td>13jan1953</td>
<td>2</td>
<td>8</td>
<td>5.659134</td>
<td>.0059433</td>
</tr>
</tbody>
</table>

. list date dayofwk t ln_dow D.ln_dow in -8/1

D.
<table>
<thead>
<tr>
<th>date</th>
<th>dayofwk</th>
<th>t</th>
<th>ln_dow</th>
<th>D.ln_dow</th>
</tr>
</thead>
<tbody>
<tr>
<td>08feb1990</td>
<td>4</td>
<td>9334</td>
<td>7.880188</td>
<td>.0016198</td>
</tr>
<tr>
<td>09feb1990</td>
<td>5</td>
<td>9335</td>
<td>7.881635</td>
<td>.0014472</td>
</tr>
<tr>
<td>12feb1990</td>
<td>1</td>
<td>9336</td>
<td>7.870601</td>
<td>-.011034</td>
</tr>
<tr>
<td>13feb1990</td>
<td>2</td>
<td>9337</td>
<td>7.872665</td>
<td>.0020638</td>
</tr>
<tr>
<td>14feb1990</td>
<td>3</td>
<td>9338</td>
<td>7.872577</td>
<td>-.0000877</td>
</tr>
<tr>
<td>15feb1990</td>
<td>4</td>
<td>9339</td>
<td>7.88213</td>
<td>.009553</td>
</tr>
<tr>
<td>16feb1990</td>
<td>5</td>
<td>9340</td>
<td>7.876863</td>
<td>-.0052876</td>
</tr>
<tr>
<td>20feb1990</td>
<td>2</td>
<td>9341</td>
<td>7.862054</td>
<td>-.0148082</td>
</tr>
</tbody>
</table>
```

The difference operator $D$ spans weekends because the specified time variable, `t`, is not a true date and has a difference of 1 for all observations. We must leave this contrived time variable in place during estimation, or `arch` will be convinced that our dataset has gaps. If we were using calendar dates, we would indeed have gaps.

**Ding, Granger, and Engle (1993)** fit an A-PARCH model of daily returns of the Standard and Poor's 500 (S&P 500) for 3jan1928–30aug1991. We will fit the same model for the Dow data shown above. The model includes an AR(1) term as well as the A-PARCH specification of conditional variance.
. arch D.ln_dow, ar(1) aparch(1) pgarch(1)

(setting optimization to BHHH)

Iteration 0:  log likelihood =  31139.547
Iteration 1:  log likelihood =  31350.751

(output omitted)

Iteration 68: log likelihood = 32273.555 (backed up)

(output omitted)

ARCH family regression -- AR disturbances

Sample: 2 - 9341  Number of obs = 9,340
Distribution: Gaussian  Wald chi2(1) = 175.46
Log likelihood = 32273.56  Prob > chi2 = 0.0000

|                  | Coef.     | Std. Err. | z    | P>|z|   | [95% Conf. Interval] |
|------------------|-----------|-----------|------|-------|---------------------|
| ln_dow           | .0001786  | .0000875  | 2.04 | 0.041 | 7.15e-06             |
| _cons            |           |           |      |       | .00035              |
| ARMA             |           |           |      |       |                     |
| ar L1.           | .1410944  | .0106519  | 13.25| 0.000 | .1202171             |
|                  |           |           |      |       | .1619716             |
| ARCH             |           |           |      |       |                     |
| aparch L1.       | .0626323  | .0034307  | 18.26| 0.000 | .0559082             |
|                  |           |           |      |       | .0693564             |
| aparch_e L1.     | -.3645093 | .0378485  | -9.63| 0.000 | -.4386909            |
|                  |           |           |      |       | -.2903277            |
| pgarch L1.       | .9299015  | .0030998  | 299.99| 0.000 | .923826              |
|                  |           |           |      |       | .935977              |
| _cons            | 7.19e-06  | 2.53e-06  | 2.84 | 0.004 | 2.23e-06             |
|                  |           |           |      |       | .0000121             |
| POWER            |           |           |      |       |                     |
| power            | 1.585187  | .0629186  | 25.19| 0.000 | 1.461869             |
|                  |           |           |      |       | 1.708505             |

In the iteration log, the final iteration reports the message “backed up”. For most estimators, ending on a “backed up” message would be a cause for great concern, but not with arch or, for that matter, arima, as long as you do not specify the g_tolerance() option. arch and arima, by default, monitor the gradient and declare convergence only if, in addition to everything else, the gradient is small enough.

The fitted model demonstrates substantial asymmetry, with the large negative L1.aparch_e coefficient indicating that the market responds with much more volatility to unexpected drops in returns (bad news) than it does to increases in returns (good news).

Example 5: ARCH model with nonnormal errors

Stock returns tend to be leptokurtotic, meaning that large returns (either positive or negative) occur more frequently than one would expect if returns were in fact normally distributed. Here we refit the previous A-PARCH model assuming the errors follow the generalized error distribution, and we let arch estimate the shape parameter of the distribution.
. use https://www.stata-press.com/data/r16/dow1, clear
. arch D.ln_dow, ar(1) aparch(1) pgarch(1) distribution(ged)

(setting optimization to BHHH)
Iteration 0: log likelihood = 31139.547
Iteration 1: log likelihood = 31348.13
(output omitted)
Iteration 61: log likelihood = 32486.461

ARCH family regression -- AR disturbances

Sample: 2 - 9341 Number of obs = 9,340
Distribution: GED Wald chi2(1) = 178.22
Log likelihood = 32486.46 Prob > chi2 = 0.0000

| D.ln_dow | GPG | Coef. | Std. Err. | z | P>|z| | [95% Conf. Interval] |
|----------|-----|-------|-----------|---|------|----------------------|
| ln_dow   |     | .0002735 | .000078   | 3.51 | .000000 | .0001207 - .0004264 |
| _cons    |     |         |           |     |       |                      |

ARMA

|    |     | .1337479 | .0100187 | 13.35 | .000000 | .1141116 - .1533842 |
|    |     |          |          |       |          |                      |
| ar | L1.  | .0641772 | .0049401 | 12.99 | .000000 | .0544949 - .0738595 |
| Aparch | L1.  | -.405225 | .0573059 | -7.07 | .000000 | -.5175426 - -.2929074 |
| Aparch_e | L1.  | .9341739 | .0045668 | 204.56 | .000000 | .9252231 - .9431247 |
| Pgarch | L1.  | .0000216 | .0000117 | 1.84  | .066000 | -1.39e-06 - .0000446 |
| _cons  |     |         |           |     |       |                      |

POWER

|     |     | 1.32524 | .1030699 | 12.86 | .000000 | 1.123227 - 1.527253 |
|     |     |          |          |       |          |                      |
|     |     | .3527019 | .0094819 | 37.20 | .000000 | .334177 - .371286   |

shape | 1.422907 | .0134919 | 37.20 | .000000 | 1.396707 - 1.449598 |

The ARMA and ARCH coefficients are similar to those we obtained when we assumed normally distributed errors, though we do note that the power term is now closer to 1. The estimated shape parameter for the generalized error distribution is shown at the bottom of the output. Here the shape parameter is 1.42; because it is less than 2, the distribution of the errors has tails that are fatter than they would be if the errors were normally distributed.

Example 6: ARCH model with constraints

Engle’s (1982) original model, which sparked the interest in ARCH, provides an example requiring constraints. Most current ARCH specifications use GARCH terms to provide flexible dynamic properties without estimating an excessive number of parameters. The original model was limited to ARCH terms, and to help cope with the collinearity of the terms, a declining lag structure was imposed in the parameters. The conditional variance equation was specified as

\[
\sigma_t^2 = \alpha_0 + \alpha(0.4 \epsilon_{t-1} + 0.3 \epsilon_{t-2} + 0.2 \epsilon_{t-3} + 0.1 \epsilon_{t-4})
\]

\[
= \alpha_0 + 0.4 \alpha \epsilon_{t-1} + 0.3 \alpha \epsilon_{t-2} + 0.2 \alpha \epsilon_{t-3} + 0.1 \alpha \epsilon_{t-4}
\]
From the earlier arch output, we know how the coefficients will be named. In Stata, the formula is

\[ \sigma^2_t = [ARCH] \_cons + .4 [ARCH]L1.arch \epsilon_{t-1} + .3 [ARCH]L2.arch \epsilon_{t-2} \\
+ .2 [ARCH]L3.arch \epsilon_{t-3} + .1 [ARCH]L4.arch \epsilon_{t-4} \]

We could specify these linear constraints many ways, but the following seems fairly intuitive; see [R] constraint for syntax.

```stata
donotuse
.use https://www.stata-press.com/data/r16/wpi1, clear
.constraint 1 (3/4)*[ARCH]l1.arch = [ARCH]l2.arch
.constraint 3 (1/4)*[ARCH]l1.arch = [ARCH]l4.arch
```

The original model was fit on U.K. inflation; we will again use the WPI data and retain our earlier specification of the mean equation, which differs from Engle’s U.K. inflation model. With our constraints, we type

```stata
donotuse
.arch D.ln_wpi, ar(1) ma(1 4) arch(1/4) constraints(1/3)
```

```stata
donotuse
(setting optimization to BHHH)
Iteration 0: log likelihood = 396.80198
Iteration 1: log likelihood = 399.07809
(output omitted)
Iteration 9: log likelihood = 399.46243
```

```
ARCH family regression -- ARMA disturbances
Sample: 1960q2 - 1990q4 Number of obs = 123
Distribution: Gaussian Wald chi2(3) = 123.32
Log likelihood = 399.4624 Prob > chi2 = 0.0000
( 1) .75*[ARCH]L.arch - [ARCH]L2.arch = 0
( 2) .5*[ARCH]L.arch - [ARCH]L3.arch = 0
( 3) .25*[ARCH]L.arch - [ARCH]L4.arch = 0
```

| D.ln_wpi | Coef. | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|----------|-------|-----------|-----|-----|---------------------|
| ln_wpi   |       |           |     |     |                     |
| _cons    | .0077204 | .0034531 | 2.24 | 0.025 | .0009525 | .0144883 |
| ARMA     |       |           |     |     |                     |
| ar       |       |           |     |     |                     |
| L1.      | .7388168 | .1126811 | 6.56 | 0.000 | .5179659 | .9596676 |
| ma       |       |           |     |     |                     |
| L1.      | -.2559691 | .1442861 | -1.77 | 0.076 | -.5387646 | .0288264 |
| L4.      | .2528923 | .1140185 | 2.22 | 0.027 | .02942 | .4763645 |
| ARCH     |       |           |     |     |                     |
| arch     |       |           |     |     |                     |
| L1.      | .2180138 | .0737787 | 2.95 | 0.003 | .0734101 | .3626174 |
| L2.      | .1635103 | .055334  | 2.95 | 0.003 | .0550576 | .2719631 |
| L3.      | .1090069 | .0368894 | 2.95 | 0.003 | .0367051 | .1813087 |
| L4.      | .0545034 | .0184447 | 2.95 | 0.003 | .0183525 | .0906544 |
| _cons    | .0000483 | 7.66e-06 | 6.30 | 0.000 | .0000333 | .0000633 |
```

L1.arch, L2.arch, L3.arch, and L4.arch coefficients have the constrained relative sizes.
**Stored results**

`arch` stores the following in `e()`:

**Scalars**
- `e(N)` number of observations
- `e(N_gaps)` number of gaps
- `e(condobs)` number of conditioning observations
- `e(k)` number of parameters
- `e(k_eq)` number of equations in `e(b)`
- `e(k_eq_model)` number of equations in overall model test
- `e(k_dv)` number of dependent variables
- `e(k_aux)` number of auxiliary parameters
- `e(df_m)` model degrees of freedom
- `e(ll)` log likelihood
- `e(chi2)` $\chi^2$
- `e(p)` $p$-value for model test
- `e(archi)` $\sigma_0^2 = \epsilon_0^2$, priming values
- `e(archany)` 1 if model contains ARCH terms, 0 otherwise
- `e(tdf)` degrees of freedom for Student’s $t$ distribution
- `e(shape)` shape parameter for generalized error distribution
- `e(tmin)` minimum time
- `e(tmax)` maximum time
- `e(power)` $\varphi$ for power ARCH terms
- `e(rank)` rank of `e(V)`
- `e(ic)` number of iterations
- `e(rc)` return code
- `e(converged)` 1 if converged, 0 otherwise

**Macros**
- `e(cmd)` `arch`
- `e(cmdline)` command as typed
- `e(depvar)` name of dependent variable
- `e(covariates)` list of covariates
- `e(eqnames)` names of equations
- `e(wtype)` weight type
- `e(wexp)` weight expression
- `e(title)` title in estimation output
- `e(tmins)` formatted minimum time
- `e(tmaxs)` formatted maximum time
- `e(dist)` distribution for error term: `gaussian`, `t`, or `ged`
- `e(mhet)` 1 if multiplicative heteroskedasticity
- `e(dfopt)` yes if degrees of freedom for $t$ distribution or shape parameter for GED distribution was estimated, no otherwise
- `e(chi2type)` Wald; type of model $\chi^2$ test
- `e(vce)` `vcetype` specified in `vce()`
- `e(vcetyp)` title used to label Std. Err.
- `e(ar)` lags for moving-average terms
- `e(ar)` lags for autoregressive terms
- `e(arch)` lags for ARCH terms
- `e(archm)` ARCH-in-mean lags
- `e(archmexp)` ARCH-in-mean exp
- `e(earch)` lags for EARCH terms
- `e(egarch)` lags for EGARCH terms
- `e(aarch)` lags for AARCH terms
- `e(narch)` lags for NARCH terms
- `e(aparch)` lags for A-PARCH terms
- `e(nparch)` lags for NPARCH terms
- `e(saarch)` lags for SAARCH terms
- `e(parch)` lags for PARCH terms
- `e(tparch)` lags for TPARCH terms
- `e(abarch)` lags for ABARCH terms
- `e(tarch)` lags for TARCH terms
Methods and formulas

The mean equation for the model fit by `arch` and with ARMA terms can be written as

\[
y_t = x_t \beta + \sum_{i=1}^{p} \psi_i g(\sigma_{t-i}^2) + \sum_{j=1}^{p} \rho_j \left\{ y_{t-j} - x_{t-j} \beta - \sum_{i=1}^{p} \psi_i g(\sigma_{t-j-i}^2) \right\} + \sum_{k=1}^{q} \theta_k \epsilon_{t-k} + \epsilon_t
\]

(conditional mean)

where

- \( \beta \) are the regression parameters,
- \( \psi \) are the ARCH-in-mean parameters,
- \( \rho \) are the autoregression parameters,
- \( \theta \) are the moving-average parameters, and
- \( g() \) is a general function, see the `archmexp()` option.

Any of the parameters in this full specification of the conditional mean may be zero. For example, the model need not have moving-average parameters (\( \theta = 0 \)) or ARCH-in-mean parameters (\( \psi = 0 \)).

The variance equation will be one of the following:

\[
\sigma_t^2 = \gamma_0 + A(\sigma, \epsilon) + B(\sigma, \epsilon)^2 \\
\ln \sigma_t^2 = \gamma_0 + C(\ln \sigma, z) + A(\sigma, \epsilon) + B(\sigma, \epsilon)^2 \\
\sigma_t^2 = \gamma_0 + D(\sigma, \epsilon) + A(\sigma, \epsilon) + B(\sigma, \epsilon)^2
\]
where $A(\sigma, \epsilon), B(\sigma, \epsilon), C(\ln \sigma, z)$, and $D(\sigma, \epsilon)$ are linear sums of the appropriate ARCH terms; see Details of syntax for more information. Equation (1) is used if no EGARCH or power ARCH terms are included in the model, (2) if EGARCH terms are included, and (3) if any power ARCH terms are included; see Details of syntax.

Methods and formulas are presented under the following headings:

- Priming values
- Likelihood from prediction error decomposition
- Missing data

### Priming values

The above model is recursive with potentially long memory. It is necessary to assume preestimation sample values for $\epsilon_t, \epsilon^2_t$, and $\sigma^2_t$ to begin the recursions, and the remaining computations are therefore conditioned on these priming values, which can be controlled using the arch0() and arma0() options. See options discussed under the Priming tab above.

The arch0(xb0wt) and arch0(xbw) options compute a weighted sum of estimated disturbances with more weight on the early observations. With either of these options,

$$
\sigma^2_{t_0-i} = \epsilon^2_{t_0-i} = (1 - 0.7) \sum_{t=0}^{T-1} 0.7^{T-t-1} \epsilon^2_{T-t} \quad \forall i
$$

where $t_0$ is the first observation for which the likelihood is computed; see options discussed under the Priming tab above. The $\epsilon^2_t$ are all computed from the conditional mean equation. If arch0(xb0wt) is specified, $\beta, \psi_i, \rho_j$, and $\theta_k$ are taken from initial regression estimates and held constant during optimization. If arch0(xbw) is specified, the current estimates of $\beta, \psi_i, \rho_j$, and $\theta_k$ are used to compute $\epsilon^2_t$ on every iteration. If any $\psi_i$ is in the mean equation (ARCH-in-mean is specified), the estimates of $\epsilon^2_t$ from the initial regression estimates are not consistent.

### Likelihood from prediction error decomposition

The likelihood function for ARCH has a particularly simple form. Given priming (or conditioning) values of $\epsilon_t, \epsilon^2_t$, and $\sigma^2_t$, the mean equation above can be solved recursively for every $\epsilon_t$ (prediction error decomposition). Likewise, the conditional variance can be computed recursively for each observation by using the variance equation. Using these predicted errors, their associated variances, and the assumption that $\epsilon_t \sim N(0, \sigma^2_t)$, we find that the log likelihood for each observation $t$ is

$$
\ln L_t = -\frac{1}{2} \left\{ \ln (2\pi \sigma^2_t) + \frac{\epsilon^2_t}{\sigma^2_t} \right\}
$$

If we assume that $\epsilon_t \sim t(df)$, then as given in Hamilton (1994, 662),

$$
\ln L_t = \ln \Gamma \left( \frac{df + 1}{2} \right) - \ln \Gamma \left( \frac{df}{2} \right) - \frac{1}{2} \left[ \ln \left\{ (df - 2)\pi \sigma^2_t \right\} + (df + 1) \ln \left\{ 1 + \frac{\epsilon^2_t}{(df - 2)\sigma^2_t} \right\} \right]
$$

The likelihood is not defined for $df \leq 2$, so instead of estimating $df$ directly, we estimate $m = \ln(df - 2)$. Then $df = \exp(m) + 2 > 2$ for any $m$. 
Following Bollerslev, Engle, and Nelson (1994, 2978), the log likelihood for the \( t \)th observation, assuming \( \epsilon_t \sim \text{GED}(s) \), is

\[
\ln L_t = \ln s - \ln \lambda - \frac{s + 1}{s} \ln 2 - \ln \Gamma\left(s^{-1}\right) - \frac{1}{2} \frac{|\epsilon_t|}{\lambda \sigma_t^s}
\]

where

\[
\lambda = \left\{ \frac{\Gamma\left(s^{-1}\right)}{2^{s/2} \Gamma\left(3s^{-1}\right)} \right\}^{1/2}
\]

To enforce the restriction that \( s > 0 \), we estimate \( r = \ln s \).

This command supports the Huber/White/sandwich estimator of the variance using \texttt{vce(robust)}. See \texttt{[P] _robust}, particularly \textit{Maximum likelihood estimators} and \textit{Methods and formulas}.

### Missing data

ARCH allows missing data or missing observations but does not attempt to condition on the surrounding data. If a dynamic component cannot be computed—\( \epsilon_t \), \( \epsilon_t^2 \), and/or \( \sigma_t^2 \)—its priming value is substituted. If a covariate, the dependent variable, or the entire observation is missing, the observation does not enter the likelihood, and its dynamic components are set to their priming values for that observation. This is acceptable only asymptotically and should not be used with a great deal of missing data.

Robert Fry Engle (1942– ) was born in Syracuse, New York. He earned degrees in physics and economics at Williams College and Cornell and then worked at MIT and the University of California, San Diego, before moving to New York University Stern School of Business in 2000. He was awarded the 2003 Nobel Prize in Economics for research on autoregressive conditional heteroskedasticity and is a leading expert in time-series analysis, especially the analysis of financial markets.

### References


Becketti, S. 2013. \textit{Introduction to Time Series Using Stata}. College Station, TX: Stata Press.


Also see

[TS] arch postestimation — Postestimation tools for arch

[TS] arima — ARIMA, ARMAX, and other dynamic regression models

[TS] mgarch — Multivariate GARCH models

[TS] tsset — Declare data to be time-series data

[R] regress — Linear regression

[U] 20 Estimation and postestimation commands
Postestimation commands

The following postestimation commands are available after arch:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estat ic</td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td>estat summarize</td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td>estat vce</td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td>estimates</td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td>forecast</td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td>lincom</td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td>lrtest</td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td>margins</td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td>marginsplot</td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td>nlcom</td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td>predict</td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td>predictnl</td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td>test</td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td>testnl</td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

`predict` creates a new variable containing predictions such as expected values and residuals. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] newvar [if] [in] [, statistic options]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td><code>xb</code></td>
<td>predicted values for mean equation—the differenced series; the default</td>
</tr>
<tr>
<td><code>y</code></td>
<td>predicted values for the mean equation in y—the undifferenced series</td>
</tr>
<tr>
<td><code>variance</code></td>
<td>predicted values for the conditional variance</td>
</tr>
<tr>
<td><code>het</code></td>
<td>predicted values of the variance, considering only the multiplicative</td>
</tr>
<tr>
<td></td>
<td>heteroskedasticity</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>residuals or predicted innovations</td>
</tr>
<tr>
<td><code>yresiduals</code></td>
<td>residuals or predicted innovations in y—the undifferenced series</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ... if` wanted only for the estimation sample.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td><code>dynamic</code></td>
<td>how to handle the lags of $y_t$</td>
</tr>
<tr>
<td><code>at</code></td>
<td>make static predictions</td>
</tr>
<tr>
<td><code>t0</code></td>
<td>set starting point for the recursions to <code>time_constant</code></td>
</tr>
<tr>
<td><code>structural</code></td>
<td>calculate considering the structural component only</td>
</tr>
</tbody>
</table>

`time_constant` is a # or a time literal, such as td(1jan1995) or tq(1995q1), etc.; see

`Conveniently typing SIF values` in [D] Datetime.
Options for predict

Six statistics can be computed by using predict after arch: the predictions of the mean equation (option *xb*, the default), the undifferenced predictions of the mean equation (option *y*), the predictions of the conditional variance (option *variance*), the predictions of the multiplicative heteroskedasticity component of variance (option *het*), the predictions of residuals or innovations (option *residuals*), and the predictions of residuals or innovations in terms of *y* (option *yresiduals*). Given the dynamic nature of ARCH models and because the dependent variable might be differenced, there are other ways of computing each statistic. We can use all the data on the dependent variable available right up to the time of each prediction (the default, which is often called a one-step prediction), or we can use the data up to a particular time, after which the predicted value of the dependent variable is used recursively to make later predictions (option *dynamic()*). Either way, we can consider or ignore the ARMA disturbance component, which is considered by default and is ignored if you specify the structural option. We might also be interested in predictions at certain fixed points where we specify the prior values of $\epsilon_t$ and $\sigma^2_t$ (option *at()*).

---

**Main**

*xb*, the default, calculates the predictions from the mean equation. If $D.\text{depvar}$ is the dependent variable, these predictions are of $D.\text{depvar}$ and not of $\text{depvar}$ itself.

*y* specifies that predictions of $\text{depvar}$ are to be made even if the model was specified for, say, $D.\text{depvar}$.

*variance* calculates predictions of the conditional variance $\hat{\sigma}^2_t$.

*het* calculates predictions of the multiplicative heteroskedasticity component of variance.

*residuals* calculates the residuals. If no other options are specified, these are the predicted innovations $\epsilon_t$; that is, they include any ARMA component. If the structural option is specified, these are the residuals from the mean equation, ignoring any ARMA terms; see structural below. The residuals are always from the estimated equation, which may have a differenced dependent variable; if *depvar* is differenced, they are not the residuals of the undifferenced *depvar*.

*yresiduals* calculates the residuals for *depvar*, even if the model was specified for, say, $D.\text{depvar}$ As with *residuals*, the *yresiduals* are computed from the model, including any ARMA component. If the structural option is specified, any ARMA component is ignored and *yresiduals* are the residuals from the structural equation; see structural below.

---

**Options**

dynamic(*time_constant*) specifies how lags of $y_t$ in the model are to be handled. If dynamic() is not specified, actual values are used everywhere lagged values of $y_t$ appear in the model to produce one-step-ahead forecasts.

dynamic(*time_constant*) produces dynamic (also known as recursive) forecasts. *time_constant* specifies when the forecast is to switch from one step ahead to dynamic. In dynamic forecasts, references to $y_t$ evaluate to the prediction of $y_t$ for all periods at or after *time_constant*; they evaluate to the actual value of $y_t$ for all prior periods.

dynamic(10), for example, would calculate predictions where any reference to $y_t$ with $t < 10$ evaluates to the actual value of $y_t$ and any reference to $y_t$ with $t \geq 10$ evaluates to the prediction of $y_t$. This means that one-step-ahead predictions would be calculated for $t < 10$ and dynamic predictions would be calculated thereafter. Depending on the lag structure of the model, the dynamic predictions might still refer to some actual values of $y_t$.

You may also specify dynamic(.) to have predict automatically switch from one-step-ahead to dynamic predictions at $p + q$, where $p$ is the maximum AR lag and $q$ is the maximum MA lag.
at(varname_ε | #_ε varname_σ² | #_σ²) makes static predictions. at() and dynamic() may not be specified together.

Specifying at() allows static evaluation of results for a given set of disturbances. This is useful, for instance, in generating the news response function. at() specifies two sets of values to be used for $\epsilon_t$ and $\sigma^2_t$, the dynamic components in the model. These specified values are treated as given. Also, any lagged values of depvar in the model are obtained from the real values of the dependent variable. All computations are based on actual data and the given values.

at() requires that you specify two arguments, which can be either a variable name or a number. The first argument supplies the values to be used for $\epsilon_t$; the second supplies the values to be used for $\sigma^2_t$. If $\sigma^2_t$ plays no role in your model, the second argument may be specified as ‘.’ to indicate missing.

t0(time_constant) specifies the starting point for the recursions to compute the predicted statistics; disturbances are assumed to be 0 for $t < t0()$. The default is to set t0() to the minimum $t$ observed in the estimation sample, meaning that observations before that are assumed to have disturbances of 0.

t0() is irrelevant if structural is specified because then all observations are assumed to have disturbances of 0.

t0(5), for example, would begin recursions at $t = 5$. If your data were quarterly, you might instead type t0(tq(1961q2)) to obtain the same result.

Any ARMA component in the mean equation or GARCH term in the conditional-variance equation makes arch recursive and dependent on the starting point of the predictions. This includes one-step-ahead predictions.

structural makes the calculation considering the structural component only, ignoring any ARMA terms, and producing the steady-state equilibrium predictions.
margins

Description for margins

margins estimates margins of response for expected values.

Menu for margins

Statistics  >  Postestimation

Syntax for margins

\[
\text{margins} \ [\ marginlist ] \ [ , \ \text{options} ] \\
\text{margins} \ [\ marginlist ] , \ \text{predict(statistic ... )} \ [\ \text{predict(statistic ... )} \ ... ] \ [\ \text{options} ]
\]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>predicted values for mean equation—the differenced series; the default</td>
</tr>
<tr>
<td>y</td>
<td>predicted values for the mean equation in y—the undifferenced series</td>
</tr>
<tr>
<td>variance</td>
<td>predicted values for the conditional variance</td>
</tr>
<tr>
<td>het</td>
<td>predicted values of the variance, considering only the multiplicative heteroskedasticity</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>yresiduals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

Statistics not allowed with margins are functions of stochastic quantities other than \( \epsilon(b) \).

For the full syntax, see [R] margins.

Remarks and examples

Example 1

Continuing with our EGARCH model example (example 3) in [TS] arch, we can see that predict, at() calculates \( \sigma^2_t \) given a set of specified innovations \( (\epsilon_t, \epsilon_{t-1}, \ldots) \) and prior conditional variances \( (\sigma^2_{t-1}, \sigma^2_{t-2}, \ldots) \). The syntax is

\[
. \ \text{predict newvar, variance at(epsilon sigma2)}
\]

\( \epsilon(t) \) and \( \sigma^2(t) \) are either variables or numbers. Using \( \sigma^2(t) \) is a little tricky because you specify values of \( \sigma^2(t) \), which predict is supposed to predict. predict does not simply copy variable \( \sigma^2(t) \) into newvar but uses the lagged values contained in sigma2 to produce the predicted value of \( \sigma^2(t) \). It does this for all \( t \), and those results are saved in newvar. (If you are interested in dynamic predictions of \( \sigma^2(t) \), see Options for predict.)

We will generate predictions for \( \sigma^2(t) \), assuming that the lagged values of \( \sigma^2(t) \) are 1, and we will vary \( \epsilon(t) \) from -4 to 4. First, we will create variable et containing \( \epsilon(t) \), and then we will create and graph the predictions:
The positive asymmetry does indeed dominate the shape of the news response function. In fact, the response is a monotonically increasing function of news. The form of the response function shows that, for our simple model, only positive, unanticipated price increases have the destabilizing effect that we observe as larger conditional variances.

Example 2

Continuing with our ARCH model with constraints example (example 6) in [TS] arch, using lincom we can recover the \( \alpha \) parameter from the original specification.

```
. lincom [ARCH]l1.arch/.4  
( 1) 2.5*[ARCH]L.arch = 0
```

| D.ln_wpi | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|----------|--------|-----------|-------|------|---------------------|
| (1)      | .5450344 | .1844468  | 2.95  | 0.003 | .1835253            | .9065436 |

Any arch parameter could be used to produce an identical estimate.
arfima — Autoregressive fractionally integrated moving-average models

Description

arfima estimates the parameters of autoregressive fractionally integrated moving-average (ARFIMA) models.

Long-memory processes are stationary processes whose autocorrelation functions decay more slowly than short-memory processes. The ARFIMA model provides a parsimonious parameterization of long-memory processes that nests the autoregressive moving-average (ARMA) model, which is widely used for short-memory processes. By allowing for fractional degrees of integration, the ARFIMA model also generalizes the autoregressive integrated moving-average (ARIMA) model with integer degrees of integration. See [TS] arima for ARMA and ARIMA parameter estimation.

Quick start

Autoregressive fractionally integrated moving-average model for $y$ with regressor $x$ using tsset data

```
arfima y x
```

Add autoregressive components of orders 1 and 2 and a moving-average component of order 4

```
arfima y x, ar(1 2) ma(4)
```

ARIMA for $y$ with autoregressive components of orders 1 and 2

```
arfima y, ar(1 2) smemory
```

Menu

Statistics > Time series > ARFIMA models
arfima — Autoregressive fractionally integrated moving-average models 53

Syntax

arfima depvar [indepvars] [if] [in] [, options]

options Description

Model

noconstant suppress constant term
ar(numlist) autoregressive terms
ma(numlist) moving-average terms
smemory estimate short-memory model without fractional integration
mle maximum likelihood estimates; the default
mpl maximum modified-profile-likelihood estimates
constraints(numlist) apply specified linear constraints

SE/Robust
vce(vcetype) vcetype may be oim or robust

Reporting

level(#) set confidence level; default is level(95)
nocnsreport do not display constraints
display_options control columns and column formats, row spacing, line width,
  display of omitted variables and base and empty cells, and
  factor-variable labeling

Maximization

maximize_options control the maximization process; seldom used
collinear keep collinear variables
coefflegend display legend instead of statistics

You must tsset your data before using arfima; see [TS] tsset.

indepvars may contain factor variables; see [U] 11.4.3 Factor variables.
depvar and indepvars may contain time-series operators; see [U] 11.4.4 Time-series varlists.
by, fp, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.
collinear and coefflegend do not appear in the dialog box.
See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

Options

The Model

noconstant; see [R] Estimation options.

ar(numlist) specifies the autoregressive (AR) terms to be included in the model. An AR(p), \( p \geq 1 \), specification would be \( \text{ar}(1/p) \). This model includes all lags from 1 to \( p \), but not all lags need to be included. For example, the specification \( \text{ar}(1 \ p) \) would specify an AR(p) with only lags 1 and \( p \) included, setting all the other AR lag parameters to 0.

ma(numlist) specifies the moving-average terms to be included in the model. These are the terms for the lagged innovations (white-noise disturbances). \( \text{ma}(1/q), q \geq 1 \), specifies an MA(q) model, but like the \( \text{ar}() \) option, not all lags need to be included.
smemory causes arfima to fit a short-memory model with $d = 0$. This option causes arfima to estimate the parameters of an ARMA model by a method that is asymptotically equivalent to that produced by arima; see [TS] arima.

mle causes arfima to estimate the parameters by maximum likelihood. This method is the default.

mpl causes arfima to estimate the parameters by maximum modified profile likelihood (MPL). The MPL estimator of the fractional-difference parameter has less small-sample bias than the maximum likelihood estimator when there are covariates in the model. mpl may only be specified when there is a constant term or indepvars in the model, and it may not be combined with the mle option.

constraints(numlist); see [R] Estimation options.

vce(vcetype) specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that are derived from asymptotic theory (oim); see [R] vce_option.

Options vce(robust) and mpl may not be combined.

level(#), nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nolvl, fvwrap(#), fvwrapon(style), cformat(%fmt), pformat(%fmt), sformat(%fmt), and nolstretch; see [R] Estimation options.

Some special points for arfima’s maximize_options are listed below.

technique(algorithm_spec) sets the optimization algorithm. The default algorithm is BFGS and BHHH is not allowed. See [R] Maximize for a description of the available optimization algorithms.

You can specify multiple optimization methods. For example, technique(bfgs 10 nr) requests that the optimizer perform 10 BFGS iterations and then switch to Newton–Raphson until convergence.

iterate(#) sets the maximum number of iterations. When the maximization is not going well, set the maximum number of iterations to the point where the optimizer appears to be stuck and inspect the estimation results at that point.

from(matname) allows you to specify starting values for the model parameters in a row vector.

We recommend that you use the iterate(0) option, retrieve the initial estimates from e(b), and modify these elements.

The following options are available with arfima but are not shown in the dialog box:

collinear, coeflegend; see [R] Estimation options.
Remarks and examples

Long-memory processes are stationary processes whose autocorrelation functions decay more slowly than short-memory processes. Because the autocorrelations die out so slowly, long-memory processes display a type of long-run dependence. The autoregressive fractionally integrated moving-average (ARFIMA) model provides a parsimonious parameterization of long-memory processes. This parameterization nests the autoregressive moving-average (ARMA) model, which is widely used for short-memory processes.

The ARFIMA model also generalizes the autoregressive integrated moving-average (ARIMA) model with integer degrees of integration. ARFIMA models provide a solution for the tendency to overdifference stationary series that exhibit long-run dependence. In the ARIMA approach, a nonstationary time series is differenced \(d\) times until the differenced series is stationary, where \(d\) is an integer. Such series are said to be integrated of order \(d\), denoted \(I(d)\), with not differencing, \(I(0)\), being the option for stationary series. Many series exhibit too much dependence to be \(I(0)\) but are not \(I(1)\), and ARFIMA models are designed to represent these series.

The ARFIMA model allows for a continuum of fractional differences, \(-0.5 < d < 0.5\). The generalization to fractional differences allows the ARFIMA model to handle processes that are neither \(I(0)\) nor \(I(1)\), to test for overdifferencing, and to model long-run effects that only die out at long horizons.

Technical note

An ARIMA model for the series \(y_t\) is given by

\[
\rho(L)(1 - L)^d y_t = \theta(L) \epsilon_t
\]

where \(\rho(L) = (1 - \rho_1 L - \rho_2 L^2 - \cdots - \rho_p L^p)\) is the autoregressive (AR) polynomial in the lag operator \(L\); \(L y_t = y_{t-1}\); \(\theta(L) = (1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_p L^p)\) is the moving-average (MA) lag polynomial; \(\epsilon_t\) is the independent and identically distributed innovation term; and \(d\) is the integer number of differences required to make the \(y_t\) stationary. An ARFIMA model is also specified by (1) with the generalization that \(-0.5 < d < 0.5\). Series with \(d \geq 0.5\) are handled by differencing and subsequent ARFIMA modeling.

Because long-memory processes are stationary, one might be tempted to approximate the processes with many terms in an ARMA model. But these approximate models are difficult to fit and to interpret because ARMA models with many terms are difficult to estimate and the ARMA parameterization has an inherent short-run nature. In contrast, the ARFIMA model has the \(d\) parameter for the long-run dependence and ARMA parameters for short-run dependence. Using different parameters for different types of dependence facilitates estimation and interpretation, as discussed by Sowell (1992a).

Technical note

An ARFIMA model specifies a fractionally integrated ARMA process. Formally, the ARFIMA model specifies that

\[
y_t = (1 - L)^{-d} \{\rho(L)\}^{-1} \theta(L) \epsilon_t
\]

The short-run ARMA process \(\rho(L)^{-1} \theta(L) \epsilon_t\) captures the short-run effects, and the long-run effects are captured by fractionally integrating the short-run ARMA process.
Essentially, the fractional-integration parameter $d$ captures the long-run effects, and the ARMA parameters capture the short-run effects. Having separate parameters for short-run and long-run effects makes the ARFIMA model more flexible and easier to interpret than the ARMA model. After estimating the ARFIMA parameters, the short-run effects are obtained by setting $d = 0$, whereas the long-run effects use the estimated value for $d$. The short-run effects describe the behavior of the fractionally differenced process $(1 - L)^d y_t$, whereas the long-run effects describe the behavior of the fractionally integrated $y_t$.

ARFIMA models have been useful in fields as diverse as hydrology and economics. Long-memory processes were first introduced in hydrology by Hurst (1951). Hosking (1981), in hydrology, and Granger and Joyeux (1980), in economics, independently discovered the ARFIMA representation of long-memory processes. Beran (1994), Baillie (1996), and Palma (2007) provide good introductions to long-memory processes and ARFIMA models.

Example 1: Mount Campito tree ring data

Baillie (1996) discusses a time series of measurements of the widths of the annual rings of a Mount Campito Bristlecone pine. The series contains measurements on rings formed in the tree from 3436 BC to 1969 AD. Essentially, larger widths were good years for the tree and narrower widths were harsh years.

We begin by plotting the time series.

```
. use https://www.stata-press.com/data/r16/campito
(Campito Mnt. tree ring data from 3435BC to 1969AD)
. tsline width, xlabel(-3435(500)1969) ysize(2)
```

Good years and bad years seem to run together, causing the appearance of local trends. The local trends are evidence of dependence, but they are not as pronounced as those in a nonstationary series.
We plot the autocorrelations for another view:

```
    . ac width, ysize(2)
```

![Graph showing autocorrelations](image)

The autocorrelations do not start below 1 but decay very slowly.

Granger and Joyeux (1980) show that the autocorrelations from an ARMA model decay exponentially, whereas the autocorrelations from an ARFIMA process decay at the much slower hyperbolic rate. Box, Jenkins, and Reinsel (2008) define short-memory processes as those whose autocorrelations decay exponentially fast and long-memory processes as those whose autocorrelations decay at the hyperbolic rate. The above plot of autocorrelations looks closer to hyperbolic than exponential.

Together, the above plots make us suspect that the series was generated by a long-memory process. We see evidence that the series is stationary but that the autocorrelations die out much slower than a short-memory process would predict.
Given that we believe the data was generated by a stationary process, we begin by fitting the data to an ARMA model. We begin by using a short-memory model because a comparison of the results highlights the advantages of using an ARFIMA model for a long-memory process.

```
. arima width, ar(1/2) ma(1) technique(bhhh 4 nr)
(setting optimization to BHHH)
Iteration 0:  log likelihood = -18934.593
Iteration 1:  log likelihood = -18914.337
Iteration 2:  log likelihood = -18913.407
Iteration 3:  log likelihood = -18913.24
(switching optimization to Newton-Raphson)
Iteration 4:  log likelihood = -18913.214
Iteration 5:  log likelihood = -18913.208
Iteration 6:  log likelihood = -18913.208

ARIMA regression
Sample:  -3435 - 1969       Number of obs  =      5405
         Wald chi2(3)  =  133686.46
Log likelihood = -18913.21 Prob > chi2   =    0.0000

                     OIM
width     Coef.  Std. Err.     z    P>|z|     [95% Conf. Interval]
width    
    _cons   42.45055    1.02142   41.56    0.000     40.44861    44.4525

ARMA

    ar
L1.   1.264367    .0253199    49.94    0.000     1.214741    1.313993
L2.  -.2848827    .0227534   -12.52    0.000    -.3294785    -.240287

    ma
L1.  -.8066007    .0189699   -42.52    0.000    -.8437811    -.7694204

/sigma   8.005814    .0770004   103.97    0.000     7.854896    8.156732
```

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

The estimated coefficients seem high in magnitude. We use `estat aroots` to investigate further.

```
. estat aroots
Eigenvalue stability condition

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.9709661</td>
<td>.970966</td>
</tr>
<tr>
<td>.2934013</td>
<td>.293401</td>
</tr>
</tbody>
</table>

All the eigenvalues lie inside the unit circle.
AR parameters satisfy stability condition.

Eigenvalue stability condition

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.8066007</td>
<td>.806601</td>
</tr>
</tbody>
</table>

All the eigenvalues lie inside the unit circle.
MA parameters satisfy invertibility condition.
```

The roots of the AR polynomial are 0.971 and 0.293, and the root of the MA polynomial is 0.807; all of these are less than one in magnitude, indicating that the series is stationary and invertible.
but has a high level of persistence. See Hamilton (1994, 59) and [TS] estat roots for details about computing and interpreting the roots of the polynomials from the estimated ARIMA coefficients. Below we estimate the parameters of an ARFIMA model with only the fractional difference parameter and a constant.

```
. arfima width
Iteration 0: log likelihood = -18918.219
Iteration 1: log likelihood = -18916.84
Iteration 2: log likelihood = -18908.508
Iteration 3: log likelihood = -18908.508 (backed up)
Iteration 4: log likelihood = -18908.508 (backed up)
Iteration 5: log likelihood = -18907.44
Iteration 6: log likelihood = -18907.42
Iteration 7: log likelihood = -18907.283
Iteration 8: log likelihood = -18907.279
Iteration 9: log likelihood = -18907.279
Refining estimates:
Iteration 0: log likelihood = -18907.279
Iteration 1: log likelihood = -18907.279

ARFIMA regression
Sample: -3435 - 1969 Number of obs = 5,405
Wald chi2(1) = 1864.44
Log likelihood = -18907.279 Prob > chi2 = 0.0000

                   OIM
| Coef. Std. Err. z  P>|z| [95% Conf. Interval]
|-------------------|-----------------|---------------------|-------------------|
width                | width _cons     | 44.01432 9.174334 4.80 0.000 26.03296 61.99568
ARFIMA               | d               | .4468889 .0103496 43.18 0.000 .4266039 .4671738
/sigma2             | 63.92927 1.229754 51.99 0.000 61.519 66.33955

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.
```

The estimate of \( d \) is large and statistically significant. The relative parsimony of the ARFIMA model is illustrated by the fact that the estimates of the standard deviation of the idiosyncratic errors are about the same in the 5-parameter ARMA model and the 3-parameter ARFIMA model.
Let’s add an AR parameter to the above ARFIMA model:

```
.arfima width, ar(1)
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>log likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-18910.997</td>
</tr>
<tr>
<td>1</td>
<td>-18910.949    (backed up)</td>
</tr>
<tr>
<td>2</td>
<td>-18908.158    (backed up)</td>
</tr>
<tr>
<td>3</td>
<td>-18907.248</td>
</tr>
<tr>
<td>4</td>
<td>-18907.233</td>
</tr>
<tr>
<td>5</td>
<td>-18907.233</td>
</tr>
<tr>
<td>6</td>
<td>-18907.233</td>
</tr>
</tbody>
</table>

Refining estimates:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>log likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-18907.233</td>
</tr>
<tr>
<td>1</td>
<td>-18907.233</td>
</tr>
</tbody>
</table>

ARFIMA regression

Sample: -3435 - 1969 Number of obs = 5,405

Wald chi2(2) = 1875.34

Log likelihood = -18907.233 Prob > chi2 = 0.0000

| width | Coef.  | Std. Err. | z     | P>|z|    | [95% Conf. Interval] |
|-------|--------|-----------|-------|--------|----------------------|
| _cons | 43.98774 | 8.685091 | 5.06  | 0.000  | 26.96527 61.0102    |

ARFIMA

|     | Coef.  | Std. Err. | z     | P>|z|    | [95% Conf. Interval] |
|-----|--------|-----------|-------|--------|----------------------|
| ar  | .0063321 | .0209091 | 0.30  | 0.762  | -.034649 .0473133   |
| L1. | .4432472 | .0158555 | 27.96 | 0.000  | .412171 .4743234   |
| d   | .4432472 | .0158555 | 27.96 | 0.000  | .412171 .4743234   |
| /sigma2 | 63.92915 | 1.229755 | 51.99 | 0.000  | 61.51887 66.33942 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

That the estimated AR term is tiny and statistically insignificant indicates that the $d$ parameter has accounted for all the dependence in the series.

As mentioned above, there is a sense in which the main advantages of an ARFIMA model over an ARMA model for long-memory processes are the relative parsimony of the ARFIMA parameterization and the ability of the ARFIMA parameterization to separate out the long-run effects from the short-run effects. If the true process was generated from an ARFIMA model, an ARMA model with many terms can approximate the process, but the terms make estimation difficult and the lack of separate long-run and short-run parameters complicates interpretation.

This example highlights the relative parsimony of the ARFIMA model. In the examples below, we illustrate the advantages of having separate parameters for long-run and short-run effects.

Technical note

You may be wondering what long-run effects can be produced by a model for stationary processes. Because the autocorrelations of a long-memory process die out so slowly, the spectral density becomes infinite as the frequency goes to 0 and the impulse–response functions die out at a much slower rate.

The spectral density of a process describes the relative contributions of random components at different frequencies to the variance of the process, with the low-frequency components corresponding to long-run effects. See [TS] psdensity for an introduction to estimating and interpreting spectral densities implied by the estimated parameters of parametric models.
Granger and Joyeux (1980) motivate ARFIMA models by noting that their implied spectral densities are finite except at frequency 0 with \(0 < d < 0.5\), whereas stationary ARMA models have finite spectral densities at all frequencies. Granger and Joyeux (1980) argue that the ability of ARFIMA models to capture this long-range dependence, which cannot be captured by stationary ARMA models, is an important advantage of ARFIMA models over ARMA models when modeling long-memory processes.

Impulse–response functions are the coefficients on the infinite-order MA representation of a process, and they describe how a shock feeds through the dynamic system. If the process is stationary, the coefficients decay to 0 and they sum to a finite constant. As expected, the coefficients from an ARFIMA model die out at a slower rate than those from an ARMA model. Because the ARMA terms model the short-run effects and the \(d\) parameter models the long-run effects, an ARFIMA model specifies both a short-run impulse–response function and a long-run impulse–response function. When an ARMA model is used to approximate a long-memory model, the ARMA impulse–response-function coefficients confound the two effects.

### Example 2

In this example, we model the log of the monthly levels of carbon dioxide above Mauna Loa, Hawaii. To remove the seasonality, we model the twelfth seasonal difference of the log of the series. This example illustrates that the ARFIMA model parameterizes long-run and short-run effects, whereas the ARMA model confounds the two effects. (Sowell [1992a] discusses this point in greater depth.)

We begin by fitting the series to an ARMA model with an AR(1) term and an MA(2).

```plaintext
use https://www.stata-press.com/data/r16/mloa, clear
.arima S12.log, ar(1) ma(2)
(setting optimization to BHHH)
Iteration 0: log likelihood = 2000.9262
Iteration 1: log likelihood = 2001.5484
Iteration 2: log likelihood = 2001.5637
Iteration 3: log likelihood = 2001.5641
Iteration 4: log likelihood = 2001.5641
ARIMA regression
Sample: 1960m1 - 1990m12 Number of obs = 372
Wald chi2(2) = 500.41
Log likelihood = 2001.564 Prob > chi2 = 0.0000

| Coef. Std. Err. z P>|z| [95% Conf. Interval] |
|-------|-----------------|-------|-----|------------------|
| log   | .0036754        | .0002475 | 14.85 | 0.000 | .0031903 .0041605 |
| _cons |                  |          |      |       |               |

ARMA
| Coef. Std. Err. z P>|z| [95% Conf. Interval] |
|-------|-----------------|-------|-----|------------------|
| ar    | .7354346        | .0357715 | 20.56 | 0.000 | .6653237 .8055456 |
| L1.   |                  |          |      |       |               |
| ma    | .1353086        | .0513156 | 2.64  | 0.008 | .0347319 .2358853 |
| L2.   |                  |          |      |       |               |
| /sigma | .0011129 | .0000401 | 27.77 | 0.000 | .0010344 .0011914 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.
All the parameters are statistically significant, and they indicate a high degree of dependence.

Below we nest the previously fit ARMA model into an ARFIMA model.

```
. arfima S12.log, ar(1) ma(2)
Iteration 0:  log likelihood = 2006.0757
Iteration 1:  log likelihood = 2006.0774  (backed up)
Iteration 2:  log likelihood = 2006.0775  (backed up)
Iteration 3:  log likelihood = 2006.0804
Iteration 4:  log likelihood = 2006.0805
Refining estimates:
Iteration 0:  log likelihood = 2006.0805
Iteration 1:  log likelihood = 2006.0805
```

ARFIMA regression

```
Sample:  1960m1 - 1990m12  Number of obs = 372
Wald chi2(3) =  248.88
Log likelihood = 2006.0805  Prob > chi2 = 0.0000
```

|                | Coef.  | Std. Err. |    z  |   P>|z| |    [95% Conf. Interval] |
|----------------|--------|-----------|-------|-------|------------------------|
| S12.log        |        |           |       |       |                        |
| _cons          | 0.003616 | 0.0012968 | 2.79  | 0.005 | 0.0010743 – 0.0061578  |
| ARFIMA         |        |           |       |       |                        |
| ar             |        |           |       |       |                        |
| L1.            | 0.2160894 | 0.1015583 | 2.13  | 0.033 | 0.0170387 – 0.41514    |
| ma             |        |           |       |       |                        |
| L2.            | 0.1633916 | 0.0516913 | 3.16  | 0.002 | 0.0620786 – 0.2647046  |
| d              | 0.4042573 | 0.0805449 | 5.02  | 0.000 | 0.2463923 – 0.5621223  |
| /sigma2        | 1.20e-06 | 8.84e-08  | 13.63 | 0.000 | 1.03e-06 – 1.38e-06    |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

All the parameters are statistically significant at the 5% level. That the confidence interval for the fractional-difference parameter $d$ includes numbers greater than 0.5 is evidence that the series may be nonstationary. Alternatively, we proceed as if the series is stationary, and the wide confidence interval for $d$ reflects the difficulty of fitting a complicated dynamic model with only 372 observations.

With the above caveat, we can now proceed to compare the interpretations of the ARMA and ARFIMA estimates. We compare these estimates in terms of their implied spectral densities. The spectral density of a stationary time series describes the relative importance of components at different frequencies. See [TS] psdensity for an introduction to spectral densities.

Below we quietly refit the ARMA model and use psdensity to estimate the parametric spectral density implied by the ARMA parameter estimates.

```
. quietly arima S12.log, ar(1) ma(2)
. psdensity d_arma omega1
```

The `psdensity` command above put the estimated ARMA spectral density into the new variable d_arma at the frequencies stored in the new variable omega1.

Below we quietly refit the ARFIMA model and use `psdensity` to estimate the long-run parametric spectral density and then the short-run parametric spectral density implied by the ARFIMA parameter estimates. The long-run estimates use the estimated $d$, and the short-run estimates set $d$ to 0 (as is
implied by specifying the `smemory` option). The long-run estimates describe the fractionally integrated series, and the short-run estimates describe the fractionally differenced series.

```
. quietly arfima S12.log, ar(1) ma(2)
. psdensity d_arfima omega2
. psdensity ds_arfima omega3, smemory
```

Now that we have the ARMA estimates, the long-run ARFIMA estimates, and the short-run ARFIMA estimates, we graph them below.

```
. line d_arma d_arfima omega1, name(lmem) nodraw
. line d_arma ds_arfima omega1, name(smem) nodraw
. graph combine lmem smem, cols(1) xcommon
```

The top graph contains a plot of the spectral densities implied by the ARMA parameter estimates and by the long-run ARFIMA parameter estimates. As discussed by Granger and Joyeux (1980), the two models imply different spectral densities for frequencies close to 0 when $d > 0$. When $d > 0$, the spectral density implied by the ARFIMA estimates diverges to infinity, whereas the spectral density implied by the ARMA estimates remains finite at frequency 0 for stable ARMA processes. This difference reflects the ability of ARFIMA models to capture long-run effects that ARMA models only capture as the parameters approach those of an unstable model.

The bottom graph contains a plot of the spectral densities implied by the ARMA parameter estimates and by the short-run ARFIMA parameter estimates, which are the ARMA parameters for the fractionally differenced process. Comparing the two plots illustrates the ability of the short-run ARFIMA parameters to capture both low-frequency and high-frequency components in the fractionally differenced series. In contrast, the ARMA parameters captured only low-frequency components in the fractionally integrated series.

Comparing the ARFIMA and ARMA spectral densities in the two graphs illustrates that the additional fractional-difference parameter allows the ARFIMA model to identify both long-run and short-run effects, which the ARMA model confounds.
Technical note

As noted above, the spectral density of an ARFIMA process with \( d > 0 \) diverges to infinity as the frequency goes to 0. In contrast, the spectral density of an ARFIMA process with \( d < 0 \) is 0 at frequency 0.

The autocorrelation function of an ARFIMA process with \( d < 0 \) also decays at the slower hyperbolic rate. ARFIMA processes with \( d < 0 \) are sometimes called antipersistent because all the autocorrelations for lags greater than 0 are negative.

Hosking (1981), Baillie (1996), and others refer to ARFIMA processes with \( d < 0 \) as “intermediate memory” processes and ARFIMA processes with \( d > 0 \) as long-memory processes. Box, Jenkins, and Reinsel (2008, 429) define long-memory processes as those with the slower hyperbolic rate of decay, which includes ARFIMA processes with \( d < 0 \). We follow Box, Jenkins, and Reinsel (2008) and thus call ARFIMA processes for \(-0.5 < d < 0\) and \(0 < d < 0.5\) long-memory processes.

Sowell (1992a) uses the properties of ARFIMA processes with \( d < 0 \) to derive tests for whether a series was generated by an \( I(1) \) process or an \( I(d) \) process with \( d < 1 \).

Example 3

In this example, we use arfima to test whether a series is nonstationary. More specifically, we test whether the series was generated by an \( I(1) \) process by testing whether the first difference of the series is overdifferenced.

We have monthly data on the log of the number of reported cases of mumps in New York City between January 1928 and December 1972. We believe that the series is stationary, after accounting for the monthly seasonal effects. We use an ARFIMA model for differenced series to test the null hypothesis of nonstationarity. We use the confidence interval for the \( d \) parameter from an ARFIMA model for the first difference of the log of the series to perform the test. If the right-hand end of the 95% CI is less than 0, we conclude that the differenced series was overdifferenced, which implies that the original series was not nonstationary.

More formally, if \( y_t \) is \( I(1) \), then \( \Delta y_t = y_t - y_{t-1} \) must be \( I(0) \). If \( \Delta y_t \) is \( I(d) \) with \( d < 0 \), then \( \Delta y_t \) is overdifferenced and \( y_t \) is \( I(d) \) with \( d < 1 \).

We use seasonal indicators to account for the seasonal effects. In the output below, we specify the mpl option to use the MPL estimator that is less biased in the presence of covariates.

arfima computes the maximum likelihood estimates (MLE) for the parameters of this stationary and invertible Gaussian process. Alternatively, the maximum MPL estimates may be computed. See Methods and formulas for a description of these two estimation techniques, but suffice it to say that the MLE estimates for \( d \) are biased in the presence of exogenous variables, even the constant term, for small samples. The MPL estimator reduces this bias; see Hauser (1999) and Doornik and Ooms (2004).
. use https://www.stata-press.com/data/r16/mumps2, clear  

. arfima D.log i.month, ma(1 2) mpl

Iteration 0: log modified profile likelihood = 53.766763
Iteration 1: log modified profile likelihood = 54.388641
Iteration 2: log modified profile likelihood = 54.934726 (backed up)
Iteration 3: log modified profile likelihood = 54.937524 (backed up)
Iteration 4: log modified profile likelihood = 55.002186
Iteration 5: log modified profile likelihood = 55.20462
Iteration 6: log modified profile likelihood = 55.205939
Iteration 7: log modified profile likelihood = 55.205949
Iteration 8: log modified profile likelihood = 55.205949
Refining estimates:
Iteration 0: log modified profile likelihood = 55.205949
Iteration 1: log modified profile likelihood = 55.205949

ARFIMA regression
Sample: 1928m2 - 1972m6
Number of obs = 533
Wald chi2(14) = 1360.28
Log modified profile likelihood = 55.205949
Prob > chi2 = 0.0000

| D.log | Coef. Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|-------|----------------|------|------|---------------------|
| D.log |                |      |      |                     |
| month |                |      |      |                     |
| February | -.220719 | .0428112 | -5.16 | 0.000 | -.3046275 -.1368105 |
| March | .0314683 | .0424718 | 0.74 | 0.459 | -.0517749 .1147115 |
| April | -.2800296 | .0460084 | -6.09 | 0.000 | -.3702043 -.1898548 |
| May | -.3703179 | .0449932 | -8.23 | 0.000 | -.4585029 -.2821329 |
| June | -.4722035 | .0446764 | -10.57 | 0.000 | -.5597676 -.3846394 |
| July | -.9613239 | .0448375 | -21.44 | 0.000 | -1.049204 -.873444 |
| August | -1.063042 | .0449272 | -23.66 | 0.000 | -1.151098 -.9749868 |
| September | -.7577301 | .0452529 | -16.74 | 0.000 | -.8464242 -.669036 |
| October | -.3024251 | .0462887 | -6.53 | 0.000 | -.3931494 -.2117009 |
| November | -.0115317 | .0426911 | -0.27 | 0.787 | -.0952046 .0721413 |
| December | .0247135 | .0430401 | 0.57 | 0.566 | -.0596435 .1090705 |
| _cons | .3656807 | .0303215 | 12.06 | 0.000 | .3062517 .4251096 |

ARFIMA

| ma |                |      |      |                     |
| L1. | .258056 | .0684813 | 3.77 | 0.000 | .1239135 .3921985 |
| L2. | .1972011 | .0506439 | 3.89 | 0.000 | .0979409 .2964612 |
| d | -.2329426 | .067336 | -3.46 | 0.001 | -.3649186 -.1009665 |

We interpret the fact that the estimated 95% CI is strictly less than 0 to mean that the differenced series is overdifferenced, which implies that the original series is stationary.
Stored results

`arfima` stores the following in `e()`:

Scalars
- `e(N)`  number of observations
- `e(k)`  number of parameters
- `e(k_eq)`  number of equations in `e(b)`
- `e(df_m)`  model degrees of freedom
- `e(ll)`  log likelihood
- `e(chi2)`  $\chi^2$
- `e(p)`  $p$-value for model test
- `e(s2)`  idiosyncratic error variance estimate, if `e(method) = mpl`
- `e(tmin)`  minimum time
- `e(tmax)`  maximum time
- `e(ar_max)`  maximum AR lag
- `e(ma_max)`  maximum MA lag
- `e(constant)`  0 if `noconstant`, 1 otherwise
- `e(rank)`  rank of `e(V)`
- `e(ic)`  number of iterations
- `e(rc)`  return code
- `e(converged)`  1 if converged, 0 otherwise

Macros
- `e(cmd)`  `arfima`
- `e(cmdline)`  command as typed
- `e(depvar)`  name of dependent variable
- `e(tvar)`  time variable
- `e(covariates)`  list of covariates
- `e(method)`  `mle` or `mpl`
- `e(eqnames)`  names of equations
- `e(title)`  title in estimation output
- `e(tmins)`  formatted minimum time
- `e(tmaxs)`  formatted maximum time
- `e(chi2type)`  Wald; type of model $\chi^2$ test
- `e(vce)`  vcetype specified in `vce()`
- `e(vcetype)`  title used to label Std. Err.
- `e(ma)`  lags for MA terms
- `e(ar)`  lags for AR terms
- `e(technique)`  maximization technique
- `e(tech_steps)`  number of iterations performed before switching techniques
- `e(properties)`  `b V`
- `e(estat_cmd)`  program used to implement `estat`
- `e(predict)`  program used to implement `predict`
- `e(marginsok)`  predictions allowed by `margins`
- `e(marginsnotok)`  predictions disallowed by `margins`
- `e(asbalanced)`  factor variables `fvset` as `asbalanced`
- `e(asobserved)`  factor variables `fvset` as `asobserved`

Matrices
- `e(b)`  coefficient vector
- `e(Cns)`  constraints matrix
- `e(ilog)`  iteration log (up to 20 iterations)
- `e(gradient)`  gradient vector
- `e(V)`  variance–covariance matrix of the estimators
- `e(V_modelbased)`  model-based variance

Functions
- `e(sample)`  marks estimation sample
Methods and formulas

Methods and formulas are presented under the following headings:

Introduction
The likelihood function
The autocovariance function
The profile likelihood
The MPL

Introduction

We model an observed second-order stationary time-series \( y_t, t = 1, \ldots, T \), using the ARFIMA \((p,d,q)\) model defined as

\[
\rho(L^p)(1-L)^d(y_t-x_t \beta) = \theta(L^q)\epsilon_t
\]

where

\[
\rho(L^p) = 1 - \rho_1 L - \rho_2 L^2 - \cdots - \rho_p L^p
\]

\[
\theta(L^q) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q
\]

\[
(1-L)^d = \sum_{j=0}^{\infty} (-1)^j \frac{\Gamma(j+d)}{\Gamma(j+1)\Gamma(d)} L^j
\]

and the lag operator is defined as \( L^j y_t = y_{t-j}, t = 1, \ldots, T \) and \( j = 1, \ldots, t-1; \epsilon_t \sim N(0, \sigma^2) \);
\( \Gamma() \) is the gamma function; and \(-0.5 < d < 0.5, d \neq 0\). The row vector \( x_t \) contains the exogenous variables specified as \textit{indepvars} in the \texttt{arfima} syntax.

The process is stationary and invertible for \(-0.5 < d < 0.5\); the roots of the AR polynomial, \( \rho(z) = 1 - \rho_1 z - \rho_2 z^2 - \cdots - \rho_p z^p = 0 \), and the MA polynomial, \( \theta(z) = 1 + \theta_1 z + \theta_2 z^2 + \cdots + \theta_q z^q = 0 \), lie outside the unit circle and there are no common roots. When \( 0 < d < 0.5 \), the process has long memory in that the autocovariance function, \( \gamma_h \), decays to 0 at a hyperbolic rate, such that \( \sum_{h=-\infty}^{\infty} |\gamma_h| = \infty \). When \(-0.5 < d < 0 \), the process also has long memory in that the autocovariance function, \( \gamma_h \), decays to 0 at a hyperbolic rate such that \( \sum_{h=-\infty}^{\infty} |\gamma_h| < \infty \). (As discussed in the text, some authors refer to ARFIMA processes with \(-0.5 < d < 0\) as having intermediate memory, but we follow Box, Jenkins, and Reinsel [2008] and refer to them as long-memory processes.)


The likelihood function

Estimation of the ARFIMA parameters \( \rho, \theta, d, \beta \) and \( \sigma^2 \) is done by the method of maximum likelihood. The log Gaussian likelihood of \( y \) given parameter estimates \( \hat{\eta} = (\hat{\rho}', \hat{\theta}', \hat{d}, \hat{\beta}', \hat{\sigma}^2) \) is

\[
\ell(y|\hat{\eta}) = -\frac{1}{2} \{ T \log(2\pi) + \log|\hat{V}| + (y-X\hat{\beta})'\hat{V}^{-1}(y-X\hat{\beta}) \} \tag{2}
\]
where the covariance matrix $V$ has a Toeplitz structure

$$
V = \begin{pmatrix}
\gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{T-1} \\
\gamma_1 & \gamma_0 & \gamma_1 & \cdots & \gamma_{T-2} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\gamma_{T-1} & \gamma_{T-2} & \gamma_{T-3} & \cdots & \gamma_0
\end{pmatrix}
$$

$\text{Var}(y_t) = \gamma_0$, $\text{Cov}(y_t, y_{t-h}) = \gamma_h$ (for $h = 1, \ldots, t-1$), and $t = 1, \ldots, T$ (Sowell 1992b).

We use the Durbin–Levinson algorithm (Palma 2007; Golub and Van Loan 2013) to factor and invert $V$. Using only the vector of autocovariances $\gamma$, the Durbin–Levinson algorithm will compute

$$
\hat{\epsilon} = \hat{D}^{-0.5} \hat{L}^{-1} (y - X\hat{\beta}),
$$

where $L$ is lower triangular and $V = LDL'$ and $D = \text{Diag}(\nu)$, $\nu_t = \text{Var}(y_t)$. The algorithm performs these computations without generating the $T \times T$ matrix $L^{-1}$.

During optimization, we restrict the fractional-integration parameter to $(-0.5, 0.5)$ using a logistic transform,

$$
d^* = \text{log}\left\{ \frac{x + 0.5}{0.5 - x} \right\},
$$

so that the range of $d^*$ encompasses the real line. During the “Refining estimates” step, the fractional-integration parameter is transformed back to the restricted space, where we obtain its standard error from the observed information matrix.

The autocovariance function

Computation of the autocovariances $\gamma_h$ is given by Sowell (1992b) with numerical enhancements by Doornik and Ooms (2003) and is reviewed by Palma (2007, sec. 3.2.4). We reproduce it here.

The autocovariance of an ARFIMA$(0,d,0)$ process is

$$
\gamma_h^* = \sigma^2 \frac{\Gamma(1 - 2d)}{\Gamma(1-d)\Gamma(d)} \frac{\Gamma(h + d)}{\Gamma(1 + h - d)}
$$

where $h = 0, 1, \ldots$. For ARFIMA$(p,d,q)$, we have

$$
\gamma_h = \sigma^2 \sum_{i=-q}^{q} \sum_{j=1}^{p} \psi(i) \xi_j C(d, p + i - h, \rho_j)
$$

where

$$
\psi(i) = \min(q, q+i) \sum_{k=\max(0,i)}^{\min(q,q+i)} \theta_k \theta_{k-i},
$$

$$
\xi_j = \left\{ \rho_j \prod_{i=1}^{p} (1 - \rho_i \rho_j) \prod_{m \neq j} (\rho_j - \rho_m) \right\}^{-1}
$$

and

$$
C(d,h, \rho) = \frac{\gamma_h^*}{\sigma^2} \left\{ \rho^{2p} F(d+h, 1, 1-d + h, \rho) + F(d-h, 1, 1-d - h, \rho) - 1 \right\}
$$

$F(\cdot)$ is the hypergeometric series (Zwillinger [Gradshteyn and Ryzhik] 2015)

$$
F(a, b, c, x) = 1 + \frac{ab}{c \cdot 1} x + \frac{a(a+1)b(b+1)}{c(c+1) \cdot 1 \cdot 2} x^2 + \frac{a(a+1)(a+2)b(b+1)(b+2)}{c(c+1)(c+2) \cdot 1 \cdot 2 \cdot 3} x^3 + \ldots
$$

The series recursions are evaluated backward as Doornik and Ooms (2003) emphasize. Doornik and Ooms (2003) also provide other computational enhancements, such as not dividing by $\rho_j$ in (3).
The profile likelihood

Doornik and Ooms (2003) show that the parameters $\sigma^2$ and $\beta$ can be concentrated out of the likelihood. Using (2), the MLE for $\sigma^2$ is

$$\hat{\sigma}^2 = \frac{1}{T}(y - X\hat{\beta})'(\hat{R}^{-1}X)'(y - X\hat{\beta})$$

(4)

where $R = \frac{1}{\sigma^2}V$ and

$$\hat{\beta} = (X'\hat{R}^{-1}X)^{-1}X'\hat{R}^{-1}y$$

(5)

is the weighted least-squares estimates for $\beta$. Substituting (4) into (2) results in the profile likelihood

$$\ell_p(y|\hat{\eta}_r) = -\frac{T}{2}\left\{1 + \log(2\pi) + \frac{1}{T} \log|\hat{R}| + \log\hat{\sigma}^2\right\}$$

We compute the MLEs using the profile likelihood for the reduced parameter set $\eta_r = (\rho', \theta', d)$. Equations (4) and (5) provide MLEs for $\sigma^2$ and $\beta$ to create the full parameter vector $\eta = (\beta', \rho', \theta', d, \sigma^2)$. We follow with the “Refining estimates” step, optimizing on the log likelihood (1). The refining step does not change the estimates; it produces the coefficient variance–covariance matrix from the observed information matrix.

Using this profile likelihood prevents the use of the BHHH optimization method because there are no observation-level scores.

The MPL

The small-sample MLE for $d$ can be biased when there are exogenous variables in the model. The MPL reduces this bias (Hauser 1999; Doornik and Ooms 2004). The mpl option will direct arfima to use this optimization criterion. The MPL is expressed as

$$\ell_m(y|\hat{\eta}_r) = -\frac{T}{2}\left\{1 + \log(2\pi)\right\} - \left(\frac{1}{T} - \frac{1}{2}\right) \log|\hat{R}| - \left(\frac{T - k - 2}{2}\right) \log\hat{\sigma}^2 - \frac{1}{2} \log|X'\hat{R}^{-1}X|$$

where $k = \text{rank}(X)$ (An and Bloomfield 1993).

There is no MPL estimator for $\sigma^2$, and you will notice its absence from the coefficient table. However, the unbiased estimate assuming ARFIMA(0,0,0),

$$\tilde{\sigma}^2 = \frac{(y - X\hat{\beta})'(\hat{R}^{-1}X)'(y - X\hat{\beta})}{T - k}$$

is stored in e() for postestimation computation of the forecast and residual root mean squared errors.

References


Also see

[T] **arfima postestimation** — Postestimation tools for arfima

[T] **arima** — ARIMA, ARMAX, and other dynamic regression models

[T] **sspace** — State-space models

[T] **tsset** — Declare data to be time-series data

[U] **20 Estimation and postestimation commands**
Postestimation commands

The following postestimation commands are of special interest after \texttt{arfima}:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{estat acplot}</td>
<td>estimate autocorrelations and autocovariances</td>
</tr>
<tr>
<td>\texttt{irf}</td>
<td>create and analyze IRFs</td>
</tr>
<tr>
<td>\texttt{psdensity}</td>
<td>estimate the spectral density</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{contrast}</td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td>*\texttt{estat ic}</td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td>\texttt{estat summarize}</td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td>\texttt{estat vce}</td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td>\texttt{estimates}</td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td>\texttt{forecast}</td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td>\texttt{lincom}</td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td>\texttt{lrtest}</td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td>*\texttt{margins}</td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td>*\texttt{marginsplot}</td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td>*\texttt{nlcom}</td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td>\texttt{predict}</td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td>*\texttt{predictnl}</td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td>\texttt{pwcompare}</td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td>\texttt{test}</td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td>\texttt{testnl}</td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>

* \texttt{estat ic}, \texttt{margins}, \texttt{marginsplot}, \texttt{nlcom}, and \texttt{predictnl} are not appropriate after \texttt{arfima, mpl}.
predict

Description for predict

predict creates a new variable containing predictions such as expected values, fractionally differenced series, and innovations. All predictions are available as static one-step-ahead predictions, and the dependent variable is also available as a dynamic multistep prediction.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [ type ] newvar [ if ] [ in ] [ , statistic options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>predicted values; the default</td>
</tr>
<tr>
<td>residuals</td>
<td>predicted innovations</td>
</tr>
<tr>
<td>rstandard</td>
<td>standardized innovations</td>
</tr>
<tr>
<td>fdifference</td>
<td>fractionally differenced series</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ... if wanted only for the estimation sample.`

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>rmse( [ type ] newvar )</td>
<td>put the estimated root mean squared error of the predicted statistic in a new variable; only permitted with options xb and residuals</td>
</tr>
<tr>
<td>dynamic( datetime )</td>
<td>forecast the time series starting at <code>datetime</code>; only permitted with option xb</td>
</tr>
</tbody>
</table>

`datetime` is a # or a time literal, such as `td(1jan1995)` or `tq(1995q1)`; see [D] Datetime.

Options for predict

- **xb**, the default, calculates the predictions for the level of `depvar`.
- `residuals` calculates the predicted innovations.
- `rstandard` calculates the standardized innovations.
- `fdifference` calculates the fractionally differenced predictions of `depvar`.
**Options**

- `rmse(type newvar)` puts the root mean squared errors of the predicted statistics into the specified new variables. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error. `rmse()` is only permitted with the `xb` and `residuals` options.

- `dynamic(datetime)` specifies when `predict` starts producing dynamic forecasts. The specified `datetime` must be in the scale of the time variable specified in `tsset`, and the `datetime` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may only be specified with `xb`.

**margins**

**Description for margins**

`margins` estimates margins of response for expected values.

**Menu for margins**

Statistics > Postestimation

**Syntax for margins**

```
margins [marginlist] [ , options ]
margins [marginlist] , predict(statistic ...) [ options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xb</code></td>
<td>predicted values; the default</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>not allowed with <code>margins</code></td>
</tr>
<tr>
<td><code>rstandard</code></td>
<td>not allowed with <code>margins</code></td>
</tr>
<tr>
<td><code>fdifference</code></td>
<td>not allowed with <code>margins</code></td>
</tr>
</tbody>
</table>

Statistics not allowed with `margins` are functions of stochastic quantities other than `e(b)`. For the full syntax, see [R] margins.

**Remarks and examples**

Remarks are presented under the following headings:

- Forecasting after ARFIMA
- IRF results for ARFIMA
Forecasting after ARFIMA

We assume that you have already read [TS] arfima. In this section, we illustrate some of the features of predict after fitting an ARFIMA model using arfima.

Example 1

We have monthly data on the one-year Treasury bill secondary market rate imported from the Federal Reserve Bank (FRED) database using import fred; see [D] import fred. Below we fit an ARFIMA model with two autoregressive terms and one moving-average term to the data.

```
. use https://www.stata-press.com/data/r16/tb1yr
    (FRED, 1-year treasury bill; secondary market rate, monthly 1959-2001)
. arfima tb1yr, ar(1/2) ma(1)
Iteration 0:  log likelihood = -235.31856
Iteration 1:  log likelihood = -235.26104  (backed up)
Iteration 2:  log likelihood = -235.25974  (backed up)
Iteration 3:  log likelihood = -235.2544  (backed up)
Iteration 4:  log likelihood = -235.13354
Iteration 5:  log likelihood = -235.13066
Iteration 6:  log likelihood = -235.12107
Iteration 7:  log likelihood = -235.11917
Iteration 8:  log likelihood = -235.11869
Iteration 9:  log likelihood = -235.11868
Refining estimates:
Iteration 0:  log likelihood = -235.11868
Iteration 1:  log likelihood = -235.11868
ARFIMA regression
Sample: 1959m7 - 2001m8  Number of obs = 506
    Wald chi2(4) = 1864.14
Log likelihood = -235.11868  Prob > chi2 = 0.0000

    OIM
                 Coef.  Std. Err.     z  P>|z|     [95% Conf. Interval]
  tb1yr
     _cons  5.496709   2.920336   1.88   0.060    -.227044    11.22046

ARFIMA
     ar    L1.  .2326113   .113682   2.05   0.041     .0097987    .4554239
    L2.  .3885213   .083569   4.65   0.000     .2247277    .5523149
     ma    L1.  .7755848   .0696903  11.58   0.000     .6443451    .9068245
     d    L1.  .4606484   .0646628   7.12   0.000     .3339116    .5873852
    /sigma2  .1466495   .009232   15.88   0.000     .1285551    .1647439
```

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

All the parameters are statistically significant at the 5% level, and they indicate a high degree of dependence in the series. In fact, the confidence interval for the fractional-difference parameter $d$ indicates that the series may be nonstationary. We will proceed as if the series is stationary and suppose that it is fractionally integrated of order 0.46.
We begin our postestimation analysis by predicting the series in sample:

```
.predict ptb
(option xb assumed)
```

We continue by using the estimated fractional-difference parameter to fractionally difference the original series and by plotting the original series, the predicted series, and the fractionally differenced series. See [TS] arfima for a definition of the fractional-difference operator.

```
.predict fdtb, fdifference
.twoway tsline tb1yr ptb fdtb, legend(cols(1))
```

The above graph shows that the in-sample predictions appear to track the original series well and that the fractionally differenced series looks much more like a stationary series than does the original.

### Example 2

In this example, we use the above estimates to produce a dynamic forecast and a confidence interval for the forecast for the one-year treasury bill rate and plot them.

We begin by extending the dataset and using `predict` to put the dynamic forecast in the new `ftb` variable and the root mean squared error of the forecast in the new `rtb` variable. (As discussed in `Methods and formulas`, the root mean squared error of the forecast accounts for the idiosyncratic error but not for the estimation error.)

```
.tsappend, add(12)
.predict ftb, xb dynamic(tm(2001m9)) rmse(rtb)
```
Now we compute a 90% confidence interval around the dynamic forecast and plot the original series, the in-sample forecast, the dynamic forecast, and the confidence interval of the dynamic forecast.

```
. scalar z = invnormal(0.95)
. generate lb = ftb - z*rtb if month>=tm(2001m9)
(506 missing values generated)
. generate ub = ftb + z*rtb if month>=tm(2001m9)
(506 missing values generated)
. twoway tsline tb1yr ftb if month>tm(1998m12) ||
>   tsrline lb ub if month>=tm(2001m9),
>   legend(cols(1) label(3 "90% prediction interval"))
```

---

**IRF results for ARFIMA**

We assume that you have already read [TS] irf and [TS] irf create. In this section, we illustrate how to calculate the impulse–response function (IRF) of an ARFIMA model.

- **Example 3**

Here we use the estimates obtained in example 1 to calculate the IRF of the ARFIMA model; see [TS] irf and [TS] irf create for more details about IRFs.
The figure shows that a shock to $tb1yr$ causes an initial spike in $tb1yr$, after which the impact of the shock starts decaying slowly. This behavior is characteristic of long-memory processes.

Methods and formulas

Denote $\gamma_h$, $h = 1, \ldots, t$, to be the autocovariance function of the ARFIMA($p, d, q$) process for two observations, $y_t$ and $y_{t-h}$, $h$ time periods apart. The covariance matrix $V$ of the process of length $T$ has a Toeplitz structure of

$$V = \begin{pmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{T-1} \\ \gamma_1 & \gamma_0 & \gamma_1 & \cdots & \gamma_{T-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma_{T-1} & \gamma_{T-2} & \gamma_{T-3} & \cdots & \gamma_0 \end{pmatrix}$$

where the process variance is $\gamma_0 = \text{Var}(y_t)$. We factor $V = LDL'$, where $L$ is lower triangular and $D = \text{Diag}(\nu_t)$. The structure of $L^{-1}$ is of importance.

$$L^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -\tau_{1,1} & 1 & 0 & \cdots & 0 & 0 \\ -\tau_{2,2} & -\tau_{2,1} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -\tau_{T-1,T-1} & -\tau_{T-1,T-2} & -\tau_{T-1,T-3} & \cdots & -\tau_{T-1,1} & 1 \end{pmatrix}$$

Let $z_t = y_t - x_t\beta$. The best linear predictor of $z_{t+1}$ based on $z_1, z_2, \ldots, z_t$ is $\hat{z}_{t+1} = \sum_{k=1}^{t} \tau_{k,k}z_{t-k+1}$. Define $-\tau_t = (-\tau_{t,t}, -\tau_{t,t-1}, \ldots, -\tau_{t-1,1})$ to be the $t$th row of $L^{-1}$ up to, but not including, the diagonal. Then $\tau_t = V_t^{-1}\gamma_t$, where $V_t$ is the $t \times t$ upper left submatrix of $V$ and $\gamma_t = (\gamma_1, \gamma_2, \ldots, \gamma_t)'$. Hence, the best linear predictor of the innovations is computed as $\hat{\epsilon} = L^{-1}z$, and the one-step predictions are $\hat{y} = \hat{\epsilon} + X\beta$. In practice, the computation is
\[ \hat{y} = \hat{L}^{-1} (y - X\hat{\beta}) + X\hat{\beta} \]

where \( \hat{L} \) and \( \hat{V} \) are computed from the maximum likelihood estimates. We use the Durbin–Levinson algorithm (Palma 2007; Golub and Van Loan 2013) to factor \( \hat{V} \), invert \( \hat{L} \), and scale \( y - X\hat{\beta} \) using only the vector of estimated autocovariances \( \hat{\gamma} \).

The prediction error variances of the one-step predictions are computed recursively in the Durbin–Levinson algorithm. They are the \( \nu_t \) elements in the diagonal matrix \( D \) computed from the Cholesky factorization of \( V \). The recursive formula is \( \nu_0 = \gamma_0 \) and \( \nu_t = \nu_{t-1}(1 - \tau_{t,t}^2) \).

Forecasting is carried out as described by Beran (1994, sec. 8.7), \( \hat{z}_{T+k} = \hat{\gamma}_k' \hat{V}^{-1} \hat{z} \), where \( \hat{\gamma}_k' = (\hat{\gamma}_{T+k-1}, \hat{\gamma}_{T+k-2}, \ldots, \hat{\gamma}_k) \). The forecast mean squared error is computed as \( \text{MSE}(\hat{z}_{T+k}) = \gamma_0 - \hat{\gamma}_k' \hat{V}^{-1} \hat{\gamma}_k \). Computation of \( \hat{V}^{-1} \hat{\gamma}_k \) is carried out efficiently using algorithm 4.7.2 of Golub and Van Loan (2013).

### References


### Also see

[TS] arfima — Autoregressive fractionally integrated moving-average models

[TS] estat acplot — Plot parametric autocorrelation and autocovariance functions

[TS] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] psdensity — Parametric spectral density estimation after arima, arfima, and ucm

[U] 20 Estimation and postestimation commands
Description

`arima` fits univariate models for a time series, where the disturbances are allowed to follow a linear autoregressive moving-average (ARMA) specification. When independent variables are included in the specification, such models are often called ARMAX models; and when independent variables are not specified, they reduce to Box–Jenkins autoregressive integrated moving-average (ARIMA) models in the dependent variable.

Quick start

AR(1) model using `tsset` data

```plaintext
arima y, ar(1)
```

MA(1) model

```plaintext
arima y, ma(1)
```

ARMA(2,1) model

```plaintext
arima y, ar(1/2) ma(1)
```

Same as above

```plaintext
arima y, arima(2,0,1)
```

As above, and take first difference of `y` and restrict estimation to years 1990 to 2010

```plaintext
arima D.y if tin(1990,2010), ar(1/2) ma(1)
```

Same as above

```plaintext
arima y if tin(1990,2010), arima(2,1,1)
```

Multiplicative SARIMA model with quarterly data

```plaintext
arima y, arima(2,1,1) sarima(2,1,0,4)
```

ARMAX model with covariates `x1` and `x2`, an AR(1) process, and robust standard errors

```plaintext
arima y x, ar(1) vce(robust)
```
Syntax

Basic syntax for a regression model with ARMA disturbances
\[ \text{arima depvar [indepvars], ar(numlist) ma(numlist)} \]

Basic syntax for an ARIMA\((p,d,q)\) model
\[ \text{arima depvar, arima(#p,#d,#q)} \]

Basic syntax for a multiplicative seasonal ARIMA\((p,d,q) \times (P,D,Q)_s\) model
\[ \text{arima depvar, arima(#p,#d,#q) sarima(#P,#D,#Q,#s)} \]

Full syntax
\[ \text{arima depvar [indepvars] [if] [in] [weight] [ , options]} \]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noconstant</td>
<td>suppress constant term</td>
</tr>
<tr>
<td>ar(numlist)</td>
<td>specify ARIMA((p,d,q)) model for dependent variable</td>
</tr>
<tr>
<td>ma(numlist)</td>
<td>autoregressive terms of the structural model disturbance</td>
</tr>
<tr>
<td>constraints(constraints)</td>
<td>apply specified linear constraints</td>
</tr>
<tr>
<td>sarima(#P,#D,#Q,#s)</td>
<td>specify period-(s) multiplicative seasonal ARIMA term</td>
</tr>
<tr>
<td>mar(numlist,#s)</td>
<td>multiplicative seasonal autoregressive term; may be repeated</td>
</tr>
<tr>
<td>mma(numlist,#s)</td>
<td>multiplicative seasonal moving-average term; may be repeated</td>
</tr>
<tr>
<td>condition</td>
<td>use conditional MLE instead of full MLE</td>
</tr>
<tr>
<td>savespace</td>
<td>conserve memory during estimation</td>
</tr>
<tr>
<td>diffuse</td>
<td>use diffuse prior for starting Kalman filter recursions</td>
</tr>
<tr>
<td>p0(#</td>
<td>matname)</td>
</tr>
<tr>
<td>state0(#</td>
<td>matname)</td>
</tr>
<tr>
<td>vce(vcetype)</td>
<td>vcetype may be opg, robust, or oim</td>
</tr>
<tr>
<td>level(##)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>detail</td>
<td>report list of gaps in time series</td>
</tr>
<tr>
<td>noconsr</td>
<td>do not display constraints</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats, row spacing, and line width</td>
</tr>
<tr>
<td>maximize_options</td>
<td>control the maximization process; seldom used</td>
</tr>
<tr>
<td>collinear</td>
<td>keep collinear variables</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>
You must \texttt{tsset} your data before using \texttt{arima}; see \cite{tsset}.

\texttt{depvar} and \texttt{indepvars} may contain time-series operators; see \cite{tsset} \texttt{11.4.4 Time-series varlists}.

\texttt{by}, \texttt{fp}, \texttt{rolling}, \texttt{statsby}, and \texttt{xi} are allowed; see \cite{statsby} \texttt{11.1.10 Prefix commands}.

\texttt{iweights} are allowed; see \cite{iweight} \texttt{11.1.6 weight}.

\texttt{collinear} and \texttt{coeflegend} do not appear in the dialog box.

See \cite{estimation} \texttt{Estimation and postestimation commands} for more capabilities of estimation commands.

\section*{Options}

\subsubsection*{Model 1}

\textbf{noconstant}; see \cite{estimation} \texttt{Estimation options}.

\texttt{arima(}\#_p,\#_d,\#_q)\texttt{)} is an alternative, shorthand notation for specifying models with ARMA disturbances. The dependent variable and any independent variables are differenced \#_d times, and 1 through \#_p lags of autocorrelations and 1 through \#_q lags of moving averages are included in the model. For example, the specification

\begin{verbatim}
  . arima D.y, ar(1/2) ma(1/3)
\end{verbatim}

is equivalent to

\begin{verbatim}
  . arima y, arima(2,1,3)
\end{verbatim}

The latter is easier to write for simple ARMAX and ARIMA models, but if gaps in the AR or MA lags are to be modeled, or if different operators are to be applied to independent variables, the first syntax is required.

\texttt{ar(numlist)} specifies the autoregressive terms of the structural model disturbance to be included in the model. For example, \texttt{ar(1/3)} specifies that lags of 1, 2, and 3 of the structural disturbance be included in the model; \texttt{ar(1 4)} specifies that lags 1 and 4 be included, perhaps to account for additive quarterly effects.

If the model does not contain regressors, these terms can also be considered autoregressive terms for the dependent variable.

\texttt{ma(numlist)} specifies the moving-average terms to be included in the model. These are the terms for the lagged innovations (white-noise disturbances).

\texttt{constraints(constraints)}; see \cite{estimation} \texttt{Estimation options}.

If constraints are placed between structural model parameters and ARMA terms, the first few iterations may attempt steps into nonstationary areas. This process can be ignored if the final solution is well within the bounds of stationary solutions.

\subsubsection*{Model 2}

\texttt{sarima(}\#_P,\#_D,\#_Q,\#_s)\texttt{)} is an alternative, shorthand notation for specifying the multiplicative seasonal components of models with ARMA disturbances. The dependent variable and any independent variables are lag-\#_s seasonally differenced \#_D times, and 1 through \#_P seasonal lags of autoregressive terms and 1 through \#_Q seasonal lags of moving-average terms are included in the model. For example, the specification

\begin{verbatim}
  . arima DS12.y, ar(1/2) ma(1/3) mar(1/2,12) mma(1/2,12)
\end{verbatim}

is equivalent to

\begin{verbatim}
  . arima y, arima(2,1,3) sarima(2,1,2,12)
\end{verbatim}
mar(numlist, #) specifies the lag-# multiplicative seasonal autoregressive terms. For example, mar(1/2,12) requests that the first two lag-12 multiplicative seasonal autoregressive terms be included in the model.

mma(numlist, #) specified the lag-# multiplicative seasonal moving-average terms. For example, mma(1 3,12) requests that the first and third (but not the second) lag-12 multiplicative seasonal moving-average terms be included in the model.

Model 3

condition specifies that conditional, rather than full, maximum likelihood estimates be produced. The presample values for $\epsilon_t$ and $\mu_t$ are taken to be their expected value of zero, and the estimate of the variance of $\epsilon_t$ is taken to be constant over the entire sample; see Hamilton (1994, 132). This estimation method is not appropriate for nonstationary series but may be preferable for long series or for models that have one or more long AR or MA lags. diffuse, p0(), and state0() have no meaning for models fit from the conditional likelihood and may not be specified with condition.

If the series is long and stationary and the underlying data-generating process does not have a long memory, estimates will be similar, whether estimated by unconditional maximum likelihood (the default), conditional maximum likelihood (condition), or maximum likelihood from a diffuse prior (diffuse).

In small samples, however, results of conditional and unconditional maximum likelihood may differ substantially; see Ansley and Newbold (1980). Whereas the default unconditional maximum likelihood estimates make the most use of sample information when all the assumptions of the model are met, Harvey (1989) and Ansley and Kohn (1985) argue for diffuse priors often, particularly in ARIMA models corresponding to an underlying structural model.

The condition or diffuse options may also be preferred when the model contains one or more long AR or MA lags; this avoids inverting potentially large matrices (see diffuse below).

When condition is specified, estimation is performed by the arch command (see [TS] arch), and more control of the estimation process can be obtained using arch directly.

condition cannot be specified if the model contains any multiplicative seasonal terms.

savespace specifies that memory use be conserved by retaining only those variables required for estimation. The original dataset is restored after estimation. This option is rarely used and should be used only if there is not enough space to fit a model without the option. However, arima requires considerably more temporary storage during estimation than most estimation commands in Stata.

diffuse specifies that a diffuse prior (see Harvey 1989 or 1993) be used as a starting point for the Kalman filter recursions. Using diffuse, nonstationary models may be fit with arima (see the p0() option below; diffuse is equivalent to specifying p0(1e9)).

By default, arima uses the unconditional expected value of the state vector $\xi_t$ (see Methods and formulas) and the mean squared error (MSE) of the state vector to initialize the filter. When the process is stationary, this corresponds to the expected value and expected variance of a random draw from the state vector and produces unconditional maximum likelihood estimates of the parameters. When the process is not stationary, however, this default is not appropriate, and the unconditional MSE cannot be computed. For a nonstationary process, another starting point must be used for the recursions.

In the absence of nonsample or presample information, diffuse may be specified to start the recursions from a state vector of zero and a state MSE matrix corresponding to an effectively
infinite variance on this initial state. This method amounts to an uninformative and improper prior that is updated to a proper MSE as data from the sample become available; see Harvey (1989).

Nonstationary models may also correspond to models with infinite variance given a particular specification. This and other problems with nonstationary series make convergence difficult and sometimes impossible.

diffuse can also be useful if a model contains one or more long AR or MA lags. Computation of the unconditional MSE of the state vector (see Methods and formulas) requires construction and inversion of a square matrix that is of dimension \( \{\max(p, q + 1)\}^2 \), where \( p \) and \( q \) are the maximum AR and MA lags, respectively. If \( q = 27 \), for example, we would require a 784-by-784 matrix. Estimation with diffuse does not require this matrix.

For large samples, there is little difference between using the default starting point and the diffuse starting point. Unless the series has a long memory, the initial conditions affect the likelihood of only the first few observations.

\[ p0(# | matname) \] is a rarely specified option that can be used for nonstationary series or when an alternate prior for starting the Kalman recursions is desired (see diffuse above for a discussion of the default starting point and Methods and formulas for background).

\( matname \) specifies a matrix to be used as the MSE of the state vector for starting the Kalman filter recursions—\( P_{1|0} \). Instead, one number, \( # \), may be supplied, and the MSE of the initial state vector \( P_{1|0} \) will have this number on its diagonal and all off-diagonal values set to zero.

This option may be used with nonstationary series to specify a larger or smaller diagonal for \( P_{1|0} \) than that supplied by diffuse. It may also be used with state0() when you believe that you have a better prior for the initial state vector and its MSE.

\[ state0(# | matname) \] is a rarely used option that specifies an alternate initial state vector, \( \xi_{1|0} \) (see Methods and formulas), for starting the Kalman filter recursions. If \( # \) is specified, all elements of the vector are taken to be \#. The default initial state vector is state0(0).

\[ vce(vcetype) \] specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that are derived from asymptotic theory (oim, opg); see [R] vce_option.

For state-space models in general and ARMAX and ARIMA models in particular, the robust or quasi–maximum likelihood estimates (QMLEs) of variance are robust to symmetric nonnormality in the disturbances, including, as a special case, heteroskedasticity. The robust variance estimates are not generally robust to functional misspecification of the structural or ARMA components of the model; see Hamilton (1994, 389) for a brief discussion.

\[ level(#); \] see [R] Estimation options.

detail specifies that a detailed list of any gaps in the series be reported, including gaps due to missing observations or missing data for the dependent variable or independent variables.

nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, vsquish, cformat(\%fmt), pformat(\%fmt), sformat(\%fmt), and nolstretch; see [R] Estimation options.
Maximization

\[ \text{maximize} \quad \text{options: difficult, technique(algorithm_spec), iterate(#), [no]\text{log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), itolerance(#), nrtolerance(#), gtolerance(#), nonrtolerance(#), and from(init_specs)}; \text{ see } \text{[R] Maximize for all options except } \text{gtolerance, and see below for information on } \text{gtolerance.} \]

These options are sometimes more important for ARIMA models than most maximum likelihood models because of potential convergence problems with ARIMA models, particularly if the specified model and the sample data imply a nonstationary model.

Several alternate optimization methods, such as Berndt–Hall–Hall–Hausman (BHHH) and Broyden–Fletcher–Goldfarb–Shanno (BFGS), are provided for ARIMA models. Although ARIMA models are not as difficult to optimize as ARCH models, their likelihoods are nevertheless generally not quadratic and often pose optimization difficulties; this is particularly true if a model is nonstationary or nearly nonstationary. Because each method approaches optimization differently, some problems can be successfully optimized by an alternate method when one method fails.

Setting `technique()` to something other than the default or BHHH changes the `vcetype` to `vce(oim)`.

The following options are all related to maximization and are either particularly important in fitting ARIMA models or not available for most other estimators.

- `technique(algorithm_spec)` specifies the optimization technique to use to maximize the likelihood function.
  - `technique(bhhh)` specifies the Berndt–Hall–Hall–Hausman (BHHH) algorithm.
  - `technique(dfp)` specifies the Davidon–Fletcher–Powell (DFP) algorithm.
  - `technique(bfgs)` specifies the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.
  - `technique(nr)` specifies Stata’s modified Newton–Raphson (NR) algorithm.
- You can specify multiple optimization methods. For example,
  \[ \text{technique(bhhh 10 nr 20)} \]
requests that the optimizer perform 10 BHHH iterations, switch to Newton–Raphson for 20 iterations, switch back to BHHH for 10 more iterations, and so on.

The default for `arima` is `technique(bhhh 5 bfgs 10)`.

- `gtolerance(#)` specifies the tolerance for the gradient relative to the coefficients. When \[ |g_i| \leq \text{gtolerance()} \] for all parameters \( b_i \) and the corresponding elements of the gradient \( g_i \), the gradient tolerance criterion is met. The default gradient tolerance for `arima` is `gtolerance(.05)`.

- `gtolerance(999)` may be specified to disable the gradient criterion. If the optimizer becomes stuck with repeated “(backed up)” messages, the gradient probably still contains substantial values, but an uphill direction cannot be found for the likelihood. With this option, results can often be obtained, but whether the global maximum likelihood has been found is unclear.

When the maximization is not going well, it is also possible to set the maximum number of iterations (see `Maximize`) to the point where the optimizer appears to be stuck and to inspect the estimation results at that point.

- `from(init_specs)` allows you to set the starting values of the model coefficients; see `Maximize` for a general discussion and syntax options.

The standard syntax for `from()` accepts a matrix, a list of values, or coefficient name value pairs; see `Maximize`. `arima` also accepts `from(armab0)`, which sets the starting value for all ARMA parameters in the model to zero prior to optimization.
ARIMA models may be sensitive to initial conditions and may have coefficient values that correspond to local maximums. The default starting values for `arima` are generally good, particularly in large samples for stationary series.

The following options are available with `arima` but are not shown in the dialog box: `collinear`, `coeflegend`; see [R] Estimation options.

**Remarks and examples**

Remarks are presented under the following headings:
- Introduction
- ARIMA models
- Multiplicative seasonal ARIMA models
- ARMAX models
- Dynamic forecasting
- Video example

**Introduction**

`arima` fits both standard ARIMA models that are autoregressive in the dependent variable and structural models with ARMA disturbances. Good introductions to the former models can be found in Box, Jenkins, and Reinsel (2008); Hamilton (1994); Harvey (1993); Newton (1988); Diggle (1990); and many others. The latter models are developed fully in Hamilton (1994) and Harvey (1989), both of which provide extensive treatment of the Kalman filter (Kalman 1960) and the state-space form used by `arima` to fit the models. Becketti (2013) discusses ARIMA models and Stata’s `arima` command, and he devotes an entire chapter explaining how the principles of ARIMA models are applied to real datasets in practice.

Consider a first-order autoregressive moving-average process. Then `arima` estimates all the parameters in the model

\[ y_t = x_t \beta + \mu_t \]

**structural equation**

\[ \mu_t = \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \]

**disturbance, ARMA(1,1)**

where \( \rho \) is the first-order autocorrelation parameter \( \theta \) is the first-order moving-average parameter \( \epsilon_t \sim i.i.d. N(0, \sigma^2) \), meaning that \( \epsilon_t \) is a white-noise disturbance

You can combine the two equations and write a general ARMA\((p, q)\) in the disturbances process as

\[ y_t = x_t \beta + \rho_1 (y_{t-1} - x_{t-1} \beta) + \rho_2 (y_{t-2} - x_{t-2} \beta) + \cdots + \rho_p (y_{t-p} - x_{t-p} \beta) + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t \]

It is also common to write the general form of the ARMA model more succinctly using lag operator notation as

\[ \rho(L^p)(y_t - x_t \beta) = \theta(L^q) \epsilon_t \]

**ARMA\((p, q)\)**

where

\[ \rho(L^p) = 1 - \rho_1 L - \rho_2 L^2 - \cdots - \rho_p L^p \]
\[ \theta(L^q) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q \]

and \( L^j y_t = y_{t-j} \).

Multiplicative seasonal ARMAX and ARIMA models can also be fit.
For stationary series, full or unconditional maximum likelihood estimates are obtained via the Kalman filter. For nonstationary series, if some prior information is available, you can specify initial values for the filter by using `state0()` and `p0()` as suggested by Hamilton (1994) or assume an uninformative prior by using the `diffuse` option as suggested by Harvey (1989).

Missing data are allowed and are handled using the Kalman filter and methods suggested by Harvey (1989 and 1993); see Methods and formulas.

**ARIMA models**

Pure ARIMA models without a structural component do not have regressors and are often written as autoregressions in the dependent variable, rather than autoregressions in the disturbances from a structural equation. For example, an ARMA(1,1) model can be written as

\[ y_t = \alpha + \rho y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \]  

(1a)

Other than a scale factor for the constant term \( \alpha \), these models are equivalent to the ARMA in the disturbances formulation estimated by `arima`, though the latter are more flexible and allow a wider class of models.

To see this effect, replace \( x_t \beta \) in the structural equation above with a constant term \( \beta_0 \) so that

\[
\begin{align*}
 y_t &= \beta_0 + \mu_t \\
 &= \beta_0 + \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \\
 &= \beta_0 + \rho (y_{t-1} - \beta_0) + \theta \epsilon_{t-1} + \epsilon_t \\
 &= (1 - \rho)\beta_0 + \rho y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t
\end{align*}
\]

(1b)

Equations (1a) and (1b) are equivalent, with \( \alpha = (1 - \rho)\beta_0 \), so whether we consider an ARIMA model as autoregressive in the dependent variable or disturbances is immaterial. Our illustration can easily be extended from the ARMA(1,1) case to the general ARIMA\((p, d, q)\) case.
Example 1: ARIMA model

Enders (2004, 87–93) considers an ARIMA model of the U.S. Wholesale Price Index (WPI) using quarterly data over the period 1960q1 through 1990q4. The simplest ARIMA model that includes differencing and both autoregressive and moving-average components is the ARIMA(1,1,1) specification. We can fit this model with `arima` by typing

```
  . use https://www.stata-press.com/data/r16/wpi1
  . arima wpi, arima(1,1,1)
```

The estimation results are shown below:

```
ARIMA regression
Sample: 1960q2 - 1990q4    Number of obs = 123
Wald chi2(2) = 310.64
Log likelihood = -135.3513  Prob > chi2 = 0.0000

                     OPG
                   Coef.  Std. Err.     z  P>|z|     [95% Conf. Interval]
D.wpi
   wpi     .7498197   .3340968   2.24  0.025     .0950019   1.404637
_cons    2.24  0.025
```

Examining the estimation results, we see that the AR(1) coefficient is 0.874, the MA(1) coefficient is \(-0.412\), and both are highly significant. The estimated standard deviation of the white-noise disturbance \(\epsilon\) is 0.725.

This model also could have been fit by typing

```
  . arima D.wpi, ar(1) ma(1)
```

The D. placed in front of the dependent variable WPI is the Stata time-series operator for differencing. Thus we would be modeling the first difference in WPI from the second quarter of 1960 through the fourth quarter of 1990 because the first observation is lost because of differencing. This second syntax allows a richer choice of models.
Example 2: ARIMA model with additive seasonal effects

After examining first differences of WPI, Enders chose a model of differences in the natural logarithms to stabilize the variance in the differenced series. The raw data and first difference of the logarithms are graphed below.

On the basis of the autocorrelations, partial autocorrelations (see graphs below), and the results of preliminary estimations, Enders identified an ARMA model in the log-differenced series.

\[ \Delta \ln(wpi_t) = \beta_0 + \rho_1 \{ \Delta \ln(wpi_{t-1}) - \beta_0 \} + \theta_1 \epsilon_{t-1} + \theta_4 \epsilon_{t-4} + \epsilon_t \]
We can fit this model with `arima` and Stata’s standard difference operator:

```
. arima D.ln_wpi, ar(1) ma(1 4)
```

(setting optimization to BHHH)

Iteration 0: log likelihood = 382.67447
Iteration 1: log likelihood = 384.80754
Iteration 2: log likelihood = 384.84749
Iteration 3: log likelihood = 385.39213
Iteration 4: log likelihood = 385.40983
(switching optimization to BFGS)
Iteration 5: log likelihood = 385.9021
Iteration 6: log likelihood = 385.95646
Iteration 7: log likelihood = 386.02979
Iteration 8: log likelihood = 386.03326
Iteration 9: log likelihood = 386.03354
Iteration 10: log likelihood = 386.03357

ARIMA regression
Sample: 1960q2 - 1990q4 Number of obs = 123

Wald chi2(3) = 333.60 Log likelihood = 386.0336 Prob > chi2 = 0.0000

| OPG | D.ln_wpi Coef. Std. Err. z P>|z| [95% Conf. Interval] |
|-----|----------------------|------------------|------------------|------------------|------------------|
| ln_wpi | .0110493 .0048349 2.29 0.022 .0015731 .0205255 |
| _cons | .0110493 .0048349 2.29 0.022 .0015731 .0205255 |

ARMA

| OPG | D.ln_wpi Coef. Std. Err. z P>|z| [95% Conf. Interval] |
|-----|----------------------|------------------|------------------|------------------|------------------|
| ar | .7806991 .0944946 8.26 0.000 .5954931 .965905 |
| L1. | .7806991 .0944946 8.26 0.000 .5954931 .965905 |
| ma | -.3990039 .1258753 -3.17 0.002 -.6457149 -.1522928 |
| L1. | -.3990039 .1258753 -3.17 0.002 -.6457149 -.1522928 |
| L4. | .3090813 .1200945 2.57 0.010 .0737003 .5444622 |
| /sigma | .0104394 .0004702 22.20 0.000 .0095178 .0113609 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

In this final specification, the log-differenced series is still highly autocorrelated at a level of 0.781, though innovations have a negative impact in the ensuing quarter (−0.399) and a positive seasonal impact of 0.309 in the following year.

Technical note

In one way, the results differ from most of Stata’s estimation commands: the standard error of the coefficients is reported as OPG Std. Err. The default standard errors and covariance matrix for `arima` estimates are derived from the outer product of gradients (OPG). This is one of three asymptotically equivalent methods of estimating the covariance matrix of the coefficients (only two of which are usually tractable to derive). Discussions and derivations of all three estimates can be found in Davidson and MacKinnon (1993), Greene (2018), and Hamilton (1994). Bollerslev, Engle, and Nelson (1994) suggest that the OPG estimates are more numerically stable in time-series regressions when the likelihood and its derivatives depend on recursive computations, which is certainly the case for the Kalman filter. To date, we have found no numerical instabilities in either estimate of the covariance matrix—subject to the stability and convergence of the overall model.
Most of Stata’s estimation commands provide covariance estimates derived from the Hessian of the likelihood function. These alternate estimates can also be obtained from 	exttt{arima} by specifying the 	exttt{vce(oim)} option.

### Multiplicative seasonal ARIMA models

Many time series exhibit a periodic seasonal component, and a seasonal ARIMA model, often abbreviated SARIMA, can then be used. For example, monthly sales data for air conditioners have a strong seasonal component, with sales high in the summer months and low in the winter months.

In the previous example, we accounted for quarterly effects by fitting the model

\[
(1 - \rho_1 L) \{ \Delta \ln(wpi_t) - \beta_0 \} = (1 + \theta_1 L + \theta_4 L^4) \epsilon_t
\]

This is an additive seasonal ARIMA model, in the sense that the first- and fourth-order MA terms work additively: \((1 + \theta_1 L + \theta_4 L^4)\).

Another way to handle the quarterly effect would be to fit a multiplicative seasonal ARIMA model. A multiplicative SARIMA model of order \((1, 1, 1) \times (0, 0, 1)_4\) for the \(\ln(wpi_t)\) series is

\[
(1 - \rho_1 L) \{ \Delta \ln(wpi_t) - \beta_0 \} = (1 + \theta_1 L)(1 + \theta_{4,1} L^4) \epsilon_t
\]

or, upon expanding terms,

\[
\Delta \ln(wpi_t) = \beta_0 + \rho_1 \{ \Delta \ln(wpi_t) - \beta_0 \} + \theta_1 \epsilon_{t-1} + \theta_{4,1} \epsilon_{t-4} + \theta_1 \theta_{4,1} \epsilon_{t-5} + \epsilon_t \tag{2}
\]

In the notation \((1, 1, 1) \times (0, 0, 1)_4\), the \((1, 1, 1)\) means that there is one nonseasonal autoregressive term \((1 - \rho_1 L)\) and one nonseasonal moving-average term \((1 + \theta_1 L)\) and that the time series is first-differenced one time. The \((0, 0, 1)_4\) indicates that there is no lag-4 seasonal autoregressive term, that there is one lag-4 seasonal moving-average term \((1 + \theta_{4,1} L^4)\), and that the series is seasonally differenced zero times. This is known as a multiplicative SARIMA model because the nonseasonal and seasonal factors work multiplicatively: \((1 + \theta_1 L)(1 + \theta_{4,1} L^4)\). Multiplying the terms imposes nonlinear constraints on the parameters of the fifth-order lagged values; 	exttt{arima} imposes these constraints automatically.

To further clarify the notation, consider a \((2, 1, 1) \times (1, 1, 2)_4\) multiplicative SARIMA model:

\[
(1 - \rho_1 L - \rho_2 L^2)(1 - \rho_{4,1} L^4) \Delta \Delta z_t = (1 + \theta_1 L)(1 + \theta_{4,1} L^4 + \theta_{4,2} L^8) \epsilon_t \tag{3}
\]

where \(\Delta\) denotes the difference operator \(\Delta y_t = y_t - y_{t-1}\) and \(\Delta_s\) denotes the lag-s seasonal difference operator \(\Delta_s y_t = y_t - y_{t-s}\). Expanding (3), we have

\[
\tilde{z}_t = \rho_1 \tilde{z}_{t-1} + \rho_2 \tilde{z}_{t-2} + \rho_{4,1} \tilde{z}_{t-4} - \rho_1 \rho_{4,1} \tilde{z}_{t-5} - \rho_2 \rho_{4,1} \tilde{z}_{t-6}
+ \theta_1 \epsilon_{t-1} + \theta_{4,1} \epsilon_{t-4} + \theta_1 \theta_{4,1} \epsilon_{t-5} + \theta_{4,2} \epsilon_{t-8} + \theta_1 \theta_{4,2} \epsilon_{t-9} + \epsilon_t
\]

where

\[
\tilde{z}_t = \Delta \Delta z_t = \Delta(z_t - z_{t-4}) = z_t - z_{t-1} - (z_{t-4} - z_{t-5})
\]

and \(z_t = y_t - \mathbf{x}_t \beta\) if regressors are included in the model, \(z_t = y_t - \beta_0\) if just a constant term is included, and \(z_t = y_t\) otherwise.
More generally, a \( (p, d, q) \times (P, D, Q)_s \) multiplicative SARIMA model is

\[
\rho(L^p)\rho_s(L^P)\Delta^d\Delta^D_s z_t = \theta(L^q)\theta_s(L^Q)\epsilon_t
\]

where

\[
\rho_s(L^P) = (1 - \rho_{s,1}L^s - \rho_{s,2}L^{2s} - \cdots - \rho_{s,P}L^{Ps})
\]

\[
\theta_s(L^Q) = (1 + \theta_{s,1}L^s + \theta_{s,2}L^{2s} + \cdots + \theta_{s,Q}L^{Qs})
\]

\( \rho(L^p) \) and \( \theta(L^q) \) were defined previously, \( \Delta^d \) means apply the \( \Delta \) operator \( d \) times, and similarly for \( \Delta^D_s \). Typically, \( d \) and \( D \) will be 0 or 1; and \( p, q, P, \) and \( Q \) will seldom be more than 2 or 3. \( s \) will typically be 4 for quarterly data and 12 for monthly data. In fact, the model can be extended to include both monthly and quarterly seasonal factors, as we explain below.

If a plot of the data suggests that the seasonal effect is proportional to the mean of the series, then the seasonal effect is probably multiplicative and a multiplicative SARIMA model may be appropriate. Box, Jenkins, and Reinsel (2008, sec. 9.3.1) suggest starting with a multiplicative SARIMA model with any data that exhibit seasonal patterns and then exploring nonmultiplicative SARIMA models if the multiplicative models do not fit the data well. On the other hand, Chatfield (2004, 14) suggests that taking the logarithm of the series will make the seasonal effect additive, in which case an additive SARIMA model as fit in the previous example would be appropriate. In short, the analyst should probably try both additive and multiplicative SARIMA models to see which provides better fits and forecasts.

Unless diffuse is used, arima must create square matrices of dimension \( \{\max(p, q + 1)\}^2 \), where \( p \) and \( q \) are the maximum AR and MA lags, respectively; and the inclusion of long seasonal terms can make this dimension rather large. For example, with monthly data, you might fit a \( (0, 1, 1) \times (0, 1, 2)_{12} \) SARIMA model. The maximum MA lag is \( 2 \times 12 + 1 = 25 \), requiring a matrix with \( 26^2 = 676 \) rows and columns.

\textbf{Example 3: Multiplicative SARIMA model}

One of the most common multiplicative SARIMA specifications is the \( (0, 1, 1) \times (0, 1, 1)_{12} \) “airline” model of Box, Jenkins, and Reinsel (2008, sec. 9.2). The dataset \texttt{airline.dta} contains monthly international airline passenger data from January 1949 through December 1960. After first- and seasonally differencing the data, we do not suspect the presence of a trend component, so we use the noconstant option with arima:
. use https://www.stata-press.com/data/r16/air2
(TIMESLAB: Airline passengers)
. generate lnair = ln(air)
. arima lnair, arima(0,1,1) sarima(0,1,1,12) noconstant
(setting optimization to BHHH)
Iteration 0: log likelihood = 223.8437
Iteration 1: log likelihood = 239.80405
(output omitted)
Iteration 8: log likelihood = 244.69651
ARIMA regression
Sample: 14 - 144 Number of obs = 131
Log likelihood = 244.6965
Wald chi2(2) = 84.53
Prob > chi2 = 0.0000
OPG

| Coef. | Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|-------|-----------|-------|-------|----------------------|
| ARMA  |           |       |       |                      |
| ma L1. | -.4018324 | .0730307 | -5.50 | 0.000 | -.5449698 -.2586949 |
| ARMA12|           |       |       |                      |
| ma L1. | -.5569342 | .0963129 | -5.78 | 0.000 | -.745704 -.3681644 |
| /sigma | .0367167 | .0020132 | 18.24 | 0.000 | .0327708 .0406625 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

Thus our model of the monthly number of international airline passengers is

\[
\Delta \Delta_{12} \ln \text{air}_t = -0.402 \epsilon_{t-1} - 0.557 \epsilon_{t-12} + 0.224 \epsilon_{t-13} + \epsilon_t
\]

\[
\hat{\sigma} = 0.037
\]

In (2), for example, the coefficient on \(\epsilon_{t-13}\) is the product of the coefficients on the \(\epsilon_{t-1}\) and \(\epsilon_{t-12}\) terms \((0.224 \approx -0.402 \times -0.557)\). arima labeled the dependent variable DS12.lnair to indicate that it has applied the difference operator \(\Delta\) and the lag-12 seasonal difference operator \(\Delta_{12}\) to \lnair; see [U] 11.4.4 Time-series varlists for more information.

We could have fit this model by typing

. arima DS12.lnair, ma(1) mma(1, 12) noconstant

For simple multiplicative models, using the sarima() option is easier, though this second syntax allows us to incorporate more complicated seasonal terms.

The mar() and mma() options can be repeated, allowing us to control for multiple seasonal patterns. For example, we may have monthly sales data that exhibit a quarterly pattern as businesses purchase our product at the beginning of calendar quarters when new funds are budgeted, and our product is purchased more frequently in a few months of the year than in most others, even after we control for quarterly fluctuations. Thus we might choose to fit the model

\[
(1 - \rho L)(1 - \rho_{4,1} L^4)(1 - \rho_{12,1} L^{12})(\Delta \Delta_4 \Delta_{12} \text{sales}_t - \beta_0) = (1 + \theta L)(1 + \theta_{4,1} L^4)(1 + \theta_{12,1} L^{12}) \epsilon_t
\]

Although this model looks rather complicated, estimating it using arima is straightforward:

. arima DS4S12.sales, ar(1) mar(1, 4) mar(1, 12) ma(1) mma(1, 4) mma(1, 12)
If we instead wanted to include two lags in the lag-4 seasonal AR term and the first and third (but not the second) term in the lag-12 seasonal MA term, we would type

```
.arima DS4S12.sales, ar(1) mar(1 2, 4) mar(1, 12) ma(1) mma(1, 4) mma(1 3, 12)
```

However, models with multiple seasonal terms can be difficult to fit. Usually, one seasonal factor with just one or two AR or MA terms is adequate.

**ARMAX models**

Thus far all our examples have been pure ARIMA models in which the dependent variable was modeled solely as a function of its past values and disturbances. Also, *arima* can fit ARMAX models, which model the dependent variable in terms of a linear combination of independent variables, as well as an ARMA disturbance process. The *prais* command (see [TS] *prais*), for example, allows you to control for only AR(1) disturbances, whereas *arima* allows you to control for a much richer dynamic error structure. *arima* allows for both nonseasonal and seasonal ARMA components in the disturbances.

>_Example 4: ARMAX model_

For a simple example of a model including covariates, we can estimate an update of Friedman and Meiselman’s (1963) equation representing the quantity theory of money. They postulate a straightforward relationship between personal-consumption expenditures (consp) and the money supply as measured by M2 (m2).

\[
\text{consp}_t = \beta_0 + \beta_1 \text{m2}_t + \mu_t
\]

Friedman and Meiselman fit the model over a period ending in 1956; we will refit the model over the period 1959q1 through 1981q4. We restrict our attention to the period prior to 1982 because the Federal Reserve manipulated the money supply extensively in the later 1980s to control inflation, and the relationship between consumption and the money supply becomes much more complex during the later part of the decade.

To demonstrate *arima*, we will include both an autoregressive term and a moving-average term for the disturbances in the model; the original estimates included neither. Thus we model the disturbance of the structural equation as

\[
\mu_t = \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t
\]

As per the original authors, the relationship is estimated on seasonally adjusted data, so there is no need to include seasonal effects explicitly. Obtaining seasonally unadjusted data and simultaneously modeling the structural and seasonal effects might be preferable.

We will restrict the estimation to the desired sample by using the *tin()* function in an *if* expression; see [FN] *Selecting time-span functions*. By leaving the first argument of *tin()* blank, we are including all available data through the second date (1981q4). We fit the model by typing
use https://www.stata-press.com/data/r16/friedman2, clear
. arima consump m2 if tin(, 1981q4), ar(1) ma(1)
(setting optimization to BHHH)
Iteration 0: log likelihood = -344.67575
Iteration 1: log likelihood = -341.57248
(output omitted)
Iteration 10: log likelihood = -340.50774
ARIMA regression
Sample: 1959q1 - 1981q4  Number of obs = 92
Wald chi2(3) = 4394.80
Log likelihood = -340.5077  Prob > chi2 = 0.0000

| Coef. Std. Err.  z    P>|z|  [95% Conf. Interval] |
|---------------------|---------------------|-------|-------|---------------------|
| consump             |                     |       |       |                     |
| consump             | 1.122029            | .0363563 | 30.86 | 0.000              | 1.050772 1.193286 |
| m2                  | -36.09872           | 56.56703 | -0.64 | 0.523              | -146.9681 74.77062 |
| _cons               | -36.09872           | 56.56703 | -0.64 | 0.523              | -146.9681 74.77062 |
| ARMA                |                     |       |       |                     |
| ar L1.              | .9348486            | .0411323 | 22.73 | 0.000              | .8542308 1.015467 |
| ma L1.              | .3090592            | .0885883 | 3.49  | 0.000              | .1354293 .4826891 |
| /sigma              | 9.655308            | .5635157 | 17.13 | 0.000              | 8.550837 10.75978 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

We find a relatively small money velocity with respect to consumption (1.122) over this period, although consumption is only one facet of the income velocity. We also note a very large first-order autocorrelation in the disturbances, as well as a statistically significant first-order moving average.
We might be concerned that our specification has led to disturbances that are heteroskedastic or non-Gaussian. We refit the model by using the `vce(robust)` option.

```stata
. arima consump m2 if tin(, 1981q4), ar(1) ma(1) vce(robust)

(setting optimization to BHHH)
Iteration 0: log pseudolikelihood = -344.67575
Iteration 1: log pseudolikelihood = -341.57248
(output omitted)
Iteration 10: log pseudolikelihood = -340.50774

ARIMA regression
Sample: 1959q1 - 1981q4  Number of obs = 92
Wald chi2(3) = 1176.26
Log pseudolikelihood = -340.5077  Prob > chi2 = 0.0000

                      Semirobust                  [95% Conf. Interval]
-------------  ---------------  --------  --------  --------------------
consump         Coef.  Std. Err.     z    P>|z|       [95% Conf. Interval]
               m2           1.122029    .0433302  25.89  0.000  1.037103  1.206954
               _cons        -36.09872    28.10478  -1.28  0.199  -91.18308  18.98564
ARMA
               ar L1.          .9348486    .0493428  18.95  0.000   .8381385   1.031559
               ma L1.          .3090592    .1605359   1.93  0.054   -.0055854   .6237038
/sigma         9.655308    1.082639    8.92  0.000    7.533375  11.77724

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

We note a substantial increase in the estimated standard errors, and our once clearly significant moving-average term is now only marginally significant.

Dynamic forecasting

Another feature of the `arima` command is the ability to use `predict` afterward to make dynamic forecasts. Suppose that we wish to fit the regression model

\[ y_t = \beta_0 + \beta_1 x_t + \rho y_{t-1} + \epsilon_t \]

by using a sample of data from \( t = 1 \ldots T \) and make forecasts beginning at time \( f \).

If we use `regress` or `prais` to fit the model, then we can use `predict` to make one-step-ahead forecasts. That is, `predict` will compute

\[ \hat{y}_f = \hat{\beta}_0 + \hat{\beta}_1 x_f + \hat{\rho} y_{f-1} \]

Most importantly, here `predict` will use the actual value of \( y \) at period \( f - 1 \) in computing the forecast for time \( f \). Thus, if we use `regress` or `prais`, we cannot make forecasts for any periods beyond \( f = T + 1 \) unless we have observed values for \( y \) for those periods.
If we instead fit our model with \texttt{arima}, then \texttt{predict} can produce dynamic forecasts by using the Kalman filter. If we use the \texttt{dynamic}\((f)\) option, then for period \(f\) \texttt{predict} will compute

\[
\hat{y}_f = \hat{\beta}_0 + \hat{\beta}_1 x_f + \hat{\rho} y_{f-1}
\]

by using the observed value of \(y_{f-1}\) just as \texttt{predict} after \texttt{regress} or \texttt{prais}. However, for period \(f + 1\) \texttt{predict newvar}, \texttt{dynamic}\((f)\) will compute

\[
\hat{y}_{f+1} = \hat{\beta}_0 + \hat{\beta}_1 x_{f+1} + \hat{\rho} \hat{y}_f
\]

using the \textit{predicted} value of \(y_f\) instead of the observed value. Similarly, the period \(f + 2\) forecast will be

\[
\hat{y}_{f+2} = \hat{\beta}_0 + \hat{\beta}_1 x_{f+2} + \hat{\rho} \hat{y}_{f+1}
\]

Of course, because our model includes the regressor \(x_t\), we can make forecasts only through periods for which we have observations on \(x_t\). However, for pure ARIMA models, we can compute dynamic forecasts as far beyond the final period of our dataset as desired.

For more information on \texttt{predict} after \texttt{arima}, see \cite{arima} \texttt{arima postestimation}.

\section*{Video example}

\textbf{Introduction to ARMA/ARIMA models}

\section*{Stored results}

\texttt{arima} stores the following in \texttt{e()}:

\begin{description}
\item Scalar
  \begin{description}
  \item \texttt{e(N)} \quad number of observations
  \item \texttt{e(N_gaps)} \quad number of gaps
  \item \texttt{e(k)} \quad number of parameters
  \item \texttt{e(k_eq)} \quad number of equations in \texttt{e(b)}
  \item \texttt{e(k_eq_model)} \quad number of equations in overall model test
  \item \texttt{e(k_dv)} \quad number of dependent variables
  \item \texttt{e(k1)} \quad number of variables in first equation
  \item \texttt{e(df_m)} \quad model degrees of freedom
  \item \texttt{e(ll)} \quad log likelihood
  \item \texttt{e(sigma)} \quad sigma
  \item \texttt{e(chi2)} \quad $\chi^2$
  \item \texttt{e(p)} \quad \textit{p}-value for model test
  \item \texttt{e(tmin)} \quad minimum time
  \item \texttt{e(tmax)} \quad maximum time
  \item \texttt{e(ar_max)} \quad maximum AR lag
  \item \texttt{e(ma_max)} \quad maximum MA lag
  \item \texttt{e(rank)} \quad rank of \texttt{e(V)}
  \item \texttt{e(ic)} \quad number of iterations
  \item \texttt{e(rc)} \quad return code
  \item \texttt{e(converged)} \quad 1 if converged, 0 otherwise
  \end{description}
\end{description}
Methods and formulas

Estimation is by maximum likelihood using the Kalman filter via the prediction error decomposition; see Hamilton (1994), Gourieroux and Monfort (1997), or, in particular, Harvey (1989). Any of these sources will serve as excellent background for the fitting of these models with the state-space form; each source also provides considerable detail on the method outlined below.

Methods and formulas are presented under the following headings:

ARIMA model
Kalman filter equations
Kalman filter or state-space representation of the ARIMA model
Kalman filter recursions
Kalman filter initial conditions
Likelihood from prediction error decomposition
Missing data
ARIMA model

The model to be fit is

\[ y_t = x_t \beta + \mu_t \]

\[ \mu_t = \sum_{i=1}^{p} \rho_i \mu_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t \]

which can be written as the single equation

\[ y_t = x_t \beta + \sum_{i=1}^{p} \rho_i (y_{t-i} - x_{t-i} \beta) + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t \]

Some of the \( \rho \)s and \( \theta \)s may be constrained to zero or, for multiplicative seasonal models, the products of other parameters.

Kalman filter equations

We will roughly follow Hamilton’s (1994) notation and write the Kalman filter

\[ \xi_t = F \xi_{t-1} + v_t \quad \text{(state equation)} \]

\[ y_t = A' x_t + H' \xi_t + w_t \quad \text{(observation equation)} \]

and

\[ \begin{pmatrix} v_t \\ w_t \end{pmatrix} \sim N \left\{ 0, \begin{pmatrix} Q & 0 \\ 0 & R \end{pmatrix} \right\} \]

We maintain the standard Kalman filter matrix and vector notation, although for univariate models \( y_t, w_t, \) and \( R \) are scalars.

Kalman filter or state-space representation of the ARIMA model

A univariate ARIMA model can be cast in state-space form by defining the Kalman filter matrices as follows (see Hamilton [1994], or Gourieroux and Monfort [1997], for details):

\[ F = \begin{bmatrix} \rho_1 & \rho_2 & \ldots & \rho_{p-1} & \rho_p \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix} \]

\[ v_t = \begin{bmatrix} \epsilon_{t-1} \\ 0 \\ \ldots \\ \ldots \\ 0 \end{bmatrix} \]

\[ A' = \beta \]

\[ H' = [1 \ \theta_1 \ \theta_2 \ \ldots \ \theta_q] \]

\[ w_t = 0 \]

The Kalman filter representation does not require the moving-average terms to be invertible.
Kalman filter recursions

To demonstrate how missing data are handled, the updating recursions for the Kalman filter will be left in two steps. Writing the updating equations as one step using the gain matrix $K$ is common. We will provide the updating equations with little justification; see the sources listed above for details.

As a linear combination of a vector of random variables, the state $\xi_t$ can be updated to its expected value on the basis of the prior state as

$$\xi_{t|t-1} = F\xi_{t-1} + v_{t-1}$$  \hspace{1cm} (4)

This state is a quadratic form that has the covariance matrix

$$P_{t|t-1} = FP_{t-1}F' + Q$$  \hspace{1cm} (5)

The estimator of $y_t$ is

$$\hat{y}_{t|t-1} = x_t\beta + H'\xi_{t|t-1}$$

which implies an innovation or prediction error

$$\hat{\epsilon}_t = y_t - \hat{y}_{t|t-1}$$

This value or vector has mean squared error (MSE)

$$M_t = H'P_{t|t-1}H + R$$

Now the expected value of $\xi_t$ conditional on a realization of $y_t$ is

$$\xi_t = \xi_{t|t-1} + P_{t|t-1}HM_{t-1}^{-1}\hat{\epsilon}_t$$  \hspace{1cm} (6)

with MSE

$$P_t = P_{t|t-1} - P_{t|t-1}HM_{t-1}^{-1}H'P_{t|t-1}$$  \hspace{1cm} (7)

This expression gives the full set of Kalman filter recursions.

Kalman filter initial conditions

When the series is stationary, conditional on $x_t\beta$, the initial conditions for the filter can be considered a random draw from the stationary distribution of the state equation. The initial values of the state and the state MSE are the expected values from this stationary distribution. For an ARIMA model, these can be written as

$$\xi_{1|0} = 0$$

and

$$\text{vec}(P_{1|0}) = (I_{r^2} - F \otimes F)^{-1}\text{vec}(Q)$$

where vec() is an operator representing the column matrix resulting from stacking each successive column of the target matrix.

If the series is not stationary, the initial state conditions do not constitute a random draw from a stationary distribution, and some other values must be chosen. Hamilton (1994) suggests that they be chosen based on prior expectations, whereas Harvey suggests a diffuse and improper prior having a state vector of 0 and an infinite variance. This method corresponds to $P_{1|0}$ with diagonal elements of $\infty$. Stata allows either approach to be taken for nonstationary series—initial priors may be specified with `state0()` and `p0()`, and a diffuse prior may be specified with `diffuse`. 
Likelihood from prediction error decomposition

Given the outputs from the Kalman filter recursions and assuming that the state and observation vectors are Gaussian, the likelihood for the state-space model follows directly from the resulting multivariate normal in the predicted innovations. The log likelihood for observation $t$ is

$$\ln L_t = -\frac{1}{2} \left\{ \ln(2\pi) + \ln(|M_t|) - \hat{i}_t'M_t^{-1}\hat{i}_t \right\}$$

This command supports the Huber/White/sandwich estimator of the variance using vce(robust). See [P] _robust, particularly Maximum likelihood estimators and Methods and formulas.

Missing data

Missing data, whether a missing dependent variable $y_t$, one or more missing covariates $x_t$, or completely missing observations, are handled by continuing the state-updating equations without any contribution from the data; see Harvey (1989 and 1993). That is, (4) and (5) are iterated for every missing observation, whereas (6) and (7) are ignored. Thus, for observations with missing data, $\xi_t = \xi_{t|t-1}$ and $P_t = P_{t|t-1}$. Without any information from the sample, this effectively assumes that the prediction error for the missing observations is 0. Other methods of handling missing data on the basis of the EM algorithm have been suggested, for example, Shumway (1984, 1988).

George Edward Pelham Box (1919–2013) was born in Kent, England, and earned degrees in statistics at the University of London. After work in the chemical industry, he taught and researched at Princeton and the University of Wisconsin. His many major contributions to statistics include papers and books in Bayesian inference, robustness (a term he introduced to statistics), modeling strategy, experimental design and response surfaces, time-series analysis, distribution theory, transformations, and nonlinear estimation.

Gwilym Meirion Jenkins (1933–1982) was a British mathematician and statistician who spent his career in industry and academia, working for extended periods at Imperial College London and the University of Lancaster before running his own company. His interests were centered on time series and he collaborated with G. E. P. Box on what are often called Box–Jenkins models. The last years of Jenkins’s life were marked by a slowly losing battle against Hodgkin’s disease.

References


Becketti, S. 2013. Introduction to Time Series Using Stata. College Station, TX: Stata Press.


Also see

[TS] arima postestimation — Postestimation tools for arima
[TS] arch — Autoregressive conditional heteroskedasticity (ARCH) family of estimators
[TS] dfactor — Dynamic-factor models
[TS] forecast — Econometric model forecasting
[TS] mgarch — Multivariate GARCH models
[TS] mswitch — Markov-switching regression models
[TS] prais — Prais–Winsten and Cochrane–Orcutt regression
[TS] sspace — State-space models
[TS] threshold — Threshold regression
[TS] tsset — Declare data to be time-series data
[TS] ucm — Unobserved-components model
[R] regress — Linear regression
[U] 20 Estimation and postestimation commands
# arima postestimation — Postestimation tools for arima

## Postestimation commands

The following postestimation commands are of special interest after arima:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estat acplot</td>
<td>estimate autocorrelations and autocovariances</td>
</tr>
<tr>
<td>estat aroots</td>
<td>check stability condition of estimates</td>
</tr>
<tr>
<td>irf</td>
<td>create and analyze IRFs</td>
</tr>
<tr>
<td>psdensity</td>
<td>estimate the spectral density</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estat ic</td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td>estat summarize</td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td>estat vce</td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td>estimates</td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td>forecast</td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td>lincom</td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td>lrtest</td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td>margins</td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td>marginsplot</td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td>nlcom</td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td>predict</td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td>predictnl</td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td>test</td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td>testnl</td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
**predict**

**Description for predict**

`predict` creates a new variable containing predictions such as expected values and mean squared errors. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

**Menu for predict**

Statistics > Postestimation

**Syntax for predict**

```
predict [type] newvar [if] [in] [, statistic options]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xb</code></td>
<td>predicted values for mean equation—the differenced series; the default</td>
</tr>
<tr>
<td><code>stdp</code></td>
<td>standard error of the linear prediction</td>
</tr>
<tr>
<td><code>y</code></td>
<td>predicted values for the mean equation in <code>y</code>—the undifferenced series</td>
</tr>
<tr>
<td><code>mse</code></td>
<td>mean squared error of the predicted values</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>residuals or predicted innovations</td>
</tr>
<tr>
<td><code>yresiduals</code></td>
<td>residuals or predicted innovations in <code>y</code>, reversing any time-series operators</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Predictions are not available for conditional ARIMA models fit to panel data.

**Options for predict**

Six statistics can be computed using `predict` after `arima`: the predictions from the model (the default also given by `xb`), the standard error of the linear prediction (`stdp`), the predictions after reversing any time-series operators applied to the dependent variable (`y`), the MSE of `xb` (`mse`), the predictions of residuals or innovations (`residuals`), and the predicted residuals or innovations in terms of `y` (`yresiduals`). Given the dynamic nature of the ARMA component and because the dependent variable might be differenced, there are other ways of computing each. We can use all the data on the dependent variable that is available right up to the time of each prediction (the default, which is...
often called a one-step prediction), or we can use the data up to a particular time, after which the predicted value of the dependent variable is used recursively to make later predictions (dynamic()). Either way, we can consider or ignore the ARMA disturbance component (the component is considered by default and is ignored if you specify structural).

All calculations can be made in or out of sample.

**Main**

- **xb**, the default, calculates the predictions from the model. If D.\_depvar is the dependent variable, these predictions are of D.\_depvar and not of depvar itself.

- **stdp** calculates the standard error of the linear prediction xb. stdp does not include the variation arising from the disturbance equation; use mse to calculate standard errors and confidence bands around the predicted values.

- **y** specifies that predictions of depvar be made, even if the model was specified in terms of, say, D.\_depvar.

- **mse** calculates the MSE of the predictions.

- **residuals** calculates the residuals. If no other options are specified, these are the predicted innovations $\epsilon_t$; that is, they include the ARMA component. If structural is specified, these are the residuals $\mu_t$ from the structural equation; see structural below.

- **yresiduals** calculates the residuals in terms of depvar, even if the model was specified in terms of, say, D.\_depvar. As with residuals, the yresiduals are computed from the model, including any ARMA component. If structural is specified, any ARMA component is ignored, and yresiduals are the residuals from the structural equation; see structural below.

**Options**

- **dynamic(time_constant)** specifies how lags of $y_t$ in the model are to be handled. If dynamic() is not specified, actual values are used everywhere that lagged values of $y_t$ appear in the model to produce one-step-ahead forecasts.

- **dynamic(time_constant)** produces dynamic (also known as recursive) forecasts. time_constant specifies when the forecast is to switch from one step ahead to dynamic. In dynamic forecasts, references to $y_t$ evaluate to the prediction of $y_t$ for all periods at or after time_constant; they evaluate to the actual value of $y_t$ for all prior periods.

  For example, dynamic(10) would calculate predictions in which any reference to $y_t$ with $t < 10$ evaluates to the actual value of $y_t$ and any reference to $y_t$ with $t \geq 10$ evaluates to the prediction of $y_t$. This means that one-step-ahead predictions are calculated for $t < 10$ and dynamic predictions thereafter. Depending on the lag structure of the model, the dynamic predictions might still refer some actual values of $y_t$.

  You may also specify dynamic(.) to have predict automatically switch from one-step-ahead to dynamic predictions at $p + q$, where $p$ is the maximum AR lag and $q$ is the maximum MA lag.

- **t0(time_constant)** specifies the starting point for the recursions to compute the predicted statistics; disturbances are assumed to be 0 for $t < t0()$. The default is to set t0() to the minimum $t$ observed in the estimation sample, meaning that observations before that are assumed to have disturbances of 0.

  t0() is irrelevant if structural is specified because then all observations are assumed to have disturbances of 0.

  t0(5) would begin recursions at $t = 5$. If the data were quarterly, you might instead type t0(tq(1961q2)) to obtain the same result.
The ARMA component of ARIMA models is recursive and depends on the starting point of the predictions. This includes one-step-ahead predictions.

`structural` specifies that the calculation be made considering the structural component only, ignoring the ARMA terms, producing the steady-state equilibrium predictions.

**margins**

**Description for margins**

`margins` estimates margins of response for expected values.

**Menu for margins**

Statistics > Postestimation

**Syntax for margins**

```
margins [marginlist] [ , options ]
margins [marginlist], predict(statistic ...) [predict(statistic ...) ...] [options]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>predicted values for mean equation—the differenced series; the default</td>
</tr>
<tr>
<td>y</td>
<td>predicted values for the mean equation in (y)—the undifferenced series</td>
</tr>
<tr>
<td>stdp</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>mse</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>yresiduals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

Statistics not allowed with `margins` are functions of stochastic quantities other than \(e(b)\).

For the full syntax, see [R] `margins`.

**Remarks and examples**

Remarks are presented under the following headings:

*Forecasting after ARIMA*

*IRF results for ARIMA*

**Forecasting after ARIMA**

We assume that you have already read [TS] `arima`. In this section, we illustrate some of the features of `predict` after fitting ARIMA, ARMAX, and other dynamic models by using `arima`. In example 2 of [TS] `arima`, we fit the model

\[
\Delta \ln(wpi_t) = \beta_0 + \rho_1 \{\Delta \ln(wpi_{t-1}) - \beta_0\} + \theta_4 \epsilon_{t-4} + \epsilon_t
\]
by typing

```
  . use https://www.stata-press.com/data/r16/wpi1
  . arima D.ln_wpi, ar(1) ma(1 4)
```

(output omitted)

If we use the command

```
  . predict xb, xb
```

then Stata computes \( \hat{x}_t \) as

\[
\hat{x}_t = \hat{\beta}_0 + \hat{\rho}_1 \{ \Delta \ln(wpi_{t-1}) - \hat{\beta}_0 \} + \hat{\theta}_1 \hat{\epsilon}_{t-1} + \hat{\theta}_4 \hat{\epsilon}_{t-4}
\]

where

\[
\hat{\epsilon}_{t-j} = \begin{cases} 
\Delta \ln(wpi_{t-j}) - \hat{x}_{t-j} & t-j > 0 \\
0 & \text{otherwise}
\end{cases}
\]

meaning that `predict newvar, xb` calculates predictions by using the metric of the dependent variable. In this example, the dependent variable represented changes in \( \ln(wpi_t) \), and so the predictions are likewise for changes in that variable.

If we instead use

```
  . predict y, y
```

Stata computes \( y_t \) as \( y_t = \hat{x}_t + \ln(wpi_{t-1}) \) so that \( y_t \) represents the predicted levels of \( \ln(wpi_t) \). In general, `predict newvar, y` will reverse any time-series operators applied to the dependent variable during estimation.

If we want to ignore the ARMA error components when making predictions, we use the `structural` option,

```
  . predict xbs, xb structural
```

which generates \( \hat{x}_{bst} = \hat{\beta}_0 \) because there are no regressors in this model, and

```
  . predict ys, y structural
```

generates \( y_{st} = \hat{\beta}_0 + \ln(wpi_{t-1}) \)

---

### Example 1: Dynamic forecasts

An attractive feature of the `arima` command is the ability to make dynamic forecasts. In example 4 of [TS] arima, we fit the model

\[
\text{consump}_t = \beta_0 + \beta_1 m_2 + \mu_t \\
\mu_t = \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t
\]

First, we refit the model by using data up through the first quarter of 1978, and then we will evaluate the one-step-ahead and dynamic forecasts.

```
  . use https://www.stata-press.com/data/r16/friedman2
  . keep if time<=tq(1981q4)
     (67 observations deleted)
  . arima consump m2 if tin(, 1978q1), ar(1) ma(1)
     (output omitted)
```
To make one-step-ahead forecasts, we type

```
predict chat, y
(52 missing values generated)
```

(Because our dependent variable contained no time-series operators, we could have instead used `predict chat, xb` and accomplished the same thing.) We will also make dynamic forecasts, switching from observed values of `consump` to forecasted values at the first quarter of 1978:

```
predict chatdy, dynamic(tq(1978q1)) y
(52 missing values generated)
```

The following graph compares the forecasted values to the observed values for the first few years following the estimation sample:

![Graph comparing observed and forecasted values of personal consumption](image)

The one-step-ahead forecasts never deviate far from the observed values, though over time the dynamic forecasts have larger errors. To understand why that is the case, rewrite the model as

\[ \text{consump}_t = \beta_0 + \beta_1 \text{m}_2t + \rho \epsilon_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \]

\[ = \beta_0 + \beta_1 \text{m}_2t + \rho (\text{consump}_{t-1} - \beta_0 - \beta_1 \text{m}_2t_{-1}) + \theta \epsilon_{t-1} + \epsilon_t \]

This form shows that the forecasted value of consumption at time \( t \) depends on the value of consumption at time \( t - 1 \). When making the one-step-ahead forecast for period \( t \), we know the actual value of consumption at time \( t - 1 \). On the other hand, with the `dynamic(tq(1978q1))` option, the forecasted value of consumption for period 1978q1 is based on the observed value of consumption in period 1977q4, but the forecast for 1978q2 is based on the forecast value for 1978q1, the forecast for 1978q3 is based on the forecast value for 1978q2, and so on. Thus, with dynamic forecasts, prior forecast errors accumulate over time. The following graph illustrates this effect.
IRF results for ARIMA

We assume that you have already read [TS] irf and [TS] irf create. In this section, we illustrate how to calculate the impulse–response function (IRF) of an ARIMA model.

Example 2

Consider a model of the quarterly U.S. money supply, as measured by M1, from Enders (2004). Enders (2004, 93–97) discusses why seasonal shopping patterns cause seasonal effects in M1. The variable \( \ln m1 \) contains data on the natural log of the money supply. We fit seasonal and nonseasonal ARIMA models and compare the IRFs calculated from both models.

We fit the following nonseasonal ARIMA model

\[
\Delta \Delta_4 \ln m1_t = \rho_1 (\Delta \Delta_4 \ln m1_{t-1}) + \rho_4 (\Delta \Delta_4 \ln m1_{t-4}) + \epsilon_t
\]

The code below fits the above model and saves a set of IRF results to a file called myirf.irf.
. use https://www.stata-press.com/data/r16/m1nsa, clear
(U.S. money supply (M1) from Enders (2004), 95–99.)
. arima DS4.lnm1, ar(1 4) noconstant nolog

**ARIMA** regression
Sample: 1961q2 - 2008q2 Number of obs = 189
Wald chi2(2) = 78.34
Log likelihood = 579.3036 Prob > chi2 = 0.0000

| Coef. | Std. Err. | z | P>|z| | [95% Conf. Interval] |
|-------|-----------|---|-----|----------------------|
| **ARMA** | | | | |
| L1. | .3551862 | .0503011 | 7.06 | 0.000 | .2565979 | .4537745 |
| L4. | -.3275808 | .0594953 | -5.51 | 0.000 | -.4441895 | -.210972 |
| /sigma | .0112678 | .0004882 | 23.08 | 0.000 | .0103109 | .0122246 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

. irf create nonseasonal, set(myirf) step(30)
(file myirf.irf created)
(file myirf.irf now active)
(file myirf.irf updated)

We fit the following seasonal ARIMA model

\[(1 - \rho_1 L)(1 - \rho_{4,1} L^4)\Delta \Delta lnm1_t = \epsilon_t\]

The code below fits this nonseasonal ARIMA model and saves a set of IRF results to the active IRF file, which is myirf.irf.

. arima DS4.lnm1, ar(1) mar(1,4) noconstant nolog

**ARIMA** regression
Sample: 1961q2 - 2008q2 Number of obs = 189
Wald chi2(2) = 119.78
Log likelihood = 588.6689 Prob > chi2 = 0.0000

| Coef. | Std. Err. | z | P>|z| | [95% Conf. Interval] |
|-------|-----------|---|-----|----------------------|
| **ARMA** | | | | |
| L1. | .489277 | .0538033 | 9.09 | 0.000 | .3838245 | .5947296 |
| L4. | -.4688653 | .0601248 | -7.80 | 0.000 | -.5867076 | -.3510229 |
| /sigma | .0107075 | .0004747 | 22.56 | 0.000 | .0097771 | .0116379 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

. irf create seasonal, step(30)
(file myirf.irf updated)

We now have two sets of IRF results in the file myirf.irf. We can graph both IRF functions side by side by calling irf graph.
The trajectories of the IRF functions are similar: each figure shows that a shock to lnm1 causes a temporary oscillation in lnm1 that dies out after about 15 time periods. This behavior is characteristic of short-memory processes.

See [TS] psdensity for an introduction to estimating spectral densities using the parameters estimated by arima.

Reference


Also see

[Ts] arima — ARIMA, ARMAX, and other dynamic regression models
[Ts] estat acplot — Plot parametric autocorrelation and autocovariance functions
[Ts] estat aroots — Check the stability condition of ARIMA estimates
[Ts] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[Ts] psdensity — Parametric spectral density estimation after arima, arfima, and ucm
[Us] 20 Estimation and postestimation commands
**Description**

corrgram produces a table of the autocorrelations, partial autocorrelations, and portmanteau ($Q$) statistics. It also displays a character-based plot of the autocorrelations and partial autocorrelations. See [TS] wntestq for more information on the $Q$ statistic.

ac produces a correlogram (a graph of autocorrelations) with pointwise confidence intervals that is based on Bartlett’s formula for MA(q) processes.

pac produces a partial correlogram (a graph of partial autocorrelations) with confidence intervals calculated using a standard error of $1/\sqrt{n}$. The residual variances for each lag may optionally be included on the graph.

**Quick start**

Produce correlogram for $y$ using `tsset` data

corrgram $y$

As above, but limit the number of computed autocorrelations to 10

corrgram $y$, lags(10)

Plot the autocorrelation function for $y$

ac $y$

As above, and generate `newv` to hold the autocorrelations

ac $y$, generate(newv)

Plot partial autocorrelation function for $y$, and include standardized residual variances in the graph

pac $y$, srv

**Menu**

corrgram

Statistics > Time series > Graphs > Autocorrelations & partial autocorrelations

ac

Statistics > Time series > Graphs > Correlogram (ac)

pac

Statistics > Time series > Graphs > Partial correlogram (pac)
## Syntax

Autocorrelations, partial autocorrelations, and portmanteau \((Q)\) statistics

```
corrgram varname [if] [in] [, corrgram_options]
```

Graph autocorrelations with confidence intervals

```
ac varname [if] [in] [, ac_options]
```

Graph partial autocorrelations with confidence intervals

```
pac varname [if] [in] [, pac_options]
```

### `corrgram_options` Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lags(#)</code></td>
<td>calculate # autocorrelations</td>
</tr>
<tr>
<td><code>noplot</code></td>
<td>suppress character-based plots</td>
</tr>
<tr>
<td><code>yw</code></td>
<td>calculate partial autocorrelations by using Yule–Walker equations</td>
</tr>
</tbody>
</table>

### `ac_options` Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lags(#)</code></td>
<td>calculate # autocorrelations</td>
</tr>
<tr>
<td><code>generate(newvar)</code></td>
<td>generate a variable to hold the autocorrelations</td>
</tr>
<tr>
<td><code>level(#)</code></td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td><code>fft</code></td>
<td>calculate autocorrelation by using Fourier transforms</td>
</tr>
</tbody>
</table>

### Plot

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>line_options</code></td>
<td>change look of dropped lines</td>
</tr>
<tr>
<td><code>marker_options</code></td>
<td>change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td><code>marker_label_options</code></td>
<td>add marker labels; change look or position</td>
</tr>
</tbody>
</table>

### CI plot

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ciopts(area_options)</code></td>
<td>affect rendition of the confidence bands</td>
</tr>
</tbody>
</table>

### Add plots

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addplot(plot)</code></td>
<td>add other plots to the generated graph</td>
</tr>
</tbody>
</table>

### Y axis, X axis, Titles, Legend, Overall

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>twoway_options</code></td>
<td>any options other than by() documented in [G-3] <code>twoway_options</code></td>
</tr>
</tbody>
</table>
### corrgram — Tabulate and graph autocorrelations

#### pac_options

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
</tr>
<tr>
<td><strong>lags(#)</strong> calculate # partial autocorrelations</td>
</tr>
<tr>
<td><strong>generate(newvar)</strong> generate a variable to hold the partial autocorrelations</td>
</tr>
<tr>
<td><strong>yw</strong> calculate partial autocorrelations by using Yule–Walker equations</td>
</tr>
<tr>
<td><strong>level(#)</strong> set confidence level; default is level(95)</td>
</tr>
<tr>
<td><strong>Plot</strong></td>
</tr>
<tr>
<td><strong>line_options</strong> change look of dropped lines</td>
</tr>
<tr>
<td><strong>marker_options</strong> change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td><strong>marker_label_options</strong> add marker labels; change look or position</td>
</tr>
<tr>
<td><strong>CI plot</strong></td>
</tr>
<tr>
<td><strong>ciopts(area_options)</strong> affect rendition of the confidence bands</td>
</tr>
<tr>
<td><strong>SRV plot</strong></td>
</tr>
<tr>
<td><strong>srv</strong> include standardized residual variances in graph</td>
</tr>
<tr>
<td><strong>srvopts(marker_options)</strong> affect rendition of the plotted standardized residual variances (SRVs)</td>
</tr>
<tr>
<td><strong>Add plots</strong></td>
</tr>
<tr>
<td><strong>addplot(plot)</strong> add other plots to the generated graph</td>
</tr>
<tr>
<td><strong>twoway_options</strong> any options other than by() documented in [G-3] twoway_options</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `corrgram`, `ac`, or `pac`; see [TS] `tsset`. Also, the time series must be dense (nonmissing and no gaps in the time variable) in the sample if you specify the `fft` option. `varname` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

### Options for corrgram

**lags(#)** specifies the number of autocorrelations to calculate. The default is to use \( \min(\left\lfloor n/2 \right\rfloor - 2, 40) \), where \( \left\lfloor n/2 \right\rfloor \) is the greatest integer less than or equal to \( n/2 \).

**noplot** prevents the character-based plots from being in the listed table of autocorrelations and partial autocorrelations.

**yw** specifies that the partial autocorrelations be calculated using the Yule–Walker equations instead of using the default regression-based technique. `yw` cannot be used if `srv` is used.

### Options for ac and pac

**lags(#)** specifies the number of autocorrelations to calculate. The default is to use \( \min(\left\lfloor n/2 \right\rfloor - 2, 40) \), where \( \left\lfloor n/2 \right\rfloor \) is the greatest integer less than or equal to \( n/2 \).

**generate(newvar)** specifies a new variable to contain the autocorrelation (`ac` command) or partial autocorrelation (`pac` command) values. This option is required if the `nograph` option is used.
nograph (implied when using \texttt{generate()} in the dialog box) prevents \texttt{ac} and \texttt{pac} from constructing a graph. This option requires the \texttt{generate()} option.

\texttt{yw} (\texttt{pac} only) specifies that the partial autocorrelations be calculated using the Yule–Walker equations instead of using the default regression-based technique. \texttt{yw} cannot be used if \texttt{srv} is used.

\texttt{level(\#)} specifies the confidence level, as a percentage, for the confidence bands in the \texttt{ac} or \texttt{pac} graph. The default is \texttt{level(95)} or as set by \texttt{set level}; see [R] \texttt{level}.

\texttt{fft} (\texttt{ac} only) specifies that the autocorrelations be calculated using two Fourier transforms. This technique can be faster than simply iterating over the requested number of lags.

\texttt{line\_options}, \texttt{marker\_options}, and \texttt{marker\_label\_options} affect the rendition of the plotted autocorrelations (with \texttt{ac}) or partial autocorrelations (with \texttt{pac}).

\texttt{line\_options} specify the look of the dropped lines, including pattern, width, and color; see [G-3] \texttt{line\_options}.

\texttt{marker\_options} specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] \texttt{marker\_options}.

\texttt{marker\_label\_options} specify if and how the markers are to be labeled; see [G-3] \texttt{marker\_label\_options}.

\texttt{ci\_opts(area\_options)} affects the rendition of the confidence bands; see [G-3] \texttt{area\_options}.

\texttt{srv} (\texttt{pac} only) specifies that the standardized residual variances be plotted with the partial autocorrelations. \texttt{srv} cannot be used if \texttt{yw} is used.

\texttt{srv\_opts(marker\_options)} (\texttt{pac} only) affects the rendition of the plotted standardized residual variances; see [G-3] \texttt{marker\_options}. This option implies the \texttt{srv} option.

\texttt{addplot(plot)} adds specified plots to the generated graph; see [G-3] \texttt{addplot\_option}.

\texttt{twoway\_options} are any of the options documented in [G-3] \texttt{twoway\_options}, excluding \texttt{by()}. These include options for titling the graph (see [G-3] \texttt{title\_options}) and for saving the graph to disk (see [G-3] \texttt{saving\_option}).

\section*{Remarks and examples}

Remarks are presented under the following headings:

\begin{itemize}
  \item \texttt{Basic examples}
  \item \texttt{Video example}
\end{itemize}
### Basic examples

`corrgram` tabulates autocorrelations, partial autocorrelations, and portmanteau (Q) statistics and plots the autocorrelations and partial autocorrelations. The Q statistics are the same as those produced by [TS] `wntestq`. `ac` produces graphs of the autocorrelations, and `pac` produces graphs of the partial autocorrelations. See Becketti (2013) for additional examples of how these commands are used in practice.

#### Example 1

Here we use the international airline passengers dataset (Box, Jenkins, and Reinsel 2008, Series G). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. We can list the autocorrelations and partial autocorrelations by using `corrgram`.

```stata
. use https://www.stata-press.com/data/r16/air2
   (TIMESLAB: Airline passengers)
. corrgram air, lags(20)
```

<table>
<thead>
<tr>
<th>LAG</th>
<th>AC</th>
<th>PAC</th>
<th>Q</th>
<th>Prob&gt;Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9480</td>
<td>0.9589</td>
<td>132.14</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>0.8756</td>
<td>-0.3298</td>
<td>245.65</td>
<td>0.0000</td>
</tr>
<tr>
<td>3</td>
<td>0.8067</td>
<td>0.2018</td>
<td>342.67</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.7526</td>
<td>0.1450</td>
<td>427.74</td>
<td>0.0000</td>
</tr>
<tr>
<td>5</td>
<td>0.7138</td>
<td>0.2585</td>
<td>504.8</td>
<td>0.0000</td>
</tr>
<tr>
<td>6</td>
<td>0.6817</td>
<td>-0.0269</td>
<td>575.6</td>
<td>0.0000</td>
</tr>
<tr>
<td>7</td>
<td>0.6629</td>
<td>0.2043</td>
<td>643.04</td>
<td>0.0000</td>
</tr>
<tr>
<td>8</td>
<td>0.6556</td>
<td>0.1561</td>
<td>709.48</td>
<td>0.0000</td>
</tr>
<tr>
<td>9</td>
<td>0.6709</td>
<td>0.5686</td>
<td>779.59</td>
<td>0.0000</td>
</tr>
<tr>
<td>10</td>
<td>0.7027</td>
<td>0.2926</td>
<td>857.07</td>
<td>0.0000</td>
</tr>
<tr>
<td>11</td>
<td>0.7432</td>
<td>0.8402</td>
<td>944.39</td>
<td>0.0000</td>
</tr>
<tr>
<td>12</td>
<td>0.7604</td>
<td>0.6127</td>
<td>1036.5</td>
<td>0.0000</td>
</tr>
<tr>
<td>13</td>
<td>0.7127</td>
<td>-0.6660</td>
<td>1118</td>
<td>0.0000</td>
</tr>
<tr>
<td>14</td>
<td>0.6463</td>
<td>-0.3846</td>
<td>1185.6</td>
<td>0.0000</td>
</tr>
<tr>
<td>15</td>
<td>0.5859</td>
<td>0.0787</td>
<td>1241.5</td>
<td>0.0000</td>
</tr>
<tr>
<td>16</td>
<td>0.5380</td>
<td>-0.0266</td>
<td>1289</td>
<td>0.0000</td>
</tr>
<tr>
<td>17</td>
<td>0.4997</td>
<td>-0.0581</td>
<td>1330.4</td>
<td>0.0000</td>
</tr>
<tr>
<td>18</td>
<td>0.4687</td>
<td>-0.0435</td>
<td>1367</td>
<td>0.0000</td>
</tr>
<tr>
<td>19</td>
<td>0.4499</td>
<td>0.2773</td>
<td>1401.1</td>
<td>0.0000</td>
</tr>
<tr>
<td>20</td>
<td>0.4416</td>
<td>-0.0405</td>
<td>1434.1</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
We can use `ac` to produce a graph of the autocorrelations.

```
    . ac air, lags(20)
```

The data probably have a trend component as well as a seasonal component. First-differencing will mitigate the effects of the trend, and seasonal differencing will help control for seasonality. To accomplish this goal, we can use Stata’s time-series operators. Here we graph the partial autocorrelations after controlling for trends and seasonality. We also use `srv` to include the standardized residual variances.

```
    . pac DS12.air, lags(20) srv
```

See [U] 11.4.4 Time-series varlists for more information about time-series operators.
Video example

Correlograms and partial correlograms

Stored results

corrgram stores the following in r():

Scalars
- r(lags) number of lags
- r(ac#) AC for lag #
- r(pac#) PAC for lag #
- r(q#) Q for lag #

Matrices
- r(AC) vector of autocorrelations
- r(PAC) vector of partial autocorrelations
- r(Q) vector of Q statistics

Methods and formulas

Box, Jenkins, and Reinsel (2008, sec. 2.1.4); Newton (1988); Chatfield (2004); and Hamilton (1994) provide excellent descriptions of correlograms. Newton (1988) also discusses the calculation of the various quantities.

The autocovariance function for a time series \(x_1, x_2, \ldots, x_n\) is defined for \(|v| < n\) as

\[
\hat{R}(v) = \frac{1}{n} \sum_{i=1}^{n-|v|} (x_i - \bar{x})(x_{i+v} - \bar{x})
\]

where \(\bar{x}\) is the sample mean, and the autocorrelation function is then defined as

\[
\hat{\rho}_v = \frac{\hat{R}(v)}{\hat{R}(0)}
\]

The variance of \(\hat{\rho}_v\) is given by Bartlett’s formula for MA(q) processes. From Brockwell and Davis (2016, 92), we have

\[
\text{Var}(\hat{\rho}_v) = \begin{cases} 
\frac{1}{n} & v = 1 \\
\frac{1}{n} \left( 1 + 2 \sum_{i=1}^{v-1} \hat{\rho}^2(i) \right) & v > 1
\end{cases}
\]

The partial autocorrelation at lag \(v\) measures the correlation between \(x_t\) and \(x_{t+v}\) after the effects of \(x_{t+1}, \ldots, x_{t+v-1}\) have been removed. By default, corrgram and pac use a regression-based method to estimate it. We run an OLS regression of \(x_t\) on \(x_{t-1}, \ldots, x_{t-v}\) and a constant term. The estimated coefficient on \(x_{t-v}\) is our estimate of the \(v\)th partial autocorrelation. The residual variance is the estimated variance of that regression, which we then standardize by dividing by \(\hat{R}(0)\).

If the yw option is specified, corrgram and pac use the Yule–Walker equations to estimate the partial autocorrelations. Per Enders (2015, 64–65), let \(\phi_{vv}\) denote the \(v\)th partial autocorrelation coefficient. We then have

\[
\hat{\phi}_{11} = \hat{\rho}_1
\]
and for $v > 1$

$$
\hat{\phi}_{vv} = \frac{\hat{\rho}_v - \sum_{j=1}^{v-1} \hat{\phi}_{v-1,j} \hat{\rho}_{v-j}}{1 - \sum_{j=1}^{v-1} \hat{\phi}_{v-1,j} \hat{\rho}_j}
$$

and

$$
\hat{\phi}_{vj} = \hat{\phi}_{v-1,j} - \hat{\phi}_{vv} \hat{\phi}_{v-1,v-j} \quad j = 1, 2, \ldots, v - 1
$$

Unlike the regression-based method, the Yule–Walker equations-based method ensures that the first-sample partial autocorrelation equal the first-sample autocorrelation coefficient, as must be true in the population; see Greene (2008, 725).

McCullough (1998) discusses other methods of estimating $\hat{\phi}_{vv}$; he finds that relative to other methods, such as linear regression, the Yule–Walker equations-based method performs poorly, in part because it is susceptible to numerical error. Box, Jenkins, and Reinsel (2008, 69) also caution against using the Yule–Walker equations-based method, especially with data that are nearly nonstationary.

Acknowledgment

The ac and pac commands are based on the ac and pac commands written by Sean Becketti (1992), a past editor of the Stata Technical Bulletin and author of the Stata Press book Introduction to Time Series Using Stata.

References


——. 2013. Introduction to Time Series Using Stata. College Station, TX: Stata Press.


Also see

[TS] pergram — Periodogram
[TS] tsset — Declare data to be time-series data
[TS] wntestq — Portmanteau (Q) test for white noise
**cumsp — Graph cumulative spectral distribution**

**Description**

cumsp plots the cumulative sample spectral-distribution function evaluated at the natural frequencies for a (dense) time series.

**Quick start**

Plot cumulative sample spectral-distribution function for y using `tsset` data

cumsp y

As above, and create `newv` containing the cumulative distribution estimates

cumsp y, generate(newv)

**Menu**

Statistics > Time series > Graphs > Cumulative spectral distribution
cumsp — Graph cumulative spectral distribution

Syntax

```
cumsp varname  [ if ]  [ in ] [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>generate(newvar) create newvar holding distribution values</td>
</tr>
<tr>
<td>Plot</td>
<td>cline_options affect rendition of the plotted points connected by lines</td>
</tr>
<tr>
<td></td>
<td>marker_options change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td></td>
<td>marker_label_options add marker labels; change look or position</td>
</tr>
<tr>
<td>Add plots</td>
<td>addplot(plot) add other plots to the generated graph</td>
</tr>
<tr>
<td>Y axis, X axis, Titles, Legend, Overall</td>
<td>twoway_options any options other than by() documented in [G-3] twoway_options</td>
</tr>
</tbody>
</table>

You must tsset your data before using cumsp; see [TS] tsset. Also, the time series must be dense (nonmissing with no gaps in the time variable) in the sample specified.

varname may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

Main

```
generate(newvar) specifies a new variable to contain the estimated cumulative spectral-distribution values.
```

Plot

```
cline_options affect the rendition of the plotted points connected by lines; see [G-3] cline_options.
marker_options specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] marker_options.
marker_label_options specify if and how the markers are to be labeled; see [G-3] marker_label_options.
```

Add plots

```
addplot(plot) provides a way to add other plots to the generated graph; see [G-3] addplot_option.
```

Y axis, X axis, Titles, Legend, Overall

```
twoway_options are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option).
```
Remarks and examples

Example 1

Here we use the international airline passengers dataset (Box, Jenkins, and Reinsel 2008, Series G).
This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. In the cumulative sample spectral distribution function for these data, we also request a vertical line at frequency 1/12. Because the data are monthly, there will be a pronounced jump in the cumulative sample spectral-distribution plot at the 1/12 value if there is an annual cycle in the data.

```
. use https://www.stata-press.com/data/r16/air2
   (TIMESLAB: Airline passengers)
. cumsp air, xline(.083333333)
```

The cumulative sample spectral-distribution function clearly illustrates the annual cycle.

Methods and formulas

A time series of interest is decomposed into a unique set of sinusoids of various frequencies and amplitudes.

A plot of the sinusoidal amplitudes versus the frequencies for the sinusoidal decomposition of a time series gives us the spectral density of the time series. If we calculate the sinusoidal amplitudes for a discrete set of “natural” frequencies \(1/n, 2/n, \ldots, q/n\), we obtain the periodogram.

Let \(x(1), \ldots, x(n)\) be a time series, and let \(\omega_k = (k - 1)/n\) denote the natural frequencies for \(k = 1, \ldots, \lfloor n/2 \rfloor + 1\) where \(\lfloor \rfloor\) indicates the greatest integer function. Define

\[
C_k^2 = \frac{1}{n^2} \left| \sum_{t=1}^{n} x(t) e^{2\pi i (t-1)\omega_k} \right|^2
\]

A plot of \(nC_k^2\) versus \(\omega_k\) is then called the periodogram.
The sample spectral density may then be defined as $\hat{f}(\omega_k) = nC_k^2$.

If we let $\hat{f}(\omega_1), \ldots, \hat{f}(\omega_Q)$ be the sample spectral density function of the time series evaluated at the frequencies $\omega_j = (j - 1)/Q$ for $j = 1, \ldots, Q$ and we let $q = \lfloor Q/2 \rfloor + 1$, then

$$\hat{F}(\omega_k) = \frac{1}{q} \sum_{i=1}^{k} \hat{f}(\omega_j) \sum_{i=1}^{q} \hat{f}(\omega_j)$$

is the sample spectral-distribution function of the time series.

Reference


Also see

[TS] corrgram — Tabulate and graph autocorrelations
[TS] pergram — Periodogram
[TS] tsset — Declare data to be time-series data
dfactor estimates the parameters of dynamic-factor models by maximum likelihood. Dynamic-factor models are flexible models for multivariate time series in which unobserved factors have a vector autoregressive structure, exogenous covariates are permitted in both the equations for the latent factors and the equations for observable dependent variables, and the disturbances in the equations for the dependent variables may be autocorrelated.

Quick start

Dynamic-factor model with y1 and y2 a function of x and an unobserved factor that follows a third-order autoregressive process using tsset data

```
dfactor (y1 y2=x) (f=, ar(1/3))
```

As above, but with equations for the observed variables following an autoregressive process of order 1

```
dfactor (y1 y2=x, ar(1)) (f=, ar(1/3))
```

As above, but with an unstructured covariance matrix for the errors of y1 and y2

```
dfactor (y1 y2=x, ar(1) covstructure(unstructured)) (f=, ar(1/3))
```
## dfactor — Dynamic-factor models

### Syntax

```
dfactor obs_eq [fac_eq] [if] [in] [, options]
```

`obs_eq` specifies the equation for the observed dependent variables, and it has the form

\[(depvars = [exog_d] [, sopts])\]

`fac_eq` specifies the equation for the unobserved factors, and it has the form

\[(facvars = [exog_f] [, sopts])\]

`depvars` are the observed dependent variables. `exog_d` are the exogenous variables that enter into the equations for the observed dependent variables. (All factors are automatically entered into the equations for the observed dependent variables.) `facvars` are the names for the unobserved factors in the model. You may specify the names of existing variables in `facvars`, but `dfactor` treats them only as names and takes no notice that they are also variables. `exog_f` are the exogenous variables that enter into the equations for the factors.

### options

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
</tr>
<tr>
<td><code>constraints(constraints)</code></td>
</tr>
<tr>
<td><strong>SE/Robust</strong></td>
</tr>
<tr>
<td><code>vce(vcetype)</code></td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
</tr>
<tr>
<td><code>level(#)</code></td>
</tr>
<tr>
<td><code>nocnsreport</code></td>
</tr>
<tr>
<td><code>display_options</code></td>
</tr>
<tr>
<td><strong>Maximization</strong></td>
</tr>
<tr>
<td><code>maximize_options</code></td>
</tr>
<tr>
<td><code>from(matname)</code></td>
</tr>
<tr>
<td><strong>Advanced</strong></td>
</tr>
<tr>
<td><code>method(method)</code></td>
</tr>
<tr>
<td><code>coeflegend</code></td>
</tr>
</tbody>
</table>
sopts

Model

noconstant  suppress constant term from the equation; allowed only in \texttt{obs\_eq}

\texttt{ar(numlist)}  autoregressive terms

\texttt{arstructure(arstructure)}  structure of autoregressive coefficient matrices

\texttt{covstructure(covstructure)}  covariance structure

\begin{tabular}{ll}
\texttt{arstructure} & Description \\
\hline
diagonal & diagonal matrix; the default \\
ltriangular & lower triangular matrix \\
general & general matrix \\
\hline
\texttt{covstructure} & Description \\
\hline
identity & identity matrix \\
dscalar & diagonal scalar matrix \\
diagonal & diagonal matrix \\
unstructured & symmetric, positive-definite matrix \\
\hline
\texttt{method} & Description \\
\hline
hybrid & use the stationary Kalman filter and the De Jong diffuse Kalman filter; the default \\
dejong & use the stationary De Jong method and the De Jong diffuse Kalman filter \\
\hline
\end{tabular}

You must \texttt{tsset} your data before using \texttt{dfactor}; see [TS] \texttt{tsset}.
\texttt{exog\_d} and \texttt{exog\_f} may contain factor variables; see [U] \texttt{11.4.3 Factor variables}.
\texttt{depvars}, \texttt{exog\_d}, and \texttt{exog\_f} may contain time-series operators; see [U] \texttt{11.4.4 Time-series varlists}.
by, \texttt{fp}, \texttt{rolling}, and \texttt{statsby} are allowed; see [U] \texttt{11.1.10 Prefix commands}.
\texttt{coeflegend} does not appear in the dialog box.
See [U] \texttt{20 Estimation and postestimation commands} for more capabilities of estimation commands.

**Options**

\begin{itemize}
\item \texttt{constraints(constraints)} apply linear constraints. Some specifications require linear constraints for parameter identification.
\item \texttt{noconstant} suppresses the constant term.
\item \texttt{ar(numlist)} specifies the vector autoregressive lag structure in the equation. By default, no lags are included in either the observable or the factor equations.
\item \texttt{arstructure(diagonal\vert ltriangular\vert general)} specifies the structure of the matrices in the vector autoregressive lag structure.
\end{itemize}
arstructure(diagonal) specifies the matrices to be diagonal—separate parameters for each lag, but no cross-equation autocorrelations. arstructure(diagonal) is the default for both the observable and the factor equations.

arstructure(ltriangular) specifies the matrices to be lower triangular—parameterizes a recursive, or Wold causal, structure.

arstructure(general) specifies the matrices to be general matrices—separate parameters for each possible autocorrelation and cross-correlation.

covstructure(identity | dscalar | diagonal | unstructured) specifies the covariance structure of the errors.

covstructure(identity) specifies a covariance matrix equal to an identity matrix, and it is the default for the errors in the factor equations.

covstructure(dscalar) specifies a covariance matrix equal to $\sigma^2$ times an identity matrix.

covstructure(diagonal) specifies a diagonal covariance matrix, and it is the default for the errors in the observable variables.

covstructure(unstructured) specifies a symmetric, positive-definite covariance matrix with parameters for all variances and covariances.

vce(vcetype) specifies the estimator for the variance–covariance matrix of the estimator.

vce(oim), the default, causes dfactor to use the observed information matrix estimator.

vce(robust) causes dfactor to use the Huber/White/sandwich estimator.

level(#) ; see [R] Estimation options.

nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, noclabel, fvwrap(#), fvwrapon(style), cformat(%fmt), pformat(%fmt), and sformat(%fmt); see [R] Estimation options.

Maximization

maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), and from(matname); see [R] Maximize for all options except from(), and see below for information on from(). These options are seldom used.

from(matname) specifies initial values for the maximization process. from(b0) causes dfactor to begin the maximization algorithm with the values in b0. b0 must be a row vector; the number of columns must equal the number of parameters in the model; and the values in b0 must be in the same order as the parameters in e(b). This option is seldom used.

Advanced

method(method) specifies how to compute the log likelihood. dfactor writes the model in state-space form and uses sspace to estimate the parameters; see [TS] sspace. method() offers two methods for dealing with some of the technical aspects of the state-space likelihood. This option is seldom used.
method(hybrid), the default, uses the Kalman filter with model-based initial values when the
model is stationary and uses the De Jong (1988, 1991) diffuse Kalman filter when the model
is nonstationary.

method(dejong) uses the De Jong (1988) method for estimating the initial values for the Kalman
filter when the model is stationary and uses the De Jong (1988, 1991) diffuse Kalman filter
when the model is nonstationary.

The following option is available with dfactor but is not shown in the dialog box:

coeflegend; see [R] Estimation options.

Remarks and examples

Remarks are presented under the following headings:

An introduction to dynamic-factor models
Some examples

An introduction to dynamic-factor models

dfactor estimates the parameters of dynamic-factor models by maximum likelihood (ML). Dynamic-
factor models represent a vector of \( k \) endogenous variables as linear functions of \( n_f < k \) unobserved
factors and some exogenous covariates. The unobserved factors and the disturbances in the equations
for the observed variables may follow vector autoregressive structures.

Dynamic-factor models have been developed and applied in macroeconomics; see Geweke (1977),

Dynamic-factor models are very flexible; in a sense, they are too flexible. Constraints must be
imposed to identify the parameters of dynamic-factor and static-factor models. The parameters in the
default specifications in dfactor are identified, but other specifications require additional restrictions.
The factors are identified only up to a sign, which means that the coefficients on the unobserved factors
can flip signs and still produce the same predictions and the same log likelihood. The flexibility of
the model sometimes produces convergence problems.

dfactor is designed to handle cases in which the number of modeled endogenous variables, \( k \),
is small. The ML estimator is implemented by writing the model in state-space form and by using
the Kalman filter to derive and implement the log likelihood. As \( k \) grows, the number of parameters
quickly exceeds the number that can be estimated.
A dynamic-factor model has the form

\[
\begin{align*}
y_t &= P f_t + Q x_t + u_t \\
f_t &= R w_t + A_1 f_{t-1} + A_2 f_{t-2} + \cdots + A_p f_{t-p} + \nu_t \\
u_t &= C_1 u_{t-1} + C_2 u_{t-2} + \cdots + C_q u_{t-q} + \epsilon_t
\end{align*}
\]

where the definitions are given in the following table:

<table>
<thead>
<tr>
<th>Item</th>
<th>Dimension</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_t)</td>
<td>(k \times 1)</td>
<td>vector of dependent variables</td>
</tr>
<tr>
<td>(P)</td>
<td>(k \times n_f)</td>
<td>matrix of parameters</td>
</tr>
<tr>
<td>(f_t)</td>
<td>(n_f \times 1)</td>
<td>vector of unobservable factors</td>
</tr>
<tr>
<td>(Q)</td>
<td>(k \times n_x)</td>
<td>matrix of parameters</td>
</tr>
<tr>
<td>(x_t)</td>
<td>(n_x \times 1)</td>
<td>vector of exogenous variables</td>
</tr>
<tr>
<td>(u_t)</td>
<td>(k \times 1)</td>
<td>vector of disturbances</td>
</tr>
<tr>
<td>(R)</td>
<td>(n_f \times n_w)</td>
<td>matrix of parameters</td>
</tr>
<tr>
<td>(w_t)</td>
<td>(n_w \times 1)</td>
<td>vector of exogenous variables</td>
</tr>
<tr>
<td>(A_i)</td>
<td>(n_f \times n_f)</td>
<td>matrix of autocorrelation parameters for (i \in {1, 2, \ldots, p})</td>
</tr>
<tr>
<td>(\nu_t)</td>
<td>(n_f \times 1)</td>
<td>vector of disturbances</td>
</tr>
<tr>
<td>(C_i)</td>
<td>(k \times k)</td>
<td>matrix of autocorrelation parameters for (i \in {1, 2, \ldots, q})</td>
</tr>
<tr>
<td>(\epsilon_t)</td>
<td>(k \times 1)</td>
<td>vector of disturbances</td>
</tr>
</tbody>
</table>

By selecting different numbers of factors and lags, the dynamic-factor model encompasses the six models in the table below:

<table>
<thead>
<tr>
<th>Model</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic factors with vector autoregressive errors</td>
<td>(DFAR)</td>
</tr>
<tr>
<td>Dynamic factors</td>
<td>(DF)</td>
</tr>
<tr>
<td>Static factors with vector autoregressive errors</td>
<td>(SFAR)</td>
</tr>
<tr>
<td>Static factors</td>
<td>(SF)</td>
</tr>
<tr>
<td>Vector autoregressive errors</td>
<td>(VAR)</td>
</tr>
<tr>
<td>Seemingly unrelated regression</td>
<td>(SUR)</td>
</tr>
</tbody>
</table>

In addition to the time-series models, d`factor` can estimate the parameters of SF models and SUR models. d`factor` can place equality constraints on the disturbance covariances, which `sureg` and `var` do not allow.

### Some examples

#### Example 1: Dynamic-factor model

Stock and Watson (1989, 1991) wrote a simple macroeconomic model as a DF model, estimated the parameters by ML, and extracted an economic indicator. In this example, we estimate the parameters of a DF model. In [TS] d`factor postestimation, we extend this example and extract an economic indicator for the differenced series.

We have data on an industrial-production index, `ipman`; real disposable income, `income`; an aggregate weekly hours index, `hours`; and aggregate unemployment, `unemp`. We believe that these variables are first-difference stationary. We model their first differences as linear functions of an unobserved factor that follows a second-order autoregressive process.
. use https://www.stata-press.com/data/r16/dfex
(St. Louis Fed (FRED) macro data)
. dfactor (D.(ipman income hours unemp) = , noconstant) (f = , ar(1/2))
searching for initial values ........................
(setting technique to bhhh)
Iteration 0: log likelihood = -675.19823
Iteration 1: log likelihood = -666.74344
(output omitted)
Refining estimates:
Iteration 0: log likelihood = -662.09507
Iteration 1: log likelihood = -662.09507
Dynamic-factor model
Sample: 1972m2 - 2008m11 Number of obs = 442
Wald chi2(6) = 751.95
Log likelihood = -662.09507 Prob > chi2 = 0.0000

| OIM                      | Coef.   | Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|--------------------------|---------|-----------|-------|-------|---------------------|
| f                        |         |           |       |       |                     |
| f                        | 0.2651932 | 0.0568663 | 4.66  | 0.000 | 0.1537372 - 0.3766491 |
| L1.                      | 0.4820398 | 0.0624635 | 7.72  | 0.000 | 0.3596136 - 0.604466  |
| L2.                      |         |           |       |       |                     |
| D.ipman                  |         |           |       |       |                     |
| f                        | 0.3502249 | 0.0287389 | 12.19 | 0.000 | 0.2938976 - 0.4065522 |
| D.income                 |         |           |       |       |                     |
| f                        | 0.0746338 | 0.0217319 | 3.43  | 0.001 | 0.0320401 - 0.1172276 |
| D.hours                  |         |           |       |       |                     |
| f                        | 0.2177469 | 0.0186769 | 11.66 | 0.000 | 0.1811407 - 0.254353  |
| D.unemp                  |         |           |       |       |                     |
| f                        | -0.0676016 | 0.0071022 | -9.52 | 0.000 | -0.0815217 - -0.0536816 |
| /observable              |         |           |       |       |                     |
| var(De.ipman)            | 0.1383158 | 0.0167086 | 8.28  | 0.000 | 0.1055675 - 0.1710641 |
| var(De.income)           | 0.2773808 | 0.0188302 | 14.73 | 0.000 | 0.2404743 - 0.3142873 |
| var(De.hours)            | 0.0911446 | 0.0080847 | 11.27 | 0.000 | 0.0752988 - 0.1069903 |
| var(De.unemp)            | 0.0237232 | 0.0017932 | 13.23 | 0.000 | 0.0202086 - 0.0272378 |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

For a discussion of the atypical iteration log, see example 1 in [TS] sspace.

The header in the output describes the estimation sample, reports the log-likelihood function at the maximum, and gives the results of a Wald test against the null hypothesis that the coefficients on the independent variables, the factors, and the autoregressive components are all zero. In this example, the null hypothesis that all parameters except for the variance parameters are zero is rejected at all conventional levels.

The results in the estimation table indicate that the unobserved factor is quite persistent and that it is a significant predictor for each of the observed variables.
dfactor writes the DF model as a state-space model and uses the same methods as sspace to estimate the parameters. Example 5 in [TS] sspace writes the model considered here in state-space form and uses sspace to estimate the parameters.

Technical note

The signs of the coefficients on the unobserved factors are not identified. They are not identified because we can multiply the unobserved factors and the coefficients on the unobserved factors by negative one without changing the log likelihood or any of the model predictions.

Altering either the starting values for the maximization process, the maximization technique() used, or the platform on which the command is run can cause the signs of the estimated coefficients on the unobserved factors to change.

Changes in the signs of the estimated coefficients on the unobserved factors do not alter the implications of the model or the model predictions.

Example 2: Dynamic-factor model with covariates

Here we extend the previous example by allowing the errors in the equations for the observables to be autocorrelated. This extension yields a constrained VAR model with an unobserved autocorrelated factor.
We estimate the parameters by typing

```
.dfactor (D.(ipman income hours unemp) = , noconstant ar(1)) (f = , ar(1/2))
```

searching for initial values ..............

(setting technique to bhhh)

Iteration 0: log likelihood = -659.68789
Iteration 1: log likelihood = -631.6043

(output omitted)

Refining estimates:
Iteration 0: log likelihood = -610.28846
Iteration 1: log likelihood = -610.28846

Dynamic-factor model

Sample: 1972m2 - 2008m11
Number of obs = 442

Wald chi2(10) = 990.91
Log likelihood = -610.28846
Prob > chi2 = 0.0000

| Coef. | Std. Err. | z | P>|z| | [95% Conf. Interval] |
|-------|-----------|---|--------|-----------------|
| f     |           |   |        |                 |
| L1.   | .4058457  | .0906183 | 4.48  | 0.000 | .2282371 .5834544 |
| L2.   | .3663499  | .0849584 | 4.31  | 0.000 | .1998344 .5328654 |

De.ipman
e.ipman
LD. -.2772149 .068808 -4.03 0.000 -.4120761 -.1423538

De.income
e.income
LD. -.2213824 .0470578 -4.70 0.000 -.3136141 -.1291508

De.hours
e.hours
LD. -.3969317 .0504256 -7.87 0.000 -.495764 -.2980994

De.unemp
e.unemp
LD. -.1736835 .0532071 -3.26 0.001 -.2779675 -.0693995

D.ipman
f .3214972 .027982 11.49 0.000 .2666535 .3763408

D.income
f .0760412 .0173844 4.37 0.000 .0419684 .110114

D.hours
f .1933165 .0172969 11.18 0.000 .1594151 .2272179

D.unemp
f -.0711994 .0066553 -10.70 0.000 -.0842435 -.0581553

/observable

| var(De.ipman) | .1387909  | .0154558 | 8.98  | 0.000 | .1084981 .1690837 |
| var(De.income) | .2636239  | .0179043 | 14.72  | 0.000 | .2285322 .2987157 |
| var(De.hours)  | .0822919  | .0071096 | 11.57  | 0.000 | .0683574 .0962265 |
| var(De.unemp)  | .0218056  | .0016658 | 13.09  | 0.000 | .0185407 .0250704 |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
The autoregressive (AR) terms are displayed in error notation. \texttt{e.varname} stands for the error in the equation for \texttt{varname}. The estimate of the $p$th AR term from \texttt{y1} on \texttt{y2} is reported as \texttt{Lpe.y1} in equation \texttt{e.y2}. In the above output, the estimated first-order AR term of \texttt{D.ipman} on \texttt{D.ipman} is $-0.277$ and is labeled as \texttt{LDe.ipman} in equation \texttt{De.ipman}.

The previous two examples illustrate how to use \texttt{dfactor} to estimate the parameters of DF models. Although the previous example indicates that the more general DFAR model fits the data well, we use these data to illustrate how to estimate the parameters of more restrictive models.

\begin{itemize}
  \item \textbf{Example 3: A VAR with constrained error variance}
\end{itemize}

In this example, we use \texttt{dfactor} to estimate the parameters of a SUR model with constraints on the error-covariance matrix. The model is also a constrained VAR with constraints on the error-covariance matrix, because we include the lags of two dependent variables as exogenous variables to model the dynamic structure of the data. Previous exploratory work suggested that we should drop the lag of \texttt{D.unemp} from the model.
constraint 1: \[ \text{cov}(\text{De.unemp}, \text{De.income}) = 0 \]

\[ \text{dfactor (D.(ipman income unemp) = LD.(ipman income), noconstant} \]
\[ > \text{covstructure(unstructured)), constraints(1)} \]

searching for initial values ............
(setting technique to bhhh)

Iteration 0: log likelihood = -569.34353
Iteration 1: log likelihood = -548.7669
(output omitted)

Refining estimates:
Iteration 0: log likelihood = -535.12973
Iteration 1: log likelihood = -535.12973

Dynamic-factor model
Sample: 1972m3 - 2008m11 Number of obs = 441

Wald chi2(6) = 88.32
Log likelihood = -535.12973 Prob > chi2 = 0.0000

( 1) \[ \text{cov}(\text{De.income, De.unemp}) = 0 \]

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{Coef.} & \text{Std. Err.} & \text{z} & \text{P>|z|} & \text{[95\% Conf. Interval]} \\
\hline
\text{D.ipman (ipman LD.)} & 0.206276 & 0.0471654 & 4.37 & 0.000 & 0.1138335 & 0.2987185 \\
\text{income LD.)} & 0.1867384 & 0.0512139 & 3.65 & 0.000 & 0.086361 & 0.2871158 \\
\hline
\text{D.income (ipman LD.)} & 0.1043733 & 0.0434048 & 2.40 & 0.016 & 0.0193015 & 0.1894451 \\
\text{income LD.)} & -0.1957893 & 0.0471305 & -4.15 & 0.000 & -0.2881634 & -0.1034153 \\
\hline
\text{D.unemp (ipman LD.)} & -0.0865823 & 0.0140747 & -6.15 & 0.000 & -0.1141681 & -0.0589964 \\
\text{income LD.)} & -0.0200749 & 0.0152828 & -1.31 & 0.189 & -0.0500285 & 0.0098788 \\
\hline
\text{/observable (var(De.ipman) = 0.3243902 \[0.0218533 \[14.84 \[0.000 \[0.2815584 \[0.3672219 \\
\text{cov(De.ipman, De.income) = 0.0445794 \[0.013696 \[3.25 \[0.001 \[0.0177358 \[0.071423 \\
\text{cov(De.ipman, De.unemp) = -0.2998076 \[0.0047755 \[-6.24 \[0.000 \[-0.0391674 \[-0.0204478 \\
\text{var(De.income, De.unemp) = 0.2747234 \[0.0185008 \[14.85 \[0.000 \[0.2384624 \[0.3109844 \\
\text{var(De.unemp) = 0.0288866 \[0.0019453 \[14.85 \[0.000 \[0.0250738 \[0.0326994 \\
\hline
\end{array}
\]

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The output indicates that the model fits well, except that the lag of first-differenced income is not a significant predictor of first-differenced unemployment.
The previous example shows how to use `dfactor` to estimate the parameters of a SUR model with constraints on the error-covariance matrix. Neither `sureg` nor `var` allows for constraints on the error-covariance matrix. Without the constraints on the error-covariance matrix and including the lag of `D.unemp`,

```
. dfactor (D.(ipman income unemp) = LD.(ipman income unemp),
    > noconstant covstructure(unstructured))
    (output omitted)
```

```
. var D.(ipman income unemp), lags(1) noconstant
    (output omitted)
```

and

```
. sureg (D.ipman LD.(ipman income unemp), noconstant)
    > (D.income LD.(ipman income unemp), noconstant)
    > (D.unemp LD.(ipman income unemp), noconstant)
    (output omitted)
```

produce the same estimates after allowing for small numerical differences.

Example 4: A lower-triangular VAR with constrained error variance

The previous example estimated the parameters of a constrained VAR model with a constraint on the error-covariance matrix. This example makes two refinements on the previous one: we use an unconditional estimator instead of a conditional estimator, and we constrain the AR parameters to have a lower triangular structure. (See the next technical note for a discussion of conditional and unconditional estimators.) The results are
. constraint 1 [observable]cov(De.unemp,De.income) = 0
. dfactor (D.(ipman income unemp) = , ar(1) arstructure(ltriangular) noconstant > covstructure(unstructured)), constraints(1)
searching for initial values ........................
(setting technique to bhhh)
Iteration 0: log likelihood = -543.89917
Iteration 1: log likelihood = -541.47792
(output omitted)
Refining estimates:
Iteration 0: log likelihood = -540.36159
Iteration 1: log likelihood = -540.36159

Dynamic-factor model
Sample: 1972m2 - 2008m11 Number of obs = 442
Wald chi2(6) = 75.48
Log likelihood = -540.36159 Prob > chi2 = 0.0000
( 1) [observable]cov(De.income,De.unemp) = 0

| Coef. Std. Err. z P>|z| [95% Conf. Interval] |
|---|---|---|---|---|
| De.ipman | | | | |
| e.ipman | .2297308 | .0473147 | 4.86 | 0.000 | .1369957 | .3224659 |
| LD. | | | | |
| De.income | | | | |
| e.ipman | .1075441 | .0433357 | 2.48 | 0.013 | .0226077 | .1924805 |
| LD. | | | | |
| e.income | -.2209485 | .0471116 | -4.69 | 0.000 | -.3132943 | -.1286028 |
| LD. | | | | |
| De.unemp | | | | |
| e.ipman | -.0975759 | .0151301 | -6.45 | 0.000 | -.1272304 | -.0679215 |
| LD. | | | | |
| e.income | -.0000467 | .0147848 | -0.00 | 0.997 | -.0290244 | .0289309 |
| LD. | | | | |
| e.unemp | -.0795348 | .0482213 | -1.65 | 0.099 | -.1740469 | .0149773 |
| LD. | | | | |
| /observable | var(De.ipman) | .3335286 | .0224282 | 14.87 | 0.000 | .2895702 | .377487 |
| cov(De.ipman, De.income) | .0457804 | .0139123 | 3.29 | 0.001 | .0185127 | .0730481 |
| cov(De.ipman, De.unemp) | -.0329438 | .0051423 | -6.41 | 0.000 | -.0430226 | -.022865 |
| var(De.income) | .2743375 | .0184657 | 14.86 | 0.000 | .2381454 | .3105296 |
| cov(De.income, De.unemp) | 0 (constrained) | | | | |
| var(De.unemp) | .0292088 | .001999 | 14.68 | 0.000 | .0253083 | .0331092 |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The estimated AR term of D.income on D.unemp is essentially 0. The estimated AR term for D.unemp on D.unemp is −0.0795, and it is not significant at the 1% or 5% levels. The estimated AR term of D.ipman on D.income is 0.1075 and is significant at the 5% level but not at the 1% level.
Technical note

We obtained the unconditional estimator in example 4 by specifying the `ar()` option instead of including the lags of the endogenous variables as exogenous variables, as we did in example 3. The unconditional estimator has an additional observation and is more efficient. This change is analogous to estimating an AR coefficient by `arima` instead of using `regress` on the lagged endogenous variable. For example, to obtain the unconditional estimator in a univariate model, typing

```
. arima D.ipman, ar(1) noconstant technique(nr)
(output omitted)
```

will produce the same estimated AR coefficient as

```
. dfactor (D.ipman, ar(1) noconstant)
(output omitted)
```

We obtain the conditional estimator by typing either

```
. regress D.ipman LD.ipman, noconstant
(output omitted)
```

or

```
. dfactor (D.ipman = LD.ipman, noconstant)
(output omitted)
```

Example 5: A static factor model

In this example, we fit regional unemployment data to an SF model. We have data on the unemployment levels for the four regions in the U.S. census: `west` for the West, `south` for the South, `ne` for the Northeast, and `midwest` for the Midwest. We treat the variables as first-difference stationary and model the first differences of these variables. Using `dfactor` yields
. use https://www.stata-press.com/data/r16/urate
(Monthly unemployment rates in US Census regions)
. dfactor (D.(west south ne midwest) = , noconstant ) (z = )
searching for initial values ................
(setting technique to bhhh)
Iteration 0: log likelihood = 872.71993
Iteration 1: log likelihood = 873.04786
(output omitted)
Refining estimates:
Iteration 0: log likelihood = 873.0755
Iteration 1: log likelihood = 873.0755
Dynamic-factor model
Sample: 1990m2 - 2008m12 Number of obs = 227
Wald chi2(4) = 342.56
Log likelihood = 873.0755 Prob > chi2 = 0.0000

| OIM          | Coef. | Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|--------------|-------|-----------|-------|-------|----------------------|
| D.west z     | 0.0978324 | 0.0065644 | 14.90 | 0.000 | 0.0849664 - 0.1106983 |
| D.south z    | 0.0859494 | 0.0061762 | 13.92 | 0.000 | 0.0738442 - 0.0980546 |
| D.ne z       | 0.0918607 | 0.0072814 | 12.62 | 0.000 | 0.0775893 - 0.106132  |
| D.midwest z  | 0.0861102 | 0.0074652 | 11.53 | 0.000 | 0.0714787 - 0.1007417 |

/observable
var(De.west)  | 0.0036887 | 0.0005834 | 6.32  | 0.000 | 0.0025453 - 0.0048322 |
var(De.south) | 0.0038902 | 0.0005228 | 7.44  | 0.000 | 0.0028656 - 0.0049149 |
var(De.ne)    | 0.0064074 | 0.0007558 | 8.48  | 0.000 | 0.0049261 - 0.0078887 |
var(De.midw-t)| 0.0074749 | 0.0008271 | 9.04  | 0.000 | 0.0058538 - 0.009096  |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The estimates indicate that we could reasonably suppose that the unobserved factor has the same effect on the changes in unemployment in all four regions. The output below shows that we cannot reject the null hypothesis that these coefficients are the same.

   ( 1) [D.west]z - [D.south]z = 0
   ( 2) [D.west]z - [D.ne]z = 0
   ( 3) [D.west]z - [D.midwest]z = 0
   chi2( 3) = 3.58
   Prob > chi2 = 0.3109

Example 6: A static factor with constraints

In this example, we impose the constraint that the unobserved factor has the same impact on changes in unemployment in all four regions. This constraint was suggested by the results of the previous example. The previous example did not allow for any dynamics in the variables, a problem we alleviate by allowing the disturbances in the equation for each observable to follow an AR(1) process.
. constraint 2 [D.west]z = [D.south]z
. constraint 3 [D.west]z = [D.ne]z
. constraint 4 [D.west]z = [D.midwest]z
. dfactor (D.(west south ne midwest) = , noconstant ar(1)) (z = ),
> constraints(2/4)
searching for initial values .............
(setting technique to bhhh)
Iteration 0: log likelihood = 827.97004
Iteration 1: log likelihood = 874.74471
(output omitted)
Refining estimates:
Iteration 0: log likelihood = 880.97488
Iteration 1: log likelihood = 880.97488

Dynamic-factor model
Sample: 1990m2 - 2008m12 Number of obs = 227
Log likelihood = 880.97488 Prob > chi2 = 0.0000
( 1) [D.west]z - [D.south]z = 0
( 2) [D.west]z - [D.ne]z = 0
( 3) [D.west]z - [D.midwest]z = 0

<table>
<thead>
<tr>
<th></th>
<th>OIM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coef.</td>
</tr>
<tr>
<td>De.west</td>
<td></td>
</tr>
<tr>
<td>e.west</td>
<td>.1297</td>
</tr>
<tr>
<td>LD.</td>
<td></td>
</tr>
<tr>
<td>De.south</td>
<td></td>
</tr>
<tr>
<td>e.south</td>
<td>-.283</td>
</tr>
<tr>
<td>LD.</td>
<td></td>
</tr>
<tr>
<td>De.ne</td>
<td></td>
</tr>
<tr>
<td>e.ne</td>
<td>.287</td>
</tr>
<tr>
<td>LD.</td>
<td></td>
</tr>
<tr>
<td>De.midwest</td>
<td></td>
</tr>
<tr>
<td>e.midwest</td>
<td>.0049</td>
</tr>
<tr>
<td>LD.</td>
<td></td>
</tr>
<tr>
<td>D.west</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>.0904</td>
</tr>
<tr>
<td>D.south</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>.0904</td>
</tr>
<tr>
<td>D.ne</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>.0904</td>
</tr>
<tr>
<td>D.midwest</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>.0904</td>
</tr>
<tr>
<td>/observable</td>
<td></td>
</tr>
<tr>
<td>var(De.west)</td>
<td>.0039</td>
</tr>
<tr>
<td>var(De.south)</td>
<td>.0035</td>
</tr>
<tr>
<td>var(De.ne)</td>
<td>.0058</td>
</tr>
<tr>
<td>var(De.midw&amp;nt)</td>
<td>.0075</td>
</tr>
</tbody>
</table>

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
The results indicate that the model might not fit well. Two of the four AR coefficients are statistically insignificant, while the two significant coefficients have opposite signs and sum to about zero. We suspect that a DF model might fit these data better than an SF model with autocorrelated disturbances.

 Stored results

dfactor stores the following in e():

Scalars

- e(N): number of observations
- e(k): number of parameters
- e(k_eq): number of equations in e(b)
- e(k_eq_model): number of equations in overall model test
- e(k_dv): number of dependent variables
- e(k_obser): number of observation equations
- e(k_factor): number of factors specified
- e(o_ar_max): number of AR terms for the disturbances
- e(f_ar_max): number of AR terms for the factors
- e(df_m): model degrees of freedom
- e(ll): log likelihood
- e(chi2): \( \chi^2 \)
- e(p): p-value for model test
- e(tmin): minimum time in sample
- e(tmax): maximum time in sample
- e(stationary): 1 if the estimated parameters indicate a stationary model, 0 otherwise
- e(rank): rank of VCE
- e(ic): number of iterations
- e(rc): return code
- e(converged): 1 if converged, 0 otherwise

Macros

- e(cmd): dfactor
- e(cmdline): command as typed
- e(depvar): unoperated names of dependent variables in observation equations
- e(obser_deps): names of dependent variables in observation equations
- e(covariates): list of covariates
- e(factor_deps): names of unobserved factors in model
- e(tvar): variable denoting time within groups
- e(eqnames): names of equations
- e(model): type of dynamic-factor model specified
- e(title): title in estimation output
- e(tmins): formatted minimum time
- e(tmaxs): formatted maximum time
- e(o_ar): list of AR terms for disturbances
- e(f_ar): list of AR terms for factors
- e(observ_cov): structure of observation-error covariance matrix
- e(factor_cov): structure of factor-error covariance matrix
- e(chi2type): Wald; type of model \( \chi^2 \) test
- e(vc): \( vce \) type specified in \( vce() \)
- e(vcetype): title used to label Std. Err.
- e(opt): type of optimization
- e(method): likelihood method
- e(initial_values): type of initial values
- e(technique): maximization technique
- e(tech_steps): iterations taken in maximization technique(s)
- e(datasignature): the checksum
- e(datasignaturevars): variables used in calculation of checksum
- e(properties): b V
- e(estat_cmd): program used to implement estat
- e(predict): program used to implement predict
Methods and formulas

dfactor writes the specified model as a state-space model and uses sspace to estimate the parameters by maximum likelihood. See Lütkepohl (2005, 619–621) for how to write the DF model in state-space form. See [TS] sspace for the technical details.

References


Also see

[TS] dfactor postestimation — Postestimation tools for dfactor
[TS] arima — ARIMA, ARMAX, and other dynamic regression models
[TS] sspace — State-space models
[TS] tsset — Declare data to be time-series data
[TS] var — Vector autoregressive models
[R] regress — Linear regression
[R] sureg — Zellner’s seemingly unrelated regression
[U] 20 Estimation and postestimation commands
Postestimation commands

The following standard postestimation commands are available after `dfactor`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear</td>
</tr>
<tr>
<td></td>
<td>combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear</td>
</tr>
<tr>
<td></td>
<td>combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic</td>
</tr>
<tr>
<td></td>
<td>measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized</td>
</tr>
<tr>
<td></td>
<td>predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as expected values, unobserved factors, autocorrelated disturbances, and innovations. The root mean squared error is available for all predictions. All predictions are also available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] { stub* | newvarlist } [ if ] [ in ] [ , statistic options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>y dependent variable, which is $\text{xb} + \text{residuals}$</td>
</tr>
<tr>
<td></td>
<td>xb linear predictions using the observable independent variables</td>
</tr>
<tr>
<td></td>
<td>xbf linear predictions using the observable independent variables plus the factor contributions</td>
</tr>
<tr>
<td></td>
<td>factors unobserved factor variables</td>
</tr>
<tr>
<td></td>
<td>residuals autocorrelated disturbances</td>
</tr>
<tr>
<td></td>
<td>innovations innovations, the observed dependent variable minus the predicted y</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

```
<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>equation(eqnames)</td>
<td>specify name(s) of equation(s) for which predictions are to be made</td>
</tr>
<tr>
<td>rmse(stub*</td>
<td>newvarlist)</td>
</tr>
<tr>
<td>dynamic(time_constant)</td>
<td>begin dynamic forecast at specified time</td>
</tr>
<tr>
<td>Advanced</td>
<td></td>
</tr>
<tr>
<td>smethod(method)</td>
<td>method for predicting unobserved states</td>
</tr>
</tbody>
</table>

method          | Description                                                                |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>onestep</td>
<td>predict using past information</td>
</tr>
<tr>
<td>smooth</td>
<td>predict using all sample information</td>
</tr>
<tr>
<td>filter</td>
<td>predict using past and contemporaneous information</td>
</tr>
</tbody>
</table>
Options for predict

The mathematical notation used in this section is defined in Description of [TS] dfactor.

Main

y, xb, xbf, factors, residuals, and innovations specify the statistic to be predicted.

y, the default, predicts the dependent variables. The predictions include the contributions of the unobserved factors, the linear predictions by using the observable independent variables, and any autocorrelation, $\hat{P}\hat{f}_t + \hat{Q}x_t + \hat{u}_t$.

xb calculates the linear prediction by using the observable independent variables, $\hat{Q}x_t$.

xbf calculates the contributions of the unobserved factors plus the linear prediction by using the observable independent variables, $\hat{P}\hat{f}_t + \hat{Q}x_t$.

factors estimates the unobserved factors, $\hat{f}_t = \hat{R}w_t + \hat{A}_1\hat{f}_{t-1} + \hat{A}_2\hat{f}_{t-2} + \cdots + \hat{A}_{t-p}\hat{f}_{t-p}$.

residuals calculates the autocorrelated residuals, $\hat{u}_t = \hat{C}_1\hat{u}_{t-1} + \hat{C}_2\hat{u}_{t-2} + \cdots + \hat{C}_{t-q}\hat{u}_{t-q}$.

innovations calculates the innovations, $\hat{\epsilon}_t = y_t - \hat{P}\hat{f}_t + \hat{Q}x_t - \hat{u}_t$.

Options

equation(eqnames) specifies the equation(s) for which the predictions are to be calculated.

You specify equation names, such as equation(income consumption) or equation(factor1 factor2), to identify the equations. For the factors statistic, you must specify names of equations for factors; for all other statistics, you must specify names of equations for observable variables.

If you do not specify equation() and do not specify stub*, the results are the same as if you had specified the name of the first equation for the predicted statistic.

equation() may not be specified with stub*.

rmse(stub* | newvarlist) puts the root mean squared errors of the predicted objects into the specified new variables. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error.

dynamic(time_constant) specifies when predict starts producing dynamic forecasts. The specified time_constant must be in the scale of the time variable specified in tsset, and the time_constant must be inside a sample for which observations on the dependent variables are available. For example, dynamic(tq(2008q4)) causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly, see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. dynamic() may not be specified with xb, xbf, innovations, smethod(filter), or smethod(smooth).

Advanced

smethod(method) specifies the method used to predict the unobserved states in the model. smethod() may not be specified with xb.

smethod(onestep), the default, causes predict to use previous information on the dependent variables. The Kalman filter is performed on previous periods, but only the one-step predictions are made for the current period.

smethod(smooth) causes predict to estimate the states at each time period using all the sample data by the Kalman smoother.
smethod(filter) causes predict to estimate the states at each time period using previous and contemporaneous data by the Kalman filter. The Kalman filter is performed on previous periods and the current period. smethod(filter) may be specified only with factors and residuals.

Remarks and examples

We assume that you have already read [TS] dfactor. In this entry, we illustrate some of the features of predict after using dfactor.

dfactor writes the specified model as a state-space model and estimates the parameters by maximum likelihood. The unobserved factors and the residuals are states in the state-space form of the model, and they are estimated by the Kalman filter or the Kalman smoother. The smethod() option controls how these states are estimated.

The Kalman filter or Kalman smoother is run over the specified sample. Changing the sample can alter the predicted value for a given observation, because the Kalman filter and Kalman smoother are recursive algorithms.

After estimating the parameters of a dynamic-factor model, there are many quantities of potential interest. Here we will discuss several of these statistics and illustrate how to use predict to compute them.

Example 1: One-step, out-of-sample forecasts

Let’s begin by estimating the parameters of the dynamic-factor model considered in example 2 in [TS] dfactor.
While several of the six statistics computed by `predict` might be of interest, we will look only at a few of these statistics for `D.ipman`. We begin by obtaining one-step predictions in the estimation sample and a six-month dynamic forecast for `D.ipman`. The graph of the in-sample predictions indicates that our model accounts only for a small fraction of the variability in `D.ipman`.

```stata
.tsappend, add(6)
.predict Dipman_f, dynamic(tm(2008m12)) equation(D.ipman)
(option y assumed; fitted values)
.tsline D.ipman Dipman_f if month<=tm(2008m11), lcolor(gs13) xtitle("") legend(rows(2))
```

Graphing the last year of the sample and the six-month out-of-sample forecast yields

```stata
.tsline D.ipman Dipman_f if month>=tm(2008m1), xtitle("") legend(rows(2))
```

![Graph of in-sample predictions](image1)

![Graph of last year of sample and six-month out-of-sample forecast](image2)
Example 2: Estimating an unobserved factor

Another common task is to estimate an unobserved factor. We can estimate the unobserved factor at each time period by using only previous information (the smethod(onestep) option), previous and contemporaneous information (the smethod(filter) option), or all the sample information (the smethod(smooth) option). We are interested in the one-step predictive power of the unobserved factor, so we use the default, smethod(onestep).

```
.predict fac if e(sample), factor
.tsline D.ipman fac, lcolor(gs10) xtitle("") legend(rows(2))
```

Methods and formulas

dfactor estimates the parameters by writing the model in state-space form and using sspace. Analogously, predict after dfactor uses the methods described in [TS] sspace postestimation. The unobserved factors and the residuals are states in the state-space form of the model.

See Methods and formulas of [TS] sspace postestimation for how predictions are made after estimating the parameters of a state-space model.

Also see

[TS] dfactor — Dynamic-factor models
[TS] sspace — State-space models
[TS] sspace postestimation — Postestimation tools for sspace
[U] 20 Estimation and postestimation commands
**Description**

`dfgls` performs a modified Dickey–Fuller $t$ test for a unit root in which the series has been transformed by a generalized least-squares regression.

**Quick start**

Modified Dickey–Fuller unit-root test for $y_1$ using GLS-transformed series using `tsset` data

\ `<dfgls y1>`

As above, for series $y_2$ that has no linear time trend

\ `<dfgls y2, notrend>`

As above, but with at most 2 lags

\ `<dfgls y2, notrend maxlag(2)>`

**Menu**

Statistics $>$ Time series $>$ Tests $>$ DF-GLS test for a unit root
**Syntax**

```stata
dfgls varname [if] [in] [ , options ]
```

**options**

<table>
<thead>
<tr>
<th>Main</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxlag(#)</td>
<td>use # as the highest lag order for Dickey–Fuller GLS regressions</td>
</tr>
<tr>
<td>notrend</td>
<td>series is stationary around a mean instead of around a linear time trend</td>
</tr>
<tr>
<td>ers</td>
<td>present interpolated critical values from Elliott, Rothenberg, and Stock (1996)</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `dfgls`; see [TS] `tsset`.

`varname` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

**Options**

- `maxlag(#)`: sets the value of $k$, the highest lag order for the first-differenced, detrended variable in the Dickey–Fuller regression. By default, `dfgls` sets $k$ according to the method proposed by Schwert (1989); that is, `dfgls` sets $k_{max} = \text{floor}[12\{(T + 1)/100\}]^{0.25}$.

- `notrend`: specifies that the alternative hypothesis be that the series is stationary around a mean instead of around a linear time trend. By default, a trend is included.

- `ers`: specifies that `dfgls` should present interpolated critical values from tables presented by Elliott, Rothenberg, and Stock (1996), which they obtained from simulations. See Critical values under Methods and formulas for details.

**Remarks and examples**

`dfgls` tests for a unit root in a time series. It performs the modified Dickey–Fuller $t$ test (known as the DF-GLS test) proposed by Elliott, Rothenberg, and Stock (1996). Essentially, the test is an augmented Dickey–Fuller test, similar to the test performed by Stata’s `dfuller` command, except that the time series is transformed via a generalized least squares (GLS) regression before performing the test. Elliott, Rothenberg, and Stock and later studies have shown that this test has significantly greater power than the previous versions of the augmented Dickey–Fuller test.

`dfgls` performs the DF-GLS test for the series of models that include 1 to $k$ lags of the first-differenced, detrended variable, where $k$ can be set by the user or by the method described in Schwert (1989). Stock and Watson (2019, 622–624) provide an excellent discussion of the approach.

As discussed in [TS] `dfuller`, the augmented Dickey–Fuller test involves fitting a regression of the form

$$
\Delta y_t = \alpha + \beta y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \cdots + \zeta_k \Delta y_{t-k} + \epsilon_t
$$

and then testing the null hypothesis $H_0: \beta = 0$. The DF-GLS test is performed analogously but on GLS-detrended data. The null hypothesis of the test is that $y_t$ is a random walk, possibly with drift. There are two possible alternative hypotheses: $y_t$ is stationary about a linear time trend or $y_t$ is stationary with a possibly nonzero mean but with no linear time trend. The default is to use the former. To specify the latter alternative, use the `notrend` option.
Example 1

Here we use the German macroeconomic dataset and test whether the natural log of investment exhibits a unit root. We use the default options with \texttt{dfgls}.

```stata
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. dfgls ln_inv
DF-GLS for ln_inv Number of obs = 80
Maxlag = 11 chosen by Schwert criterion

<table>
<thead>
<tr>
<th>[lags]</th>
<th>DF-GLS tau</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>-2.925</td>
<td>-3.610</td>
<td>-2.763</td>
<td>-2.489</td>
</tr>
<tr>
<td>10</td>
<td>-2.671</td>
<td>-3.610</td>
<td>-2.798</td>
<td>-2.523</td>
</tr>
<tr>
<td>9</td>
<td>-2.766</td>
<td>-3.610</td>
<td>-2.832</td>
<td>-2.555</td>
</tr>
<tr>
<td>8</td>
<td>-3.259</td>
<td>-3.610</td>
<td>-2.865</td>
<td>-2.587</td>
</tr>
<tr>
<td>7</td>
<td>-3.536</td>
<td>-3.610</td>
<td>-2.905</td>
<td>-2.617</td>
</tr>
<tr>
<td>6</td>
<td>-3.115</td>
<td>-3.610</td>
<td>-2.909</td>
<td>-2.646</td>
</tr>
<tr>
<td>5</td>
<td>-3.054</td>
<td>-3.610</td>
<td>-2.958</td>
<td>-2.674</td>
</tr>
<tr>
<td>3</td>
<td>-2.071</td>
<td>-3.610</td>
<td>-3.012</td>
<td>-2.723</td>
</tr>
<tr>
<td>2</td>
<td>-1.675</td>
<td>-3.610</td>
<td>-3.035</td>
<td>-2.744</td>
</tr>
<tr>
<td>1</td>
<td>-1.752</td>
<td>-3.610</td>
<td>-3.055</td>
<td>-2.762</td>
</tr>
</tbody>
</table>

Opt Lag (Ng-Perron seq t) = 7 with RMSE .0388771
Min SC = -6.169137 at lag 4 with RMSE .0398949
Min MAIC = -6.136371 at lag 1 with RMSE .0440319
```

The null hypothesis of a unit root is not rejected for lags 1–3, it is rejected at the 10% level for lags 9–10, and it is rejected at the 5% level for lags 4–8 and 11. For comparison, we also test for a unit root in log of investment by using \texttt{dfuller} with two different lag specifications. We need to use the \texttt{trend} option with \texttt{dfuller} because it is not included by default.
Augmented Dickey-Fuller test for unit root

<table>
<thead>
<tr>
<th>Test Statistic</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z(t)</td>
<td>-3.133</td>
<td>-4.069</td>
<td>-3.463</td>
</tr>
</tbody>
</table>

MacKinnon approximate p-value for Z(t) = 0.0987

Augmented Dickey-Fuller test for unit root

<table>
<thead>
<tr>
<th>Test Statistic</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z(t)</td>
<td>-3.994</td>
<td>-4.075</td>
<td>-3.466</td>
</tr>
</tbody>
</table>

MacKinnon approximate p-value for Z(t) = 0.0090

The critical values and the test statistic produced by `dfuller` with 4 lags do not support rejecting the null hypothesis, although the MacKinnon approximate p-value is less than 0.1. With 7 lags, the critical values and the test statistic reject the null hypothesis at the 5% level, and the MacKinnon approximate p-value is less than 0.01.

That the `dfuller` results are not as strong as those produced by `dfgls` is not surprising because the DF-GLS test with a trend has been shown to be more powerful than the standard augmented Dickey–Fuller test.

Stored results

If `maxlag(0)` is specified, `dfgls` stores the following in `r()`:

Scalars
- `r(rmse0)`: RMSE
- `r(dft0)`: DF-GLS statistic

Otherwise, `dfgls` stores the following in `r()`:

Scalars
- `r(maxlag)`: highest lag order $k$
- `r(N)`: number of observations
- `r(sclag)`: lag chosen by Schwarz criterion
- `r(maiclag)`: lag chosen by modified AIC method
- `r(optlag)`: lag chosen by sequential-t method

Matrices
- `r(results)`: $k$, MAIC, SIC, RMSE, and DF-GLS statistics

Methods and formulas

`dfgls` tests for a unit root. There are two possible alternative hypotheses: $y_t$ is stationary around a linear trend or $y_t$ is stationary with no linear time trend. Under the first alternative hypothesis, the DF-GLS test is performed by first estimating the intercept and trend via GLS. The GLS estimation is performed by generating the new variables, $\tilde{y}_t$, $x_t$, and $z_t$, where
\begin{equation}
\tilde{y}_1 = y_1 \\
\tilde{y}_t = y_t - \alpha^* y_{t-1}, \quad t = 2, \ldots, T \\
x_1 = 1 \\
x_t = 1 - \alpha^*, \quad t = 2, \ldots, T \\
z_1 = 1 \\
z_t = t - \alpha^*(t-1)
\end{equation}

and \(\alpha^* = 1 - (13.5/T)\). An OLS regression is then estimated for the equation

\[ \tilde{y}_t = \delta_0 x_t + \delta_1 z_t + \epsilon_t \]

The OLS estimators \(\hat{\delta}_0\) and \(\hat{\delta}_1\) are then used to remove the trend from \(y_t\); that is, we generate

\[ y^* = y_t - (\hat{\delta}_0 + \hat{\delta}_1 t) \]

Finally, we perform an augmented Dickey–Fuller test on the transformed variable by fitting the OLS regression

\[ \Delta y^*_t = \alpha + \beta y^*_{t-1} + \sum_{j=1}^{k} \zeta_j \Delta y^*_{t-j} + \epsilon_t \] (1)

and then test the null hypothesis \(H_0: \beta = 0\) by using tabulated critical values.

To perform the DF-GLS test under the second alternative hypothesis, we proceed as before but define \(\alpha^* = 1 - (7/T)\), eliminate \(z\) from the GLS regression, compute \(y^* = y_t - \delta_0\), fit the augmented Dickey–Fuller regression by using the newly transformed variable, and perform a test of the null hypothesis that \(\beta = 0\) by using the tabulated critical values.

dfgls reports the DF-GLS statistic and its critical values obtained from the regression in (1) for \(k \in \{1, 2, \ldots, k_{\text{max}}\}\). By default, dfgls sets \(k_{\text{max}} = \text{floor}[12\{\frac{T+1}{100}\}^{0.25}]\) as proposed by Schwert (1989), although you can override this choice with another value. The sample size available with \(k_{\text{max}}\) lags is used in all the regressions. Because there are \(k_{\text{max}}\) lags of the first-differenced series, \(k_{\text{max}} + 1\) observations are lost, leaving \(T - k_{\text{max}}\) observations. dfgls requires that the sample of \(T + 1\) observations on \(y_t = (y_0, y_1, \ldots, y_T)\) have no gaps.

dfgls reports the results of three different methods for choosing which value of \(k\) to use. These are method 1, the Ng–Perron sequential \(t\); method 2, the minimum Schwarz information criterion (SIC); and method 3, the Ng–Perron modified Akaike information criterion (MAIC). Although the SIC has a long history in time-series modeling, the Ng–Perron sequential \(t\) was developed by Ng and Perron (1995), and the MAIC was developed by Ng and Perron (2000).

The SIC can be calculated using either the log likelihood or the sum-of-squared errors from a regression; dfgls uses the latter definition. Specifically, for each \(k\)

\[ \text{SIC} = \ln(\hat{\text{rmse}}^2) + (k + 1) \frac{\ln(T - k_{\text{max}})}{(T - k_{\text{max}})} \]
where
\[
\hat{\text{rmse}} = \frac{1}{(T - k_{\text{max}})} \sum_{t=k_{\text{max}}+1}^{T} \hat{e}_t^2
\]
dfgls reports the value of the smallest SIC and the \( k \) that produced it.

Ng and Perron (1995) derived a sequential-\( t \) algorithm for choosing \( k \):

i. Set \( n = 0 \) and run the regression in method 2 with all \( k_{\text{max}} - n \) lags. If the coefficient on \( \beta_{k_{\text{max}}} \) is significantly different from zero at level \( \alpha \), choose \( k \) to \( k_{\text{max}} \). Otherwise, continue to ii.

ii. If \( n < k_{\text{max}} \), set \( n = n + 1 \) and continue to iii. Otherwise, set \( k = 0 \) and stop.

iii. Run the regression in method 2 with \( k_{\text{max}} - n \) lags. If the coefficient on \( \beta_{k_{\text{max}}-n} \) is significantly different from zero at level \( \alpha \), choose \( k \) to \( k_{\text{max}} - n \). Otherwise, return to ii.

Per Ng and Perron (1995), dfgls uses \( \alpha = 10\% \). dfgls reports the \( k \) selected by this sequential-\( t \) algorithm and the \( \hat{\text{rmse}} \) from the regression.

Method 3 is based on choosing \( k \) to minimize the MAIC. The MAIC is calculated as
\[
\text{MAIC}(k) = \ln(\hat{\text{rmse}}^2) + \frac{2\{\tau(k) + k\}}{T - k_{\text{max}}}
\]
where
\[
\tau(k) = \frac{1}{\hat{\text{rmse}}^2} \hat{\beta}_0^2 \sum_{t=k_{\text{max}}+1}^{T} \tilde{y}_t^2
\]
and \( \tilde{y} \) was defined previously.

Critical values

By default, dfgls uses the 5\% and 10\% critical values computed from the response surface analysis of Cheung and Lai (1995). Because Cheung and Lai (1995) did not present results for the 1\% case, the 1\% critical values are always interpolated from the critical values presented by ERS.

ERS presented critical values, obtained from simulations, for the DF-GLS test with a linear trend and showed that the critical values for the mean-only DF-GLS test were the same as those for the ADF test. If dfgls is run with the \texttt{ers} option, dfgls will present interpolated critical values from these tables. The method of interpolation is standard. For the trend case, below 50 observations and above 200 there is no interpolation; the values for 50 and \( \infty \) are reported from the tables. For a value \( N \) that lies between two values in the table, say, \( N_1 \) and \( N_2 \), with corresponding critical values \( CV_1 \) and \( CV_2 \), the critical value
\[
CV = CV_1 + \frac{N - N_1}{N_1}(CV_2 - CV_1)
\]
is presented. The same method is used for the mean-only case, except that interpolation is possible for values between 50 and 500.
Acknowledgments

We thank Christopher F. Baum of the Department of Economics at Boston College and author of the Stata Press books *An Introduction to Modern Econometrics Using Stata* and *An Introduction to Stata Programming* and the late Richard Sperling for a previous version of dfgls.

References


Also see

[TS] dfuller — Augmented Dickey–Fuller unit-root test

[TS] pperron — Phillips–Perron unit-root test

[TS] tsset — Declare data to be time-series data

[XT] xtunitroot — Panel-data unit-root tests
**dfuller** performs the augmented Dickey–Fuller test that a variable follows a unit-root process. The null hypothesis is that the variable contains a unit root, and the alternative is that the variable was generated by a stationary process. You may optionally exclude the constant, include a trend term, and include lagged values of the difference of the variable in the regression.

### Quick start

Augmented Dickey–Fuller test for presence of a unit root in \( y \) using \texttt{tsset} data

\[
\texttt{dfuller} \ y
\]

As above, but with a trend term

\[
\texttt{dfuller} \ y, \texttt{trend}
\]

Augmented Dickey–Fuller test for presence of a unit root in \( y \) with a drift term

\[
\texttt{dfuller} \ y, \texttt{drift}
\]

As above, but include 3 lagged differences and display the regression table

\[
\texttt{dfuller} \ y, \texttt{drift} \ \texttt{lags(3)} \ \texttt{regress}
\]

### Menu

Statistics > Time series > Tests > Augmented Dickey-Fuller unit-root test
Syntax

\[
dfuller \ varname \ [\ if] \ [\ in] \ [\ , \ options]\]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noconstant</td>
<td>suppress constant term in regression</td>
</tr>
<tr>
<td>trend</td>
<td>include trend term in regression</td>
</tr>
<tr>
<td>drift</td>
<td>include drift term in regression</td>
</tr>
<tr>
<td>regress</td>
<td>display regression table</td>
</tr>
<tr>
<td>lags(#)</td>
<td>include # lagged differences</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `dfuller`; see [TS] `tsset`.
`varname` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

**Main**

- `noconstant` suppresses the constant term (intercept) in the model and indicates that the process under the null hypothesis is a random walk without drift. `noconstant` cannot be used with the `trend` or `drift` option.

- `trend` specifies that a trend term be included in the associated regression and that the process under the null hypothesis is a random walk, perhaps with drift. This option may not be used with the `noconstant` or `drift` option.

- `drift` indicates that the process under the null hypothesis is a random walk with nonzero drift. This option may not be used with the `noconstant` or `trend` option.

- `regress` specifies that the associated regression table appear in the output. By default, the regression table is not produced.

- `lags(#)` specifies the number of lagged difference terms to include in the covariate list.

Remarks and examples

Dickey and Fuller (1979) developed a procedure for testing whether a variable has a unit root or, equivalently, that the variable follows a random walk. Hamilton (1994, 528–529) describes the four different cases to which the augmented Dickey–Fuller test can be applied. The null hypothesis is always that the variable has a unit root. They differ in whether the null hypothesis includes a drift term and whether the regression used to obtain the test statistic includes a constant term and time trend. Becketti (2013, chap. 9) provides additional examples showing how to conduct these tests.

The true model is assumed to be

\[
y_t = \alpha + y_{t-1} + u_t
\]

where \(u_t\) is an independent and identically distributed zero-mean error term. In cases one and two, presumably \(\alpha = 0\), which is a random walk without drift. In cases three and four, we allow for a drift term by letting \(\alpha\) be unrestricted.
The Dickey–Fuller test involves fitting the model

\[ y_t = \alpha + \rho y_{t-1} + \delta t + u_t \]

by ordinary least squares (OLS), perhaps setting \( \alpha = 0 \) or \( \delta = 0 \). However, such a regression is likely to be plagued by serial correlation. To control for that, the augmented Dickey–Fuller test instead fits a model of the form

\[ \Delta y_t = \alpha + \beta y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \cdots + \zeta_k \Delta y_{t-k} + \epsilon_t \]

(1)

where \( k \) is the number of lags specified in the \texttt{lags()} option. The \texttt{noconstant} option removes the constant term \( \alpha \) from this regression, and the \texttt{trend} option includes the time trend \( \delta t \), which by default is not included. Testing \( \beta = 0 \) is equivalent to testing \( \rho = 1 \), or, equivalently, that \( y_t \) follows a unit root process.

In the first case, the null hypothesis is that \( y_t \) follows a random walk without drift, and (1) is fit without the constant term \( \alpha \) and the time trend \( \delta t \). The second case has the same null hypothesis as the first, except that we include \( \alpha \) in the regression. In both cases, the population value of \( \alpha \) is zero under the null hypothesis. In the third case, we hypothesize that \( y_t \) follows a unit root with drift, so that the population value of \( \alpha \) is nonzero; we do not include the time trend in the regression. Finally, in the fourth case, the null hypothesis is that \( y_t \) follows a unit root with or without drift so that \( \alpha \) is unrestricted, and we include a time trend in the regression.

The following table summarizes the four cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>Process under null hypothesis</th>
<th>Regression restrictions</th>
<th>dfuller option</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Random walk without drift</td>
<td>( \alpha = 0, \delta = 0 )</td>
<td>noconstant</td>
</tr>
<tr>
<td>2</td>
<td>Random walk without drift</td>
<td>( \delta = 0 )</td>
<td>(default)</td>
</tr>
<tr>
<td>3</td>
<td>Random walk with drift</td>
<td>( \delta = 0 )</td>
<td>drift</td>
</tr>
<tr>
<td>4</td>
<td>Random walk with or without drift</td>
<td>(none)</td>
<td>trend</td>
</tr>
</tbody>
</table>

Except in the third case, the \( t \)-statistic used to test \( H_0: \beta = 0 \) does not have a standard distribution. Hamilton (1994, chap. 17) derives the limiting distributions, which are different for each of the three other cases. The critical values reported by \texttt{dfuller} are interpolated based on the tables in Fuller (1996). MacKinnon (1994) shows how to approximate the \( p \)-values on the basis of a regression surface, and \texttt{dfuller} also reports that \( p \)-value. In the third case, where the regression includes a constant term and under the null hypothesis the series has a nonzero drift parameter \( \alpha \), the \( t \) statistic has the usual \( t \) distribution; \texttt{dfuller} reports the one-sided critical values and \( p \)-value for the test of \( H_0 \) against the alternative \( H_a: \beta < 0 \), which is equivalent to \( \rho < 1 \).

Deciding which case to use involves a combination of theory and visual inspection of the data. If economic theory favors a particular null hypothesis, the appropriate case can be chosen based on that. If a graph of the data shows an upward trend over time, then case four may be preferred. If the data do not show a trend but do have a nonzero mean, then case two would be a valid alternative.

> Example 1

In this example, we examine the international airline passengers dataset from Box, Jenkins, and Reinsel (2008, Series G). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. Because the data show a clear upward trend, we use the \texttt{trend} option with \texttt{dfuller} to include a constant and time trend in the augmented Dickey–Fuller regression.
. use https://www.stata-press.com/data/r16/air2
(TIMESLAB: Airline passengers)
. dfuller air, lags(3) trend regress

Augmented Dickey-Fuller test for unit root
Number of obs = 140

<table>
<thead>
<tr>
<th></th>
<th>Interpolated Dickey-Fuller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>1% Critical Value</td>
</tr>
<tr>
<td>Test</td>
<td>Value</td>
</tr>
<tr>
<td>Z(t)</td>
<td>-6.936</td>
</tr>
</tbody>
</table>

MacKinnon approximate p-value for Z(t) = 0.000

| D.air     | Coef.  | Std. Err. | t     | P>|t| | [95% Conf. Interval] |
|-----------|--------|-----------|-------|-----|---------------------|
| air       | -0.521 | 0.075      | -6.94 | 0.000 | -0.67048            | -0.3729379 |
| L1.       | 0.557  | 0.079      | 6.97  | 0.000 | 0.399082            | 0.7154923  |
| LD.       | 0.096  | 0.088      | 1.09  | 0.276 | -0.0774825          | 0.2693065  |
| L2D.      | 0.145  | 0.088      | 1.65  | 0.101 | -0.0289232          | 0.3191433  |
| L3D.      | 1.407  | 0.209      | 6.71  | 0.000 | 0.9925118           | 1.822557   |
| _trend    | 44.491 | 7.783      | 5.72  | 0.000 | 29.09753            | 59.88575   |

Here we can overwhelmingly reject the null hypothesis of a unit root at all common significance levels. From the regression output, the estimated \( \beta \) of \(-0.522\) implies that \( \rho = (1 - 0.522) = 0.478 \). Experiments with fewer or more lags in the augmented regression yield the same conclusion.

Example 2

In this example, we use the German macroeconomic dataset to determine whether the log of consumption follows a unit root. We will again use the `trend` option, because consumption grows over time.
dfuller — Augmented Dickey–Fuller unit-root test

. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. tsset qtr
time variable: qtr, 1960q1 to 1982q4
delta: 1 quarter
. dfuller ln_consump, lags(4) trend
Augmented Dickey–Fuller test for unit root
Number of obs = 87

<table>
<thead>
<tr>
<th>Test Statistic</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z(t)</td>
<td>-1.318</td>
<td>-4.069</td>
<td>-3.463</td>
</tr>
</tbody>
</table>

MacKinnon approximate p-value for Z(t) = 0.8834

As we might expect from economic theory, here we cannot reject the null hypothesis that log consumption exhibits a unit root. Again using different numbers of lag terms yield the same conclusion.

Stored results
dfuller stores the following in r():

Scalars:
r(N) number of observations
r(lags) number of lagged differences
r(Zt) Dickey–Fuller test statistic
r(cv1) 1% critical value
r(cv5) 5% critical value
r(cv10) 10% critical value
r(p) MacKinnon approximate p-value (if there is a constant or trend in associated regression)

Methods and formulas

In the OLS estimation of an AR(1) process with Gaussian errors,

\[ y_t = \rho y_{t-1} + \epsilon_t \]

where \( \epsilon_t \) are independent and identically distributed as \( N(0, \sigma^2) \) and \( y_0 = 0 \), the OLS estimate (based on an \( n \)-observation time series) of the autocorrelation parameter \( \rho \) is given by

\[ \hat{\rho}_n = \frac{\sum_{t=1}^{n} y_{t-1} y_t}{\sum_{t=1}^{n} y_t^2} \]

If \( |\rho| < 1 \), then

\[ \sqrt{n}(\hat{\rho}_n - \rho) \to N(0, 1 - \rho^2) \]

If this result were valid when \( \rho = 1 \), the resulting distribution would have a variance of zero. When \( \rho = 1 \), the OLS estimate \( \hat{\rho} \) still converges in probability to one, though we need to find a suitable nondegenerate distribution so that we can perform hypothesis tests of \( H_0: \rho = 1 \). Hamilton (1994, chap. 17) provides a superb exposition of the requisite theory.
To compute the test statistics, we fit the augmented Dickey–Fuller regression

$$\Delta y_t = \alpha + \beta y_{t-1} + \delta t + \sum_{j=1}^{k} \zeta_j \Delta y_{t-j} + e_t$$

via OLS where, depending on the options specified, the constant term $\alpha$ or time trend $\delta t$ is omitted and $k$ is the number of lags specified in the $\text{lags()}$ option. The test statistic for $H_0: \beta = 0$ is $Z_t = \hat{\beta}/\hat{\sigma}_\beta$, where $\hat{\sigma}_\beta$ is the standard error of $\hat{\beta}$.

The critical values included in the output are linearly interpolated from the table of values that appears in Fuller (1996), and the MacKinnon approximate $p$-values use the regression surface published in MacKinnon (1994).

David Alan Dickey (1945– ) was born in Ohio and obtained degrees in mathematics at Miami University and a PhD in statistics at Iowa State University in 1976 as a student of Wayne Fuller. He works at North Carolina State University and specializes in time-series analysis.

Wayne Arthur Fuller (1931– ) was born in Iowa, obtained three degrees at Iowa State University and then served on the faculty between 1959 and 2001. He has made many distinguished contributions to time series, measurement-error models, survey sampling, and econometrics.

References

Becketti, S. 2013. *Introduction to Time Series Using Stata*. College Station, TX: Stata Press.


Also see

[TS] *dfgls* — DF-GLS unit-root test

[TS] *pperron* — Phillips–Perron unit-root test

[TS] *tsset* — Declare data to be time-series data

[XT] *xtunitroot* — Panel-data unit-root tests
estat acplot — Plot parametric autocorrelation and autocovariance functions

Description

estat acplot plots the estimated autocorrelation and autocovariance functions of a stationary process using the parameters of a previously fit parametric model.

estat acplot is available after arima and arfima; see [TS] arima and [TS] arfima.

Quick start

Autocorrelation function using estimates from arima or arfima
estat acplot

Autocovariance function using estimates from arima or arfima
estat acplot, covariance

As above, and save results in mydata.dta
estat acplot, covariance saving(mydata)

Menu for estat

Statistics  >  Postestimation
Syntax

estat acplot [, options]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saving(filename[, ...])</td>
<td>save results to filename; save variables in double precision;</td>
</tr>
<tr>
<td></td>
<td>save variables with prefix stubname</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>lags(#)</td>
<td>use # autocorrelations</td>
</tr>
<tr>
<td>covariance</td>
<td>calculate autocovariances; the default is to calculate autocorrelations</td>
</tr>
<tr>
<td>smemory</td>
<td>report short-memory ACF; only allowed after arfima</td>
</tr>
</tbody>
</table>

CI plot

ciopts(rcap_options) affects rendition of the confidence bands

Plot

marker_options            | change look of markers (color, size, etc.)                                 |
marker_label_options      | add marker labels; change look or position                                 |
cline_options             | affect rendition of the plotted points                                     |

Y axis, X axis, Titles, Legend, Overall
twoway_options           | any options other than by() documented in [G-3] twoway_options             |

Options

saving(filename[, suboptions]) creates a Stata data file (.dta file) consisting of the autocorrelation estimates, standard errors, and confidence bounds.

Five variables are saved: lag (lag number), ac (autocorrelation estimate), se (standard error), ci_l (lower confidence bound), and ci_u (upper confidence bound).

double specifies that the variables be saved as doubles, meaning 8-byte reals. By default, they are saved as floats, meaning 4-byte reals.

name(stubname) specifies that variables be saved with prefix stubname.

replace indicates that filename be overwritten if it exists.

level(#) specifies the confidence level, as a percentage, for confidence intervals. The default is level(95) or as set by set level; see [R] level.

lags(#) specifies the number of autocorrelations to calculate. The default is to use min{floor(n/2) − 2, 40}, where floor(n/2) is the greatest integer less than or equal to n/2 and n is the number of observations.

covariance specifies the calculation of autocovariances instead of the default autocorrelations.

smemory specifies that the ARFIMA fractional integration parameter be ignored. The computed autocorrelations are for the short-memory ARMA component of the model. This option is allowed only after arfima.

ciopts(rcap_options) affects the rendition of the confidence bands; see [G-3] rcap_options.
Plot marker options affect the rendition of markers drawn at the plotted points, including their shape, size, color, and outline; see [G-3] marker_options.

marker_label_options specify if and how the markers are to be labeled; see [G-3] marker_label_options.

cline_options affect whether lines connect the plotted points and the rendition of those lines; see [G-3] cline_options.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] twoway_options, except by(). These include options for titling the graph (see [G-3] title_options) and options for saving the graph to disk (see [G-3] saving_option).

Remarks and examples

The dependent variable evolves over time because of random shocks in the time domain representation. The autocovariances $\gamma_j, j \in \{0, 1, \ldots, \infty\}$, of a covariance-stationary process $y_t$ specify its variance and dependence structure, and the autocorrelations $\rho_j, j \in \{1, 2, \ldots, \infty\}$, provide a scale-free measure of $y_t$’s dependence structure. The autocorrelation at lag $j$ specifies whether realizations at time $t$ and realizations at time $t - j$ are positively related, unrelated, or negatively related. estat acplot uses the estimated parameters of a parametric model to estimate and plot the autocorrelations and autocovariances of a stationary process.
Example 1

In example 1 of [TS] arima, we fit an ARIMA(1,1,1) model of the U.S. Wholesale Price Index (WPI) using quarterly data over the period 1960q1 through 1990q4.

```
  . use https://www.stata-press.com/data/r16/wpi1
  . arima wpi, arima(1,1,1)
     (setting optimization to BHHH)
  Iteration 0: log likelihood = -139.80133
  Iteration 1: log likelihood = -135.6278
  Iteration 2: log likelihood = -135.41838
  Iteration 3: log likelihood = -135.36691
  Iteration 4: log likelihood = -135.35892
     (switching optimization to BFGS)
  Iteration 5: log likelihood = -135.35471
  Iteration 6: log likelihood = -135.35135
  Iteration 7: log likelihood = -135.35132
  Iteration 8: log likelihood = -135.35131

ARIMA regression
Sample: 1960q2 - 1990q4 Number of obs = 123
Wald chi2(2) = 310.64
Log likelihood = -135.3513 Prob > chi2 = 0.0000

                      D.wpi                      OPG                      
               Coef.     Std. Err.     z    P>|z|     [95% Conf. Interval]
---------------------------------------------------------------
     wpi     _cons      .7498197     .3340968     2.24  0.025     .0950019    1.404637

               ARMA                      OPG                      
                      Coef.     Std. Err.     z    P>|z|     [95% Conf. Interval]
---------------------------------------------------------------
               ar L1.        .8742288     .0545435    16.03  0.000      .7673256    .981132
               ma L1.     -.4120458     .1000284    -4.12  0.000     -.6080979   -.2159938
   /sigma      .7250436     .0368065    19.70  0.000      .6529042    .7971829
```

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

Now we use estat acplot to estimate the autocorrelations implied by the estimated ARMA parameters. We include lags(50) to indicate that autocorrelations be computed for 50 lags. By default, a 95% confidence interval is provided for each autocorrelation.
. estat acplot, lags(50)

The graph is similar to a typical autocorrelation function of an AR(1) process with a positive coefficient. The autocorrelations of a stationary AR(1) process decay exponentially toward zero.

Methods and formulas

The autocovariance function for ARFIMA models is described in Methods and formulas of [TS] arfima. The autocovariance function for ARIMA models is obtained by setting the fractional difference parameter to zero.

Box, Jenkins, and Reinsel (2008) provide excellent descriptions of the autocovariance function for ARIMA and seasonal ARIMA models. Palma (2007) provides an excellent summary of the autocovariance function for ARFIMA models.

References


Also see

[TS] arfima — Autoregressive fractionally integrated moving-average models

[TS] arima — ARIMA, ARMAX, and other dynamic regression models
estat aroots — Check the stability condition of ARIMA estimates

Description
estat aroots checks the eigenvalue stability condition after estimating the parameters of an ARIMA model using arima. A graph of the eigenvalues of the companion matrices for the AR and MA polynomials is also produced.

estat aroots is available only after arima; see [TS] arima.

Quick start
Verify that all eigenvalues of the autoregressive polynomial lie inside the unit circle after arima
estat aroots

As above, but suppress the graph
estat aroots, nograph

Label each plotted eigenvalue with its distance from the unit circle
estat aroots, dlabel

Menu for estat
Statistics > Postestimation
Syntax

```
estat aroots [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nograph</td>
<td>suppress graph of eigenvalues for the companion matrices</td>
</tr>
<tr>
<td>dlabel</td>
<td>label eigenvalues with the distance from the unit circle</td>
</tr>
<tr>
<td>modlabel</td>
<td>label eigenvalues with the modulus</td>
</tr>
</tbody>
</table>

**Grid**

- **nogrid** suppresses the polar grid circles.
- **pgrid(numlist)** specifies radii and appearance of polar grid circles; see [Options] for details.

**Plot**

- **marker_options** change look of markers (color, size, etc.)

**Reference unit circle**

- **rlopts(cline_options)** affect rendition of reference unit circle

**Y axis, X axis, Titles, Legend, Overall**

- **twoway_options** any options other than by() documented in [G-3] `twoway_options`

---

**Options**

- **nograph** specifies that no graph of the eigenvalues of the companion matrices be drawn.
- **dlabel** labels each eigenvalue with its distance from the unit circle. **dlabel** cannot be specified with **modlabel**.
- **modlabel** labels the eigenvalues with their moduli. **modlabel** cannot be specified with **dlabel**.

**Grid**

- **nogrid** suppresses the polar grid circles.

**Plot**

- **marker_options** specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] `marker_options`.

**Reference unit circle**

- **rlopts(cline_options)** affect the rendition of the reference unit circle; see [G-3] `cline_options`.
Remarks and examples

Inference after \texttt{arima} requires that the variable $y_t$ be covariance stationary. The variable $y_t$ is covariance stationary if its first two moments exist and are time invariant. More explicitly, $y_t$ is covariance stationary if

1. $E(y_t)$ is finite and not a function of $t$;
2. $\text{Var}(y_t)$ is finite and independent of $t$; and
3. $\text{Cov}(y_t, y_s)$ is a finite function of $|t - s|$ but not of $t$ or $s$ alone.

The stationarity of an ARMA process depends on the autoregressive (AR) parameters. If the inverse roots of the AR polynomial all lie inside the unit circle, the process is stationary, invertible, and has an infinite-order moving-average (MA) representation. Hamilton (1994, chap. 1) shows that if the modulus of each eigenvalue of the matrix $F(\rho)$ is strictly less than 1, the estimated ARMA is stationary; see \textit{Methods and formulas} for the definition of the matrix $F(\rho)$.

The MA part of an ARMA process can be rewritten as an infinite-order AR process provided that the MA process is invertible. Hamilton (1994, chap. 1) shows that if the modulus of each eigenvalue of the matrix $F(\theta)$ is strictly less than 1, the estimated ARMA is invertible; see \textit{Methods and formulas} for the definition of the matrix $F(\theta)$.

\section*{Example 1}

In this example, we check the stability condition of the SARIMA model that we fit in example 3 of [TS] \texttt{arima}. We begin by reestimating the parameters of the model.

\begin{verbatim}
. use https://www.stata-press.com/data/r16/air2
   (TIMESLAB: Airline passengers)
. generate lnair = ln(air)
\end{verbatim}
. arima lnair, arima(0,1,1) sarima(0,1,1,12) noconstant

(setting optimization to BHHH)

Iteration 0:  log likelihood =  223.8437
Iteration 1:  log likelihood =  239.80405
Iteration 2:  log likelihood =  244.10265
Iteration 3:  log likelihood =  244.65895
Iteration 4:  log likelihood =  244.68945

(switching optimization to BFGS)
Iteration 5:  log likelihood =  244.69431
Iteration 6:  log likelihood =  244.69647
Iteration 7:  log likelihood =  244.69651
Iteration 8:  log likelihood =  244.69651

ARIMA regression
Sample:  14 - 144  Number of obs =  131
Log likelihood =  244.6965  Prob > chi2 =  0.0000

\[
\begin{array}{l}
\text{Wald chi2(2) = 84.53} \\
\text{Log likelihood = 244.6965} \\
\text{Prob > chi2 = 0.0000}
\end{array}
\]

| DS12.lnair | Coef. | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|------------|-------|-----------|-------|------|----------------------|
| ARMA       |       |           |       |      |                      |
| ma         | L1.   | -.4018324 | .0730307 | -5.50 | 0.000 | -.5449698 | -.2586949 |
| ARMA12     |       |           |       |      |                      |
| ma         | L1.   | -.5569342 | .0963129 | -5.78 | 0.000 | -.745704 | -.3681644 |
| /sigma     |       | .0367167  | .0020132 | 18.24 | 0.000 | .0327708 | .0406625 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

We can now use estat aroots to check the stability condition of the MA part of the model.

. estat aroots

Eigenvalue stability condition

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.824798 +  .4761974i</td>
<td>.952395</td>
</tr>
<tr>
<td>.824798 -  .4761974i</td>
<td>.952395</td>
</tr>
<tr>
<td>.9523947</td>
<td>.952395</td>
</tr>
<tr>
<td>-.824798 + .4761974i</td>
<td>.952395</td>
</tr>
<tr>
<td>-.824798 - .4761974i</td>
<td>.952395</td>
</tr>
<tr>
<td>-.4761974 + .824798i</td>
<td>.952395</td>
</tr>
<tr>
<td>-.4761974 - .824798i</td>
<td>.952395</td>
</tr>
<tr>
<td>2.776e-16  + .9523947i</td>
<td>.952395</td>
</tr>
<tr>
<td>2.776e-16  - .9523947i</td>
<td>.952395</td>
</tr>
<tr>
<td>.4761974 + .824798i</td>
<td>.952395</td>
</tr>
<tr>
<td>.4761974 - .824798i</td>
<td>.952395</td>
</tr>
<tr>
<td>-.9523947</td>
<td>.952395</td>
</tr>
<tr>
<td>.4018324</td>
<td>.401832</td>
</tr>
</tbody>
</table>

All the eigenvalues lie inside the unit circle.
MA parameters satisfy invertibility condition.
Because the modulus of each eigenvalue is strictly less than 1, the MA process is invertible and can be represented as an infinite-order AR process.

The graph produced by `estat aroots` displays the eigenvalues with the real components on the $x$ axis and the imaginary components on the $y$ axis. The graph indicates visually that these eigenvalues are just inside the unit circle.

### Stored results

`estat aroots` stores the following in `r()`:

<table>
<thead>
<tr>
<th>Matrices</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r(Re_ar)</code></td>
<td>real part of the eigenvalues of $F(\rho)$</td>
</tr>
<tr>
<td><code>r(Im_ar)</code></td>
<td>imaginary part of the eigenvalues of $F(\rho)$</td>
</tr>
<tr>
<td><code>r(Modulus_ar)</code></td>
<td>modulus of the eigenvalues of $F(\rho)$</td>
</tr>
<tr>
<td><code>r(ar)</code></td>
<td>$F(\rho)$, the AR companion matrix</td>
</tr>
<tr>
<td><code>r(Re_ma)</code></td>
<td>real part of the eigenvalues of $F(\theta)$</td>
</tr>
<tr>
<td><code>r(Im_ma)</code></td>
<td>imaginary part of the eigenvalues of $F(\theta)$</td>
</tr>
<tr>
<td><code>r(Modulus_ma)</code></td>
<td>modulus of the eigenvalues of $F(\theta)$</td>
</tr>
<tr>
<td><code>r(ma)</code></td>
<td>$F(\theta)$, the MA companion matrix</td>
</tr>
</tbody>
</table>

### Methods and formulas

Recall the general form of the ARMA model,

$$
\rho(L^p)(y_t - x_t\beta) = \theta(L^q)\epsilon_t
$$

where

$$
\rho(L^p) = 1 - \rho_1 L - \rho_2 L^2 - \cdots - \rho_p L^p
$$

$$
\theta(L^q) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q
$$

and $L^j y_t = y_{t-j}$.
estat aroots forms the companion matrix

\[
F(\gamma) = \begin{pmatrix}
\gamma_1 & \gamma_2 & \ldots & \gamma_{r-1} & \gamma_r \\
1 & 0 & \ldots & 0 & 0 \\
0 & 1 & \ldots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & 1 & 0
\end{pmatrix}
\]

where \( \gamma = \rho \) and \( r = p \) for the AR part of ARMA, and \( \gamma = -\theta \) and \( r = q \) for the MA part of ARMA. aroots obtains the eigenvalues of \( F \) by using matrix eigenvalues. The modulus of the complex eigenvalue \( r + ci \) is \( \sqrt{r^2 + c^2} \). As shown by Hamilton (1994, chap. 1), a process is stable and invertible if the modulus of each eigenvalue of \( F \) is strictly less than 1.

Reference

Also see
[TS] arima — ARIMA, ARMAX, and other dynamic regression models
Description

`estat sbcusum` performs a test of whether the coefficients in a time-series regression are stable over time. The test statistic is constructed from the cumulative sum of either the recursive residuals or the ordinary least-squares (OLS) residuals.

`estat sbcusum` requires that the current estimation results be from `regress`.

Quick start

Test stability of parameters based on the cumulative sum of recursive residuals, and plot the cumulative sum with 95% confidence bands

`estat sbcusum`

As above, but use OLS residuals

`estat sbcusum, ols`

Menu for estat

Statistics ➤ Postestimation
Syntax

`estat sbcusum [, options]`

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive</td>
<td>use cumulative sum of recursive residuals; the default</td>
</tr>
<tr>
<td>ols</td>
<td>use cumulative sum of OLS residuals</td>
</tr>
<tr>
<td>generate(newvar)</td>
<td>create newvar containing the cumulative sum of the residuals</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
</tbody>
</table>

Plot

- recast(plottype) plot cumulative sum statistic using plottype
- cline_options affect rendition of the line plotting the cumulative sum statistic

Confidence bands

- cbopts(area_options) affect rendition of the confidence bands

Add plots

- addplot(plot) add other plots to the graph of the cumulative sum

Y axis, X axis, Titles, Legend, Overall

- twoway_options any options other than by() documented in [G-3] twoway_options

You must `tsset` your data before using `estat sbcusum`; see [TS] tsset.

Options

- **recursive** computes the cumulative sum (cusum) test statistic and draws a cusum plot using the recursive residuals. This is the default.
- **ols** computes the cusum test statistic and draws a cusum plot using the OLS residuals.
- **generate(newvar)** creates a new variable containing the cusum of the residuals at each time period.
- **level(#)** specifies the default confidence level, as a percentage, for confidence bands, when they are reported. The default is level(95) or as set by `set level`; see [U] 20.8 Specifying the width of confidence intervals.

- **recast(plottype)** specifies that the cusum statistic be plotted using plottype. plottype may be scatter, line, connected, area, bar, spike, or dropline; see [G-2] graph twoway.

- **cline_options** affect the rendition of the plotted cusum statistic. cline_options are as described in [G-3] cline_options. When recast(plottype) is specified, the plot options are whatever is appropriate for the specified plottype.

- **cbopts(area_options)** affects the rendition of the confidence bands for the cusum statistic. area_options are as described in [G-3] area_options.
addplot(plot) allows adding more graph twoway plots to the graph; see [G-3] addplot_option.

twoway_options are any of the options documented in [G-3] twoway_options, except by(). These include options for titling the graph (see [G-3] title_options) and options for saving the graph to disk (see [G-3] saving_option).

Remarks and examples

Remarks are presented under the following headings:

Introduction

Cusum of recursive residuals

Cusum of OLS residuals

Introduction

estat sbcusum performs a test of parameter stability for linear regression coefficients. For instance, estat sbcusum can test for structural breaks due to changes in regression coefficients over time. The test can be performed using the cusum from recursive residuals (the default) or using the cusum from OLS residuals.

In addition to reporting a test statistic, estat sbcusum plots the cusum of residuals and corresponding confidence bands over the sample period.

Both cusum of residuals tests have the null hypothesis that all parameters are stable or constant over time, but these tests do not require a specific alternative hypothesis. We reject this null hypothesis based on the test statistic being larger than a critical value or based on the plotted cusum being outside the confidence bands.

In their simulation study, Ploberger and Krämer (1992) show that the cusum test based on recursive residuals has better power to detect parameter instability occurring early in the sample than the test based on OLS residuals. By contrast, the OLS cusum statistic has better power to detect parameter instabilities for breaks occurring later in the sample than the test based on recursive residuals.

Cusum of recursive residuals

The test statistic based on the cusum of recursive residuals was introduced in Brown, Durbin, and Evans (1975). Under the null hypothesis, the recursive residuals are shown to be independent and identically distributed as normal with 0 mean and constant variance. The cusum of the recursive residuals also has a mean of 0 under the null hypothesis. If the coefficients change after a certain time period, the plot of the recursive cusum process will drift away from the expected value of 0.

The limiting distribution of the sequence of the recursive cusum statistic is approximated by a Brownian motion. The bounds of the Brownian process at a specified significance level are approximated by a linear function. The sample cusum process crossing the theoretical bounds at any time point signifies rejection of the null hypothesis. This is easily identified from a graph that plots the recursive cusum statistic and the bounds. Alternatively, a test statistic based on the maximum of the recursive cusum statistic can be used to test the null hypothesis.
Example 1: Recursive cusum test

wpi1.dta contains quarterly data on the log of the U.S. Wholesale Price Index (WPI) from the first quarter of 1960 to the fourth quarter of 1990 (Enders 2004). We want to know whether the growth rate in the WPI has changed over time. To do this, we take the first differences of ln_wpi; see [U] 11.4.4 Time-series varlists.

The graph below shows a time-series plot of the growth rate of the WPI.

![Time-series plot of growth rate of WPI]

The growth rate changes in the middle of the sample, which leads us to believe that the mean of the series may not be stable during the sample period. Because we are only interested in the average growth rate of the series, we fit a constant-only linear regression model. We fit the model with the quietly command because we are not interested in the coefficient estimates themselves. We then use estat sbcusum to check the stability of the mean in our sample.

```
. quietly regress D.ln_wpi
. estat sbcusum, level(99)
```

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Test Statistic</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive</td>
<td>1.9030</td>
<td>1.1430</td>
<td>0.9479</td>
<td>0.850</td>
</tr>
</tbody>
</table>

We can interpret the results by comparing the absolute value of the modified cusum statistic with critical values; see Methods and formulas. We reject the null hypothesis of a constant mean at the 1% level because the test statistic value of 1.9030 exceeds the 1% critical level of 1.1430.
We can also examine the cusum plot. From the graph above, we see that the plot of the recursive cusum process crosses the 99% confidence bands, which implies that the mean of the regression model is not stable at the 1%, 5%, or 10% significance levels.

Moreover, the graph also provides information on the instability of the coefficients in the regression model. Note that under the null hypothesis of no parameter instability, the expected value of the cusum of recursive residuals is 0. In the graph above, the cusum of the recursive residuals starts crossing the 99% confidence bands in the middle of the sample, which also indicates that the parameters of the regression model become unstable during that time.

Cusum of OLS residuals

A similar test statistic is based on the cusum of OLS residuals. Under the null hypothesis, the OLS residuals are correlated and heteroskedastic. The OLS residuals have an expected value of 0 in the presence of an intercept or if the underlying process has a mean of 0 without an intercept. Their cumulative sum always returns to 0, unlike the recursive cusum process, which will drift away from 0 after a structural break.

The limiting distribution of the OLS cusum statistic under the null hypothesis is approximated by a Brownian bridge process (Ploberger and Krämer 1992). If there is a structural break at a certain time period, the absolute value of the OLS cusum statistic peaks at that time and then returns to the expected value of 0.

A plot with a constant upper bound and lower bound for a specified significance level is used as a test for parameter stability. If the cusum statistic goes outside of the bounds, we reject the null hypothesis of parameter stability. The plot also provides additional information on the timing of the structural break. Alternatively, the absolute value of the OLS cusum statistic may also be used to test the null hypothesis at a specified significance level.

Example 2: OLS cusum test

Continuing with example 1, we also test the null hypothesis of parameter stability by using the cusum of OLS residuals.
. estat sbcusum, ols level(99)
Cumulative sum test for parameter stability
Sample: 1960q2 - 1990q4 Number of obs = 123
Ho: No structural break

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Test Statistic</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ols</td>
<td>1.9854</td>
<td>1.6276</td>
<td>1.3581</td>
<td>1.224</td>
</tr>
</tbody>
</table>

We reject the null hypothesis of parameter stability at the 1%, 5%, and 10% levels. The graph below plots the OLS cusum process crossing the 99% band, which also implies rejection at the 1%, 5%, and 10% levels, confirming our results from the table.

Stored results

estat sbcusum stores the following in r():

Scalars:
- r(cusum) value of the test statistic
- r(level) confidence level of confidence intervals
- r(df) degrees of freedom

Macros:
- r(statistic) name of the statistic used; recursive or ols
- r(tmins) formatted minimum time
- r(tmaxs) formatted maximum time

Matrices:
- r(cvalues) vector of critical values
Consider a linear regression model with \( k \) variables,

\[ y_t = x_t' \beta_t + e_t \quad t = 1, \ldots, T \]

where \( y_t \) is the dependent variable and \( x_t \) is a vector of covariates that may contain lags of the dependent variable. \( \beta_t \) is a vector of time-varying parameters and \( e_t \) is an independent and identically distributed error term.

The cusum test based on recursive residuals is constructed using the one-step-ahead standardized forecast error

\[ e_{t}^{\text{rec}} = \frac{y_t - x_t' \hat{\beta}_{t-1}}{\sqrt{1 + x_t' (X_{t-1}'X_{t-1}) x_t}} \quad t = k + 1, \ldots, T \]

where \( \hat{\beta}_{t-1} \) is the estimate of \( \beta \) for the first \( t - 1 \) observations and \( X_{t-1}' = (x_1, \ldots, x_{t-1}) \). The null hypothesis for the test of parameter stability is \( H_0: \beta_t = \beta \), which implies constant parameters \( \beta \).

Under the null hypothesis, the sequence \( \{e_{t}^{\text{rec}}\} \) is independent and identically distributed \( N(0, \sigma^2) \).

The cusum statistic is formed as

\[ C_t^{\text{rec}} = \frac{1}{\hat{\sigma}} \sum_{j=k+1}^{t} e_{j}^{\text{rec}} \]

where \( \hat{\sigma}^2 = \{1/(T - k)\} \sum_{t=k+1}^{T} (e_{t}^{\text{rec}} - \bar{e}_{t}^{\text{rec}})^2 \). The limiting distribution of the sequence \( \{C_t^{\text{rec}}\} \) is a Brownian motion starting from 0 at time \( t = k + 1 \) with an expected value of 0 and variance \( t - k \).

Inference is based on the path of the sequence \( \{C_t^{\text{rec}}\} \) crossing a theoretical boundary of a Brownian motion with probability \( \alpha \). Brown, Durbin, and Evans (1975) approximate the nonlinear boundaries by a pair of linear functions \( \pm c \{1 + 2(t - k)/(T - k)\} \), where the value of \( c \) is obtained by solving the following equation:

\[ \Phi(3c) + \exp(-4c^2) \{1 - \Phi(c)\} = 0.5\alpha \]

The test statistic is formed as

\[ \max_{k+1 \leq t \leq T} \left| \frac{C_t^{\text{rec}}}{1 + 2\frac{t-k}{T-k}} \right| \]

The cusum statistic based on OLS residuals is constructed as

\[ C_t^{\text{ols}} = \frac{1}{\tilde{\sigma} \sqrt{T}} \sum_{j=1}^{T} e_{j}^{\text{ols}} \]

where \( e_{t}^{\text{ols}} \) is the OLS residual and \( \tilde{\sigma}^2 = \{1/(T - k)\} \sum_{t=k+1}^{T} (e_{t}^{\text{ols}} - \bar{e}_{t}^{\text{ols}})^2 \). Ploberger and Krämer (1992) show the limiting distribution of the sequence \( \{C_t^{\text{ols}}\} \) to be a Brownian bridge process that starts at 0 at time \( t = k + 1 \) and ends at 0 at time \( T \). The boundaries for the cusum statistic are approximated by a pair of parallel lines with the probability of a sample process crossing the boundaries being \( \alpha \).

The critical value \( c \) is obtained by solving the following equation:

\[ 2 \sum_{j=1}^{\infty} (-1)^{j+1} \exp(-2j^2c^2) = \alpha \]

The test statistic is formed as the maximum of \( |C_t^{\text{ols}}| \).
References


Also see

[TS] estat sbknown — Test for a structural break with a known break date
[TS] estat sbsingle — Test for a structural break with an unknown break date
[TS] tsset — Declare data to be time-series data
[R] regress — Linear regression
**Description**

`estat sbknown` performs a Wald or a likelihood-ratio (LR) test of whether the coefficients in a time-series regression vary over the periods defined by known break dates.

`estat sbknown` requires that the current estimation results be from `regress` or `ivregress 2sls`.

**Quick start**

Test for a structural break at January 1983 for current estimation results

```stata
estat sbknown, break(tm(1983m1))
```

As above, but for the first quarter of 1997

```stata
estat sbknown, break(tq(1997q1))
```

As above, but perform an LR test instead of a Wald test

```stata
estat sbknown, break(tq(1997q1)) lr
```

Perform a Wald test for multiple breaks at dates 1997q1 and 2005q1

```stata
estat sbknown, break(tq(1997q1) tq(2005q1))
```

**Menu for estat**

Statistics  >  Postestimation
Syntax

\texttt{estat sbknown, break\{time\_constant\_list\} \ [options]} \]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*break{time_constant_list}</td>
<td>specify one or more break dates</td>
</tr>
<tr>
<td>breakvars{\ [varlist], \ constant }</td>
<td>specify variables to be included in the test; by default, all coefficients are tested</td>
</tr>
<tr>
<td>wald</td>
<td>request a Wald test; the default</td>
</tr>
<tr>
<td>lr</td>
<td>request an LR test</td>
</tr>
</tbody>
</table>

*break() is required.

You must \texttt{tsset} your data before using \texttt{estat sbknown}; see \[TS\] \texttt{tsset}.

Options

\texttt{break\{time\_constant\_list\}} specifies a list of one or more hypothesized break dates. \texttt{break()} is required with at least one break date.

\texttt{time\_constant\_list} is a list of one or more \texttt{time\_constant} elements specified using dates in Stata internal form (SIF) or human-readable form (HRF) format. If you specify the \texttt{time\_constant\_list} using HRF, you must use one of the datetime pseudofunctions; see \[D\] \texttt{Datetime}.

\texttt{breakvars\{\ [varlist], \ constant \}} specifies variables to be included in the test. By default, all the coefficients are tested.

\texttt{constant} specifies that a constant be included in the list of variables to be tested. \texttt{constant} may be specified only if the original model was fit with a constant term.

\texttt{wald} requests that a Wald test be performed. This is the default.

\texttt{lr} requests that an LR test be performed instead of a Wald test.

Remarks and examples

\texttt{estat sbknown} performs a test of the null hypothesis that the coefficients do not vary over the subsamples defined by the specified known break dates. The null hypothesis of no structural break can be tested using a Wald or an LR test.

Consider the linear regression

\[ y_t = x_t \beta + \epsilon_t \]

A model with a structural break allows the coefficients to change after a break date. If \( b \) is the break date, the model is

\[ y_t = \begin{cases} x_t \beta + \epsilon_t & \text{if } t \leq b \\ x_t (\beta + \delta) + \epsilon_t & \text{if } t > b \end{cases} \]

For this model, the null and alternative hypotheses are \( H_0 : \delta = 0 \) and \( H_a : \delta \neq 0 \).

For a classical linear model where \( \epsilon_t \) is independent and identically distributed, this test is known as the Chow (1960) test.
Example 1: Test for a single known break date

In usmacro.dta, we have data for the fedfunds series from the third quarter of 1954 to the fourth quarter of 2010 from the Federal Reserve Economic Database (FRED), a macroeconomic database provided by the Federal Reserve Bank of Saint Louis.

We want to model the federal funds rate as a function of its first lag, but we are concerned that there may be a structural break after 1981q2. We fit the model parameters using regress, and then we use estat sbknown to test for a structural break.

```
use https://www.stata-press.com/data/r16/usmacro
(output omitted)
regress fedfunds L.fedfunds
(output omitted)
estat sbknown, break(tq(1981q2))
```

Wald test for a structural break: Known break date

<table>
<thead>
<tr>
<th>Sample:</th>
<th>1954q4 - 2010q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break date:</td>
<td>1981q2</td>
</tr>
<tr>
<td>Ho: No structural break</td>
<td></td>
</tr>
<tr>
<td>chi2(2)</td>
<td>6.4147</td>
</tr>
<tr>
<td>Prob &gt; chi2</td>
<td>0.0405</td>
</tr>
</tbody>
</table>

Exogenous variables: L.fedfunds

Coefficients included in test: L.fedfunds _cons

We reject the null hypothesis of no structural break at the 5% level.

Example 2: Test for multiple known breaks

Suppose we divide the data into three subsamples for the periods 1954q4 to 1970q1, 1970q2 to 1995q1, and 1995q2 to 2010q4, specified by the break dates at 1970q1 and 1995q1. We would like to test whether the coefficients are the same in these subsamples. We do this by specifying multiple dates in the break() option.

```
estat sbknown, break(tq(1970q1) tq(1995q1))
```

Wald test for a structural break: Known break date

<table>
<thead>
<tr>
<th>Sample:</th>
<th>1954q4 - 2010q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break date:</td>
<td>1970q1 1995q1</td>
</tr>
<tr>
<td>Ho: No structural break</td>
<td></td>
</tr>
<tr>
<td>chi2(4)</td>
<td>4.6739</td>
</tr>
<tr>
<td>Prob &gt; chi2</td>
<td>0.3224</td>
</tr>
</tbody>
</table>

Exogenous variables: L.fedfunds

Coefficients included in test: L.fedfunds _cons

We fail to reject the null hypothesis of no structural break for the specified dates.
### Stored results

`estat sbknown` stores the following in `r()`:

- **Scalars**
  - `r(chi2)`: $\chi^2$ test statistic
  - `r(p)`: $p$-value for $\chi^2$ test
  - `r(df)`: degrees of freedom

- **Macros**
  - `r(breakdate)`: list of break dates
  - `r(breakvars)`: list of variables whose coefficients are included in the test
  - `r(test)`: type of test

### Methods and formulas

A test for a structural break with a known break date can be constructed by fitting a linear regression with an indicator variable as

$$y_t = x_t \beta + (b \leq t)x_t \delta + \epsilon_t$$

The null hypothesis of no structural break is $H_0: \delta = 0$. This can be tested by constructing a Wald statistic or an LR statistic, both with $\chi^2(k)$ as the limiting distribution, where $k$ is the number of parameters in the model.

A regression model with multiple breaks may be expressed as

$$y_t = x_t \beta + x_t \{(b_1 \leq t < b_2)\delta_1 + (b_2 \leq t < b_3)\delta_2 + \cdots + (b_m \leq t)\delta_m\} + \epsilon_t$$

where $b_1, \ldots, b_m$ are $m \geq 2$ break dates. The null hypothesis of no structural break is a joint test given by $H_0: \delta_1 = \cdots = \delta_m = 0$.

### References


### Also see

- [TS] `estat sbcusum` — Cumulative sum test for parameter stability
- [TS] `estat sbsingle` — Test for a structural break with an unknown break date
- [TS] `tsset` — Declare data to be time-series data
- [R] `ivregress` — Single-equation instrumental-variables regression
- [R] `regress` — Linear regression
**estat sbsingle** — Test for a structural break with an unknown break date

**Description**

`estat sbsingle` performs a test of whether the coefficients in a time-series regression vary over the periods defined by an unknown break date.

`estat sbsingle` requires that the current estimation results be from `regress` or `ivregress 2sls`.

**Quick start**

Supremum Wald test for a structural break at an unknown break date for current estimation results using default symmetric trimming of 15%

`estat sbsingle`

Same as above

`estat sbsingle, swald`

As above, but also report average Wald test

`estat sbsingle, swald awald`

Supremum Wald test with symmetric trimming of 20%

`estat sbsingle, trim(20)`

As above, but use asymmetric trimming with a left trim of 10% and a right trim of 20%

`estat sbsingle, ltrim(10) rtrim(20)`

**Menu for estat**

Statistics > Postestimation
Syntax

```plaintext
estat sbsingle [, options]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakvars([varlist][, constant])</td>
<td>specify variables to be included in the test; by default, all coefficients are tested</td>
</tr>
<tr>
<td>trim(#)</td>
<td>specify a trimming percentage; default is trim(15)</td>
</tr>
<tr>
<td>ltrim(#l)</td>
<td>specify a left trimming percentage</td>
</tr>
<tr>
<td>rtrim(#r)</td>
<td>specify a right trimming percentage</td>
</tr>
<tr>
<td>swald</td>
<td>request a supremum Wald test; the default</td>
</tr>
<tr>
<td>awald</td>
<td>request an average Wald test</td>
</tr>
<tr>
<td>ewald</td>
<td>request an exponential Wald test</td>
</tr>
<tr>
<td>all</td>
<td>report all tests</td>
</tr>
<tr>
<td>slr</td>
<td>request a supremum likelihood-ratio (LR) test</td>
</tr>
<tr>
<td>alr</td>
<td>request an average LR test</td>
</tr>
<tr>
<td>elr</td>
<td>request an exponential LR test</td>
</tr>
<tr>
<td>generate(newvarlist)</td>
<td>create newvarlist containing Wald or LR test statistics</td>
</tr>
<tr>
<td>nodots</td>
<td>suppress iteration dots</td>
</tr>
</tbody>
</table>

You must tsset your data before using estat sbsingle; see [TS] tsset.

Options

`breakvars([varlist][, constant])` specifies variables to be included in the test. By default, all the coefficients are tested.

`constant` specifies that a constant be included in the list of variables to be tested. `constant` may be specified only if the original model was fit with a constant term.

`trim(#)` specifies an equal left and right trimming percentage as an integer. Specifying `trim(#)` causes the observation at the #th percentile to be treated as the first possible break date and the observation at the (100 − #)th percentile to be treated as the last possible break date. By default, the trimming percentage is set to 15 but may be set to any value between 1 and 49.

`ltrim(#l)` specifies a left trimming percentage as an integer. Specifying `ltrim(#l)` causes the observation at the #lth percentile to be treated as the first possible break date. This option must be specified with `rtrim(#r)` and may not be combined with `trim(#r)`. #l must be between 1 and 99.

`rtrim(#r)` specifies a right trimming percentage as an integer. Specifying `rtrim(#r)` causes the observation at the (100 − #r)th percentile to be treated as the last possible break date. This option must be specified with `ltrim(#l)` and may not be combined with `trim(#r)`. #r must be less than (100 − #l). Specifying #l = #r is equivalent to specifying `trim(#)` with # = #l = #r.

`swald` requests that a supremum Wald test be performed. This is the default.

`awald` requests that an average Wald test be performed.

`ewald` requests that an exponential Wald test be performed.

`all` specifies that all tests be displayed in a table.

`slr` requests that a supremum LR test be performed.
alr requests that an exponential LR test be performed.

generate(newvarlist) creates either one or two new variables containing the Wald statistics, LR statistics, or both that are transformed and used to calculate the requested Wald or LR tests. If you request only Wald-type tests (swald, awald, or ewald) or only LR-type tests (slr, alr, or elr), then you may specify only one varname in generate(). By default, newvar will contain Wald or LR statistics, depending on the type of test specified.

A variable containing Wald statistics and a variable containing LR statistics are created if you specify both Wald-type and LR-type tests and specify two varnames in generate(). If you only specify one varname in generate() with Wald-type and LR-type tests specified, then Wald statistics are returned.

If no test is specified and generate() is specified, Wald statistics are returned.
	nodots suppresses display of the iteration dots. By default, one dot character is displayed for each iteration in the range of possible break dates.

Remarks and examples

estat sbsingle constructs a test statistic for a structural break without imposing a known break date by combining the test statistics computed for each possible break date in the sample. estat sbsingle uses the maximum, an average, or the exponential of the average of the tests computed at each possible break date. The test at each possible break date can be either a Wald or an LR test.

The limiting distribution of each of these test statistics is known but nonstandard. Not only is each test statistic a function of many sample statistics, but each of these test statistics also depends on the unknown break date, which is not identified under the null hypothesis; see Davies (1987) for a seminal treatment.

Tests that use the maximum of the sample tests are known as supremum tests. Specifically, the supremum Wald test uses the maximum of the sample Wald tests, and the supremum LR test uses the maximum of the sample LR tests. The intuition behind these tests is to compare the maximum sample test with what could be expected under the null hypothesis of no break (Quandt [1960], Kim and Siegmund [1989], and Andrews [1993]).

Supremum tests have much less power than average tests and exponential tests. Average tests use the average of the sample tests, and exponential tests use the natural log of the average of the exponential of the sample tests. An average test is optimal when the alternative hypothesis is a small change in parameter values at the structural break. An exponential test is optimal when the alternative hypothesis is a larger structural break. See Andrews and Ploberger (1994) for details about the properties of average and exponential tests.

All tests implemented in estat sbsingle are a function of the sample statistics computed over a range of possible break dates. However, not all sample observations can be tested as break dates because there are insufficient observations to estimate the parameters for dates too near the beginning or the end of the sample. This identification problem is solved by trimming, which excludes observations too near the beginning or the end of the sample from the set of possible break dates. Andrews (1993) recommends a symmetric trimming of 15% when the researcher has no other information on good trimming values.

Much research went into deriving the properties of the implemented tests, and we have cited only a few of the many papers on the subject. See Perron (2006) for an excellent survey.
Example 1: Test for a structural break with unknown break date

In usmacro.dta, we have data for the fedfunds series from the third quarter of 1954 to the fourth quarter of 2010 from the Federal Reserve Economic Database (FRED) provided by the Federal Reserve Bank of Saint Louis.

Consider a model for the federal funds rate as a function of its first lag and the inflation rate (inflation). We want to test whether coefficients changed at an unknown break date. Below, we fit the model using regress and perform the test using estat sbsingle.

```
use https://www.stata-press.com/data/r16/usmacro
(regression data - St. Louis Fed)
regress fedfunds L.fedfunds inflation
(estimation results omitted)
estat sbsingle
```

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test for a structural break: Unknown break date

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of obs</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>Full sample:</td>
<td>1955q3 - 2010q4</td>
<td></td>
</tr>
<tr>
<td>Trimmed sample:</td>
<td>1964q1 - 2002q3</td>
<td></td>
</tr>
<tr>
<td>Estimated break date:</td>
<td>1980q4</td>
<td></td>
</tr>
<tr>
<td>Ho: No structural break</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test</th>
<th>Statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>swald</td>
<td>14.1966</td>
<td>0.0440</td>
</tr>
</tbody>
</table>

Exogenous variables: L.fedfunds inflation
Coefficients included in test: L.fedfunds inflation _cons

By default, a supremum Wald test is performed. The output indicates that we reject the null hypothesis of no structural break at the 5% level and that the estimated break date is 1980q4.

Some researchers perform more than one test. Below, we present results for the supremum Wald, average Wald, and average LR tests.

```
estat sbsingle, swald awald alr
```

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test for a structural break: Unknown break date

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of obs</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>Full sample:</td>
<td>1955q3 - 2010q4</td>
<td></td>
</tr>
<tr>
<td>Trimmed sample:</td>
<td>1964q1 - 2002q3</td>
<td></td>
</tr>
<tr>
<td>Ho: No structural break</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test</th>
<th>Statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>swald</td>
<td>14.1966</td>
<td>0.0440</td>
</tr>
<tr>
<td>awald</td>
<td>4.5673</td>
<td>0.1474</td>
</tr>
<tr>
<td>alr</td>
<td>4.6319</td>
<td>0.1411</td>
</tr>
</tbody>
</table>

Exogenous variables: L.fedfunds inflation
Coefficients included in test: L.fedfunds inflation _cons
Only the supremum Wald test rejects the null hypothesis of no break.

Example 2: Testing for a structural break in a subset of coefficients

Below, we test the null hypothesis that there is a break in the intercept, when we assume that there is no break in either the autoregressive coefficient or the coefficient on inflation.

```
. estat sbsingle, breakvars(, constant)
```

```
 Test for a structural break: Unknown break date

 Number of obs = 222

 Full sample: 1955q3 - 2010q4
 Trimmed sample: 1964q1 - 2002q3
 Estimated break date: 2001q1
 Ho: No structural break

 Test Statistic p-value
 swald 6.7794 0.1141

 Exogenous variables: L.fedfunds inflation
 Coefficients included in test: _cons
```

We fail to reject the null hypothesis of no structural break in the intercept when there is no break in any other coefficient.

Example 3: Reviewing sample test statistics

The observation-level Wald or LR test statistics sometimes provide useful diagnostic information. Below, we use the `generate()` option to store the observation-level Wald statistics in the new variable `wald`, which we subsequently plot using `tsline`.

```
. estat sbsingle, breakvars(, constant)
    ------------- 50
    ------------- 100
    ------------- 150
```

```
 Test for a structural break: Unknown break date

 Number of obs = 222

 Full sample: 1955q3 - 2010q4
 Trimmed sample: 1964q1 - 2002q3
 Estimated break date: 2001q1
 Ho: No structural break

 Test Statistic p-value
 swald 6.7794 0.1141

 Exogenous variables: L.fedfunds inflation
 Coefficients included in test: _cons
```

The observation-level Wald or LR test statistics sometimes provide useful diagnostic information. Below, we use the `generate()` option to store the observation-level Wald statistics in the new variable `wald`, which we subsequently plot using `tsline`.
. estat sbsingle, breakvars(L.fedfunds) generate(wald)
  (output omitted)
. tsline wald, title("Wald test statistics")

We see a spike in the value of the test statistic at the estimated break date of 1980q4. The bump to left of the spike may indicate a second break.

Example 4: Structural break test with an endogenous regressor

We can use `estat sbsingle` to test for a structural break in a regression with endogenous variables. Suppose we want to estimate the New Keynesian hybrid Phillips curve, which defines inflation as a function of the lagged value of inflation (L.inflation), the output gap (ogap), and the expected value of inflation in $t+1$ \{\text{E}_t(\text{F.inflation})\}, conditional on information available at time $t$ (Gali and Gertler 1999). See [U] 11.4.4 Time-series varlists.

Expected future inflation cannot be directly observed, so macroeconomists use instruments to predict the one-step-ahead inflation rate. This prediction is obtained by regressing the one-step ahead inflation rate on a set of instruments.

We can write this mathematically as

\[
\text{inflation} = \alpha + \text{L.inflation} \beta + \text{ogap} \delta + \text{E}_t(\text{F.inflation}) \gamma + \epsilon_t
\]

and

\[
\text{F.inflation} = \text{z}_t \theta + \nu_{t+1}
\]

where $\text{z}_t$ is a vector of instruments. The forecasted values given by $\text{E}_t(\text{F.inflation}|\text{z}_t) = \text{z}_t \hat{\theta}$ are uncorrelated with $\nu_{t+1}$ by construction.

In this example, we fit the Phillips curve model for the period 1970q1 to 1997q4. We are interested in testing whether expectation of future inflation changed during this period. We instrument the future value of inflation with the first two lags of inflation, the federal funds rate, and the output gap. We use `ivregress 2sls` to fit the model.
. ivregress 2sls inflation L.inflation ogap
> (F.inflation = L(1/2).inflation L(1/2).ogap L(1/2).fedfunds)
> if tin(1970q1,1997q4)
(output omitted)
. estat sbsingle, breakvars(F.inflation)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>50</td>
</tr>
</tbody>
</table>

Test for a structural break: Unknown break date

Number of obs = 112
Full sample: 1970q1 - 1997q4
Trimmed sample: 1974q2 - 1993q4
Estimated break date: 1981q3
Ho: No structural break

<table>
<thead>
<tr>
<th>Test</th>
<th>Statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>swald</td>
<td>6.7345</td>
<td>0.1164</td>
</tr>
</tbody>
</table>

Coefficients included in test: F.inflation

We fail to reject the null hypothesis of no structural break in the coefficient of expected future inflation.

Stored results

`estat sbsingle` stores the following in r():

Scalars
- r(chi2): $\chi^2$ test statistic
- r(p): p-value for $\chi^2$ test
- r(df): degrees of freedom

Macros
- r(ltrim): start of trim date
- r(rtrim): end of trim date
- r(breakvars): list of variables whose coefficients are included in the test
- r(breakdate): estimated break date only after supremum tests
- r(test): type of test

Methods and formulas

Each supremum test statistic is the maximum value of the test statistic that is obtained from a series of Wald or LR tests over a range of possible break dates in the sample. Let $b$ denote a possible break date in the range $[b_1, b_2]$ for a sample size $T$.

The supremum test statistic for testing the null hypothesis of no structural change in $k$ coefficients is given by

$$\sup_{b_1 \leq b \leq b_2} S_T(b)$$
where $S_T(b)$ is the Wald or LR test statistic evaluated at a potential break date $b$. The average and the exponential versions of the test statistic are

$$\text{average } S_T = \frac{1}{b_2 - b_1 + 1} \sum_{b=b_1}^{b_2} S_T(b)$$

$$\text{exponential } S_T = \ln \left[ \frac{1}{b_2 - b_1 + 1} \sum_{b=b_1}^{b_2} \exp \left\{ \frac{1}{2} S_T(b) \right\} \right]$$

respectively.

The limiting distributions of the test statistics are given by

$$\text{supremum } S_T \rightarrow_d \sup_{\lambda \in [\varepsilon_1, \varepsilon_2]} S(\lambda)$$

$$\text{average } S_T \rightarrow_d \frac{1}{\varepsilon_2 - \varepsilon_1} \int_{\varepsilon_1}^{\varepsilon_2} S(\lambda) \, d\lambda$$

$$\text{exponential } S_T \rightarrow_d \ln \left[ \frac{1}{\varepsilon_2 - \varepsilon_1} \int_{\varepsilon_1}^{\varepsilon_2} \exp \left\{ \frac{1}{2} S(\lambda) \, d\lambda \right\} \right]$$

where

$$S(\lambda) = \frac{\{B_k(\lambda) - \lambda B_k(1)\}'\{B_k(\lambda) - \lambda B_k(1)\}}{\lambda(1 - \lambda)}$$

$B_k(\lambda)$ is a vector of $k$-dimensional independent Brownian motions, $\varepsilon_1 = b_1/T$, $\varepsilon_2 = b_2/T$, and $\lambda = \varepsilon_2 (1 - \varepsilon_1)/(\varepsilon_1 (1 - \varepsilon_2))$.

Computing the $p$-values for the nonstandard limiting distributions is computationally complicated. For each test, the reported $p$-value is computed using the method in Hansen (1997).

References


Davies, R. B. 1987. Hypothesis testing when a nuisance parameter is present only under the alternative. *Biometrika* 74: 33–43.


Also see

[TS] estat sbcusum — Cumulative sum test for parameter stability
[TS] estat sbknown — Test for a structural break with a known break date
[TS] tsset — Declare data to be time-series data
[R] ivregress — Single-equation instrumental-variables regression
[R] regress — Linear regression
**fcast compute** — Compute dynamic forecasts after var, svar, or vec

**Description**

*fcast compute* produces dynamic forecasts of the dependent variables in a model previously fit by *var*, *svar*, or *vec*. *fcast compute* creates new variables and, if necessary, extends the time frame of the dataset to contain the prediction horizon.

**Quick start**

Dynamic forecasts stored in *f_y1*, *f_y2*, and *f_y3* after fitting a model with *var* for dependent variables *y1*, *y2*, and *y3*

```plaintext
fcast compute f_
```

As above, but begin forecast on the first quarter of 1979

```plaintext
fcast compute f_, dynamic(q(1979q1))
```

As above, but specify that 10 periods should be forecasted

```plaintext
fcast compute f_, dynamic(q(1979q1)) step(10)
```

**Menu**

Statistics > Multivariate time series > VEC/VAR forecasts > Compute forecasts (required for graph)
Syntax

After var and svar

\[ \texttt{fcast compute prefix[, options1]} \]

After vec

\[ \texttt{fcast compute prefix[, options2]} \]

prefix is the prefix appended to the names of the dependent variables to create the names of the variables holding the dynamic forecasts.

<table>
<thead>
<tr>
<th>options (_1)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td></td>
</tr>
<tr>
<td>\texttt{step(#)}</td>
<td>set # periods to forecast; default is \texttt{step(1)}</td>
</tr>
<tr>
<td>\texttt{dynamic(time_constant)}</td>
<td>begin dynamic forecasts at \texttt{time_constant}</td>
</tr>
<tr>
<td>\texttt{estimate(estname)}</td>
<td>use previously stored results \texttt{estname}; default is to use active results</td>
</tr>
<tr>
<td>\texttt{replace}</td>
<td>replace existing forecast variables that have the same prefix</td>
</tr>
<tr>
<td><strong>Std. Errors</strong></td>
<td></td>
</tr>
<tr>
<td>\texttt{nose}</td>
<td>suppress asymptotic standard errors</td>
</tr>
<tr>
<td>\texttt{bs}</td>
<td>obtain standard errors from bootstrapped residuals</td>
</tr>
<tr>
<td>\texttt{bsp}</td>
<td>obtain standard errors from parametric bootstrap</td>
</tr>
<tr>
<td>\texttt{bscentile}</td>
<td>estimate bounds by using centiles of bootstrapped dataset</td>
</tr>
<tr>
<td>\texttt{reps(#)}</td>
<td>perform # bootstrap replications; default is \texttt{reps(200)}</td>
</tr>
<tr>
<td>\texttt{nodots}</td>
<td>suppress the usual dot after each bootstrap replication</td>
</tr>
<tr>
<td>\texttt{saving(filename[, replace])}</td>
<td>save bootstrap results as \texttt{filename}; use \texttt{replace} to overwrite existing \texttt{filename}</td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td></td>
</tr>
<tr>
<td>\texttt{level(#)}</td>
<td>set confidence level; default is \texttt{level(95)}</td>
</tr>
</tbody>
</table>
options

Main

step(#) specifies the number of periods to be forecast. The default is step(1).

dynamic(time_constant) specifies the period to begin the dynamic forecasts. The default is the period after the last observation in the estimation sample. The dynamic() option accepts either a Stata date function that returns an integer or an integer that corresponds to a date using the current tsset format. dynamic() must specify a date in the range of two or more periods into the estimation sample to one period after the estimation sample.

estimate(estname) specifies that fcast compute use the estimation results stored as estname. By default, fcast compute uses the active estimation results. See [R] estimates for more information on manipulating estimation results.

replace causes fcast compute to replace the variables in memory with the specified predictions.

differences specifies that fcast compute also save dynamic predictions of the first-differenced variables. differences can be specified only with vec estimation results.

Std. Errors

nose specifies that the asymptotic standard errors of the forecasted levels, and thus the asymptotic confidence intervals for the levels, not be calculated. By default, the asymptotic standard errors and the asymptotic confidence intervals of the forecasted levels are calculated.

bs specifies that fcast compute use confidence bounds estimated by a simulation method based on bootstrapping the residuals.

bsp specifies that fcast compute use confidence bounds estimated via simulation in which the innovations are drawn from a multivariate normal distribution.

bscentile specifies that fcast compute use centiles of the bootstrapped dataset to estimate the bounds of the confidence intervals. By default, fcast compute uses the estimated standard errors and the quantiles of the standard normal distribution determined by level().

Reporting

level(#) set confidence level; default is level(95)
reps(#) gives the number of repetitions used in the simulations. The default is reps(200).

nodots specifies that no dots be displayed while obtaining the simulation-based standard errors. By default, for each replication, a dot is displayed.

saving(filename[, replace]) specifies the name of the file to hold the dataset that contains the bootstrap replications. The replace option overwrites any file with this name.

replace specifies that filename be overwritten if it exists. This option is not shown in the dialog box.

level(#) specifies the confidence level, as a percentage, for confidence intervals. The default is level(95) or as set by set level; see [U] 20.8 Specifying the width of confidence intervals.

Remarks and examples

Researchers often use VARs and VECMs to construct dynamic forecasts. fcast compute computes dynamic forecasts of the dependent variables in a VAR or VECM previously fit by var, svar, or vec. If you are interested in conditional, one-step-ahead predictions, use predict (see [TS] var, [TS] var svar, and [TS] vec).

To obtain and analyze dynamic forecasts, you fit a model, use fcast compute to compute the dynamic forecasts, and use fcast graph to graph the results.

Example 1

Typing

```
  use https://www.stata-press.com/data/r16/lutkepohl2
  var dln_inc dln_consump dln_inv if qtr<tfq(1979q1)
  fcast compute m2_, step(8)
  fcast graph m2_dln_inc m2_dln_inv m2_dln_consump, observed
```

fits a VAR with two lags, computes eight-step dynamic predictions for each endogenous variable, and produces the graph.
The graph shows that the model is better at predicting changes in income and investment than in consumption. The graph also shows how quickly the predictions from the two-lag model settle down to their mean values.

`fcast compute` creates new variables in the dataset. If there are $K$ dependent variables in the previously fitted model, `fcast compute` generates $4K$ new variables:

- $K$ new variables that hold the forecasted levels, named by appending the specified prefix to the name of the original variable
- $K$ estimated lower bounds for the forecast interval, named by appending the specified prefix and the suffix “_LB” to the name of the original variable
- $K$ estimated upper bounds for the forecast interval, named by appending the specified prefix and the suffix “_UB” to the name of the original variable
- $K$ estimated standard errors of the forecast, named by appending the specified prefix and the suffix “_SE” to the name of the original variable

If you specify options so that `fcast compute` does not calculate standard errors, the $3K$ variables that hold them and the bounds of the confidence intervals are not generated.

If the model previously fit is a VECM, specifying `differences` generates another $K$ variables that hold the forecasts of the first differences of the dependent variables, named by appending the prefix “prefixD_” to the name of the original variable.

**Example 2**

Plots of the forecasts from different models along with the observations from a holdout sample can provide insights to their relative forecasting performance. Continuing the previous example,
The model with six lags predicts changes in investment better than the two-lag model in some periods but markedly worse in other periods.

Methods and formulas

Methods and formulas are presented under the following headings:

Predictions after var and svar
Dynamic forecasts after vec

Predictions after var and svar

A VAR with endogenous variables $y_t$ and exogenous variables $x_t$ can be written as

$$ y_t = v + A_1 y_{t-1} + \cdots + A_p y_{t-p} + B x_t + u_t $$

where

$t = 1, \ldots, T$

$y_t = (y_{1t}, \ldots, y_{Kt})'$ is a $K \times 1$ random vector, the $A_i$ are fixed $(K \times K)$ matrices of parameters, $x_t$ is an $(M \times 1)$ vector of exogenous variables, $B$ is a $(K \times M)$ matrix of coefficients, $v$ is a $(K \times 1)$ vector of fixed parameters, and $u_t$ is assumed to be white noise; that is,

$$ E(u_t) = 0_K $$

$$ E(u_t u_t') = \Sigma $$

$$ E(u_t u_s') = 0_K \text{ for } t \neq s $$
The `fcast compute` function will dynamically predict the variables in the vector $y_t$ conditional on $p$ initial values of the endogenous variables and any exogenous $x_t$. Adopting the notation from Lütkepohl (2005, 402) to fit the case at hand, the optimal $h$-step-ahead forecast of $y_{t+h}$ conditional on $x_t$ is

$$y_{t}(h) = \hat{v} + \hat{A}_1 y_{t}(h-1) + \cdots + \hat{A}_p y_{t}(h-p) + \hat{B} x_t$$

(1)

If there are no exogenous variables, (1) becomes

$$y_{t}(h) = \hat{v} + \hat{A}_1 y_{t}(h-1) + \cdots + \hat{A}_p y_{t}(h-p)$$

When there are no exogenous variables, `fcast compute` can compute the asymptotic confidence bounds.

As shown by Lütkepohl (2005, 204–205), the asymptotic estimator of the covariance matrix of the prediction error is given by

$$\hat{\Sigma}_{\hat{y}}(h) = \hat{\Sigma}_{\hat{y}}(h) + \frac{1}{T} \hat{\Omega}(h)$$

(2)

where

$$\hat{\Sigma}_{\hat{y}}(h) = \sum_{i=0}^{h-1} \hat{\Phi}_i \hat{\Sigma} \hat{\Phi}_i'$$

$$\hat{\Omega}(h) = \frac{1}{T} \sum_{t=0}^{T} \left\{ \sum_{i=0}^{h-1} Z_t (\hat{B}')^{h-1-i} \otimes \hat{\Phi}_i \right\} \hat{\Sigma}_{\beta} \left\{ \sum_{i=0}^{h-1} Z_t (\hat{B}')^{h-1-i} \otimes \hat{\Phi}_i \right\}$$

$$\hat{\Phi}_0 = I_K$$

$$\hat{\Phi}_i = \sum_{j=1}^{i} \hat{\Phi}_{i-j} \hat{A}_j \quad i = 1, 2, \ldots$$

$$\hat{A}_j = 0 \quad \text{for} \ j > p$$

$\hat{\Sigma}$ is the estimate of the covariance matrix of the innovations, and $\hat{\Sigma}_{\beta}$ is the estimated VCE of the coefficients in the VAR. The formula in (2) is general enough to handle the case in which constraints are placed on the coefficients in the VAR($p$).

Equation (2) is made up of two terms. $\hat{\Sigma}_{\hat{y}}(h)$ is the estimated mean squared error (MSE) of the forecast. $\hat{\Sigma}_{\hat{y}}(h)$ estimates the error in the forecast arising from the unseen innovations. $T^{-1} \hat{\Omega}(h)$ estimates the error in the forecast that is due to using estimated coefficients instead of the true coefficients. As the sample size grows, uncertainty with respect to the coefficient estimates decreases, and $T^{-1} \hat{\Omega}(h)$ goes to zero.
If $y_t$ is normally distributed, the bounds for the asymptotic $(1 - \alpha)100\%$ interval around the forecast for the $k$th component of $y_t$, $h$ periods ahead, are

$$\hat{y}_{k,t}(h) \pm z(\frac{\alpha}{2})\hat{\sigma}_k(h)$$

(3)

where $\hat{\sigma}_k(h)$ is the $k$th diagonal element of $\hat{\Sigma}_y(h)$.

Specifying the bs option causes the standard errors to be computed via simulation, using bootstrapped residuals. Both var and svar contain estimators for the coefficients of a VAR that are conditional on the first $p$ observations on the endogenous variables in the data. Similarly, these algorithms are conditional on the first $p$ observations of the endogenous variables in the data. However, the simulation-based estimates of the standard errors are also conditional on the estimated coefficients. The asymptotic standard errors are not conditional on the coefficient estimates because the second term on the right-hand side of (2) accounts for the uncertainty arising from using estimated parameters.

For a simulation with $R$ repetitions, this method uses the following algorithm:

1. Fit the model and save the estimated coefficients.
2. Use the estimated coefficients to calculate the residuals.
3. Repeat steps 3a–3c $R$ times.
   3a. Draw a simple random sample with replacement of size $T + h$ from the residuals. When the $t$th observation is drawn, all $K$ residuals are selected, preserving any contemporaneous correlation among the residuals.
   3b. Use the sampled residuals, $p$ initial values of the endogenous variables, any exogenous variables, and the estimated coefficients to construct a new sample dataset.
   3c. Save the simulated endogenous variables for the $h$ forecast periods in the bootstrapped dataset.
4. For each endogenous variable and each forecast period, the simulated standard error is the estimated standard error of the $R$ simulated forecasts. By default, the upper and lower bounds of the $(1 - \alpha)100\%$ are estimated using the simulation-based estimates of the standard errors and the normality assumption, as in (3). If the bcentile option is specified, the sample centiles for the upper and lower bounds of the $R$ simulated forecasts are used for the upper and lower bounds of the confidence intervals.

If the bsp option is specified, a parametric simulation algorithm is used. Specifically, everything is as above except that 3a is replaced by 3a(bsp) as follows:

3a(bsp). Draw $T + h$ observations from a multivariate normal distribution with covariance matrix $\hat{\Sigma}$.

The algorithm above assumes that $h$ forecast periods come after the original sample of $T$ observations. If the $h$ forecast periods lie within the original sample, smaller simulated datasets are sufficient.

**Dynamic forecasts after vec**

Methods and formulas of [TS] vec discusses how to obtain the one-step predicted differences and levels. fcast compute uses the previous dynamic predictions as inputs for later dynamic predictions.
Per Lütkepohl (2005, sec. 6.5), fcast compute uses

\[
\hat{\Sigma}_y(h) = \left( \frac{T}{T-d} \right)^{h-1} \sum_{i=0}^{h-1} \hat{\Phi}_i \hat{\Omega} \hat{\Phi}_i
\]

where the \( \hat{\Phi}_i \) are the estimated matrices of impulse–response functions, \( T \) is the number of observations in the sample, \( d \) is the number of degrees of freedom, and \( \hat{\Omega} \) is the estimated cross-equation variance matrix. The formulas for \( d \) and \( \hat{\Omega} \) are given in Methods and formulas of [TS] vec.

The estimated standard errors at step \( h \) are the square roots of the diagonal elements of \( \hat{\Sigma}_y(h) \).

Per Lütkepohl (2005), the estimated forecast-error variance does not consider parameter uncertainty. As the sample size gets infinitely large, the importance of parameter uncertainty diminishes to zero.

References


Also see

[TS] fcast graph — Graph forecasts after fcast compute
[TS] var intro — Introduction to vector autoregressive models
[TS] vec intro — Introduction to vector error-correction models
**fcast graph** graphs dynamic forecasts of the endogenous variables from a VAR(p) or VECM that has already been obtained from *fcast compute*; see [TS] *fcast compute*.

### Quick start

Graph forecasts in \( f_{y1} \) after *fcast compute*

\[
fcast graph \ f_{y1}
\]

As above, and include observed values of the predicted variable

\[
fcast graph \ f_{y1}, \ observed
\]

As above, but suppress confidence bands

\[
fcast graph \ f_{y1}, \ observed \ noci
\]
fcast graph — Graph forecasts after fcast compute

Syntax

```
fcast graph varlist [ if ] [ in ] [, options]
```

where `varlist` contains one or more forecasted variables generated by `fcast compute`.

**options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
</tr>
<tr>
<td>differences</td>
</tr>
<tr>
<td>noci</td>
</tr>
<tr>
<td>observed</td>
</tr>
<tr>
<td><strong>Forecast plot</strong></td>
</tr>
<tr>
<td>cline_options</td>
</tr>
<tr>
<td><strong>CI plot</strong></td>
</tr>
<tr>
<td>ciopts(area_options)</td>
</tr>
<tr>
<td><strong>Observed plot</strong></td>
</tr>
<tr>
<td>obopts(cline_options)</td>
</tr>
<tr>
<td><strong>Y axis, Time axis, Titles, Legend, Overall</strong></td>
</tr>
<tr>
<td>twoway_options</td>
</tr>
<tr>
<td>byopts(by_option)</td>
</tr>
</tbody>
</table>

**Options**

- **Main**
  - `differences`: specifies that the forecasts of the first-differenced variables be graphed. This option is available only with forecasts computed by `fcast compute` after `vec`. The `differences` option implies `noci`.
  - `noci`: specifies that the confidence intervals be suppressed. By default, the confidence intervals are included.
  - `observed`: specifies that observed values of the predicted variables be included in the graph. By default, observed values are not graphed.

- **Forecast plot**
  - `cline_options`: affect the rendition of the plotted lines corresponding to the forecast; see [G-3] `cline_options`.

- **CI plot**
  - `ciopts(area_options)`: affects the rendition of the confidence bands for the forecasts; see [G-3] `area_options`.

- **Observed plot**
  - `obopts(cline_options)`: affects the rendition of the observed values of the predicted variables; see [G-3] `cline_options`. This option implies the `observed` option.
Remarks and examples

*Remarks and examples*

fcast graph graphs dynamic forecasts created by fcast compute.

Example 1

In this example, we use a cointegrating VECM to model the state-level unemployment rates in Missouri, Indiana, Kentucky, and Illinois, and we graph the forecasts against a 6-month holdout sample.

```
use https://www.stata-press.com/data/r16/urates
vec missouri indiana kentucky illinois if t < tm(2003m7), trend(rconstant) > rank(2) lags(4)
  (output omitted)
fcast compute m1_, step(6)
fcast graph m1_missouri m1_indiana m1_kentucky m1_illinois, observed
```

Because the 95% confidence bands for the predicted unemployment rates in Missouri and Indiana do not contain all their observed values, the model does not reliably predict these unemployment rates.
Also see

- [TS] fcast compute — Compute dynamic forecasts after var, svar, or vec
- [TS] var intro — Introduction to vector autoregressive models
- [TS] vec intro — Introduction to vector error-correction models
**Description**

`forecast` is a suite of commands for obtaining forecasts by solving models, collections of equations that jointly determine the outcomes of one or more variables. Equations can be stochastic relationships fit using estimation commands such as `regress`, `ivregress`, `var`, or `reg3`; or they can be nonstochastic relationships, called identities, that express one variable as a deterministic function of other variables. Forecasting models may also include exogenous variables whose values are already known or determined by factors outside the purview of the system being examined. The `forecast` commands can also be used to obtain dynamic forecasts in single-equation models.

The `forecast` suite lets you incorporate outside information into your forecasts through the use of add factors and similar devices, and you can specify the future path for some model variables and obtain forecasts for other variables conditional on that path. Each set of forecast variables has its own name prefix or suffix, so you can compare forecasts based on alternative scenarios. Confidence intervals for forecasts can be obtained via stochastic simulation and can incorporate both parameter uncertainty and additive error terms.

`forecast` works with both time-series and panel datasets. Time-series datasets may not contain any gaps, and panel datasets must be strongly balanced.

This manual entry provides an overview of forecasting models and several examples showing how the `forecast` commands are used together. See the individual subcommands’ manual entries for detailed discussions of the various options available and specific remarks about those subcommands.

**Quick start**

Estimate a linear and an ARIMA regression, and store their results as `myreg` and `myarima`, respectively

```
regress y1 x1 x2
estimates store myreg
arima y2 x3 y1, ar(1) ma(1)
estimates store myarima
```

Create a forecast model with the name `mymodel`

```
forecast create mymodel
```

Add stored estimates `myreg` and `myarima` to forecast model `mymodel`

```
forecast estimates myreg
forecast estimates myarima
```

Compute dynamic forecasts from 2012 through 2020 of `y1` and `y2` using `mymodel` with nonmissing values of `x1`, `x2`, and `x3` for the entire forecast horizon

```
forecast solve, begin(2012) end(2020)
```


206
Syntax

```
forecast subcommand ... [, options ]
```

<table>
<thead>
<tr>
<th>subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>create a new model</td>
</tr>
<tr>
<td>estimates</td>
<td>add estimation result to current model</td>
</tr>
<tr>
<td>identity</td>
<td>specify an identity (nonstochastic equation)</td>
</tr>
<tr>
<td>coefvector</td>
<td>specify an equation via a coefficient vector</td>
</tr>
<tr>
<td>exogenous</td>
<td>declare exogenous variables</td>
</tr>
<tr>
<td>solve</td>
<td>obtain one-step-ahead or dynamic forecasts</td>
</tr>
<tr>
<td>adjust</td>
<td>adjust a variable by add factoring, replacing, etc.</td>
</tr>
<tr>
<td>describe</td>
<td>describe a model</td>
</tr>
<tr>
<td>list</td>
<td>list all forecast commands composing current model</td>
</tr>
<tr>
<td>clear</td>
<td>clear current model from memory</td>
</tr>
<tr>
<td>drop</td>
<td>drop forecast variables</td>
</tr>
<tr>
<td>query</td>
<td>check whether a forecast model has been started</td>
</tr>
</tbody>
</table>

Remarks and examples

A forecasting model is a system of equations that jointly determine the outcomes of one or more endogenous variables, whereby the term *endogenous* variables contrasts with *exogenous* variables, whose values are not determined by the interplay of the system’s equations. A model, in the context of the forecast commands, consists of

1. zero or more stochastic equations fit using Stata estimation commands and added to the current model using `forecast estimates`. These stochastic equations describe the behavior of endogenous variables.
2. zero or more nonstochastic equations (identities) defined using `forecast identity`. These equations often describe the behavior of endogenous variables that are based on accounting identities or adding-up conditions.
3. zero or more equations stored as coefficient vectors and added to the current model using `forecast coefvector`. Typically, you will fit your equations in Stata and use `forecast estimates` to add them to the model. `forecast coefvector` is used to add equations obtained elsewhere.
4. zero or more exogenous variables declared using `forecast exogenous`.
5. at least one stochastic equation or identity.
6. optional adjustments to be made to the variables of the model declared using `forecast adjust`.

One use of adjustments is to produce forecasts under alternative scenarios.

The `forecast` commands are designed to be easy to use, so without further ado, we dive headfirst into an example.
Example 1: Klein’s model

Example 3 of [R] reg3 shows how to fit Klein’s (1950) model of the U.S. economy using the three-stage least-squares estimator (3SLS). Here we focus on how to make forecasts from that model once the parameters have been estimated. In Klein’s model, there are seven equations that describe the seven endogenous variables. Three of those equations are stochastic relationships, while the rest are identities:

\[
\begin{align*}
    c_t &= \beta_0 + \beta_1 p_t + \beta_2 p_{t-1} + \beta_3 w_t + \epsilon_{1t} \\
    i_t &= \beta_4 + \beta_5 p_t + \beta_6 p_{t-1} + \beta_7 k_{t-1} + \epsilon_{2t} \\
    w_{Pt} &= \beta_8 + \beta_9 y_t + \beta_{10} y_{t-1} + \beta_{11} y_{r_t} + \epsilon_{3t}
\end{align*}
\]

The variables in the model are defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Consumption</td>
<td>endogenous</td>
</tr>
<tr>
<td>p</td>
<td>Private-sector profits</td>
<td>endogenous</td>
</tr>
<tr>
<td>wp</td>
<td>Private-sector wages</td>
<td>endogenous</td>
</tr>
<tr>
<td>wg</td>
<td>Government-sector wages</td>
<td>exogenous</td>
</tr>
<tr>
<td>w</td>
<td>Total wages</td>
<td>endogenous</td>
</tr>
<tr>
<td>i</td>
<td>Investment</td>
<td>endogenous</td>
</tr>
<tr>
<td>k</td>
<td>Capital stock</td>
<td>endogenous</td>
</tr>
<tr>
<td>y</td>
<td>National income</td>
<td>endogenous</td>
</tr>
<tr>
<td>g</td>
<td>Government spending</td>
<td>exogenous</td>
</tr>
<tr>
<td>t</td>
<td>Indirect bus. taxes + net exports</td>
<td>exogenous</td>
</tr>
<tr>
<td>yr</td>
<td>Time trend = Year − 1931</td>
<td>exogenous</td>
</tr>
</tbody>
</table>

Our model has four exogenous variables: government-sector wages (wg), government spending (g), a time-trend variable (yr), and, for simplicity, a variable that lumps indirect business taxes and net exports together (t). To make out-of-sample forecasts, we must populate those variables over the entire forecast horizon before solving our model. (We use the phrases “solve our model” and “obtain forecasts from our model” interchangeably.)

We will illustrate the entire process of fitting and forecasting our model, though our focus will be on the latter task. See [R] reg3 for a more in-depth look at fitting models like this one. Before we solve our model, we first estimate the parameters of the stochastic equations by loading the dataset and calling reg3:
The output from `reg3` indicates that we have a total of six endogenous variables even though our model in fact has seven. The discrepancy stems from (6) of our model. The capital stock variable ($k_t$) is a function of the endogenous investment variable and is therefore itself endogenous. However, $k_t$ does not appear in any of our model’s stochastic equations, so we did not declare it in the `endog()` option of `reg3`; from a purely estimation perspective, the contemporaneous value of the capital stock variable is irrelevant, though it does play a role in terms of solving our model. We next store the estimation results using `estimates store`:

```
. estimates store klein
```

Now we are ready to define our model using the `forecast` commands. We first tell Stata to initialize a new model; we will call our model `kleinmodel`:

```
. forecast create kleinmodel
Forecast model kleinmodel started.
```
The name you give the model mainly controls how output from \texttt{forecast} commands is labeled. More importantly, \texttt{forecast create} creates the internal data structures Stata uses to keep track of your model.

The next step is to add all the equations to the model. To add the three stochastic equations we fit using \texttt{reg3}, we use \texttt{forecast estimates}:

\begin{verbatim}
. forecast estimates klein
    Added estimation results from \texttt{reg3}.
    Forecast model kleinmodel now contains 3 endogenous variables.
\end{verbatim}

That command tells Stata to find the estimates stored as \texttt{klein} and add them to our model. \texttt{forecast estimates} uses those estimation results to determine that there are three endogenous variables (c, i, and \( w_p \)), and it will save the estimated parameters and other information that \texttt{forecast solve} will later need to obtain predictions for those variables. \texttt{forecast estimates} confirmed our request by reporting that the estimation results added were from \texttt{reg3}.

\texttt{forecast estimates} reports that our forecast model has three endogenous variables because our \texttt{reg3} command included three left-hand-side variables. The fact that we specified three additional endogenous variables in the \texttt{endog()} option of \texttt{reg3} so that \texttt{reg3} reports a total of six endogenous variables is irrelevant to \texttt{forecast}. All that matters is the number of left-hand-side variables in the model.

We also need to specify the four identities, equations (4) through (7), that determine the other four endogenous variables in our model. To do that, we use \texttt{forecast identity}:

\begin{verbatim}
. forecast identity y = c + i + g
    Forecast model kleinmodel now contains 4 endogenous variables.
. forecast identity p = y - t - wp
    Forecast model kleinmodel now contains 5 endogenous variables.
. forecast identity k = L.k + i
    Forecast model kleinmodel now contains 6 endogenous variables.
. forecast identity w = wg + wp
    Forecast model kleinmodel now contains 7 endogenous variables.
\end{verbatim}

You specify identities similarly to how you use the \texttt{generate} command, except that the left-hand-side variable is an endogenous variable in your model rather than a new variable you want to create in your dataset. Time-series operators often come in handy when specifying identities; here we expressed capital, a stock variable, as its previous value plus current-period investment, a flow variable. An identity defines an endogenous variable, so each time we use \texttt{forecast identity}, the number of endogenous variables in our forecast model increases by one.

Finally, we will tell Stata about the four exogenous variables. We do that with the \texttt{forecast exogenous} command:

\begin{verbatim}
. forecast exogenous wg
    Forecast model kleinmodel now contains 1 declared exogenous variable.
. forecast exogenous g
    Forecast model kleinmodel now contains 2 declared exogenous variables.
. forecast exogenous t
    Forecast model kleinmodel now contains 3 declared exogenous variables.
. forecast exogenous yr
    Forecast model kleinmodel now contains 4 declared exogenous variables.
\end{verbatim}
forecast keeps track of the exogenous variables that you declare using the forecast exogenous command and reports the number currently in the model. When you later use forecast solve, forecast verifies that these variables contain nonmissing data over the forecast horizon. In fact, we could have instead typed

```
.forecast exogenous wg g t yr
```

but to avoid confusing ourselves, we prefer to issue one command for each variable in our model.

Now Stata knows everything it needs to know about the structure of our model. klein2.dta in memory contains annual observations from 1920 to 1941. Before we make out-of-sample forecasts, we should first see how well our model works by comparing its forecasts with actual data. There are a couple of ways to do that. The first is to produce static forecasts. In static forecasts, actual values of all lagged variables that appear in the model are used. Because actual values will be missing beyond the last historical time period in the dataset, static forecasts can only forecast one period into the future (assuming only first lags appear in the model); for that reason, they are often called one-step-ahead forecasts. To obtain these one-step-ahead forecasts, we type

```
.forecast solve, prefix(s_) begin(1921) static
```

We specified `begin(1921)` to request that the first year for which forecasts are produced be 1921. Our model includes variables that are lagged one period; because our data start in 1920, 1921 is the first year in which we can evaluate all the equations of the model. If we did not specify the `begin(1921)` option, `forecast solve` would have started forecasting in 1941. By default, `forecast solve` looks for the earliest time period in which any of the endogenous variables contains a missing value and begins forecasting in that period. In `klein2.dta`, k is missing in 1941.

The header of the output confirms that we requested static forecasts for our model, and it indicates that it will produce forecasts from 1921 through 1941, the last year in our dataset. By default, `forecast solve` produces a status report in which the time period being forecast is displayed along with a dot for each iteration the equation solver performs. The footer of the output confirms that we forecast seven endogenous variables for 21 years.
The command we just typed will create seven new variables in our dataset, one for each endogenous variable, containing the static forecasts. Because we specified `prefix(s_)`, the seven new variables will be named `s_c`, `s_i`, `s_wp`, `s_y`, `s_p`, `s_k`, and `s_w`. Here we graph a subset of the variables and their forecasts:

Our static forecasts appear to fit the data relatively well. Had they not fit well, we would have to go back and reexamine the specification of our model. If the static forecasts are poor, then the dynamic forecasts that use previous periods’ forecast values are unlikely to work well either. On the other hand, even if the model produces good static forecasts, it may not produce accurate dynamic forecasts more than one or two periods into the future.

Another way to check how well a model forecasts is to produce dynamic forecasts for time periods in which observed values are available. Here we begin dynamic forecasts in 1936, giving us six years’ data with which to compare actual and forecast values and then graph our results:

```
. forecast solve, prefix(d_) begin(1936)
Computing dynamic forecasts for model kleinmodel.
```

Starting period: 1936
Ending period: 1941
Forecast prefix: d_
1936: ............................................
1937: ..........................................
1938: ............................................
1939: ............................................
1940: ............................................
1941: ............................................
Forecast 7 variables spanning 6 periods.
Most of the in-sample forecasts look okay, though our model was unable to predict the outsized increase in investment in 1936 and the sharp drop in 1938.

Our first example was particularly easy because all the endogenous variables appeared in levels. However, oftentimes the endogenous variables are better modeled using mathematical transformations such as logarithms, first differences, or percentage changes; transformations of the endogenous variables may appear as explanatory variables in other equations. The next few examples illustrate these complications.

Example 2: Models with transformed endogenous variables

hardware.dta contains hypothetical quarterly sales data from the Hughes Hardware Company, a huge regional distributor of building products. Hughes Hardware has three main product lines: dimensional lumber (dim), sheet goods such as plywood and fiberboard (sheet), and miscellaneous hardware, including fasteners and hand tools (misc). Based on past experience, we know that dimensional lumber sales are closely tied to the level of new home construction and that other product lines’ sales can be modeled in terms of the quantity of lumber sold. We are going to use the following set of equations to model sales of the three product lines:

\[
\begin{align*}
%\Delta \text{dim}_t &= \beta_{10} + \beta_{11} \ln(\text{starts}_t) + \beta_{12} %\Delta \text{gdp}_t + \beta_{13} \text{unrate}_t + \epsilon_{1t} \\
\text{sheet}_t &= \beta_{20} + \beta_{21} \text{dim}_t + \beta_{22} %\Delta \text{gdp}_t + \beta_{23} \text{unrate}_t + \epsilon_{2t} \\
\text{misc}_t &= \beta_{30} + \beta_{31} \text{dim}_t + \beta_{32} %\Delta \text{gdp}_t + \beta_{33} \text{unrate}_t + \epsilon_{3t}
\end{align*}
\]

Here \text{starts}_t represents the number of new homes for which construction began in quarter \( t \), \text{gdp}_t denotes real (inflation-adjusted) gross domestic product (GDP), and \text{unrate}_t represents the quarterly average unemployment rate. Our equation for \text{dim}_t is written in terms of percentage changes from quarter to quarter rather than in levels, and the percentage change in GDP appears as a regressor in each equation rather than the level of GDP itself. In our model, these three macroeconomic factors are exogenous, and here we will reserve the last few years’ data to make forecasts; in practice, we would need to make our own forecasts of these macroeconomic variables or else purchase a forecast.

We will approximate the percentage change variables by taking first differences of the natural logarithms of the respective underlying variables. In terms of estimation, this does not present any challenges. Here we load the dataset into memory, create the necessary log-transformed variables,
and fit the three equations using *regress* with the data through the end of 2009. We use *quietly* to suppress the output from *regress* to save space, and we store each set of estimation results as we go. In Stata, we type

```stata
use https://www.stata-press.com/data/r16/hardware, clear
(g Hughes Hardware sales data)
generate lndim = ln(dim)
generate lngdp = ln(gdp)
generate lnstarts = ln(starts)
quietly regress D.lndim lnstarts D.lngdp unrate if qdate <= tq(2009q4)
.estimates store dim
quietly regress sheet dim D.lngdp unrate if qdate <= tq(2009q4)
.estimates store sheet
quietly regress misc dim D.lngdp unrate if qdate <= tq(2009q4)
.estimates store misc
```

The equations for sheet goods and miscellaneous items do not present any challenges for *forecast*, so we proceed by creating a new forecast model named *salesfcast* and adding those two equations:

```stata
.forecast create salesfcast, replace
Forecast model kleinmodel ended.
Forecast model salesfcast started.
.forecast estimates sheet
Added estimation results from *regress*.
Forecast model salesfcast now contains 1 endogenous variable.
.forecast estimates misc
Added estimation results from *regress*.
Forecast model salesfcast now contains 2 endogenous variables.
```

The equation for dimensional lumber requires more finesse. First, because our dependent variable contains a time-series operator, we must use the *names()* option of *forecast estimates* to specify a valid name for the endogenous variable being added:

```stata
.forecast estimates dim, names(dlndim)
Added estimation results from *regress*.
Forecast model salesfcast now contains 3 endogenous variables.
```

We have entered the endogenous variable *dlndim* into our model, but it represents the left-hand-side variable of the regression equation we just added. That is, *dlndim* is the first difference of the logarithm of *dim*, the sales variable we ultimately want to forecast. We can specify an identity to reverse the first-differencing, providing us with a variable containing the logarithm of *dim*:

```stata
.forecast identity lndim = L.lndim + dlndim
Forecast model salesfcast now contains 4 endogenous variables.
```

Finally, we can specify another identity to obtain *dim* from *lndim*:

```stata
.forecast identity dim = exp(lndim)
Forecast model salesfcast now contains 5 endogenous variables.
```
Now we can solve the model. We will obtain dynamic forecasts starting in the first quarter of 2010, and we will use the `log(off)` option to suppress the iteration log:

```
.forecast solve, begin(tq(2010q1)) log(off)
```

Computing dynamic forecasts for model `salesfcast`.

Starting period: 2010q1
Ending period: 2012q3
Forecast prefix: f_
Forecast 5 variables spanning 11 periods.

We did not specify the `prefix()` or `suffix()` option, so by default, `forecast` prefixed our forecast variables with `f_`. The following graph illustrates our forecasts:

Our model performed well in 2010, but it did not forecast the pickup in sales that occurred in 2011 and 2012.

> Technical note

For more information about working with log-transformed variables, see the second technical note in [TS] `forecast estimates`.

The `forecast` commands can also be used to make forecasts for strongly balanced panel datasets. A panel dataset is strongly balanced when all the panels have the same number of observations, and the observations for different panels were all made at the same times. Our next example illustrates how to produce a forecast with panel data and highlights a couple of key assumptions one must make.

> Example 3: Forecasting a panel dataset

In the previous example, we mentioned that Hughes Hardware was a regional distributor of building products. In fact, Hughes Hardware operates in five states across the southern United States: Texas, Oklahoma, Louisiana, Arkansas, and Mississippi. The company is in the process of deciding whether it should open additional distribution centers or move existing ones to new locations. As part of the process, we need to make sales forecasts for each of the states the company serves.
To make our state-level forecasts, we will use essentially the same model that we did for the company-wide forecast, though we will also include state-specific effects. The model we will use is

\[
\begin{align*}
\%\Delta \text{dim}_i &= \beta_{10} + \beta_{11} \ln(\text{starts}_i) + \beta_{12} \text{rgspgrowth}_i + \beta_{13} \text{unrate}_i + u_{1i} + \epsilon_{1i} \\
\text{sheet}_i &= \beta_{20} + \beta_{21} \text{dim}_i + \beta_{22} \text{rgspgrowth}_i + \beta_{23} \text{unrate}_i + u_{2i} + \epsilon_{2i} \\
\text{misc}_i &= \beta_{30} + \beta_{31} \text{dim}_i + \beta_{32} \text{rgspgrowth}_i + \beta_{33} \text{unrate}_i + u_{3i} + \epsilon_{3i}
\end{align*}
\]

The subscript \(i\) indexes states, and we have replaced the \text{gdp} variable that was in our previous model with \text{rgspgrowth}, which measures the annual growth rate in real gross state product (GSP), the state-level analogue to national GDP. The GSP data are released only annually, so we have replicated the annual growth rate for all four quarterly observations in a given year. For example, \text{rgspgrowth} is about 5.3 for the four observations for the state of Texas in the year 2007; in 2007, Texas’s real GSP was 5.3% higher than in 2006.

The state-level error terms are \(u_{1i}, u_{2i},\) and \(u_{3i}\). Here we will use the fixed-effects estimator and fit the three equations via \texttt{xtreg}, \textit{fe}, again using data only through the end of 2009 so that we can examine how well our model forecasts. Our first task is to fit the three equations and store the estimation results. At the same time, we will also use \texttt{predict} to obtain the predicted fixed-effects terms. You will see why in just a moment. Because the regression results are not our primary concern here, we will use \texttt{quietly} to suppress the output.

In Stata, we type

```stata
. use https://www.stata-press.com/data/r16/statehardware, clear
  (Hughes state-level sales data)
. generate lndim = ln(dim)
. generate lnstarts = ln(starts)
. quietly xtreg D.lndim lnstarts rgspgrowth unrate if qdate <= tq(2009q4), fe
. predict dlndim_u, u
  (45 missing values generated)
. estimates store dim
. quietly xtreg sheet dim rgspgrowth unrate if qdate <= tq(2009q4), fe
. predict sheet_u, u
  (40 missing values generated)
. estimates store sheet
. quietly xtreg misc dim rgspgrowth unrate if qdate <= tq(2009q4), fe
. predict misc_u, u
  (40 missing values generated)
. estimates store misc
```

Having fit the model, we are almost ready to make forecasts. First, though, we need to consider how to handle the state-level error terms. If we simply created a forecast model, added our three estimation results, then called \texttt{forecast solve}, Stata would forecast \text{misc}_i, for example, as a function of \text{dim}_i, \text{rgspgrowth}_i, \text{unrate}_i, and the estimate of the constant term \(\beta_{30}\). However, our model implies that \text{misc}_i also depends on \(u_{3i}\) and the idiosyncratic error term \(\epsilon_{3i}\). We will ignore the idiosyncratic error for now (but see the discussion of simulations in \textit{TS} \texttt{forecast solve}).

By construction, \(u_{3i}\) has a mean of zero when averaged across all panels, but in general, \(u_{3i}\) is nonzero for any individual panel. Therefore, we should include it in our forecasts.

After you fit a model with \texttt{xtreg}, you can predict the panel-specific error component for the subset of observations in the estimation sample. Typically, \texttt{xtreg} is used in situations where the number of observations per panel \(T\) is modest. In those cases, the estimates of the panel-specific error components are likely to be “noisy” (analogous to estimating a sample mean with just a few observations). Often asymptotic analyses of panel-data estimators assume \(T\) is fixed, and in those cases, the estimators of the panel-specific errors are inconsistent.
However, in forecasting applications, the number of observations per panel is usually larger than in most other panel-data applications. With enough observations, we can have more confidence in the estimated panel-specific errors. If we are willing to assume that we have decent estimates of the panel-specific errors and that those panel-level effects will remain constant over the forecast horizon, then we can incorporate them into our forecasts. Because `predict` only provided us with estimates of the panel-level effects for the estimation sample, we need to extend them into the forecast horizon. An easy way to do that is to use `egen` to create a new set of variables:

```
    by state: egen dlndim_u2 = mean(dlndim_u)
    by state: egen sheet_u2 = mean(sheet_u)
    by state: egen misc_u2 = mean(misc_u)
```

We can use `forecast adjust` to incorporate these terms into our forecasts. The following commands define our forecast model, including the estimated panel-specific terms:

```
    forecast create statemodel, replace
    (Forecast model salesfcast ended.)
    Forecast model statemodel started.
    . forecast estimates dim, name(dlndim)
    Added estimation results from `xtreg`.
    Forecast model statemodel now contains 1 endogenous variable.
    . forecast adjust dlndim = dlndim + dlndim_u2
    Endogenous variable dlndim now has 1 adjustment.
    . forecast identity lndim = L.lndim + dlndim
    Forecast model statemodel now contains 2 endogenous variables.
    . forecast identity dim = exp(lndim)
    Forecast model statemodel now contains 3 endogenous variables.
    . forecast estimates sheet
    Forecast model statemodel now contains 4 endogenous variables.
    . forecast adjust sheet = sheet + sheet_u2
    Endogenous variable sheet now has 1 adjustment.
    . forecast estimates misc
    Forecast model statemodel now contains 5 endogenous variables.
    . forecast adjust misc = misc + misc_u2
    Endogenous variable misc now has 1 adjustment.
```

We used `forecast adjust` to perform our adjustment to `dlndim` immediately after we added those estimation results so that we would not forget to do so and before we used identities to obtain the actual `dim` variable. However, we could have specified the adjustment at any time. Regardless of when you specify an adjustment, `forecast solve` performs those adjustments immediately after the variable being adjusted is computed.
Now we can solve our model. Here we obtain dynamic forecasts beginning in the first quarter of 2010:

```
.forecast solve, begin(tq(2010q1))
```

Computing dynamic forecasts for model statemodel.

Starting period: 2010q1
Ending period: 2011q4
Number of panels: 5
Forecast prefix: f_

Solving panel 1
Solving panel 2
Solving panel 3
Solving panel 4
Solving panel 5

Forecast 5 variables spanning 8 periods for 5 panels.

Here is our state-level forecast for sheet goods:

```
<table>
<thead>
<tr>
<th>Year</th>
<th>AR</th>
<th>LA</th>
<th>MS</th>
<th>OK</th>
<th>TX</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2010</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2012</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```

Similar to our company-wide forecast, our state-level forecast failed to call the bottom in sales that occurred in 2011. Because our model missed the shift in sales momentum in every one of the five states, we would be inclined to go back and try respecifying one or more of the equations in our model. On the other hand, if our model forecasted most of the states well but performed poorly in just a few states, then we would first want to investigate whether any events in those states could account for the unexpected results.

**Technical note**

Stata also provides the `areg` command for fitting a linear regression with a large dummy-variable set and is designed for situations where the number of groups (panels) is fixed, while the number of observations per panel increases with the sample size. When the goal is to create a forecast model for panel data, you should nevertheless use `xtreg` rather than `areg`. The `forecast` commands require knowledge of the panel-data settings declared using `xtset` as well as panel-related estimation information saved by the other panel-data commands in order to produce forecasts with panel datasets.
In the previous example, none of our equations contained lagged dependent variables as regressors. If an equation did contain a lagged dependent variable, then one could use a dynamic panel-data (DPD) estimator such as *xtabond*, *xtdpd*, or *xtdpdsys*. DPD estimators are designed for cases where the number of observations per panel $T$ is small. As shown by Nickell (1981), the bias of the standard fixed- and random-effects estimators in the presence of lagged dependent variables is of order $1/T$ and is thus particularly severe when each panel has relatively few observations. Judson and Owen (1999) perform Monte Carlo experiments to examine the relative performance of different panel-data estimators in the presence of lagged dependent variables when used with panel datasets having dimensions more commonly encountered in macroeconomic applications. Based on their results, while the bias of the standard fixed-effects estimator (LSDV in their notation) is not inconsequential even when $T = 20$, for $T = 30$, the fixed-effects estimator does work as well as most alternatives. The only estimator that appreciably outperformed the standard fixed-effects estimator when $T = 30$ is the least-squares dummy variable corrected estimator (LSDVC in their notation). Bruno (2005) provides a Stata implementation of that estimator. Many datasets used in forecasting situations contain even more observations per panel, so the “Nickell bias” is unlikely to be a major concern.

In this manual entry, we have provided an overview of the *forecast* commands and provided several examples to get you started. The command-specific entries fill in the details.

**Video example**

Tour of forecasting

**References**


**Also see**

[TS] *tsset* — Declare data to be time-series data

[TS] *var* — Vector autoregressive models

[R] *ivregress* — Single-equation instrumental-variables regression

[R] *reg3* — Three-stage estimation for systems of simultaneous equations

[R] *regress* — Linear regression

[XT] *xtreg* — Fixed-, between-, and random-effects and population-averaged linear models

[XT] *xtset* — Declare data to be panel data
**Description**

`forecast adjust` specifies an adjustment to be applied to an endogenous variable in the model. Adjustments are typically used to produce alternative forecast scenarios or to incorporate outside information into a model. For example, you could use `forecast adjust` with a macroeconomic model to simulate the effect of an oil price shock whereby the price of oil spikes $50 higher than your model otherwise predicts in a given quarter.

**Quick start**

Adjust the endogenous variable `y` in forecast to account for the variable `shock` in 1990

`forecast adjust y = y + shock if year==1990`

Adjust the endogenous variable `y` in forecast to account for a structural change in its mean that occurred in year 2000

`forecast adjust y = y + 400000 if year > 2000`

**Syntax**

```
forecast adjust varname = exp [if] [in]
```

*varname* is the name of an endogenous variable that has been previously added to the model using `forecast estimates` or `forecast coefvector`.

*exp* represents a Stata expression; see [U] 13 Functions and expressions.

**Remarks and examples**

When preparing a forecast, you often want to produce several different scenarios. The baseline scenario is the default forecast that your model produces. It reflects the interplay among the equations and exogenous variables without any outside forces acting on the model. Users of forecasts often want answers to questions like “What happens to the economy if housing prices decline 10% more than your baseline forecast suggests they will?” or “What happens to unemployment and interest rates if tax rates increase?” `forecast adjust` lets you explore such questions by specifying alternative paths for one or more endogenous variables in your model.

**Example 1: Revisiting the Klein model**

In example 1 of [TS] `forecast`, we produced a baseline forecast for the classic Klein (1950) model. We noted that investment declined quite substantially in 1938. Suppose the government had a plan such as a one-year investment tax credit that it could enact in 1939 to stimulate investment. Based on discussions with accountants, tax experts, and business leaders, say this plan would encourage an additional $1 billion in investment in 1939. How would this additional investment affect the economy?
To answer this question, we first refit the Klein (1950) model from [TS] forecast using the data through 1938 and then obtain dynamic forecasts starting in 1939. We will prefix these forecast variables with bl_ to indicate they are the baseline forecasts. In Stata, we type

```
use https://www.stata-press.com/data/r16/klein2
quietly reg3 (c p L.p w) (i p L.p L.k) (wp y L.y yr) if year < 1939,
> endog(w p y) exog(t wg g)
estimates store klein
forecast create kleinmodel
Forecast model kleinmodel started.
forecast estimates klein
Added estimation results from reg3.
Forecast model kleinmodel now contains 3 endogenous variables.
forecast identity y = c + i + g
Forecast model kleinmodel now contains 4 endogenous variables.
forecast identity p = y - t - wp
Forecast model kleinmodel now contains 5 endogenous variables.
forecast identity k = L.k + i
Forecast model kleinmodel now contains 6 endogenous variables.
forecast identity w = wg + wp
Forecast model kleinmodel now contains 7 endogenous variables.
forecast exogenous wg
Forecast model kleinmodel now contains 1 declared exogenous variable.
forecast exogenous g
Forecast model kleinmodel now contains 2 declared exogenous variables.
forecast exogenous t
Forecast model kleinmodel now contains 3 declared exogenous variables.
forecast exogenous yr
Forecast model kleinmodel now contains 4 declared exogenous variables.
forecast solve, prefix(bl_) begin(1939)
Computing dynamic forecasts for model kleinmodel.
```

Starting period: 1939
Ending period: 1941
Forecast prefix: bl_

1939: .................................................................
       .................................................................
1940: .................................................................
       .................................................................
1941: .................................................................
       .................................................................
Forecast 7 variables spanning 3 periods.

To model our $1 billion increase in investment in 1939, we type

```
forecast adjust i = i + 1 if year == 1939
Endogenous variable i now has 1 adjustment.
```

While computing the forecasts for 1939, whenever forecast evaluates the equation for i, it will set i to be higher than it would otherwise be by 1. Now we re-solve our model using the prefix alt_ to indicate this is an alternative forecast:
222 forecast adjust — Adjust a variable by add factoring, replacing, etc.

`. forecast solve, prefix(alt_) begin(1939)
Computing dynamic forecasts for model kleinmodel.

Starting period: 1939
Ending period: 1941
Forecast prefix: alt_
1939: .......................................................... ..........................................................
1940: .......................................................... ..........................................................
1941: .......................................................... ..........................................................
Forecast 7 variables spanning 3 periods.

The following graph shows how investment and total income respond to this policy shock.

![Graph showing the effect of $1 billion investment tax credit](image)

Both investment and total income would be higher not just in 1939 but also in 1940; the higher capital stock implied by the additional investment raises total output (and hence income) even after the tax credit expires. Let’s look at these two variables in more detail:

`. list year bl_i alt_i bl_y alt_y if year >= 1938, sep(0)

<table>
<thead>
<tr>
<th>year</th>
<th>bl_i</th>
<th>alt_i</th>
<th>bl_y</th>
<th>alt_y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1938</td>
<td>-1.9</td>
<td>-1.9</td>
<td>60.9</td>
<td>60.9</td>
</tr>
<tr>
<td>1939</td>
<td>3.757227</td>
<td>6.276423</td>
<td>75.57685</td>
<td>80.71709</td>
</tr>
<tr>
<td>1940</td>
<td>7.971523</td>
<td>9.501909</td>
<td>89.67435</td>
<td>94.08473</td>
</tr>
<tr>
<td>1941</td>
<td>16.16375</td>
<td>16.20362</td>
<td>123.0809</td>
<td>124.238</td>
</tr>
</tbody>
</table>

Although we simulated a policy that we thought would encourage $1 billion in investment, investment in fact rises about $2.5 billion in 1939 according to our model. That is because higher investment raises total income, which also affects private-sector profits, which beget further changes in investment, and so on.

The investment multiplier in this example might strike you as implausibly large, but it highlights an important attribute of forecasting models. Studying each equation’s estimated coefficients in isolation can help to unveil some specification errors, but one must also consider how those equations interact.
It is possible to construct models in which each equation appears to be well specified, but the model nevertheless forecasts poorly or suggests unlikely behavior in response to policy shocks.

In the previous example, we applied a single adjustment to a single endogenous variable in a single time period. However, `forecast` allows you to specify `forecast adjust` multiple times with each endogenous variable, and many real-world policy simulations require adjustments to multiple variables. You can also consider policies that affect variables for multiple periods.

For example, suppose we wanted to see what would happen if our investment tax credit lasted two years instead of one. One way would be to use `forecast adjust` twice:

```
.forecast adjust i = i + 1 if year == 1939
.forecast adjust i = i + 1 if year == 1940
```

A second way would be to make that adjustment using one command:

```
.forecast adjust i = i + 1 if year == 1939 | year == 1940
```

To make adjustments lasting more than one or two periods, you should create an adjustment variable, which makes more sense. A third way to simulate our two-year tax credit is

```
.generate i_adj = 0
.replace i_adj = 1 if year == 1939 | year == 1940
.forecast adjust i = i + i_adj
```

So far in our discussion of `forecast adjust`, we have always shown an endogenous variable being adjusted by adding a number or variable to it. However, any valid expression is allowed on the right-hand side of the equals sign. If you want to explore the effects of a policy that will increase investment by 10% in 1939, you could type

```
.forecast adjust i = 1.1*i if year == 1939
```

If you believe investment will be $-2.0$ in 1939, you could type

```
.forecast adjust i = -2.0 if year == 1939
```

An alternative way to force forecasts of endogenous variables to take on prespecified values is discussed in example 1 of [TS] `forecast solve`.

### Stored results

`forecast adjust` stores the following in `r()`:

- **Macros**
  - `r(lhs)` left-hand-side (endogenous) variable
  - `r(rhs)` right-hand side of identity
  - `r(basenames)` base names of variables found on right-hand side
  - `r(fullnames)` full names of variables found on right-hand side

### Reference


### Also see

- [TS] `forecast` — Econometric model forecasting
- [TS] `forecast solve` — Obtain static and dynamic forecasts
**forecast clear — Clear current model from memory**

**Description**

*forecast clear* removes the current forecast model from memory.

**Syntax**

```
forecast clear
```

**Remarks and examples**

For an overview of the *forecast* commands, see [TS] *forecast*. This manual entry assumes you have already read that manual entry. *forecast* allows you to have only one model in memory at a time. You use *forecast clear* to remove the current model from memory. Forecast models themselves do not consume a significant amount of memory, so there is no need to clear a model from memory unless you intend to create a new one. An alternative to *forecast clear* is the replace option with *forecast create*.

Calling *forecast clear* when no forecast model exists in memory does not result in an error.

**Also see**

[TS] *forecast* — Econometric model forecasting

[TS] *forecast create* — Create a new forecast model
**forecast coeffvector — Specify an equation via a coefficient vector**

**Description**

`forecast coeffvector` adds equations that are stored as coefficient vectors to your forecast model. Typically, equations are added using `forecast estimates` and `forecast identity`. `forecast coeffvector` is used in less-common situations where you have a vector of parameters that represent a linear equation.

Most users of the `forecast` commands will not need to use `forecast coeffvector`. We recommend skipping this manual entry until you are familiar with the other features of `forecast`.

**Quick start**

Incorporate coefficient vector of the endogenous equation of \( y \) to be used by `forecast solve`

\[
\text{forecast coeffvector} \ y
\]

As above, but include the variance of the estimated parameters stored in matrix `mymat`

\[
\text{forecast coeffvector} \ y, \text{variance}(mymat)
\]

**Syntax**

\[
\text{forecast coeffvector} \ cname [, \ options]
\]

`cname` is a Stata matrix with one row.

<table>
<thead>
<tr>
<th><strong>options</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>variance(vname)</code></td>
<td>specify parameter variance matrix</td>
</tr>
<tr>
<td><code>errorvariance(ename)</code></td>
<td>specify additive error variance matrix</td>
</tr>
<tr>
<td><code>names(namelist[, replace])</code></td>
<td>use <code>namelist</code> for names of left-hand-side variables</td>
</tr>
</tbody>
</table>

**Options**

`variance(vname)` specifies that Stata matrix `vname` contains the variance matrix of the estimated parameters. This option only has an effect if you specify the `simulate()` option when calling `forecast solve` and request `sim_technique’s betas` or `residuals`. See `TS forecast solve`.

`errorvariance(ename)` specifies that the equations being added include an additive error term with variance \( ename \), where `ename` is the name of a Stata matrix. The number of rows and columns in `ename` must match the number of equations represented by coefficient vector `cname`. This option only has an effect if you specify the `simulate()` option when calling `forecast solve` and request `sim_technique’s errors` or `residuals`. See `TS forecast solve`.

225
names(namelist[, replace]) instructs `forecast coefvector` to use namelist as the names of the left-hand-side variables in the coefficient vector being added. By default, `forecast coefvector` uses the equation names on the column stripe of `cname`. You must use this option if any of the equation names stored with `cname` contains time-series operators.

**Remarks and examples**

For an overview of the `forecast` commands, see [TS] `forecast`. This manual entry assumes you have already read that manual entry. This manual entry also assumes that you are familiar with Stata’s matrices and the concepts of row and column names that can be attached to them; see [P] `matrix`. You use `forecast coefvector` to add endogenous variables to your model that are defined by linear equations, where the linear equations are stored in a coefficient (parameter) vector.

Remarks are presented under the following headings:

- *Introduction*
- *Simulations with coefficient vectors*

**Introduction**

`forecast coefvector` can be used to add equations that you obtained elsewhere to your model. For example, you might see the estimated coefficients for an equation in an article and want to add that equation to your model. User-written estimators that do not implement a `predict` command can also be included in forecast models via `forecast coefvector`. `forecast coefvector` can also be useful in situations where you want to simulate time-series data, as the next example illustrates.

▌ Example 1: A shock to an autoregressive process

Consider the following autoregressive process:

\[ y_t = 0.9y_{t-1} - 0.6y_{t-2} + 0.3y_{t-3} \]

Suppose \( y_t \) is initially equal to zero. How does \( y_t \) evolve in response to a one-unit shock at time \( t = 5 \)? We can use `forecast coefvector` to find out. First, we create a small dataset with time variable \( t \) and set our target variable \( y \) equal to zero:

```
. set obs 20
    number of observations (_N) was 0, now 20
. generate t = _n
. tsset t
    time variable: t, 1 to 20
    delta: 1 unit
. generate y = 0
```

Now let’s think about our coefficient vector. The only tricky part is in labeling the columns. We can represent the lagged values of \( y_t \) using time-series operators; there is just one equation, corresponding to variable \( y \). We can use `matrix coleq` to apply both variable and equation names to the columns of our matrix. In Stata, we type
. matrix y = (.9, -.6, 0.3)
. matrix coleq y = y:L.y y:L2.y y:L3.y
. matrix list y
   y[1,3]
     y: y: y:
     L. L2. L3.
     y y y
   r1 .9 -.6 .3

`forecast coefvector` ignores the row name of the vector being added (r1 here), so we can leave it as is. Next we create a forecast model and add y:

```
. forecast create
Forecast model started.
. forecast coefvector y
Forecast model now contains 1 endogenous variable.
```

To shock our system at \( t = 5 \), we can use `forecast adjust`:

```
. forecast adjust y = 1 in 5
Endogenous variable y now has 1 adjustment.
```

Now we can solve our model. Because our \( y \) variable is filled in for the entire dataset, `forecast solve` will not be able to automatically determine when forecasting should commence. We have three lags in our process, so we will start at \( t = 4 \). To reduce the amount of output, we specify `log(off)`:

```
. forecast solve, begin(4) log(off)
Computing dynamic forecasts for current model.
```

Starting period: 4
Ending period: 20
Forecast prefix: f_
Forecast 1 variable spanning 17 periods.

The graph shows our shock causing \( y \) to jump to 1 at \( t = 5 \). At \( t = 6 \), we can see that \( y = 0.9 \), and at \( t = 7 \), we can see that \( y = 0.9 \times 0.9 - 0.6 \times 1 = 0.21 \).
The previous example used a coefficient vector representing a single equation. However, coefficient vectors can contain multiple equations. For example, say we read an article and saw the following results displayed:

\[
\begin{align*}
  x_t &= 0.2 + 0.3x_{t-1} - 0.8z_t \\
  z_t &= 0.1 + 0.7z_{t-1} + 0.3x_t - 0.2x_{t-1}
\end{align*}
\]

We can add both equations at once to our forecast model. Again the key is in labeling the columns. `forecast coefvector` understands `_cons` to mean a constant term, and it looks at the equation names on the vector’s columns to determine how many equations there are and to what endogenous variables they correspond:

```plaintext
. matrix eqvector = (0.2, 0.3, -0.8, 0.1, 0.7, 0.3, -0.2)
. matrix coleq eqvector = x:_cons x:L.x x:y y:_cons y:L.y y:x y:L.x
. matrix list eqvector
```

We could then type

```
. forecast coefvector y
```

to add our coefficient vector to a model.

Just like with estimation results whose left-hand-side variables contain time-series operators, if any of the equation names of the coefficient vector being added contains time-series operators, you must use the `names()` option of `forecast coefvector` to specify alternative names.

### Simulations with coefficient vectors

The `forecast solve` command provides the option `simulate(sim_technique, ...)` to perform stochastic simulations and obtain measures of forecast uncertainty. How `forecast solve` handles coefficient vectors when performing these simulations depends on the options provided with `forecast coefvector`. There are four cases to consider:

1. You specify neither `variance()` nor `errorvariance()` with `forecast coefvector`. You have provided no measures of uncertainty with this coefficient vector. Therefore, `forecast solve` treats it like an identity. No random errors or residuals are added to this coefficient vector’s linear combination, nor are the coefficients perturbed in any way.

2. You specify `variance()` but not `errorvariance()`. The `variance()` option provides the covariance matrix of the estimated parameters in the coefficient vector. Therefore, the coefficient vector is taken to be stochastic. If you request `sim_technique betas`, this coefficient vector is assumed to be distributed multivariate normal with a mean equal to the original value of the vector and covariance matrix as specified in the `variance()` option, and random draws are taken from this distribution. If you request `sim_technique residuals`, randomly chosen static residuals are added to this coefficient vector’s linear combination. Because you did not specify a covariance matrix for the error terms with the `errorvariance()` option, `sim_technique errors` cannot draw random errors for this coefficient vector’s linear combination, so `sim_technique errors` has no impact on the equations.
3. You specify `errorvariance()` but not `variance()`. Because you specified a covariance matrix for the assumed additive error term, the equations represented by this coefficient vector are stochastic. If you request `sim_technique residuals`, randomly chosen static residuals are added to this coefficient vector’s linear combination. If you request `sim_technique errors`, multivariate normal errors with mean zero and covariance matrix as specified in the `errorvariance()` option are added during the simulations. However, specifying `sim_technique betas` does not affect the equations because there is no covariance matrix associated with the coefficients.

4. You specify both `variance()` and `errorvariance()`. The equations represented by this coefficient vector are stochastic, and `forecast solve` treats the coefficient vector just like an estimation result. `sim_technique’s betas, residuals, and errors` all work as expected.

**Methods and formulas**

Let $\beta$ denote the $1 \times k$ coefficient vector being added. Then the matrix specified in the `variance()` option must be $k \times k$. Row and column names for that matrix are ignored.

Let $m$ denote the number of equations represented by $\beta$. That is, if $\beta$ is stored as Stata matrix `beta` and local macro `m` is to hold the number of equations, then in Stata parlance,

```stata
    . local eqnames : coleq beta
    . local eq : list uniq eqnames
    . local m : list sizeof eq
```

Then the matrix specified in the `errorvariance` option must be $m \times m$. Row and column names for that matrix are ignored.

**Also see**

- [TS] `forecast` — Econometric model forecasting
- [TS] `forecast solve` — Obtain static and dynamic forecasts
- [P] `matrix` — Introduction to matrix commands
- [P] `matrix rownames` — Name rows and columns
**Title**

`forecast create` — Create a new forecast model

---

**Description**

`forecast create` creates a new forecast model in Stata.

**Quick start**

Start a forecast model called `myforecast`

```
forecast create myforecast
```

As above, but clear the existing model `myforecast` from memory if it exists

```
forecast create myforecast, replace
```

**Syntax**

```
forecast create [name] [, replace]
```

`name` is an optional name that can be given to the model. `name` must follow the naming conventions described in [U] 11.3 Naming conventions.

**Option**

`replace` causes Stata to clear the existing model from memory before creating `name`. You may have only one model in memory at a time. By default, `forecast create` issues an error message if another model is already in memory.

**Remarks and examples**

For an overview of the `forecast` commands, see [TS] `forecast`. This manual entry assumes you have already read that manual entry. The `forecast create` command creates a new forecast model in Stata. You must create a model before you can add equations or solve it. You can have only one model in memory at a time.

You may optionally specify a name for your model. That name will appear in the output produced by the various `forecast` subcommands.
Example 1

Here we create a model named `salesfcast`:

```
. forecast create salesfcast
Forecast model salesfcast started.
```

Technical note

Warning: Do not type `clear all`, `clear mata`, or `clear results` after creating a forecast model with `forecast create` unless you intend to remove your forecast model. Typing `clear all` or `clear mata` eliminates the internal structures used to store your forecast model. Typing `clear results` clears all estimation results from memory. If your forecast model includes estimation results that rely on the ability to call `predict`, you will not be able to solve your model.

Also see

[TS] `forecast` — Econometric model forecasting

[TS] `forecast clear` — Clear current model from memory
### forecast describe — Describe features of the forecast model

**Description**

forecast describe displays information about the forecast model currently in memory. For example, you can obtain information regarding all the endogenous or exogenous variables in the model, the adjustments used for alternative scenarios, or the solution method used. Typing `forecast describe` without specifying a particular aspect of the model is equivalent to typing `forecast describe` for every available aspect and can result in more output than you want, particularly if you also request a detailed description.

**Quick start**

- Display information about the estimates in the current forecast
  ```
  forecast describe estimates
  ```

- Display information about coefficient vectors
  ```
  forecast describe coefvector
  ```

- Display endogenous variables defined by identities
  ```
  forecast describe identity
  ```

- Display names of declared exogenous variables
  ```
  forecast describe exogenous
  ```

- Display information about the solution method used
  ```
  forecast describe solve
  ```

- Display information about endogenous variables
  ```
  forecast describe endogenous
  ```

- All the above
  ```
  forecast describe
  ```
## Syntax

*Describe the current forecast model*

```
forecast describe [, options]
```

*Describe particular aspects of the current forecast model*

```
forecast describe aspect [, options]
```

<table>
<thead>
<tr>
<th>aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimates</td>
<td>estimation results</td>
</tr>
<tr>
<td>coefvector</td>
<td>coefficient vectors</td>
</tr>
<tr>
<td>identity</td>
<td>identities</td>
</tr>
<tr>
<td>exogenous</td>
<td>declared exogenous variables</td>
</tr>
<tr>
<td>adjust</td>
<td>adjustments to endogenous variables</td>
</tr>
<tr>
<td>solve</td>
<td>forecast solution information</td>
</tr>
<tr>
<td>endogenous</td>
<td>all endogenous variables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>brief</td>
<td>provide a one-line summary</td>
</tr>
<tr>
<td><em>detail</em></td>
<td>provide more-detailed information</td>
</tr>
</tbody>
</table>

* Specifying *detail* provides no additional information with *aspects* exogenous, endogenous, and solve.

### Options

*brief* requests that *forecast describe* produce a one-sentence summary of the aspect specified. For example, *forecast describe exogenous, brief* will tell you just the current forecast model’s name and the number of exogenous variables in the model.

*detail* requests a more-detailed description of the aspect specified. For example, typing *forecast describe estimates* lists all the estimation results added to the model using *forecast estimates*, the estimation commands used, and the number of left-hand-side variables in each estimation result. When you specify *forecast describe estimates, detail*, the output includes a list of all the left-hand-side variables entered with *forecast estimates*.

### Remarks and examples

For an overview of the *forecast* commands, see [TS] *forecast*. This manual entry assumes you have already read that manual entry. *forecast describe* displays information about the forecast model currently in memory. You can obtain either all the information at once or information about individual aspects of your model, whereby we use the word “aspect” to refer to, for example, just the estimation results, identities, or solution information.
Example 1

In example 1 of [TS] forecast, we created and forecasted Klein’s (1950) model of the U.S. economy. Here we obtain information about all the endogenous variables in the model:

```
. forecast describe endogenous
Forecast model kleinmodel contains 7 endogenous variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Source</th>
<th># adjustments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. c</td>
<td>estimates</td>
<td>0</td>
</tr>
<tr>
<td>2. i</td>
<td>estimates</td>
<td>0</td>
</tr>
<tr>
<td>3. wp</td>
<td>estimates</td>
<td>0</td>
</tr>
<tr>
<td>4. y</td>
<td>identity</td>
<td>0</td>
</tr>
<tr>
<td>5. p</td>
<td>identity</td>
<td>0</td>
</tr>
<tr>
<td>6. k</td>
<td>identity</td>
<td>0</td>
</tr>
<tr>
<td>7. w</td>
<td>identity</td>
<td>0</td>
</tr>
</tbody>
</table>
```

As we mentioned in [TS] forecast, there are seven endogenous variables in this model. Three of those variables (c, i, and wp) were left-hand-side variables in equations we fitted and added to our forecast model with forecast estimates. The other four variables were defined by identities added with forecast identity. The right-hand column of the table indicates that none of our endogenous variables contains adjustments specified using forecast adjust.

We can obtain more information about the estimated equations in our model using forecast describe estimates:

```
. forecast describe estimates, detail
Forecast model kleinmodel contains 1 estimation result:

<table>
<thead>
<tr>
<th>Estimation result</th>
<th>Command</th>
<th>LHS variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. klein</td>
<td>reg3</td>
<td>c, i, wp</td>
</tr>
</tbody>
</table>
```

Our model has one estimation result, klein, containing results produced by the reg3 command. If we had not specified the detail option, forecast describe estimates would have simply stated the number of left-hand-side variables (3) rather than listing them.
At the end of example 1 in [TS] forecast, we obtained dynamic forecasts beginning in 1936. Here we obtain information about the solution:

```
. forecast describe solve
Forecast model kleinmodel has been solved:

<table>
<thead>
<tr>
<th>Forecast horizon</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
<td>1936</td>
</tr>
<tr>
<td>End</td>
<td>1941</td>
</tr>
<tr>
<td>Number of periods</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Forecast variables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>d_</td>
</tr>
<tr>
<td>Number of variables</td>
<td>7</td>
</tr>
<tr>
<td>Storage type</td>
<td>float</td>
</tr>
<tr>
<td>Type of forecast</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique</td>
<td>Damped Gauss-Seidel (0.200)</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>500</td>
</tr>
<tr>
<td>Tolerance for function values</td>
<td>1.0e-09</td>
</tr>
<tr>
<td>Tolerance for function zero</td>
<td>(not applicable)</td>
</tr>
</tbody>
</table>

We obtain information about the forecast horizon, how the variables holding our forecasts were created and stored, and the solution technique used. If we had used the simulate() option with forecast solve, we would have obtained information about the types of simulations performed and the variables used to hold the results.

### Stored results

When you specify option brief, only a limited number of results are stored. In the tables below, a superscript \(B\) indicates results that are available even after brief is specified. forecast coefvector saves certain results only if detail is specified; these are indicated by superscript \(D\). Typing forecast describe without specifying an aspect does not return any results.

**forecast describe estimates** stores the following in \(r()\):

**Scalars**
- \(r(n\text{-estimates})^B\) number of estimation results
- \(r(n\text{-lhs})\) number of left-hand-side variables defined by estimation results

**Macros**
- \(r(model)^B\) name of forecast model, if named
- \(r(lhs)\) left-hand-side variables
- \(r(estimates)\) names of estimation results

**forecast describe identity** stores the following in \(r()\):

**Scalars**
- \(r(n\text{-identities})^B\) number of identities

**Macros**
- \(r(model)^B\) name of forecast model, if named
- \(r(lhs)\) left-hand-side variables
- \(r(identities)\) list of identities
**forecast describe coefvector** stores the following in \( r() \):

Scalars
- \( r(\text{n_coefvectors}) \): number of coefficient vectors
- \( r(\text{n_lhs}) \): number of left-hand-side variables defined by coefficient vectors

Macros
- \( r(\text{model}) \): name of forecast model, if named
- \( r(\text{lhs}) \): left-hand-side variables
- \( r(\text{rhs}) \): right-hand-side variables
- \( r(\text{names}) \): names of coefficient vectors
- \( r(\text{Vnames}) \): names of variance matrices ("." if not specified)
- \( r(\text{Enames}) \): names of error variance matrices ("." if not specified)

**forecast describe exogenous** stores the following in \( r() \):

Scalars
- \( r(\text{n_exogenous}) \): number of declared exogenous variables

Macros
- \( r(\text{model}) \): name of forecast model, if named
- \( r(\text{exogenous}) \): declared exogenous variables

**forecast describe endogenous** stores the following in \( r() \):

Scalars
- \( r(\text{n_endogenous}) \): number of endogenous variables

Macros
- \( r(\text{model}) \): name of forecast model, if named
- \( r(\text{varlist}) \): endogenous variables
- \( r(\text{source_list}) \): sources of endogenous variables (estimates, identity, coefvector)
- \( r(\text{adjust_cnt}) \): number of adjustments per endogenous variable

**forecast describe solve** stores the following in \( r() \):

Scalars
- \( r(\text{periods}) \): number of periods forecast per panel
- \( r(\text{Npanels}) \): number of panels forecast
- \( r(\text{Nvar}) \): number of forecast variables
- \( r(\text{damping}) \): damping parameter for damped Gauss–Seidel
- \( r(\text{maxiter}) \): maximum number of iterations
- \( r(\text{vtolerance}) \): tolerance for forecast values
- \( r(\text{ztolerance}) \): tolerance for function zero
- \( r(\text{sim_nreps}) \): number of simulations

Macros
- \( r(\text{solved}) \): solved, if the model has been solved
- \( r(\text{model}) \): name of forecast model, if named
- \( r(\text{actuals}) \): actuals, if specified with \texttt{forecast solve}
- \( r(\text{double}) \): double, if specified with \texttt{forecast solve}
- \( r(\text{static}) \): static, if specified with \texttt{forecast solve}
- \( r(\text{begin}) \): first period in forecast horizon
- \( r(\text{end}) \): last period in forecast horizon
- \( r(\text{technique}) \): solver technique
- \( r(\text{sim_technique}) \): specified \texttt{sim_technique}
- \( r(\text{prefix}) \): forecast variable prefix
- \( r(\text{suffix}) \): forecast variable suffix
- \( r(\text{sim_prefix}_{i}) \): \( i \)th simulation statistic prefix
- \( r(\text{sim_suffix}_{j}) \): \( j \)th simulation statistic suffix
- \( r(\text{sim_stat}_{i}) \): \( i \)th simulation statistic
forecast describe adjust stores the following in r():

Scalars
- r(n_adjustments) \( B \) total number of adjustments
- r(n_adjust_vars) \( B \) number of variables with adjustments

Macros
- r(model) \( B \) name of forecast model, if named
- r(varlist) variables with adjustments
- r(adjust_cnt) number of adjustments per endogenous variable
- r(adjust_list) list of adjustments

Reference

Also see

[Ts] forecast — Econometric model forecasting
[Ts] forecast list — List forecast commands composing current model
**Title**

**forecast drop — Drop forecast variables**

### Description

*forecast drop* drops variables previously created by *forecast solve*.

### Quick start

Remove all variables created by *forecast solve* from the current dataset

```
forecast drop
```

Remove only forecast variables starting with *f_*

```
forecast drop, prefix(f_)
```

Remove only forecast variables ending with *_f*

```
forecast drop, suffix(_f)
```

### Syntax

```
forecast drop [ , options ]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>prefix(string)</em></td>
<td>specify prefix for forecast variables</td>
</tr>
<tr>
<td><em>suffix(string)</em></td>
<td>specify suffix for forecast variables</td>
</tr>
</tbody>
</table>

* You can specify *prefix()* or *suffix()* but not both.

### Options

*prefix(string)* and *suffix(string)* specify either a name prefix or a name suffix that will be used to identify forecast variables to be dropped. You may specify *prefix()* or *suffix()* but not both.

By default, *forecast drop* removes all forecast variables produced by the previous invocation of *forecast solve*.

Suppose, however, that you previously specified the simulate() option with *forecast solve* and wish to remove variables containing simulation results but retain the variables containing the point forecasts. Then you can use the *prefix()* or *suffix()* option to identify the simulation variables you want dropped.
Remarks and examples

For an overview of the forecast commands, see [TS] forecast. This manual entry assumes you have already read that manual entry. forecast drop safely removes variables previously created using forecast solve. Say you previously solved your model and created forecast variables that were suffixed with \_f. Do not type

```
    . drop *_f
```

to remove those variables from the dataset. Rather, type

```
    . forecast drop
```

The former command is dangerous: Suppose you were given the dataset and asked to produce the forecast. The person who previously worked with the dataset created other variables that ended with \_f. Using drop would remove those variables as well. forecast drop removes only those variables that were previously created by forecast solve based on the model in memory.

If you do not specify any options, forecast drop removes all the forecast variables created by the current model, including the variables that contain the point forecasts as well as any variables that contain simulation results specified by the simulate() option with forecast solve. Suppose you had typed

```
    . forecast solve, prefix(s_) simulate(betas, statistic(stddev, prefix(sd_)))
```

Then if you type

```
    . forecast drop, prefix(sd_)
```

forecast drop will remove the variables containing the standard deviations of the forecasts and will leave the variables containing the point forecasts (prefixed with s_) untouched.

forecast drop does not exit with an error if a variable it intends to drop does not exist in the dataset.

Stored results

forecast drop stores the following in r():

Scalars

\r(n\_dropped) number of variables dropped

Also see

[TS] forecast — Econometric model forecasting

[TS] forecast solve — Obtain static and dynamic forecasts
**Title**

*forecast estimates* — Add estimation results to a forecast model

### Description

*forecast estimates* adds estimation results to the forecast model currently in memory. You must first create a new model using *forecast create* before you can add estimation results with *forecast estimates*. After estimating the parameters of an equation or set of equations, you must use *estimates store* to store the estimation results in memory or use *estimates save* to save them on disk before adding them to the model.

### Quick start

Add estimation results stored in *myestimates* to the forecast model in memory

```
forecast estimates myestimates
```

As above, but specify the prediction produced by *predict, pr outcome(#1)*

```
forecast estimates myestimates, predict("pr outcome(#1)")
```

Add estimates from *var* estimation stored in memory as *varest*

```
forecast estimates varest
```

Also add the second estimation result saved on disk as *notcurrent.ster* to the forecast model

```
forecast estimates using notcurrent, number(2)
```
**Syntax**

Add estimation result currently in memory to model

```
forecast estimates name [ , options ]
```

`name` is the name of a stored estimation result; see [R] estimates store.

Add estimation result currently saved on disk to model

```
forecast estimates using filename [ , number(#) options ]
```

`filename` is an estimation results file created by `estimates save`; see [R] estimates save. If no file extension is specified, `.ster` is assumed.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict(<code>p_options</code>)</td>
<td>call predict using <code>p_options</code></td>
</tr>
<tr>
<td>names(<code>namelist</code>[, replace])</td>
<td>use <code>namelist</code> for names of left-hand-side variables</td>
</tr>
<tr>
<td>advise</td>
<td>advise whether estimation results can be dropped from memory</td>
</tr>
</tbody>
</table>

**Options**

`predict(`p_options`)` specifies the `predict` options to use when predicting the dependent variables. For a single-equation estimation command, you simply specify the appropriate options to pass to `predict`. If multiple options are required, enclose them in quotation marks:

```
. forecast estimates ..., predict("pr outcome(#1)")
```

For a multiple-equation estimation command, you can either specify one set of options that will be applied to all equations or specify `p` options, where `p` is the number of endogenous variables being added. If multiple options are required for each equation, enclose each equation’s options in quotes:

```
. forecast estimates ..., predict("pr eq(#1)" "pr eq(#2)")
```

If you do not specify the `eq()` option for any of the equations, `forecast` automatically includes it for you.

If you are adding results from a linear estimation command that `forecast` recognizes as one whose predictions can be calculated as $x_t'\beta$, do not specify the `predict()` option, because this will slow `forecast`’s computation time substantially. Use the `advise` option to determine whether `forecast` needs to call `predict`.

If you do not specify any `predict` options, `forecast` uses the default type of prediction for the command whose results are being added.

`names(`namelist`[, replace])` instructs `forecast estimates` to use `namelist` as the names of the left-hand-side variables in the estimation result being added. You must use this option if any of the left-hand-side variables contains time-series operators. By default, `forecast estimates` uses the names stored in the `e(depvar)` macro of the results being added.

`forecast estimates` creates a new variable in the dataset for each element of `namelist`. If a variable of the same name already exists in your dataset, `forecast estimates` exits with an error unless you specify the `replace` option, in which case existing variables are overwritten.
advise requests that *forecast estimates* report a message indicating whether the estimation results being added can be removed from memory. This option is useful if you expect your model to contain more than 300 sets of estimation results, the maximum number that Stata allows you to store in memory; see [R] *Limits*. This option also provides an indication of the speed with which the model can be solved: *forecast* executes much more slowly with estimation results that must remain in memory.

`number(#),` for use with *forecast estimates* using, specifies that the #th set of estimation results from *filename* be loaded. This assumes that multiple sets of estimation results have been saved in *filename*. The default is `number(1)`. See [R] *estimates save* for more information on saving multiple sets of estimation results in a single file.

**Remarks and examples**

For an overview of the *forecast* commands, see [TS] *forecast*. This manual entry assumes you have already read that manual entry. *forecast estimates* adds stochastic equations previously fit by Stata estimation commands to a forecast model.

Remarks are presented under the following headings:

- **Introduction**
- The advise option
- Using saved estimation results
- The predict option
- Forecasting with ARIMA models

**Introduction**

After you fit an equation that will become a part of your model, you must use either `estimates store` to store the estimation results in memory or `estimates save` to save the estimation results to disk. Then you can use *forecast estimates* to add that equation to your model.

We usually refer to “equation” in the singular, but of course, you can also use a multiple-equation estimation command to fit several equations at once and add them to the model. When we discuss adding a stochastic equation to a model, we really mean adding a single estimation result.

In this discussion, we also need to make a distinction between making a forecast and obtaining a prediction. We use the word “predict” to refer to the process of obtaining a fitted value for a single equation, just as you can use the `predict` command to obtain fitted values, residuals, or other statistics after fitting a model with an estimation command. We use the word “forecast” to mean finding a solution to the complete set of equations that compose the forecast model. The iterative techniques we use to solve the model and produce forecasts require that we be able to obtain predictions from each of the equations in the model.

**Example 1: A simple example**

Here we illustrate how to add estimation results from a regression model in which none of the left-hand-side variables contains time-series operators or mathematical transformations. We use quietly with the estimation command because the output is not relevant here. We type
. use https://www.stata-press.com/data/r16/klein2
. quietly reg3 (c p L.p w) (i p L.p L.k) (wp y L.y yr), endog(w p y) exog(t wg g)
. estimates store klein
. forecast create kleinmodel
Forecast model kleinmodel started.
. forecast estimates klein
Added estimation results from `reg3'.
Forecast model kleinmodel now contains 3 endogenous variables.

`forecast estimates' indicated that three endogenous variables were added to the forecast model.
That is because we specified three equations in our call to `reg3'. As we mentioned in example 1 in
[TS] `forecast', the `endog()' option of `reg3' has no bearing on `forecast'. All that matters are the
three left-hand-side variables.

Technical note

When you add an estimation result to your forecast model, `forecast' looks at the macro `e(depvar)
` to determine the endogenous variables being added. If that macro is empty, `forecast' tries a few
other macros to account for nonstandard commands. The number of endogenous variables being added
to the model is based on the number of words found in the macro containing the dependent variables.

You can fit equations with the `D.' and `S.' first- and seasonal-difference time-series operators
adorning the left-hand-side variables, but in those cases, when you add the equations to the model,
you must use the `names()' option of `forecast estimates'. When you specify `names(namelist),
`forecast estimates' uses `namelist' as the names of the newly declared endogenous variables and
ignores what is in `e(depvar)'. Moreover, `forecast' does not automatically “undo” the operators on
left-hand-side variables. For example, you might fit a regression with `D.x' as the regressand and then
add it to the model using `forecast estimates ... , name(Dx)'. In that case, `forecast' will solve
the model in terms of `Dx'. You must add an identity to convert `Dx' to the corresponding level variable
`x', as the next example illustrates.

Of course, you are free to use the `D.', `S.', and `L.' time-series operators on endogenous variables
when they appear on the right-hand sides of equations. It is only when `D.' or `S.' appears on the
left-hand side that you must use the `names()' option to provide alternative names for them. You
cannot add equations to models for which the `L.' operator appears on left-hand-side variables. You
cannot use the `F.' forward operator anywhere in forecast models.

Example 2: Differenced and log-transformed dependent variables

Consider the following model:

\begin{align}
\text{D.logC} &= \beta_{10} + \beta_{11}\text{D.logW} + \beta_{12}\text{D.logY} + u_{1t} \\
\text{logW} &= \beta_{20} + \beta_{21}\text{L.logW} + \beta_{22}M + \beta_{23}\text{logY} + \beta_{24}\text{logC} + u_{2t}
\end{align}

Here \text{logY} and \text{M} are exogenous variables, so we will assume they are filled in over the forecast
horizon before solving the model. Ultimately, we are interested in forecasting \text{C} and \text{W}. However,
the first equation is specified in terms of changes in the logarithm of \text{C}, and the second equation is
specified in terms of the logarithm of \text{W}.
We will refer to variables and transformations like $\log C$, $D.\log C$, and $C$ as “related” variables because they are related to one another by simple mathematical functions. Including the related variables, we in fact have a five-equation model with two stochastic equations and three identities:

$$
\begin{align*}
\text{dlog}C &= \beta_{10} + \beta_{11}D.\log W + \beta_{12}D.\log Y + u_{1t} \\
\log C &= \text{L.log}C + \text{dlog}C \\
C &= \exp(\log C) \\
\log W &= \beta_{20} + \beta_{21}\text{L.log}W + \beta_{22}M + \beta_{23}\log Y + \beta_{24}\log C + u_{2t} \\
W &= \exp(\log W)
\end{align*}
$$

To fit (1) and (2) in Stata and create a forecast model, we type

```
. use https://www.stata-press.com/data/r16/fcestimates, clear
   (1978 Automobile Data)
. quietly regress D.\log C D.\log W D.\log Y
. estimates store dlogceq
. quietly regress \log W \text{L.log}W M \log Y \log C
. estimates store logweq
. forecast create cwmodel, replace
   (Forecast model kleinmodel ended.)
Forecast model cwmodel started.
. forecast estimates dlogceq, names(dlogC)
   Added estimation results from `regress'.
Forecast model cwmodel now contains 1 endogenous variable.
. forecast identity \log C = \text{L.log}C + \text{dlog}C
   Forecast model cwmodel now contains 2 endogenous variables.
. forecast identity C = \exp(\log C)
   Forecast model cwmodel now contains 3 endogenous variables.
. forecast estimates logweq
   Added estimation results from `regress'.
Forecast model cwmodel now contains 4 endogenous variables.
. forecast identity W = \exp(\log W)
   Forecast model cwmodel now contains 5 endogenous variables.
```

Because the left-hand-side variable in (1) contains a time-series operator, we had to use the names() option of forecast estimates when adding that equation’s estimation results to our forecast model. Here we named this endogenous variable dlogC. We then added the other four equations to our model. In general, when we have a set of related variables, we prefer to specify the identities right after we add the stochastic equation so that we do not forget about them.

⚠️ Technical note

In the previous example, we “undid” the log-transformations by simply exponentiating the logarithmic variable. However, that is only an approximation that does not work well in many applications. Suppose we fit the linear regression model

$$
\ln y_t = x'_t \beta + u_t
$$

...
where \( u_t \) is a zero-mean regression error term. Then \( E(y_t|x_t) = \exp(x_t'\beta) \times E\{\exp(u_t)\} \). Although \( E(u_t) = 0 \), Jensen’s inequality suggests that \( E\{\exp(u_t)\} \neq 1 \), implying that we cannot predict \( y_t \) by simply taking the exponential of the linear prediction \( x_t'\beta \).

If we assume that \( u_t \sim N(0, \sigma^2) \), then \( E\{\exp(u_t)\} = \exp(\sigma^2/2) \). Moreover, many estimation commands like `regress` provide an estimate \( \hat{\sigma}^2 \) of \( \sigma^2 \), so for regression models that contain a logarithmic dependent variable, we can obtain better forecasts for the dependent variable in levels if we approximate \( E\{\exp(u_t)\} \) as \( \exp(\hat{\sigma}^2/2) \). Suppose we run the regression

```
. regress lny x1 x2 x3
. estimates store myreg
```

then we could add \( \text{lny} \) and \( y \) as endogenous variables like this:

```
. forecast estimates lny
. forecast identity y = exp(lny)*'=e(rmse)^2 / 2'
```

In the second command, Stata will first evaluate the expression ‘\( =e(rmse)^2 / 2 \)’ and replace it with its numerical value. After `regress`, the macro `e(rmse)` contains the square root of the estimate of \( \hat{\sigma}^2 \), so the value of this expression will be our estimate of \( E\{\exp(u_t)\} \). Then `forecast` will forecast \( y \) as the product of this number and \( \exp(\text{lny}) \). Here we had to use a macro expression including an equals sign to force Stata to evaluate the expression immediately and obtain the expression’s value. Identities are not associated with estimation results, so as soon as we used another estimation command or restored some other estimation results (perhaps unknowingly by invoking `forecast solve`), our reference to `e(rmse)` would no longer be meaningful. See [U] 18.3.8 Macro expressions for more information on macro evaluation.

Another alternative would be to use Duan’s (1983) smearing technique. Stata code for this is provided in Cameron and Trivedi (2010).

A third alternative is to use the generalized linear model (GLM) as implemented by the `glm` command with a log-link function. In a GLM framework, we would be modeling \( \ln(E(y_t)) \) rather than \( E\{\ln(y_t)\} \) because we would be using `regress`, but oftentimes, the two quantities are similar. Moreover, obtaining predicted values for \( y_t \) in the GLM does not present the transformation problem as happens with linear regression. The `forecast` commands contain special code to handle estimation results obtained by using `glm` with the `link(log)` option, and you do not need to specify an identity to obtain \( y \) as a function of \( \text{lny} \). All you would need to do is

```
. glm y x1 x2 x3, link(log)
. estimates store myglm
. forecast estimates myglm
```

The advise option

To produce forecasts from your model, `forecast` must be able to obtain predictions for each estimation result that you have added. For many of the most commonly used estimation commands such as `regress`, `ivregress`, and `var`, `forecast` includes special code to quickly obtain these predictions. For estimation commands that either require more involved computations to obtain predictions or are not widely used in forecasting, `forecast` instead relies on the `predict` command to obtain predictions.

The `advise` option of `forecast estimates` advises you as to whether `forecast` includes the special code to obtain fast predictions for the command whose estimation results are being added to the model. For example, here we use `advise` with `forecast estimates` when building the Klein (1950) model.
Example 3: Using the advise option

```
. use https://www.stata-press.com/data/r16/klein2, clear
. quietly reg3 (c p L.p w) (i p L.p L.k) (wp y L.y yr), endog(w p y) exog(t wg g)
. estimates store klein
. forecast create kleinmodel, replace
   (Forecast model cwmodel ended.)
Forecast model kleinmodel started.
. forecast estimates klein, advise
   (These estimation results are no longer needed; you can drop them.)
   Added estimation results from `reg3'.
Forecast model kleinmodel now contains 3 endogenous variables.
```

After we typed `forecast estimates', Stata advised us that “[t]hese estimation results are no longer needed; you can drop them”. That means `forecast' includes code to obtain predictions from `reg3' without having to call `predict'. `forecast' has recorded all the information it needs about the estimation results stored in `klein', and we could type

```
. estimates drop klein
```

to remove those estimates from memory.

For relatively small models, there is no need to use `estimates drop' to remove estimation results from memory. However, Stata allows no more than 300 sets of estimation results to be in memory at once, and `forecast solve' requires estimation results to be in memory (and not merely saved on disk) before it can produce forecasts. For very large models in which that limit may bind, you can use the `advise' option to determine which estimation results are needed to solve the model and which can be dropped.

Suppose we had estimation results from a command for which `forecast' must call `predict' to obtain predictions. Then instead of obtaining the note saying the estimation results were no longer needed, we would obtain a note stating

```
. forecast estimates IUsePredict
   (These estimation results are needed to solve the model.)
```

In that case, the estimation results would need to be in memory before calling `forecast solve'.

The `advise' option also provides an indication of how quickly forecasts can be produced from the model. Models for which `forecast' never needs to call `predict' can be solved much more quickly than models that include equations for which `forecast' must restore estimation results and call `predict' to obtain predictions.

Using saved estimation results

Stata’s `estimates' commands allow you to save estimation results to disk so that they are available in subsequent Stata sessions. You can use the `using' option of `forecast estimates' to use estimation results saved on disk without having to first call `estimates use'. In fact, `estimates use' can even retrieve estimation results stored on a website, as the next example demonstrates.
Example 4: Adding saved estimation results

The file `klein.ster` contains the estimation results produced by `reg3` for the three stochastic equations of Klein’s (1950) model. That file is stored on the Stata Press website in the same location as the example datasets. Here we create a forecast model and add those results:

```stata
.forecast create example4, replace
(Forecast model kleinmodel ended.)
Forecast model example4 started.
Added estimation results from `reg3'.
Forecast model example4 now contains 3 endogenous variables.
```

If you do not specify a file extension, `forecast estimates` assumes the file ends in `.ster`. You are more likely to save your estimation results on your computer’s disk drive rather than a web server, but in either case, this example shows that you can fit equations in one session of Stata, save the results to disk, and then build your forecast model later.

The `estimates save` command allows you to save multiple estimation results to the same file and numbers them sequentially starting at 1. You can use the `number()` option of `forecast estimates using` to specify which set of estimation results from the specified file you wish to add to the forecast model. If you do not specify `number()`, `forecast estimates using` uses the first set of results.

When you use `forecast estimates using`, `forecast` loads the estimation results from disk and stores them in memory using a temporary name. Later, when you proceed to solve your model, `forecast` checks to see whether those estimation results are still in memory. If not, it will attempt to reload them from the file you had specified. You should therefore not move or rename estimation result files between the time you add them to your model and the time you solve the model.

The predict option

As we mentioned while discussing the `advise` option, the `forecast` commands include code to quickly obtain predictions from some of the most commonly used commands, while they use `predict` to obtain predictions from other estimation commands. When you add estimation results that require `forecast` to use `predict`, by default, `forecast` assumes that it can pass the option `xb` on to `predict` to obtain the appropriate predicted values. You use the `predict()` option of `forecast estimates` to specify the option that `predict` must use to obtain predicted values from the estimates being added.

For example, suppose you used `tobit` to fit an equation whose dependent variable is left-censored at zero and then stored the estimation results under the name `tobitreg`. When solving the model, you want to use the predicted values of the left-truncated mean, the expected value of the dependent variable conditional on its being greater than zero. Looking at the `Syntax for predict in [R] tobit postestimation`, we see that the appropriate option we must pass to `predict` is `e(0,)`. To add this estimation result to an existing forecast model, we would therefore type

```stata
.forecast estimates tobitreg, predict(e(0,))
```

Now, whenever `forecast` calls `predict` with those estimation results, it will pass the option `e(0,)` so that we obtain the appropriate predictions. If you are adding results from a multiple-equation estimation command with \( k \) dependent variables, then you must specify \( k \)`predict` options within the `predict()` option, separated by spaces.
Forecasting with ARIMA models

Practitioners often use ARIMA models to forecast some of the variables in their models, and you can certainly use estimation results produced by commands such as `arima` with `forecast`. There are just two rules to follow when using commands that use the Kalman filter to obtain predictions. First, do not specify the `predict()` option with `forecast estimates`. The `forecast` commands know how to handle these estimators automatically. Second, as we stated earlier, the `forecast` commands do not “undo” any time-series operators that may adorn the left-hand-side variables of estimation results, so you must use `forecast identity` to specify identities to recover the underlying variables in levels.

Example 5: An ARIMA model with first- and seasonal-differencing

`wpi1.dta` contains quarterly observations on the variable `wpi`. First, let’s fit a multiplicative seasonal ARIMA model with both first- and seasonal-difference operators applied to the dependent variable and store the estimation results:

```
. use https://www.stata-press.com/data/r16/wpi1
. arima wpi, arima(1, 1, 1) sarima(1, 1, 1, 4)
(output omitted)
. estimates store arima
```
(For details on fitting seasonal ARIMA models, see [TS] arima).

With the difference operators used here, when `forecast` calls `predict`, it will obtain predictions in terms of `DS4.wpi`. Using the definitions of time-series operators in [TS] tsset, we have

\[
DS4.wpi_t = (wpi_t - wpi_{t-4}) - (wpi_{t-1} - wpi_{t-5})
\]

so that

\[
wpi_t = DS4.wpi_t + wpi_{t-4} + (wpi_{t-1} - wpi_{t-5})
\]

Because our `arima` results include a dependent variable with time-series operators, we must use the `name()` option of `forecast estimates` to specify an alternative variable name. We will name ours `ds4wpi`. Then we can specify an identity by using the previous equation to recover our forecasts in terms of `wpi`. We type

```
. forecast create arimaexample, replace
(Forecast model example4 ended.)
Forecast model arimaexample started.
. forecast estimates arima, name(ds4wpi)
Added estimation results from arima.
Forecast model arimaexample now contains 1 endogenous variable.
. forecast identity wpi = ds4wpi + L4.wpi + (L.wpi - L5.wpi)
Forecast model arimaexample now contains 2 endogenous variables.
```
. forecast solve, begin(tq(1988q1))
Computing dynamic forecasts for model arimaexample.

Starting period: 1988q1
Ending period:  1990q4
Forecast prefix:  f_

1988q1: ............
1988q2: ............
1988q3: ............
(output omitted)
1990q4: ............

Forecast 2 variables spanning 12 periods.

Because our entire forecast model consists of a single equation fit by arima, we can also call predict to obtain forecasts:

. predict a_wpi, y dynamic(tq(1988q1))
(5 missing values generated)
. list t f_wpi a_wpi in -5/1

<table>
<thead>
<tr>
<th>t</th>
<th>f_wpi</th>
<th>a_wpi</th>
</tr>
</thead>
<tbody>
<tr>
<td>120.</td>
<td>1989q4</td>
<td>110.2182</td>
</tr>
<tr>
<td>121.</td>
<td>1990q1</td>
<td>111.6782</td>
</tr>
<tr>
<td>122.</td>
<td>1990q2</td>
<td>112.9945</td>
</tr>
<tr>
<td>123.</td>
<td>1990q3</td>
<td>114.3281</td>
</tr>
<tr>
<td>124.</td>
<td>1990q4</td>
<td>115.5142</td>
</tr>
</tbody>
</table>

Looking at the last few observations in the dataset, we see that the forecasts produced by forecast (f_wpi) match those produced by predict (a_wpi). Of course, the advantage of forecast is that we can combine multiple sets of estimation results and obtain forecasts for an entire system of equations.

Technical note

Do not add estimation results to your forecast model that you have stored after calling an estimation command with the by: prefix. The stored estimation results will contain information from only the last group on which the estimation command was executed. forecast will then use those results for all observations in the forecast horizon regardless of the value of the group variable you specified with by:.

References

Cameron, A. C., and P. K. Trivedi. 2010. Microeconometrics Using Stata. Rev. ed. College Station, TX: Stata Press.
Also see

[TS] forecast — Econometric model forecasting

[R] estimates — Save and manipulate estimation results

[R] predict — Obtain predictions, residuals, etc., after estimation
**Title**

**Forecast exogenous — Declare exogenous variables**

### Description

*forecast exogenous* declares exogenous variables in the current forecast model.

### Syntax

```
forecast exogenous varlist
```

### Remarks and examples

For an overview of the *forecast* commands, see [TS] **forecast**. This manual entry assumes you have already read that manual entry. *forecast exogenous* declares exogenous variables in your forecast model.

Before you can solve your model, all the exogenous variables must be filled in with nonmissing values over the entire forecast horizon. When you use *forecast solve*, Stata first checks your exogenous variables and exits with an error message if any of them contains missing values for any periods being forecast. When you assemble a large model with many variables, it is easy to forget some variables and then have problems obtaining forecasts. *forecast exogenous* provides you with a mechanism to explicitly declare the exogenous variables in your model so that you do not forget about them.

Declaring exogenous variables with *forecast exogenous* is not explicitly necessary, but we nevertheless strongly encourage doing so. Stata can check the exogenous variables before solving the model and issue an appropriate error message if missing values are found, whereas troubleshooting models for which forecasting failed is more difficult after the fact.

#### Example 1

Here we fit a simple single-equation dynamic model with two exogenous variables, \(x_1\) and \(x_2\):

```
. use https://www.stata-press.com/data/r16/forecastex1
. quietly regress y L.y x1 x2
. estimates store exregression
. forecast create myexample
   Forecast model myexample started.
. forecast estimates exregression
   Added estimation results from *regress*.
   Forecast model myexample now contains 1 endogenous variable.
. forecast exogenous x1
   Forecast model myexample now contains 1 declared exogenous variable.
. forecast exogenous x2
   Forecast model myexample now contains 2 declared exogenous variables.
```
Instead of using `forecast exogenous` twice, we could have instead typed

```
   . forecast exogenous x1 x2
```

**Also see**

[Ts] `forecast` — Econometric model forecasting
**Title**

**forecast identity — Add an identity to a forecast model**

### Description

`forecast identity` adds an identity to the forecast model currently in memory. You must first create a new model using `forecast create` before you can add an identity with `forecast identity`. An identity is a nonstochastic equation that expresses an endogenous variable in the model as a function of other variables in the model. Identities often describe the behavior of endogenous variables that are based on accounting identities or adding-up conditions.

### Quick start

Add an identity to the forecast that states that `y3` is the sum of `y1` and `y2`

```
forecast identity y3=y1+y2
```

As above, and create new variable `newy` before adding it to the forecast

```
forecast identity newy=y1+y2, generate
```

### Syntax

```
forecast identity varname = exp [, options]
```

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generate</code></td>
<td>create new variable <code>varname</code></td>
</tr>
<tr>
<td><code>*double</code></td>
<td>store new variable as a double instead of as a float</td>
</tr>
</tbody>
</table>

`varname` is the name of an endogenous variable to be added to the forecast model.

* You can only specify `double` if you also specify `generate`.

### Options

- `generate` specifies that the new variable `varname` be created equal to `exp` for all observations in the current dataset.

- `double`, for use in conjunction with the `generate` option, requests that the new variable be created as a double instead of as a float. See [D] Data types.

### Remarks and examples

For an overview of the `forecast` commands, see [TS] forecast. This manual entry assumes you have already read that manual entry. `forecast identity` specifies a nonstochastic equation that determines the value of an endogenous variable in the model. When you type

```
.forecast identity varname = exp
```

253
forecast identity registers `varname` as an endogenous variable in your forecast model that is equal to `exp`, where `exp` is a valid Stata expression that is typically a function of other endogenous variables and exogenous variables in your model and perhaps lagged values of `varname` as well. `forecast identity` was used in all the examples in [TS] `forecast`.

**Example 1: Variables with constant growth rates**

Some models contain variables that you are willing to assume will grow at a constant rate throughout the forecast horizon. For example, say we have a model using annual data and want to assume that our population variable `pop` grows at 0.75% per year. Then we can declare endogenous variable `pop` by using `forecast identity`:

```
. forecast identity pop = 1.0075*L.pop
```

Typically, you use `forecast identity` to define the relationship that determines an endogenous variable that is already in your dataset. For example, in example 1 of [TS] `forecast`, we used `forecast identity` to define total wages as the sum of government and private-sector wages, and the total wage variable already existed in our dataset.

The `generate` option of `forecast identity` is useful when you wish to use a transformation of one or more endogenous variables as a right-hand-side variable in a stochastic equation that describes another endogenous variable. For example, say you want to use `regress` to model variable `y` as a function of the ratio of two endogenous variables, `u` and `w`, as well as other covariates. Without the `generate` option of `forecast identity`, you would have to define the variable `y = u/w` twice: first, you would have to use the `generate` command to create the variable before fitting your regression model, and then you would have to use `forecast identity` to add an identity to your forecast model to define `y` in terms of `u` and `w`. Assuming you have already created your forecast model, the `generate` option allows you to define the ratio variable just once, before you fit the regression equation. In this example, the ratio variable is easy enough to specify twice, but it is very easy to forget to include identities that define regressors used in estimation results while building large forecast models. In other cases, an endogenous variable may be a more complicated function of other endogenous variables, so having to specify the function only once reduces the chance for error.

**Stored results**

`forecast identity` stores the following in `r()`:

- **Macros**
  - `r(lhs)` left-hand-side (endogenous) variable
  - `r(rhs)` right-hand side of identity
  - `r(basenames)` base names of variables found on right-hand side
  - `r(fullnames)` full names of variables found on right-hand side

**Also see**

[TS] `forecast` — Econometric model forecasting
forecast list — List forecast commands composing current model

Description

forecast list produces a list of forecast commands that compose the current model.

Quick start

List all forecast commands that compose the current model

```
forecast list
```

Save a list of commands to replicate the current forecast model to myforecast.do

```
forecast list, saving(myforecast)
```

As above, but save the commands as myforecast.txt

```
forecast list, saving(myforecast.txt)
```

Syntax

```
forecast list [, options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saving(filename [, replace] )</td>
<td>save list of commands to file</td>
</tr>
<tr>
<td>notrim</td>
<td>do not remove extraneous white space</td>
</tr>
</tbody>
</table>

Options

saving(filename [, replace] ) requests that forecast list write the list of commands to disk with filename. If no extension is specified, .do is assumed. If filename already exists, an error is issued unless you specify replace, in which case the file is overwritten.

notrim requests that forecast list not remove any extraneous spaces and that commands be shown exactly as they were originally entered. By default, superfluous white space is removed.

Remarks and examples

For an overview of the forecast commands, see [TS] forecast. This manual entry assumes you have already read that manual entry. forecast list produces a list of all the forecast commands you would need to enter to re-create the forecast model currently in memory. Unlike using a command log, forecast list only shows the forecast-related commands but not any estimation command or other commands you may have issued. If you specify saving(filename), forecast list saves the list as filename.do, which you can then edit using the Do-file Editor.
`forecast` creates models by accumulating estimation results, identities, and other features that you add to the model by using various `forecast` subcommands. Once you add a feature to a model, it remains a part of the model until you clear the entire model from memory. `forecast list` provides a list of all the `forecast` commands you would need to rebuild the current model.

When building all but the smallest forecast models, you will typically write a do-file to load your dataset, perhaps call some estimation commands, and issue a sequence of `forecast` commands to build and solve your forecast model. There are times, though, when you will type a `forecast` command interactively and then later want to undo the command or else wish you had not typed the command in the first place. `forecast list` provides the solution.

Suppose you use `forecast adjust` to perform some policy simulations and then decide you want to remove those adjustments from the model. `forecast list` makes this easy to do. You simply call `forecast list` with the `saving()` option to produce a do-file that contains all the `forecast` commands issued since the model was created. Then you can edit the do-file to remove the `forecast adjust` command, type `forecast clear`, and run the do-file.

Example 1: Klein’s model

In example 1 of [TS] `forecast`, we obtained forecasts from Klein’s (1950) macroeconomic model. If we type `forecast list` after typing all the commands in that example, we obtain

```
. forecast list
forecast create kleinmodel
forecast estimates klein
forecast identity y = c + i + g
forecast identity p = y - t - wp
forecast identity k = L.k + i
forecast identity w = wg + wp
forecast exogenous wg
forecast exogenous g
forecast exogenous t
forecast exogenous yr
```

The `forecast solve` command is not included in output produced by `forecast list` because solving the model does not add any features to the model.

Technical note

To prevent you from accidentally destroying the model in memory, `forecast list` does not add the `replace` option to `forecast create` even if you specified `replace` when you originally called `forecast create`.

Reference


Also see

[TS] `forecast` — Econometric model forecasting
**Title**

forecast query — Check whether a forecast model has been started

**Description**

forecast query issues a message indicating whether a forecast model has been started.

**Syntax**

```
forecast query
```

**Remarks and examples**

For an overview of the forecast commands, see [TS] forecast. This manual entry assumes you have already read that manual entry. forecast query allows you to check whether a forecast model has been started. Most users of the forecast commands will not need to use forecast query. This command is most useful to programmers.

Suppose there is no forecast model in memory:

```
.forecast query
No forecast model exists.
```

Now we create a forecast model named fcmodel:

```
.forecast create fcmodel
Forecast model fcmodel started.
.forecast query
Forecast model fcmodel exists.
```

**Stored results**

forecast query stores the following in r():

- **Scalars**
  - r(found) 1 if model started, 0 otherwise

- **Macros**
  - r(name) model name

**Also see**

- [TS] forecast — Econometric model forecasting
- [TS] forecast describe — Describe features of the forecast model
**Description**

`forecast solve` computes static or dynamic forecasts based on the model currently in memory. Before you can solve a model, you must first create a new model using `forecast create` and add equations and variables to it using the commands summarized in [TS] `forecast`.

**Quick start**

Compute dynamic forecast after `forecast create` and `forecast estimates`

```
forecast solve
```

As above, but with forecasts starting at 1990q1 and ending at 1995q3

```
forecast solve, begin(q(1990q1)) end(q(1995q3))
```

As above, and change prefix of predicted endogenous variables to `hat`

```
forecast solve, begin(q(1990q1)) end(q(1995q3)) prefix(hat)
```

As above, but forecast 11 periods starting at 1990q1

```
forecast solve, begin(q(1990q1)) prefix(hat) periods(11)
```

Incorporate forecast uncertainty via simulation, and store point forecasts and their standard deviations in variables prefixed with `d_` and `sd_`

```
forecast solve, prefix(d_) ///
    simulate(betas, statistic(stddev, prefix(sd_)))
```
**Syntax**

```
forecast solve [ , { prefix(stub) | suffix(stub) } options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>*prefix(string)</td>
<td>specify prefix for forecast variables</td>
</tr>
<tr>
<td>*suffix(string)</td>
<td>specify suffix for forecast variables</td>
</tr>
<tr>
<td>begin(time_constant)</td>
<td>specify period to begin forecasting</td>
</tr>
<tr>
<td>†end(time_constant)</td>
<td>specify period to end forecasting</td>
</tr>
<tr>
<td>†periods(#)</td>
<td>specify number of periods to forecast</td>
</tr>
<tr>
<td>double</td>
<td>store forecast variables as doubles instead of as floats</td>
</tr>
<tr>
<td>static</td>
<td>produce static forecasts instead of dynamic forecasts</td>
</tr>
<tr>
<td>actuals</td>
<td>use actual values if available instead of forecasts</td>
</tr>
</tbody>
</table>

**Simulation**

```
simulate(sim_technique, sim_statistic sim_options)
```

**Reporting**

```
log(log_level)   | specify level of logging display; log_level may be detail, on, brief, or off
```

**Solver**

```
v_tolerance(#)   | specify tolerance for forecast values            |
| z_tolerance(#)  | specify tolerance for function zero              |
| iterate(#)      | specify maximum number of iterations             |
| technique(technique) | specify solution method; may be dampedgaussseidel #, gaussseidel, broydenpowell, or newtonraphson |
```

* You can specify prefix() or suffix() but not both.
† You can specify end() or periods() but not both.

**sim_technique**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>betas</td>
</tr>
<tr>
<td>errors</td>
</tr>
<tr>
<td>residuals</td>
</tr>
</tbody>
</table>

You can specify one or two sim_methods separated by a space, though you cannot specify both errors and residuals.

**sim_statistic** is

```
statistic(statistic, { prefix(string) | suffix(string) } )
```

and may be repeated up to three times.
**statistic**

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>record the mean of the simulation forecasts</td>
</tr>
<tr>
<td>variance</td>
<td>record the variance of the simulation forecasts</td>
</tr>
<tr>
<td>stddev</td>
<td>record the standard deviation of the simulation forecasts</td>
</tr>
</tbody>
</table>

**sim_options**

<table>
<thead>
<tr>
<th>sim_options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saving(filename, ...)</td>
<td>save results to file; save statistics in double precision; save results to filename every # replications</td>
</tr>
<tr>
<td>nodots</td>
<td>suppress replication dots</td>
</tr>
<tr>
<td>reps(#)</td>
<td>perform # replications; default is reps(50)</td>
</tr>
</tbody>
</table>

**Options**

prefix(string) and suffix(string) specify a name prefix or suffix that will be used to name the variables holding the forecast values of the variables in the model. You may specify prefix() or suffix() but not both. Sometimes, it is more convenient to have all forecast variables start with the same set of characters, while other times, it is more convenient to have all forecast variables end with the same set of characters.

If you specify prefix(f_), then the forecast values of endogenous variables x, y, and z will be stored in new variables f_x, f_y, and f_z.

If you specify suffix(_g), then the forecast values of endogenous variables x, y, and z will be stored in new variables x_g, y_g, and z_g.

begin(time_constant) requests that forecast begin forecasting at period time_constant. By default, forecast determines when to begin forecasting automatically.

double requests that the forecast and simulation variables be stored in double precision. The default is to use single-precision floats. See [D] Data types for more information.

static requests that static forecasts be produced. Actual values of variables are used wherever lagged values of the endogenous variables appear in the model. By default, dynamic forecasts are produced, which use the forecast values of variables wherever lagged values of the endogenous variables appear in the model. Static forecasts are also called one-step-ahead forecasts.

actuals specifies how nonmissing values of endogenous variables in the forecast horizon are treated. By default, nonmissing values are ignored, and forecasts are produced for all endogenous variables. When you specify actuals, forecast sets the forecast values equal to the actual values if they are nonmissing. The forecasts for the other endogenous variables are then conditional on the known values of the endogenous variables with nonmissing data.
`simulate(sim Technique, sim Statistic sim Options)` allows you to simulate your model to obtain measures of uncertainty surrounding the point forecasts produced by the model. Simulating a model involves repeatedly solving the model, each time accounting for the uncertainty associated with the error terms and the estimated coefficient vectors.

`sim Technique` can be `betas`, `errors`, or `residuals`, or you can specify both `betas` and one of `errors` or `residuals` separated by a space. You cannot specify both `errors` and `residuals` simultaneously. The `sim Technique` controls how uncertainty is introduced into the model.

`sim Statistic` specifies a summary statistic to summarize the forecasts over all the simulations. `sim Statistic` takes the form

\[
\text{statistic(statistic, \{prefix(string) | suffix(string) \})}
\]

where `statistic` may be `mean`, `variance`, or `stddev`. You may specify either the prefix or the suffix that will be used to name the variables that will contain the requested `statistic`. You may specify up to three `sim Statistics`, allowing you to track the mean, variance, and standard deviations of your forecasts.

`sim Options` include `saving(filename, [suboptions])`, `nodots`, and `reps(#).`

`saving(filename, [suboptions])` creates a Stata data file (`.dta` file) consisting of (for each endogenous variable in the model) a variable containing the simulated values.

`double` specifies that the results for each replication be saved as doubles, meaning 8-byte reals. By default, they are saved as floats, meaning 4-byte reals.

`replace` specifies that `filename` be overwritten if it exists.

`every(#)` specifies that results be written to disk every `#`th replication. `every()` should be specified only in conjunction with `saving()` when the command takes a long time for each replication. This will allow recovery of partial results should some other software crash your computer. See [P] postfile.

`nodots` suppresses display of the replication dots. By default, one dot character is displayed for each successful replication. If during a replication convergence is not achieved, `forecast solve` exits with an error message.

`reps(#)` requests that `forecast solve` perform `#` replications; the default is `reps(50).

`log(log_level)` specifies the level of logging provided while solving the model. `log_level` may be `detail`, `on`, `brief`, or `off`.

`log(detail)` provides a detailed iteration log including the current values of the convergence criteria for each period in each panel (in the case of panel data) for which the model is being solved.

`log(on)`, the default, provides an iteration log showing the current panel and period for which the model is being solved as well as a sequence of dots for each period indicating the number of iterations.

`log(brief)`, when used with a time-series dataset, is equivalent to `log(on)`. When used with a panel dataset, `log(brief)` produces an iteration log showing the current panel being solved but does not show which period within the current panel is being solved.

`log(off)` requests that no iteration log be produced.
vtolerance(#), ztolerance(#), and iterate(#) control when the solver of the system of equations stops. ztolerance() is ignored if either technique(dampedgaussseidel #) or technique(gaussseidel) is specified. These options are seldom used. See [M-5] solvenl().

\texttt{\textbf{technique(technique)}} specifies the technique to use to solve the system of equations. technique may be dampedgaussseidel #, gaussseidel, broydenpowell, or newtonraphson, where $0 < # < 1$ specifies the amount of damping with smaller numbers indicating less damping.

The default is \texttt{technique(dampedgaussseidel 0.2)}, which works well in most situations. If you have convergence issues, first try continuing to use \texttt{dampedgaussseidel #} but with a larger damping factor. Techniques broydenpowell and newtonraphson usually work well, but because they require the computation of numerical derivatives, they tend to be much slower. See \texttt{[M-5] solvenl(())}.

\section{Remarks and examples}

For an overview of the \texttt{forecast} commands, see \texttt{[TS] forecast}. This manual entry assumes you have already read that manual entry. The \texttt{forecast solve} command solves a forecast model in Stata. Before you can solve a model, you must first create a model using \texttt{forecast create}, and you must add at least one equation using \texttt{forecast estimates}, \texttt{forecast coefvector}, or \texttt{forecast identity}. We covered the most commonly used options of \texttt{forecast solve} in the examples in \texttt{[TS] forecast}.

Here we focus on two sets of options that are available with \texttt{forecast solve}. First, we discuss the \texttt{actuals} option, which allows you to obtain forecasts conditional on prespecified values for one or more of the endogenous variables. Then we focus on performing simulations to obtain estimates of uncertainty around the point forecasts.

Remarks are presented under the following headings:

\begin{itemize}
  \item Performing conditional forecasts
  \item Using simulations to measure forecast accuracy
\end{itemize}

\section{Performing conditional forecasts}

Sometimes, you already know the values of some of the endogenous variables in the forecast horizon and would like to obtain forecasts for the remaining endogenous variables conditional on those known values. Other times, you may not know the values but would nevertheless like to specify a path for some endogenous variables and see how the others would evolve conditional on that path. To accomplish these types of exercises, you can use the \texttt{actuals} option of \texttt{forecast solve}.

\section*{Example 1: Specifying alternative scenarios}

\texttt{gdpoil.dta} contains quarterly data on the annualized growth rate of GDP and the percentage change in the quarterly average price of oil through the end of 2007. We want to explore how GDP would have evolved if the price of oil had risen 10\% in each of the first three quarters of 2008 and then held steady for several years. We will use a bivariate vector autoregression (VAR) to forecast the variables \texttt{gdp} and \texttt{oil}. Results obtained from the \texttt{varsoc} command indicate that the Hannan–Quinn information criterion is minimized when the VAR includes two lags. First, we fit our VAR model and store the estimation results:
. use https://www.stata-press.com/data/r16/gdpoil
. var gdp oil, lags(1 2)

Vector autoregression

Sample:  1986q4 - 2007q4    Number of obs =     85
Log likelihood = -500.0749    AIC =  12.00176
FPE =  559.0724    HQIC =  12.11735
Det(Sigma_ml) =  441.7362    SBIC =  12.28913

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>RMSE</th>
<th>R-sq</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdp</td>
<td>5</td>
<td>1.88516</td>
<td>0.1820</td>
<td>18.91318</td>
<td>0.0008</td>
</tr>
<tr>
<td>oil</td>
<td>5</td>
<td>11.8776</td>
<td>0.1140</td>
<td>10.93614</td>
<td>0.0273</td>
</tr>
</tbody>
</table>

|                     | Coef.  | Std. Err. | z    | P>|z| |  [95% Conf. Interval] |
|---------------------|--------|-----------|------|------|----------------------|
| gdp                 |        |           |      |      |                      |
| L1. gdp             | .1498285| .1015076 | 1.48 | 0.140 | -.0491227             | .3487797 |
| L2. gdp             | .3465238| .1022446 | 3.39 | 0.001 | .146128               | .5469196 |
| oil                 |        |           |      |      |                      |
| L1. oil             | -.0374609| .0167968 | -2.23| 0.026 | -.070382             | -.0045399 |
| L2. oil             | .0119564| .0164599 | 0.73 | 0.468 | -.0203043            | .0442172 |
| _cons               | 1.519983| .4288145 | 3.54 | 0.000 | .6795226             | 2.360444 |

|                     | Coef.  | Std. Err. | z    | P>|z| |  [95% Conf. Interval] |
|---------------------|--------|-----------|------|------|----------------------|
| oil                 |        |           |      |      |                      |
| L1. gdp             | .8102233| .6395579 | 1.27 | 0.205 | -.4432871            | 2.063734 |
| L2. gdp             | 1.090244| .6442017 | 1.69 | 0.091 | -.1723684            | 2.352856 |
| oil                 |        |           |      |      |                      |
| L1. oil             | .0995271| .1058295 | 0.94 | 0.347 | -.1078949            | .3069491 |
| L2. oil             | -.1870052| .103707 | -1.80| 0.071 | -.3902672            | .0162568 |
| _cons               | -4.041859| 2.701785 | -1.50| 0.135 | -9.33726             | 1.253543 |

. estimates store var

The dataset ends in the fourth quarter of 2007, so before we can produce forecasts for 2008 and beyond, we need to extend our dataset. We can do that using the \texttt{tsappend} command. Here we extend our dataset three years:

. tsappend, add(12)
Now we can create a forecast model and obtain baseline forecasts:

```
. forecast create oilmodel
Forecast model oilmodel started.
. forecast estimates var
  Added estimation results from var.
  Forecast model oilmodel now contains 2 endogenous variables.
. forecast solve, prefix(bl_)
Computing dynamic forecasts for model oilmodel.
```

Starting period: 2008q1
Ending period: 2010q4
Forecast prefix: bl_

2008q1: ...............  
  (output omitted)  
2010q4: ...............  
Forecast 2 variables spanning 12 periods.

To see how GDP evolves if oil prices increase 10% in each of the first three quarters of 2008 and then remain flat, we need to obtain a forecast for gdp conditional on a specified path for oil. The actuals option of forecast solve will do that for us. With the actuals option, if an endogenous variable contains a nonmissing value for the period currently being forecast, forecast solve will use that value as the forecast, overriding whatever value might be produced by that variable’s underlying estimation result or identity. Then the endogenous variables with missing values will be forecast conditional on the endogenous variables that do have valid data. Here we fill in oil with our hypothesized price path:

```
. replace oil = 10 if qdate == tq(2008q1)
  (1 real change made)
. replace oil = 10 if qdate == tq(2008q2)
  (1 real change made)
. replace oil = 10 if qdate == tq(2008q3)
  (1 real change made)
. replace oil = 0 if qdate > tq(2008q3)
  (9 real changes made)
```

Now we obtain forecasts conditional on our oil variable. We will use the prefix alt_ for these forecast variables:

```
. forecast solve, prefix(alt_) actuals
Computing dynamic forecasts for model oilmodel.
```

Starting period: 2008q1
Ending period: 2010q4
Forecast prefix: alt_

2008q1: ...............  
  (output omitted)  
2010q4: ...............  
Forecast 2 variables spanning 12 periods.
Forecasts used actual values if available.
Finally, we make a variable containing the difference between our alternative and our baseline \( gdp \) forecasts and graph it:

\[
\text{generate diff\_gdp = alt\_gdp - bl\_gdp}
\]

Our model indicates GDP growth would be about 0.4% less in the second through fourth quarters of 2008 than it would otherwise be, but would be mostly unaffected thereafter if oil prices followed our hypothetical path. The one-quarter lag in the response of GDP is due to our using a VAR model. In our VAR model, lagged values of \( oil \) predict the current value of \( gdp \), but the current value of \( oil \) does not.

**Technical note**

The previous example allowed us to demonstrate `forecast solve`'s `actuals` option, but in fact measuring the economy’s response to oil shocks is much more difficult than our simple VAR analysis would suggest. One obvious complication is that positive and negative oil price shocks do not have symmetric effects on the economy. In our simple model, if a 50% increase in oil prices lowers GDP by \( x\% \), then a 50% decrease in oil prices must raise GDP by \( x\% \). However, a 50% decrease in oil prices is perhaps more likely to portend weakness in the economy rather than an imminent growth spurt. See, for example, Hamilton (2003) and Kilian and Vigfusson (2013).

Another way to specify alternative scenarios for your forecasts is to use the `forecast adjust` command. That command is more flexible in the types of manipulations you can perform on endogenous variables but, depending on the task at hand, may involve more effort. The `actuals` option of the `forecast solve` and the `forecast adjust` commands are complementary. There is much overlap in what you can achieve; in some situations, specifying the `actuals` option will be easier, while in other situations, using adjustments via `forecast adjust` will prove to be easier.
Using simulations to measure forecast accuracy

To motivate the discussion, we will focus on the simple linear regression model. Even though `forecast` can handle models with many equations with equal ease, all the issues that arise can be illustrated with one equation. Suppose we have the following relationship between variables $y$ and $x$:

$$y_t = \alpha + \beta x_t + \epsilon_t$$  \hspace{1cm} (1)

where $\epsilon_t$ is a zero-mean error term. Say we fit (1) by ordinary least squares (OLS) using observations $1, \ldots, T$ and obtain the point estimates $\hat{\alpha}$ and $\hat{\beta}$. Assuming we have data for exogenous variable $x$ at time $T + 1$, we could forecast $y_{T+1}$ as

$$\hat{y}_{T+1} = \hat{\alpha} + \hat{\beta} x_{T+1}$$  \hspace{1cm} (2)

However, there are several factors that prevent us from guaranteeing ex ante that $y_{T+1}$ will indeed equal $\hat{y}_{T+1}$. We must assume that (1) specifies the correct relationship between $y$ and $x$. Even if that relationship held for times 1 through $T$, are we sure it will hold at time $T + 1$? Uncertainty due to issues like that are inherent to the type of forecasting that the `forecast` commands are designed for. Here we discuss two additional sources of uncertainty that `forecast solve` can help you measure.

First, we estimated $\alpha$ and $\beta$ by OLS to obtain $\hat{\alpha}$ and $\hat{\beta}$, but we must emphasize the word estimated. Our estimates are subject to sampling error. When you fit a regression using `regress` or any other estimation command, Stata presents not just the point estimates of the parameters but also the standard errors and confidence intervals representing the level of uncertainty surrounding those point estimates. Uncertainty surrounding the true values of $\alpha$ and $\beta$ mean that there is some level of uncertainty surrounding our predicted value $\hat{y}_{T+1}$ as well.

Second, (1) states that $y_t$ depends not just on $\alpha$, $\beta$, and $x_t$ but also on an unobserved error term $\epsilon_t$. When we make our forecast using (2), we assume that the error term will equal its expected value of zero. Saying a random error has an expected value of zero is clearly not the same as saying it will be zero every time. If a positive outside shock occurs at $T + 1$, $y_{T+1}$ will be higher than our estimate based on (2) would lead us to believe.

Fortunately, quantifying both these sources of uncertainty is straightforward using simulation. First, we solve our model as usual, providing us with our point forecasts. To see how uncertainty surrounding our estimated parameters affects our forecasts, we can take random draws from a multivariate normal distribution whose mean is $(\hat{\alpha}, \hat{\beta})$ and whose variance is the covariance matrix produced by `regress`. We then solve our model using these randomly drawn parameters rather than the original point estimates. If we repeat the process of drawing random parameters and solving the model many times, we can use the variance or standard deviation across replications for each time period as a measure of uncertainty.

To account for uncertainty surrounding the error term, we can also use simulation. Here, at each replication, we add a random noise term to our forecast for $y_{T+1}$, where we draw our random errors such that they have the same characteristics as $\epsilon_t$. There are two ways we can do that. First, all the estimation commands commonly used in forecasting provide us with an estimate of the variance or standard deviation of the error term. For example, `regress` labels the estimated standard deviation of the error term “Root RMSE” and conveniently saves it in a macro that `forecast` can access. If we are willing to assume that all the errors in the equations in our model are normally distributed, then we can use random-normal errors drawn with means equal to zero and variances as reported by the estimation command used to fit each equation.

Sometimes the assumption of normality is unpalatable. In those cases, an alternative is to solve the model to obtain static forecasts and then compute the sample residuals based on the observations for which we have nonmissing values of the endogenous variables. Then in our simulations, we randomly choose one of the residuals observed for that equation.
At each replication, whether we draw errors based on the normal errors or from the pool of static-forecast residuals, we add the drawn value to our estimate of $\hat{y}_{T+1}$ to provide a simulated value for our forecast. Then, just like when simulating parameter uncertainty, we can use the variance or standard deviation across replications to measure uncertainty. In fact, we can perform simulations that draw both random parameters and random errors to account for both sources of uncertainty at once.

**Example 2: Accounting for parameter uncertainty**

Here we revisit our Klein (1950) model from example 1 of [TS] `forecast` and perform simulations in which we account for uncertainty associated with the estimated parameters of the model. First, we load the dataset and set up our model:

```stata
use https://www.stata-press.com/data/r16/klein2, clear
quietly reg3 (c p L.p w) (i p L.p L.k) (wp y L.y yr), endog(w p y)>
> exog(t wg g)
estimates store klein
.forecast create kleinmodel, replace
    (Forecast model kleinmodel ended.)
Forecast model kleinmodel started.
.forecast estimates klein
    Added estimation results from `reg3'.
Forecast model kleinmodel now contains 3 endogenous variables.
.forecast identity y = c + i + g
    Forecast model kleinmodel now contains 4 endogenous variables.
.forecast identity p = y - t - wp
    Forecast model kleinmodel now contains 5 endogenous variables.
.forecast identity k = L.k + i
    Forecast model kleinmodel now contains 6 endogenous variables.
.forecast identity w = wg + wp
    Forecast model kleinmodel now contains 7 endogenous variables.
.forecast exogenous wg
    Forecast model kleinmodel now contains 1 declared exogenous variable.
.forecast exogenous g
    Forecast model kleinmodel now contains 2 declared exogenous variables.
.forecast exogenous t
    Forecast model kleinmodel now contains 3 declared exogenous variables.
.forecast exogenous yr
    Forecast model kleinmodel now contains 4 declared exogenous variables.
```

Now we are ready to solve our model. We are going to begin dynamic forecasts in 1936, and we are going to perform 100 replications. We will store the point forecasts in variables prefixed with `d_`, and we will store the standard deviations of our forecasts in variables prefixed with `sd_`. Because the simulations involve the use of random numbers, we must remember to set the random-number seed if we want to be able to replicate our results; see [R] `set seed`. We type

```stata
...```
The key here is the `simulate()` option. We requested that `forecast solve` perform 100 simulations by taking random draws for the parameters (`betas`), and we requested that it record the standard deviation (`stddev`) of each endogenous variable in new variables that begin with `sd_`. Next we compute the upper and lower bounds of a 95% prediction interval for our forecast of total income `y`:

```
. generate d_y_up = d_y + invnormal(0.975)*sd_y
(16 missing values generated)
. generate d_y_dn = d_y + invnormal(0.025)*sd_y
(16 missing values generated)
```

We obtained 16 missing values after each `generate` because the simulation summary variables only contain nonmissing data for the periods in which forecasts were made. The point-forecast variables that begin with `d_` in this example are filled in with the corresponding actual values of the endogenous variables for periods before the beginning of the forecast horizon; in our experience, having both the historical data and forecasts in one set of variables simplifies many tasks. Here we graph our forecast of total income along with the 95% prediction interval:
Our next example will use the same forecast model, but we will not need the forecast variables we just created. `forecast drop` makes removing those variables easy:

```
. forecast drop
  (dropped 14 variables)
```

`forecast drop` drops all variables created by the previous invocation of `forecast solve`, including both the point-forecast variables and any variables that contain simulation results. In this case, `forecast drop` will remove all the variables that begin with `sd_` as well as `d_y`, `d_c`, `d_i`, and so on. However, we are not done yet. We created the variables `d_y_dn` and `d_y_up` ourselves, and they were not part of the forecast model. Therefore, they are not removed by `forecast drop`, and we need to do that ourselves:

```
. drop d_y_dn d_y_up
```

### Example 3: Accounting for both parameter uncertainty and random errors

In the previous example, we measured uncertainty in our model stemming from the fact that our parameters were estimated. Here we not only simulate random draws for the parameters but also add random-normal errors to the stochastic equations. We type
. set seed 1
.forecast solve, prefix(d_) begin(1936)
> simulate(betas errors, statistic(stddev, prefix(sd_)) reps(100))
Computing dynamic forecasts for model kleinmodel.

Starting period: 1936
Ending period: 1941
Forecast prefix: d_
1936: ............................................
1937: ...........................................
1938: ............................................
1939: ............................................
1940: ............................................
1941: .............................................
Performing simulations (100)

Forecast 7 variables spanning 6 periods.

The only difference between this call to forecast solve and the one in the previous example is that here we specified betas errors in the simulate() option rather than just betas. Had we wanted to perform simulations involving the parameters and random draws from the pool of static-forecast residuals rather than random-normal errors, we would have specified betas residuals. After we re-create the variables containing the bounds on our prediction interval, we obtain the following graph:

Notice that by accounting for both parameter and additive error uncertainty, our prediction interval became much wider.
forecast solve — Obtain static and dynamic forecasts

Stored results

forecast solve stores the following in r():

Scalars
- r(first_obs) first observation in forecast horizon
- r(last_obs) last observation in forecast horizon
- (of first panel if forecasting panel data)
- r(Npanels) number of panels forecast
- r(Nvar) number of forecast variables
- r(vtolerance) tolerance for forecast values
- r(ztolerance) tolerance for function zero
- r(iterate) maximum number of iterations
- r(sim_nreps) number of simulations
- r(damping) damping parameter for damped Gauss–Seidel

Macros
- r(prefix) forecast variable prefix
- r(suffix) forecast variable suffix
- r(actuals) actuals, if specified
- r(static) static, if specified
- r(double) double, if specified
- r(technique) solution method
- r(sim_technique) specified sim_technique
- r(logtype) on, off, brief, or detail

Methods and formulas

Formalizing the definition of a model provided in [TS] forecast, we represent the endogenous variables in the model as the $k \times 1$ vector $y$, and we represent the exogenous variables in the model as the $m \times 1$ vector $x$. We refer to the contemporaneous values as $y_t$ and $x_t$; for notational simplicity, we refer to lagged values as $y_{t-1}$ and $x_{t-1}$ with the implication that further lags of the variables can also be included with no loss of generality. We use $\theta$ to refer to the vector of all the estimated parameters in all the equations of the model. We use $u_t$ and $u_{t-1}$ to refer to contemporaneous and lagged error terms, respectively.

The forecast commands solve models of the form

$$y_{it} = f_i(y_{-i,t}, y_{t-1}, x_t, x_{t-1}, u_t, u_{t-1}; \theta)$$

(3)

where $i = 1, \ldots, k$ and $y_{-i,t}$ refers to the $k - 1 \times 1$ vector of endogenous variables other than $y_i$ at time $t$. If equation $j$ is an identity, we take $u_{jt} = 0$ for all $t$; for stochastic equations, the errors correspond to the usual regression error terms. Equation (3) does not include subscripts indexing panels for notational simplicity, but the extension is obvious. A model is solvable if $k \geq 1$. $m$ may be zero.

Endogenous variables are added to the forecast model via forecast estimates, forecast identity, and forecast coefvector. Equations added via forecast estimates are always stochastic, while equations added via forecast identity are always nonstochastic. Equations added via forecast coefvector are treated as stochastic if options variance() or errorvariance() (or both) are specified and nonstochastic if neither is specified.

Exogenous variables are declared using forecast exogenous, but the model may contain additional exogenous variables. For example, the right-hand side of an equation may contain exogenous variables that are not declared using forecast exogenous. Before solving the model, forecast solve determines whether the declared exogenous variables contain missing values over the forecast horizon and issues an informative error message if any do. Undeclared exogenous variables that contain missing values within the forecast horizon will cause forecast solve to exit with a less-informative error message and require the user to do more work to pinpoint the problem.
Adjustments added via `forecast adjust` easily fit within the framework of (3). Simply let \( f_i(·) \) represent the value of \( y_{it} \) obtained by first evaluating the appropriate estimation result, coefficient vector, or identity and then performing the adjustments based on that intermediate result. Endogenous variables may have multiple adjustments; adjustments are made in the order in which they were specified via `forecast adjust`. For single-equation estimation results and coefficient vectors as well as identities, adjustments are performed right after the equation is evaluated. For multiple-equation estimation results and coefficient vectors, adjustments are made after all the equations within that set of results are evaluated. Suppose an estimation result that uses `predict` includes two left-hand-side variables, \( y_{1t} \) and \( y_{2t} \), and you have added two adjustments to \( y_{1t} \) and one adjustment to \( y_{2t} \). Here `forecast solve` first calls `predict` twice to obtain candidate values for \( y_{1t} \) and \( y_{2t} \); then it performs the two adjustments to \( y_{1t} \), and finally it adjusts \( y_{2t} \).

`forecast solve` offers four solution techniques: Gauss–Seidel, damped Gauss–Seidel, Broyden–Powell, and Newton–Raphson. The Gauss–Seidel techniques are simple iterative techniques that are often fast and typically work well, particularly when a damping factor is used. Gauss–Seidel is simply damped Gauss–Seidel without damping (a damping factor of 0). By default, damped Gauss–Seidel with a damping factor of 0.2 is used, representing a small amount of damping. As Fair (1984, 250) notes, while these techniques often work well, there is no guarantee that they will converge. Technique Newton–Raphson typically works well but is slow because it requires the use of numerical derivatives at every iteration to obtain a Jacobian matrix. The Broyden–Powell (Broyden 1970; Powell 1970) method is analogous to quasi-Newton methods used for function optimization in that an updating method is used at each iteration to update an estimate of the Jacobian matrix rather than actually recalculating it. For additional details as well as a discussion of the convergence criteria, see [M-5] `solvenl()`.

If you do not specify the `begin()` option, `forecast solve` uses the following algorithm to select the starting time period. Suppose the time variable \( t \) runs from 1 to \( T \). If, at time \( T \), none of the endogenous variables contains missing values, `forecast solve` exits with an error message: there are no periods in which the endogenous variables are not known; therefore, there are no periods where a forecast is obviously required. Otherwise, consider period \( T – 1 \). If none of the endogenous variables contains missing values in that period, then the only period to forecast is \( T \). Otherwise, work back through time to find the latest period in which all the endogenous variables contain nonmissing values and then begin forecasting in the subsequent period. In the case of panel datasets, the same algorithm is applied to each panel, and forecasts for all panels begin on the earliest period selected.

When you specify the `simulate()` option with `sim_technique betas`, `forecast solve` draws random vectors from the multivariate normal distribution for each estimation result individually. The mean and variance are based on the estimation result’s \( e(b) \) and \( e(V) \) macros, respectively. If the estimation result is from a multiple-equation estimator, the corresponding Stata command stores in \( e(b) \) and \( e(V) \) the full parameter vector and covariance matrix for all equations so that `forecast solve`’s simulations will account for covariances among parameters in that estimation result’s equations. However, covariances among parameters that appear in different estimation results are taken to be zero.

If you specify a coefficient vector using `forecast coeffvector` and specify a variance matrix in the `variance()` option, then those coefficient vectors are simulated just like the parameter vectors from estimation results. If you do not specify the `variance()` option, then the coefficient vector is assumed to be nonstochastic and therefore is not simulated.

When you specify the `simulate()` option with `sim_technique residuals`, `forecast solve` first obtains static forecasts from your model for all possible periods. For each endogenous variable defined by a stochastic equation, it then computes residuals as the forecast value minus the actual value for all observations with nonmissing data. At each replication and for each period in the forecast horizon, `forecast solve` randomly selects one element from each stochastic equation’s pool of residuals before solving the model for that replication and period. Then whenever `forecast solve`
evaluates a stochastic equation, it adds the chosen element to the predicted value for that equation. Suppose an estimation result represents a multiple-equation estimator with \( m \) equations, and suppose that there are \( n \) time periods for which sample residuals are available. Arrange the residuals into the \( n \times m \) matrix \( R \). Then when \texttt{forecast solve} is randomly selecting residuals for this estimation result, it will choose a random number \( j \) between 1 and \( n \) and select the entire \( j \)th row from \( R \). That preserves the correlation structure among the error terms of the estimation result’s equations.

If you specify a coefficient vector using \texttt{forecast coefvector} and specify either the \texttt{variance()} option or the \texttt{errorvariance()} option (or both), \texttt{sim_technique residuals} considers the equation represented by the coefficient vector to be stochastic and resamples residuals for that equation.

When you specify the \texttt{simulate()} option with \texttt{sim_technique errors}, \texttt{forecast solve}, for each stochastic equation, replication, and period, takes a random draw from a multivariate normal distribution with zero mean before solving the model for that replication and period. Then whenever \texttt{forecast solve} evaluates a stochastic equation, it adds that random draw to the predicted value for that equation. The variance of the distribution from which errors are drawn is based on the estimation results for that equation. The \texttt{forecast} commands look in \texttt{e(rmse)}, \texttt{e(sigma)}, and \texttt{e(Sigma)} to find the estimated variance. If you add an estimation result that does not set any of those three macros and you request \texttt{sim_technique errors}, \texttt{forecast solve} exits with an error message. Multiple-equation commands typically set \texttt{e(Sigma)} so that the randomly drawn errors reflect the estimated error correlation structure.

If you specify a coefficient vector using \texttt{forecast coefvector} and specify the \texttt{errorvariance()} option, \texttt{sim_technique errors} simulates errors for that equation. Otherwise, the equation is treated like an identity and no errors are added.

\texttt{forecast solve} solves panel-data models by solving for all periods in the forecast horizon for the first panel in the dataset, then the second dataset, and so on. When you perform simulations with panel datasets, one replication is completed for all panels in the dataset before moving to the next replication. Simulations that include residual resampling select residuals from the pool containing residuals for all panels; \texttt{forecast solve} does not restrict itself to the static-forecast residuals for a single panel when simulating that panel.

References


Also see

[TS] \texttt{forecast} — Econometric model forecasting

[TS] \texttt{forecast adjust} — Adjust a variable by add factoring, replacing, etc.

[TS] \texttt{forecast drop} — Drop forecast variables

[R] \texttt{set seed} — Specify random-number seed and state
irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

Description

irf creates and manipulates IRF files that contain estimates of the IRFs, dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs) created after estimation by var, svar, or vec; see [TS] var, [TS] var svar, or [TS] vec.

irf creates and manipulates IRF files that contain estimates of the IRFs created after estimation by arima, arfima, dsge, or dsgenl; see [TS] arima, [TS] arfima, [DSGE] dsge, or [DSGE] dsgenl.

IRFs and FEVDs are described below, and the process of analyzing them is outlined. After reading this entry, please see [TS] irf create.

Quick start

Fit a VAR model

```stata
var y1 y2 y3
```

Create impulse–response function myirf and IRF file myirfs.irf

```stata
irf create myirf, set(myirfs)
```

Graph orthogonalized impulse–response function for dependent variables y1 and y2 given a shock to y1

```stata
irf graph oirf, impulse(y1) response(y1 y2)
```

As above, but present results in a table

```stata
irf table oirf, impulse(y1) response(y1 y2)
```

Note: irf commands can be used after var, svar, vec, arima, arfima, dsge, or dsgenl; see [TS] var, [TS] var svar, [TS] vec, [TS] arima, [TS] arfima, [DSGE] dsge, or [DSGE] dsgenl.

**irf** — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

**Syntax**

```
irf subcommand ... [, ...]
```

<table>
<thead>
<tr>
<th>subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>create IRF file containing IRFs, dynamic-multiplier functions, and FEVDs</td>
</tr>
<tr>
<td>set</td>
<td>set the active IRF file</td>
</tr>
<tr>
<td>graph</td>
<td>graph results from active file</td>
</tr>
<tr>
<td>cgraph</td>
<td>combine graphs of IRFs, dynamic-multiplier functions, and FEVDs</td>
</tr>
<tr>
<td>ograph</td>
<td>graph overlaid IRFs, dynamic-multiplier functions, and FEVDs</td>
</tr>
<tr>
<td>table</td>
<td>create tables of IRFs, dynamic-multiplier functions, and FEVDs from active file</td>
</tr>
<tr>
<td>ctable</td>
<td>combine tables of IRFs, dynamic-multiplier functions, and FEVDs</td>
</tr>
<tr>
<td>describe</td>
<td>describe contents of active file</td>
</tr>
<tr>
<td>add</td>
<td>add results from an IRF file to the active IRF file</td>
</tr>
<tr>
<td>drop</td>
<td>drop IRF results from active file</td>
</tr>
<tr>
<td>rename</td>
<td>rename IRF results within a file</td>
</tr>
</tbody>
</table>

IRF stands for impulse–response function; FEVD stands for forecast-error variance decomposition. `irf` can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see `[TS] var`, `[TS] var svar`, `[TS] vec`, `[TS] arima`, `[TS] arfima`, `[DSGE] dsge`, or `[DSGE] dsgenl`.

**Remarks and examples**

An IRF measures the effect of a shock to an endogenous variable on itself or on another endogenous variable; see Lütkepohl (2005, 51–63) and Hamilton (1994, 318–323) for formal definitions. Becketti (2013) provides an approachable, gentle introduction to IRF analysis. Of the many types of IRFs, `irf create` estimates the five most important: simple IRFs, orthogonalized IRFs, cumulative IRFs, cumulative orthogonalized IRFs, and structural IRFs.

A dynamic-multiplier function, or transfer function, measures the impact of a unit increase in an exogenous variable on the endogenous variables over time; see Lütkepohl (2005, chap. 10) for formal definitions. `irf create` estimates simple and cumulative dynamic-multiplier functions after `var`.

The forecast-error variance decomposition (FEVD) measures the fraction of the forecast-error variance of an endogenous variable that can be attributed to orthogonalized shocks to itself or to another endogenous variable; see Lütkepohl (2005, 63–66) and Hamilton (1994, 323–324) for formal definitions. Of the many types of FEVDs, `irf create` estimates the two most important: Cholesky and structural.
To analyze IRFs and FEVDs in Stata, you first fit a model, then use `irf create` to estimate the IRFs and FEVDs and save them in a file, and finally use `irf graph` or any of the other `irf` analysis commands to examine results:

```
use https://www.stata-press.com/data/r16/lutkepohl2
(output omitted)
var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(irf create order1, step(10) set(myirf1)
(file myirf1.irf created)
(file myirf1.irf now active)
(file myirf1.irf updated)
.irf graph oirf, impulse(dln_inc) response(dln_consump)
```

Multiple sets of IRFs and FEVDs can be placed in the same file, with each set of results in a file bearing a distinct name. The `irf create` command above created file `myirf1.irf` and put one set of results in it, named `order1`. The `order1` results include estimates of the simple IRFs, orthogonalized IRFs, cumulative IRFs, cumulative orthogonalized IRFs, and Cholesky FEVDs.

IRF files are just files: they can be erased by `erase`, listed by `dir`, and copied by `copy`; see [D] `erase`, [D] `dir`, and [D] `copy`.

Below we use the same estimated `var` but use a different Cholesky ordering to create a second set of IRF results, which we will save as `order2` in the same file, and then we will graph both results:
We have compared results for one model under two different identification schemes. We could just as well have compared results of two different models. We now use `irf table` to display the results tabularly:

```
. irf table oirf, irf(order1 order2) impulse(dln_inc) response(dln_consump)
```

<table>
<thead>
<tr>
<th>step</th>
<th>(1)</th>
<th>(1)</th>
<th>(1)</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oirf</td>
<td>Lower</td>
<td>Upper</td>
<td>oirf</td>
<td>Lower</td>
<td>Upper</td>
</tr>
<tr>
<td>0</td>
<td>.004934</td>
<td>.003016</td>
<td>.006852</td>
<td>.005244</td>
<td>.003252</td>
<td>.007237</td>
</tr>
<tr>
<td>1</td>
<td>.001309</td>
<td>-.000931</td>
<td>.003549</td>
<td>.001235</td>
<td>-.001011</td>
<td>.003482</td>
</tr>
<tr>
<td>2</td>
<td>.003573</td>
<td>.001285</td>
<td>.005862</td>
<td>.00391</td>
<td>.001542</td>
<td>.006278</td>
</tr>
<tr>
<td>3</td>
<td>-.000692</td>
<td>-.002333</td>
<td>.0095</td>
<td>-.000677</td>
<td>-.002347</td>
<td>.000993</td>
</tr>
<tr>
<td>4</td>
<td>.000905</td>
<td>-.000541</td>
<td>.002351</td>
<td>.00094</td>
<td>-.000576</td>
<td>.002456</td>
</tr>
<tr>
<td>5</td>
<td>.000328</td>
<td>-.0005</td>
<td>.001156</td>
<td>.000341</td>
<td>-.000518</td>
<td>.001201</td>
</tr>
<tr>
<td>6</td>
<td>.000021</td>
<td>-.000675</td>
<td>.000717</td>
<td>.000042</td>
<td>-.000693</td>
<td>.000777</td>
</tr>
<tr>
<td>7</td>
<td>.000154</td>
<td>-.000206</td>
<td>.000515</td>
<td>.000161</td>
<td>-.000218</td>
<td>.00054</td>
</tr>
<tr>
<td>8</td>
<td>.000026</td>
<td>-.000248</td>
<td>.0003</td>
<td>.000027</td>
<td>-.000261</td>
<td>.000315</td>
</tr>
<tr>
<td>9</td>
<td>.000026</td>
<td>-.000121</td>
<td>.000174</td>
<td>.00003</td>
<td>-.000125</td>
<td>.000184</td>
</tr>
<tr>
<td>10</td>
<td>.000026</td>
<td>-.000061</td>
<td>.000113</td>
<td>.000027</td>
<td>-.000065</td>
<td>.00012</td>
</tr>
</tbody>
</table>

95% lower and upper bounds reported

(1) irfname = order1, impulse = dln_inc, and response = dln_consump
(2) irfname = order2, impulse = dln_inc, and response = dln_consump

Both the table and the graph show that the two orthogonalized IRFs are essentially the same. In both functions, an increase in the orthogonalized shock to dln_inc causes a short series of increases in dln_consump that dies out after four or five periods.
References

Becketti, S. 2013. *Introduction to Time Series Using Stata*. College Station, TX: Stata Press.


Also see

[TS] *arfima* — Autoregressive fractionally integrated moving-average models

[TS] *arima* — ARIMA, ARMAX, and other dynamic regression models

[TS] *var* — Vector autoregressive models

[TS] *var intro* — Introduction to vector autoregressive models

[TS] *var svar* — Structural vector autoregressive models

[TS] *varbasic* — Fit a simple VAR and graph IRFs or FEVDs

[TS] *vec* — Vector error-correction models

[TS] *vec intro* — Introduction to vector error-correction models

[DSGE] *dsge* — Linear dynamic stochastic general equilibrium models

[DSGE] *dsgenl* — Nonlinear dynamic stochastic general equilibrium models
**irf add** — Add results from an IRF file to the active IRF file

<table>
<thead>
<tr>
<th>Description</th>
<th>Quick start</th>
<th>Menu</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>irf add</td>
<td>copies results from an existing IRF file on disk to the active IRF file, set by <em>irf set</em>; see [TS] <em>irf set</em>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Quick start**

Copy the IRF results *myirf1* from *myirfs.irf* to *newirf* in the active IRF file

```
irf add newirf = myirf1, using(myirfs)
```

As above, but copy all IRF results from *myirfs.irf* to the active file

```
irf add _all, using(myirfs)
```

Note: *irf* commands can be used after *var*, *svar*, *vec*, *arima*, *arfima*, *dsge*, or *dsgenl*; see [TS] *var*, [TS] *var svar*, [TS] *vec*, [TS] *arima*, [TS] *arfima*, [DSGE] *dsge*, or [DSGE] *dsgenl*.

**Menu**

Statistics > Multivariate time series > Manage IRF results and files > Add IRF results
Syntax

```
irf add { _all | [newname=] oldname ... }, using(irf_filename)
```

Option

```
using(irf_filename)
```

specifies the file from which results are to be obtained and is required. If
`irf_filename` is specified without an extension, `.irf` is assumed.

Remarks and examples

If you have not read [TS] irf, please do so.

Example 1

After fitting a VAR model, we create two separate IRF files:

```stata
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
. irf create original, set(irf1, replace)
(file irf1.irf created)
(file irf1.irf now active)
(file irf1.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump) set(irf2, replace)
(file irf2.irf created)
(file irf2.irf now active)
(file irf2.irf updated)
```

We copy IRF results `original` to the active file giving them the name `order1`.

```stata
. irf add order1 = original, using(irf1)
(file irf2.irf updated)
```

Here we create new IRF results and save them in the new file `irf3`.

```stata
. irf create order3, order(dln_inc dln_consump dln_inv) set(irf3, replace)
(file irf3.irf created)
(file irf3.irf now active)
(file irf3.irf updated)
```

Now we copy all the IRF results in file `irf2` into the active file.

```stata
. irf add _all, using(irf2)
(file irf3.irf updated)
```

Also see

[TS] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] var intro — Introduction to vector autoregressive models
[TS] vec intro — Introduction to vector error-correction models
irf cgraph — Combined graphs of IRFs, dynamic-multiplier functions, and FEVDs

Description

.irf cgraph makes a graph or a combined graph of IRF results. A graph is drawn for specified combinations of named IRF results, impulse variables, response variables, and statistics. irf cgraph combines these graphs into one image, unless separate graphs are requested.

.irf cgraph operates on the active IRF file; see [TS] irf set.

Quick start

Combine graphs of an orthogonalized IRF myirf and cumulative IRF mycirf for dependent variables y1 and y2

.irf cgraph (myirf y1 y2 oirf) (mycirf y1 y2 cirf)

As above, but suppress confidence bands and add a title

.irf cgraph (myirf y1 y2 oirf) (mycirf y1 y2 cirf), noci ///
  title("My Title")

Note: irf commands can be used after var, svar, vec, arima, arfima, dsge, or dsgenl; see [TS] var, [TS] var svar, [TS] vec, [TS] arima, [TS] arfima, [DSGE] dsge, or [DSGE] dsgenl.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Combined graphs
Syntax

```
irf cgraph (spec_1) [(spec_2) ... (spec_N)] [, options]
```

where \( (spec_k) \) is

```
(irfname impulsevar responsevar stat [, spec_options])
```

\( irfname \) is the name of a set of IRF results in the active IRF file. \( impulsevar \) should be specified as an endogenous variable for all statistics except \( dm \) and \( cdm \); for those, specify as an exogenous variable. \( responsevar \) is an endogenous variable name. \( stat \) is one or more statistics from the list below:

<table>
<thead>
<tr>
<th>( stat )</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( irf )</td>
<td>impulse–response function</td>
</tr>
<tr>
<td>( oirf )</td>
<td>orthogonalized impulse–response function</td>
</tr>
<tr>
<td>( dm )</td>
<td>dynamic-multiplier function</td>
</tr>
<tr>
<td>( cirf )</td>
<td>cumulative impulse–response function</td>
</tr>
<tr>
<td>( coirf )</td>
<td>cumulative orthogonalized impulse–response function</td>
</tr>
<tr>
<td>( cdm )</td>
<td>cumulative dynamic-multiplier function</td>
</tr>
<tr>
<td>( fevd )</td>
<td>Cholesky forecast-error variance decomposition</td>
</tr>
<tr>
<td>( sirf )</td>
<td>structural impulse–response function</td>
</tr>
<tr>
<td>( sfevd )</td>
<td>structural forecast-error variance decomposition</td>
</tr>
</tbody>
</table>

Notes: 1. No statistic may appear more than once.
2. If confidence intervals are included (the default), only two statistics may be included.
3. If confidence intervals are suppressed (option \( noci \)), up to four statistics may be included.

<table>
<thead>
<tr>
<th>( options )</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( set(filename) )</td>
<td>make ( filename ) active</td>
</tr>
</tbody>
</table>

**Options**

- \( combine_options \) affect appearance of combined graph
- \( twoway_options \) any options other than by() documented in [G-3] `twoway_options`
- \( * spec_options \) level, steps, and rendition of plots and their CIs
- \( individual \) graph each combination individually

\* `spec_options` appear on multiple tabs in the dialog box.
\*\* `individual` does not appear in the dialog box.
spec_options | Description
---|---
Main | noci | suppress confidence bands
Options | level(#) | set confidence level; default is level(95)
 | lstep(#) | use # for first step
 | ustep(#) | use # for maximum step
Plots | plot#opts(line_options) | affect rendition of the line plotting the # stat
CI plots | ci#opts(area_options) | affect rendition of the confidence interval for the # stat

spec_options may be specified within a graph specification, globally, or in both. When specified in a graph specification, the spec_options affect only the specification in which they are used. When supplied globally, the spec_options affect all graph specifications. When supplied in both places, options in the graph specification take precedence.

Options

Main

- noci suppresses graphing the confidence interval for each statistic. noci is assumed when the model was fit by vec because no confidence intervals were estimated.

- set(filename) specifies the file to be made active; see [TS] irf set. If set() is not specified, the active file is used.

Options

- level(#) specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is level(95) or as set by set level; see [U] 20.8 Specifying the width of confidence intervals. The value set of an overall level() can be overridden by the level() inside a (spec.).

- lstep(#) specifies the first step, or period, to be included in the graph. lstep(0) is the default.

- ustep(#), # ≥ 1, specifies the maximum step, or period, to be included in the graph.

- combine_options affect the appearance of the combined graph; see [G-2] graph combine.

Plots

- plot1opts(cline_options), ..., plot4opts(cline_options) affect the rendition of the plotted statistics. plot1opts() affects the rendition of the first statistic; plot2opts(), the second; and so on. cline_options are as described in [G-3] cline_options.

CI plots

- ci1opts1(area_options) and ci2opts2(area_options) affect the rendition of the confidence intervals for the first (ci1opts1()) and second (ci2opts2()) statistics. See [TS] irf graph for a description of this option and [G-3] area_options for the suboptions that change the look of the CI.
Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option).

The following option is available with irf cgraph but is not shown in the dialog box:
individual specifies that each graph be displayed individually. By default, irf cgraph combines the subgraphs into one image.

Remarks and examples

If you have not read [TS] irf, please do so.

The relationship between irf cgraph and irf graph is syntactically and conceptually the same as that between irf ctable and irf table; see [TS] irf ctable for a description of the syntax.

irf cgraph is much the same as using irf graph to make individual graphs and then using graph combine to put them together. If you cannot use irf cgraph to do what you want, consider the other approach.

> Example 1

You have previously issued the commands:

```
. use https://www.stata-press.com/data/r16/lutkepohl2
. mat a = (., 0, 0
               0,.,0
               .,.,.)
. mat b = I(3)
. svar dln_inv dln_inc dln_consump, aeq(a) beq(b)
. irf create modela, set(results3) step(8)
. svar dln_inc dln_inv dln_consump, aeq(a) beq(b)
. irf create modelb, step(8)
```
You now type

```
.irf cgraph (modela dln_inc dln_consump oirf sirf)
> (modelb dln_inc dln_consump fevd sfevd, lstep(1))
> (modela dln_inc dln_consump fevd sfevd, lstep(1)),
> title("Results from modela and modelb")
```

Results from modela and modelb

```
Results from modela and modelb

<table>
<thead>
<tr>
<th>Step</th>
<th>95% CI for oirf</th>
<th>95% CI for sirf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-.002</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>.002</td>
</tr>
<tr>
<td>3</td>
<td>.004</td>
<td>.006</td>
</tr>
<tr>
<td>4</td>
<td>.006</td>
<td>.008</td>
</tr>
<tr>
<td>5</td>
<td>.01</td>
<td>.012</td>
</tr>
<tr>
<td>6</td>
<td>.012</td>
<td>.014</td>
</tr>
<tr>
<td>7</td>
<td>.014</td>
<td>.016</td>
</tr>
<tr>
<td>8</td>
<td>.016</td>
<td>.018</td>
</tr>
</tbody>
</table>
```

```
Results from modela and modelb

<table>
<thead>
<tr>
<th>Step</th>
<th>95% CI for fevd</th>
<th>95% CI for sfevd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-.1</td>
<td>-.2</td>
</tr>
<tr>
<td>2</td>
<td>-.2</td>
<td>-.3</td>
</tr>
<tr>
<td>3</td>
<td>-.3</td>
<td>-.4</td>
</tr>
<tr>
<td>4</td>
<td>-.4</td>
<td>-.5</td>
</tr>
<tr>
<td>5</td>
<td>-.5</td>
<td>-.6</td>
</tr>
<tr>
<td>6</td>
<td>-.6</td>
<td>-.7</td>
</tr>
<tr>
<td>7</td>
<td>-.7</td>
<td>-.8</td>
</tr>
<tr>
<td>8</td>
<td>-.8</td>
<td>-.9</td>
</tr>
</tbody>
</table>
```

Stored results

```
irf cgraph stores the following in r():
```

Scalars

- \( r(k) \) number of specific graph commands

Macros

- \( r(individual) \) individual, if specified
- \( r(save) \) filename, replace from saving() option for combined graph
- \( r(name) \) name, replace from name() option for combined graph
- \( r(title) \) title of the combined graph
- \( r(save#) \) filename, replace from saving() option for individual graphs
- \( r(name#) \) name, replace from name() option for individual graphs
- \( r(title#) \) title for the \#th graph
- \( r(ci#) \) level applied to the \#th confidence interval or noci
- \( r(response#) \) response specified in the \#th command
- \( r(impulse#) \) impulse specified in the \#th command
- \( r(irfname#) \) IRF name specified in the \#th command
- \( r(stats#) \) statistics specified in the \#th command

Also see

- [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
- [var intro](#) — Introduction to vector autoregressive models
- [vec intro](#) — Introduction to vector error-correction models
**Description**

`irf create` estimates multiple sets of impulse–response functions (IRFs), dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs). All of these estimates and their standard errors are known collectively as IRF results and are saved in an IRF file under a specified filename. Once you have created a set of IRF results, you can use the other `irf` commands to analyze them.

**Quick start**

Create impulse–response function `myirf` with 8 forecast periods in the active IRF file

```plaintext
irf create myirf
```

As above, and use IRF file `myirfs.irf`

```plaintext
irf create myirf, set(myirfs)
```

As above, but compute the IRF for 12 periods

```plaintext
irf create myirf, set(myirfs) step(12)
```

**Note:** `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see `[TS] var`, `[TS] var svar`, `[TS] vec`, `[TS] arima`, `[TS] arfima`, `[DSGE] dsge`, or `[DSGE] dsgenl`.

**Menu**

Statistics > Multivariate time series > IRF and FEVD analysis > Obtain IRFs, dynamic-multiplier functions, and FEVDs
**Syntax**

After `var`

```
irf create `irfname' [, `var_options']
```

After `svar`

```
irf create `irfname' [, `svar_options']
```

After `vec`, `dsge`, or `dsgenl`

```
irf create `irfname' [, `vec_options']
```

After `arima`

```
irf create `irfname' [, `arima_options']
```

After `arfima`

```
irf create `irfname' [, `arfima_options']
```

`irfname` is any valid name that does not exceed 15 characters.

<table>
<thead>
<tr>
<th><code>var_options</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td></td>
</tr>
<tr>
<td><code>set(filename[, replace])</code></td>
<td>make <code>filename</code> active</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>replace <code>irfname</code> if it already exists</td>
</tr>
<tr>
<td><code>step(#)</code></td>
<td>set forecast horizon to <code>#</code>; default is <code>step(8)</code></td>
</tr>
<tr>
<td><code>order(varlist)</code></td>
<td>specify Cholesky ordering of endogenous variables</td>
</tr>
<tr>
<td><code>estimates(estname)</code></td>
<td>use previously stored results <code>estname</code>; default is to use active results</td>
</tr>
<tr>
<td><strong>Std. errors</strong></td>
<td></td>
</tr>
<tr>
<td><code>nose</code></td>
<td>do not calculate standard errors</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>obtain standard errors from bootstrapped residuals</td>
</tr>
<tr>
<td><code>bsp</code></td>
<td>obtain standard errors from parametric bootstrap</td>
</tr>
<tr>
<td><code>nodots</code></td>
<td>do not display “.” for each bootstrap replication</td>
</tr>
<tr>
<td><code>reps(#)</code></td>
<td>use <code>#</code> bootstrap replications; default is <code>reps(200)</code></td>
</tr>
<tr>
<td><code>bsaving(filename[, replace])</code></td>
<td>save bootstrap results in <code>filename</code></td>
</tr>
</tbody>
</table>
### svar_options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set(filename[, replace])</code></td>
<td>make filename active</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>replace irfname if it already exists</td>
</tr>
<tr>
<td><code>step(#)</code></td>
<td>set forecast horizon to #; default is step(8)</td>
</tr>
<tr>
<td><code>estimates(estname)</code></td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
</tbody>
</table>

### Std. errors

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nose</code></td>
<td>do not calculate standard errors</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>obtain standard errors from bootstrapped residual</td>
</tr>
<tr>
<td><code>bsp</code></td>
<td>obtain standard errors from parametric bootstrap</td>
</tr>
<tr>
<td><code>nodots</code></td>
<td>do not display “.” for each bootstrap replication</td>
</tr>
<tr>
<td><code>reps(#)</code></td>
<td>use # bootstrap replications; default is reps(200)</td>
</tr>
<tr>
<td><code>bsaving(filename[, replace])</code></td>
<td>save bootstrap results in filename</td>
</tr>
</tbody>
</table>

### vec_options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set(filename[, replace])</code></td>
<td>make filename active</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>replace irfname if it already exists</td>
</tr>
<tr>
<td><code>step(#)</code></td>
<td>set forecast horizon to #; default is step(8)</td>
</tr>
<tr>
<td><code>estimates(estname)</code></td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
</tbody>
</table>

### arima_options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set(filename[, replace])</code></td>
<td>make filename active</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>replace irfname if it already exists</td>
</tr>
<tr>
<td><code>step(#)</code></td>
<td>set forecast horizon to #; default is step(8)</td>
</tr>
<tr>
<td><code>estimates(estname)</code></td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
</tbody>
</table>

### Std. errors

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nose</code></td>
<td>do not calculate standard errors</td>
</tr>
</tbody>
</table>

### arfima_options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set(filename[, replace])</code></td>
<td>make filename active</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>replace irfname if it already exists</td>
</tr>
<tr>
<td><code>step(#)</code></td>
<td>set forecast horizon to #; default is step(8)</td>
</tr>
<tr>
<td><code>smemory</code></td>
<td>calculate short-memory IRFs</td>
</tr>
<tr>
<td><code>estimates(estname)</code></td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
</tbody>
</table>

### Std. errors

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nose</code></td>
<td>do not calculate standard errors</td>
</tr>
</tbody>
</table>
The default is to use asymptotic standard errors if no options are specified.

`irf create` is for use after fitting a model with the `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl` command; see [TS] `var`, [TS] `svar`, [TS] `vec`, [TS] `arima`, [TS] `arfima`, [DSGE] `dsge`, or [DSGE] `dsgenl`.

You must `tsset` your data before using `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl` and, hence, before using `irf create`; see [TS] `tsset`.

### Options

#### Main

- `set(filename[, replace])` specifies the IRF file to be used. If `set()` is not specified, the active IRF file is used; see [TS] `irf set`.

  If `set()` is specified, the specified file becomes the active file, just as if you had issued an `irf set` command.

- `replace` specifies that the results saved under `irfname` may be replaced, if they already exist. IRF results are saved in files, and one file may contain multiple IRF results.

- `step(#)` specifies the step (forecast) horizon; the default is eight periods.

- `order(varlist)` is allowed only after estimation by `var`; it specifies the Cholesky ordering of the endogenous variables to be used when estimating the orthogonalized IRFs. By default, the order in which the variables were originally specified on the `var` command is used.

- `smemory` is allowed only after estimation by `arfima`; it specifies that the IRFs are calculated based on a short-memory model with the fractional difference parameter $d$ set to zero.

- `estimates(estname)` specifies that estimation results previously estimated by `var`, `svar`, or `vec`, and stored by `estimates`, be used. This option is rarely specified; see [R] `estimates`.

#### Std. errors

- `nose`, `bs`, and `bsp` are alternatives that specify how (whether) standard errors are to be calculated. If none of these options is specified, asymptotic standard errors are calculated, except in two cases: after estimation by `vec` and after estimation by `svar` in which long-run constraints were applied. In those two cases, the default is as if `nose` were specified, although in the second case, you could specify `bs` or `bsp`. After estimation by `vec`, standard errors are simply not available.

  - `nose` specifies that no standard errors be calculated.
  - `bs` specifies that standard errors be calculated by bootstrapping the residuals. `bs` may not be specified if there are gaps in the data.
  - `bsp` specifies that standard errors be calculated via a multivariate-normal parametric bootstrap. `bsp` may not be specified if there are gaps in the data.

- `nodots`, `reps(#)`, and `bsaving(filename[, replace])` are relevant only if `bs` or `bsp` is specified.

  - `nodots` specifies that dots not be displayed each time `irf create` performs a bootstrap replication.
  - `reps(#)`, $# > 50$, specifies the number of bootstrap replications to be performed. `reps(200)` is the default.

- `bsaving(filename[, replace])` specifies that file `filename` be created and that the bootstrap replications be saved in it. New file `filename` is just a `.dta` dataset that can be loaded later using `use`; see [D] `use`. If `filename` is specified without an extension, `.dta` is assumed.
Remarks and examples

If you have not read [TS] irf, please do so. An introductory example using IRFs is presented there.

*irf create* estimates several types of IRFs, dynamic-multiplier functions, and FEVDs. Which estimates are saved depends on the estimation method previously used to fit the model, as summarized in the table below:

<table>
<thead>
<tr>
<th>Saves</th>
<th>arima</th>
<th>arfima</th>
<th>var</th>
<th>svar</th>
<th>vec</th>
<th>dsge/ dsge/</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple IRFs</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>orthogonalized IRFs</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>dynamic multipliers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cumulative IRFs</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>cumulative orthogonalized IRFs</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>cumulative dynamic multipliers</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>structural IRFs</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cholesky FEVDs</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>structural FEVDs</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remarks are presented under the following headings:

- Introductory examples
- Technical aspects of IRF files
- IRFs and FEVDs
- IRF results for VARs
  - An introduction to impulse–response functions for VARs
  - An introduction to dynamic-multiplier functions for VARs
  - An introduction to forecast-error variance decompositions for VARs
- IRF results for VECMs
  - An introduction to impulse–response functions for VECMs
  - An introduction to forecast-error variance decompositions for VECMs
- IRF results for ARIMA and ARFIMA

Introductory examples

**Example 1: After var**

Below we compare bootstrap and asymptotic standard errors for a specific FEVD. We begin by fitting a VAR(2) model to the Lütkepohl data (we use the *var* command). We next use the *irf create* command twice, first to create results with asymptotic standard errors (saved under the name *asym*) and then to re-create the same results, this time with bootstrap standard errors (saved under the name *bs*). Because bootstrapping is a random process, we set the random-number seed (**set seed 123456**) before using *irf create* the second time; this makes our results reproducible. Finally, we compare results by using the IRF analysis command *irf ctable*. 
. use https://www.stata-press.com/data/r16/lutkepohl2
   (Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4), lags(1/2)
   (output omitted)
. irf create asymp, step(8) set(results1)
   (file results1.irf created)
   (file results1.irf now active)
   (file results1.irf updated)
. set seed 123456
. irf create bs, step(8) bs reps(250) nodots
   (file results1.irf updated)
. irf ctable (asymp dln_inc dln_consump fevd) (bs dln_inc dln_consump fevd),
>   noci stderror

<table>
<thead>
<tr>
<th>step</th>
<th>(1) fevd</th>
<th>S.E.</th>
<th>(2) fevd</th>
<th>S.E.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>.282135</td>
<td>.087373</td>
<td>.282135</td>
<td>.102756</td>
</tr>
<tr>
<td>2</td>
<td>.278777</td>
<td>.083782</td>
<td>.278777</td>
<td>.098161</td>
</tr>
<tr>
<td>3</td>
<td>.33855</td>
<td>.090006</td>
<td>.33855</td>
<td>.10586</td>
</tr>
<tr>
<td>4</td>
<td>.339942</td>
<td>.089207</td>
<td>.339942</td>
<td>.104191</td>
</tr>
<tr>
<td>5</td>
<td>.342813</td>
<td>.090494</td>
<td>.342813</td>
<td>.105351</td>
</tr>
<tr>
<td>6</td>
<td>.343119</td>
<td>.090517</td>
<td>.343119</td>
<td>.105258</td>
</tr>
<tr>
<td>7</td>
<td>.343079</td>
<td>.090499</td>
<td>.343079</td>
<td>.105266</td>
</tr>
<tr>
<td>8</td>
<td>.34315</td>
<td>.090569</td>
<td>.34315</td>
<td>.105303</td>
</tr>
</tbody>
</table>

(1) irfname = asymp, impulse = dln_inc, and response = dln_consump
(2) irfname = bs, impulse = dln_inc, and response = dln_consump

Point estimates are, of course, the same. The bootstrap estimates of the standard errors, however, are larger than the asymptotic estimates, which suggests that the sample size of 71 is not large enough for the distribution of the estimator of the FEVD to be well approximated by the asymptotic distribution. Here we would expect the bootstrap confidence interval to be more reliable than the confidence interval that is based on the asymptotic standard error.

Technical note

The details of the bootstrap algorithms are given in *Methods and formulas*. These algorithms are conditional on the first \( p \) observations, where \( p \) is the order of the fitted \( \text{VAR} \). (In an \( \text{SVAR} \) model, \( p \) is the order of the \( \text{VAR} \) that underlies the \( \text{SVAR} \).) The bootstrapped estimates are conditional on the first \( p \) observations, just as the estimators of the coefficients in \( \text{VAR} \) models are conditional on the first \( p \) observations. With bootstrap standard errors (option \( \text{bs} \)), the \( p \) initial observations are used with resampling the residuals to produce the bootstrap samples used for estimation. With the more parametric bootstrap (option \( \text{bsp} \)), the \( p \) initial observations are used with draws from a multivariate normal distribution with variance–covariance matrix \( \hat{\Sigma} \) to generate the bootstrap samples.

Technical note

For \texttt{var} and \texttt{svar e()} results, \texttt{irf} uses \( \hat{\Sigma} \), the estimated variance matrix of the disturbances, in computing the asymptotic standard errors of all the functions. The point estimates of the orthogonalized impulse–response functions, the structural impulse–response functions, and all the variance
decompositions also depend on \( \Sigma \). As discussed in \([TS]\) \texttt{var}, \texttt{var} and \texttt{svar} use the ML estimator of this matrix by default, but they have option \texttt{dfk}, which will instead use an estimator that includes a small-sample correction. Specifying \texttt{dfk} when the model is fit—when the \texttt{var} or \texttt{svar} command is given—changes the estimate of \( \Sigma \) and will change the IRF results that depend on it.

Example 2: After \texttt{var} with exogenous variables

After fitting a VAR, \texttt{irf create} computes estimates of the dynamic multipliers, which describe the impact of a unit change in an exogenous variable on each endogenous variable. For instance, below we estimate and report the cumulative dynamic multipliers from a model in which changes in investment are exogenous. The results indicate that both of the cumulative dynamic multipliers are significant.

```stata
. var dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4), lags(1/2)
    > exog(L(0/2).dln_inv)
    (output omitted)
. irf create dm, step(8)
    (file results1.irf updated)
. irf table cdm, impulse(dln_inv) irf(dm)
```

<table>
<thead>
<tr>
<th>step</th>
<th>(1)</th>
<th>(1)</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cdm</td>
<td>Lower</td>
<td>Upper</td>
</tr>
<tr>
<td>0</td>
<td>.032164</td>
<td>-.027215</td>
<td>.091544</td>
</tr>
<tr>
<td>1</td>
<td>.096658</td>
<td>.003479</td>
<td>.189656</td>
</tr>
<tr>
<td>2</td>
<td>.140107</td>
<td>.022897</td>
<td>.257317</td>
</tr>
<tr>
<td>3</td>
<td>.150527</td>
<td>.032116</td>
<td>.268938</td>
</tr>
<tr>
<td>4</td>
<td>.148979</td>
<td>.031939</td>
<td>.26602</td>
</tr>
<tr>
<td>5</td>
<td>.151247</td>
<td>.033011</td>
<td>.269482</td>
</tr>
<tr>
<td>6</td>
<td>.150267</td>
<td>.033202</td>
<td>.267331</td>
</tr>
<tr>
<td>7</td>
<td>.150336</td>
<td>.032858</td>
<td>.267813</td>
</tr>
<tr>
<td>8</td>
<td>.150525</td>
<td>.033103</td>
<td>.267948</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>step</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cdm</td>
<td>Lower</td>
<td>Upper</td>
</tr>
<tr>
<td>0</td>
<td>.058681</td>
<td>.012529</td>
<td>.104832</td>
</tr>
<tr>
<td>1</td>
<td>.062723</td>
<td>-.005058</td>
<td>.130504</td>
</tr>
<tr>
<td>2</td>
<td>.126167</td>
<td>.032497</td>
<td>.219837</td>
</tr>
<tr>
<td>3</td>
<td>.136583</td>
<td>.038691</td>
<td>.234476</td>
</tr>
<tr>
<td>4</td>
<td>.146482</td>
<td>.04442</td>
<td>.248543</td>
</tr>
<tr>
<td>5</td>
<td>.146075</td>
<td>.045201</td>
<td>.24695</td>
</tr>
<tr>
<td>6</td>
<td>.145542</td>
<td>.044988</td>
<td>.246096</td>
</tr>
<tr>
<td>7</td>
<td>.146309</td>
<td>.045315</td>
<td>.247304</td>
</tr>
<tr>
<td>8</td>
<td>.145786</td>
<td>.045206</td>
<td>.246365</td>
</tr>
</tbody>
</table>

95% lower and upper bounds reported

(1) \texttt{irfname = dm}, \texttt{impulse = dln_inv}, and \texttt{response = dln_inc}  
(2) \texttt{irfname = dm}, \texttt{impulse = dln_inv}, and \texttt{response = dln_consump}
Example 3: After vec

Although all IRFs and orthogonalized IRFs (OIRFs) from models with stationary variables will taper off to zero, some of the IRFs and OIRFs from models with first-difference stationary variables will not. This is the key difference between IRFs and OIRFs from systems of stationary variables fit by var or svar and those obtained from systems of first-difference stationary variables fit by vec. When the effect of the innovations dies out over time, the shocks are said to be transitory. In contrast, when the effect does not taper off, shocks are said to be permanent.

In this example, we look at the OIRF from one of the VECMs fit to the unemployment-rate data analyzed in example 2 of [TS] vec. We see that an orthogonalized shock to Indiana has a permanent effect on the unemployment rate in Missouri:

```
. use https://www.stata-press.com/data/r16/urates
. vec missouri indiana kentucky illinois, trend(rconstant) rank(2) lags(4)
(output omitted)
. irf create vec1, set(vecirfs) step(50)
(file vecirfs.irf created)
(file vecirfs.irf now active)
(file vecirfs.irf updated)
```

Now we can use `irf graph` to graph the OIRF of interest:

```
. irf graph oirf, impulse(indiana) response(missouri)
```

![Graph showing the OIRF of interest](image)

The graph shows that the estimated OIRF converges to a positive asymptote, which indicates that an orthogonalized innovation to the unemployment rate in Indiana has a permanent effect on the unemployment rate in Missouri.

Technical aspects of IRF files

This section is included for programmers wishing to extend the `irf` system.

`irf create` estimates a series of impulse–response functions and their standard errors. Although these estimates are saved in an IRF file, most users will never need to look at the contents of this file. The IRF commands fill in, analyze, present, and manage IRF results.
IRF files are just Stata datasets that have names ending in .irf instead of .dta. The dataset in the file has a nested panel structure.

Variable `irfname` contains the `irfname` specified by the user. Variable `impulse` records the name of the endogenous variable whose innovations are the impulse. Variable `response` records the name of the endogenous variable that is responding to the innovations. In a model with $K$ endogenous variables, there are $K^2$ combinations of `impulse` and `response`. Variable `step` records the periods for which these estimates were computed.

Below is a catalog of the statistics that `irf create` estimates and the variable names under which they are saved in the IRF file.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>impulse–response functions</td>
<td>irf</td>
</tr>
<tr>
<td>orthogonalized impulse–response functions</td>
<td>oirf</td>
</tr>
<tr>
<td>dynamic-multiplier functions</td>
<td>dm</td>
</tr>
<tr>
<td>cumulative impulse–response functions</td>
<td>cirf</td>
</tr>
<tr>
<td>cumulative orthogonalized impulse–response functions</td>
<td>coirf</td>
</tr>
<tr>
<td>cumulative dynamic-multiplier functions</td>
<td>cdm</td>
</tr>
<tr>
<td>Cholesky forecast-error decomposition</td>
<td>fevd</td>
</tr>
<tr>
<td>structural impulse–response functions</td>
<td>sirf</td>
</tr>
<tr>
<td>structural forecast-error decomposition</td>
<td>sfevd</td>
</tr>
<tr>
<td>standard error of the impulse–response functions</td>
<td>stdirf</td>
</tr>
<tr>
<td>standard error of the orthogonalized impulse–response functions</td>
<td>stdoirf</td>
</tr>
<tr>
<td>standard error of the cumulative impulse–response functions</td>
<td>stdcirf</td>
</tr>
<tr>
<td>standard error of the cumulative orthogonalized impulse–response functions</td>
<td>stdcoirf</td>
</tr>
<tr>
<td>standard error of the Cholesky forecast-error decomposition</td>
<td>stdfevd</td>
</tr>
<tr>
<td>standard error of the structural impulse–response functions</td>
<td>stdsirf</td>
</tr>
<tr>
<td>standard error of the structural forecast-error decomposition</td>
<td>stdsfevd</td>
</tr>
</tbody>
</table>
In addition to the variables, information is stored in _dta characteristics. Much of the following information is also available in r() after irf describe, where it is often more convenient to obtain the information. Characteristic _dta[version] contains the version number of the IRF file, which is currently 1.1. Characteristic _dta[irfnames] contains a list of all the irfnames in the IRF file. For each irfname, there are a series of additional characteristics:

<table>
<thead>
<tr>
<th>Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>_dta[irfname_model]</td>
<td>var, ar var, lr var, vec, arima, arfima, dsge, or dsgen</td>
</tr>
<tr>
<td>_dta[irfname_order]</td>
<td>Cholesky order used in IRF estimates</td>
</tr>
<tr>
<td>_dta[irfname_exog]</td>
<td>exogenous variables, and their lags, in VAR</td>
</tr>
<tr>
<td>_dta[irfname_exogvars]</td>
<td>exogenous variables in VAR</td>
</tr>
<tr>
<td>_dta[irfname_constant]</td>
<td>constant or noconstant, depending on whether noconstant was specified in var or svar</td>
</tr>
<tr>
<td>_dta[irfname_lags]</td>
<td>lags in model</td>
</tr>
<tr>
<td>_dta[irfname_exlags]</td>
<td>lags of exogenous variables in model</td>
</tr>
<tr>
<td>_dta[irfname_tmin]</td>
<td>minimum value of timevar in the estimation sample</td>
</tr>
<tr>
<td>_dta[irfname_tmax]</td>
<td>maximum value of timevar in the estimation sample</td>
</tr>
<tr>
<td>_dta[irfname_timevar]</td>
<td>name of tsset timevar</td>
</tr>
<tr>
<td>_dta[irfname_tsfmt]</td>
<td>format of timevar</td>
</tr>
<tr>
<td>_dta[irfname_varcns]</td>
<td>constrained or colon-separated list of constraints placed on VAR coefficients</td>
</tr>
<tr>
<td>_dta[irfname_svarcns]</td>
<td>constrained or colon-separated list of constraints placed on VAR coefficients</td>
</tr>
<tr>
<td>_dta[irfname_step]</td>
<td>maximum step in IRF estimates</td>
</tr>
<tr>
<td>_dta[irfname_stdererror]</td>
<td>asymptotic, bs, bsp, or none, depending on the type of standard errors requested</td>
</tr>
<tr>
<td>_dta[irfname_reps]</td>
<td>number of bootstrap replications performed</td>
</tr>
<tr>
<td>_dta[irfname_version]</td>
<td>version of the IRF file that originally held irfname IRF results</td>
</tr>
<tr>
<td>_dta[irfname_rank]</td>
<td>number of cointegrating equations</td>
</tr>
<tr>
<td>_dta[irfname_trend]</td>
<td>trend() specified in vec</td>
</tr>
<tr>
<td>_dta[irfname_veccns]</td>
<td>constraints placed on VECM parameters</td>
</tr>
<tr>
<td>_dta[irfname_sind]</td>
<td>normalized seasonal indicators included in vec</td>
</tr>
<tr>
<td>_dta[irfname_d]</td>
<td>fractional difference parameter d in arfima</td>
</tr>
</tbody>
</table>

**IRFs and FEVDs**

`irf create` can estimate several types of IRFs and FEVDs for VARs and VECMs. `irf create` can also estimate IRFs and cumulative IRFs for ARIMA and ARFIMA models. We first discuss IRF results for VAR and SVAR models, and then we discuss them in the context of VECMs. Because the cointegrating VECM is an extension of the stationary VAR framework, the section that discusses the IRF results for VECMs draws on the earlier VAR material. We conclude our discussion with IRF results for ARIMA and ARFIMA models.
IRF results for VARs

An introduction to impulse–response functions for VARs

A $p$th-order vector autoregressive model (VAR) with exogenous variables is given by

$$y_t = v + A_1 y_{t-1} + \cdots + A_p y_{t-p} + B x_t + u_t$$

where

$$y_t = (y_{1t}, \ldots, y_{Kt})'$$ is a $K \times 1$ random vector,  
the $A_i$ are fixed $K \times K$ matrices of parameters,  
$x_t$ is an $R_0 \times 1$ vector of exogenous variables,  
$B$ is a $K \times R_0$ matrix of coefficients,  
v is a $K \times 1$ vector of fixed parameters, and  
u$u_t$ is assumed to be white noise; that is,

$$E(u_t) = 0$$
$$E(u_t u'_s) = \Sigma$$
$$E(u_t u'_s) = 0 \text{ for } t \neq s$$

As discussed in [TS] varstable, a VAR can be rewritten in moving-average form only if it is stable. Any exogenous variables are assumed to be covariance stationary. Because the functions of interest in this section depend only on the exogenous variables through their effect on the estimated $A_i$, we can simplify the notation by dropping them from the analysis. All the formulas given below still apply, although the $A_i$ are estimated jointly with $B$ on the exogenous variables.

Below we discuss conditions under which the IRFs and forecast-error variance decompositions have a causal interpretation. Although estimation requires only that the exogenous variables be predetermined, that is, that $E(x_{jt} u_{it}) = 0$ for all $i$, $j$, and $t$, assigning a causal interpretation to IRFs and FEVDs requires that the exogenous variables be strictly exogenous, that is, that $E(x_{js} u_{it}) = 0$ for all $i$, $j$, $s$, and $t$.

IRFs describe how the innovations to one variable affect another variable after a given number of periods. For an example of how IRFs are interpreted, see Stock and Watson (2001). They use IRFs to investigate the effect of surprise shocks to the Federal Funds rate on inflation and unemployment. In another example, Christiano, Eichenbaum, and Evans (1999) use IRFs to investigate how shocks to monetary policy affect other macroeconomic variables.

Consider a VAR without exogenous variables:

$$y_t = v + A_1 y_{t-1} + \cdots + A_p y_{t-p} + u_t$$

The VAR represents the variables in $y_t$ as functions of its own lags and serially uncorrelated innovations $u_t$. All the information about contemporaneous correlations among the $K$ variables in $y_t$ is contained in $\Sigma$. In fact, as discussed in [TS] var svar, a VAR can be viewed as the reduced form of a dynamic simultaneous-equation model.

To see how the innovations affect the variables in $y_t$ after, say, $i$ periods, rewrite the model in its moving-average form

$$y_t = \mu + \sum_{i=0}^{\infty} \Phi_i u_{t-i}$$

where $\mu$ is the $K \times 1$ time-invariant mean of $y_t$, and

$$\Phi_i = \begin{cases} I_K & \text{if } i = 0 \\ \sum_{j=1}^{i} \Phi_{i-j} A_j & \text{if } i = 1, 2, \ldots \end{cases}$$
We can rewrite a VAR in the moving-average form only if it is stable. Essentially, a VAR is stable if the variables are covariance stationary and none of the autocorrelations are too high (the issue of stability is discussed in greater detail in [TS] `varstable`).

The $\Phi_i$ are the simple IRFs. The $j,k$ element of $\Phi_i$ gives the effect of a 1-time unit increase in the $k$th element of $u_t$ on the $j$th element of $y_t$ after $i$ periods, holding everything else constant. Unfortunately, these effects have no causal interpretation, which would require us to be able to answer the question, “How does an innovation to variable $k$, holding everything else constant, affect variable $j$ after $i$ periods?” Because the $u_t$ are contemporaneously correlated, we cannot assume that everything else is held constant. Contemporaneous correlation among the $u_t$ implies that a shock to one variable is likely to be accompanied by shocks to some of the other variables, so it does not make sense to shock one variable and hold everything else constant. For this reason, (2) cannot provide a causal interpretation.

This shortcoming may be overcome by rewriting (2) in terms of mutually uncorrelated innovations. Suppose that we had a matrix $P$, such that $\Sigma = PP'$. If we had such a $P$, then $P^{-1} \Sigma P'^{-1} = I_K$, and

$$E\{P^{-1}u_t(P^{-1}u_t)\}' = P^{-1}E\{(u_t'u_t)'P'^{-1}\} = P^{-1} \Sigma P'^{-1} = I_K$$

We can thus use $P^{-1}$ to orthogonalize the $u_t$ and rewrite (2) as

$$y_t = \mu + \sum_{i=0}^{\infty} \Phi_iPP^{-1}u_{t-i}$$

$$= \mu + \sum_{i=0}^{\infty} \Theta_iP^{-1}u_{t-i}$$

$$= \mu + \sum_{i=0}^{\infty} \Theta_iw_{t-i}$$

where $\Theta_i = \Phi_iP$ and $w_t = P^{-1}u_t$. If we had such a $P$, the $w_k$ would be mutually orthogonal, and no information would be lost in the holding-everything-else-constant assumption, implying that the $\Theta_i$ would have the causal interpretation that we seek.

Choosing a $P$ is similar to placing identification restrictions on a system of dynamic simultaneous equations. The simple IRFs do not identify the causal relationships that we wish to analyze. Thus we seek at least as many identification restrictions as necessary to identify the causal IRFs.

So, where do we get such a $P$? Sims (1980) popularized the method of choosing $P$ to be the Cholesky decomposition of $\hat{\Sigma}$. The IRFs based on this choice of $P$ are known as the orthogonalized IRFs. Choosing $P$ to be the Cholesky decomposition of $\hat{\Sigma}$ is equivalent to imposing a recursive structure for the corresponding dynamic structural equation model. The ordering of the recursive structure is the same as the ordering imposed in the Cholesky decomposition. Because this choice is arbitrary, some researchers will look at the OIRFs with different orderings assumed in the Cholesky decomposition. The order() option available with `irf create` facilitates this type of analysis.

The SVAR approach integrates the need to identify the causal IRFs into the model specification and estimation process. Sufficient identification restrictions can be obtained by placing either short-run or long-run restrictions on the model. The VAR in (1) can be rewritten as

$$y_t - v - A_1y_{t-1} - \cdots - A_p y_{t-p} = u_t$$

Similarly, a short-run SVAR model can be written as

$$A(y_t - v - A_1y_{t-1} - \cdots - A_p y_{t-p}) = Au_t = Be_t$$

(3)
where \(A\) and \(B\) are \(K \times K\) nonsingular matrices of parameters to be estimated, \(e_t\) is a \(K \times 1\) vector of disturbances with \(e_t \sim N(0, I_K)\), and \(E(e_t e'_s) = 0_k\) for all \(s \neq t\). Sufficient constraints must be placed on \(A\) and \(B\) so that \(P\) is identified. One way to see the connection is to draw out the implications of the latter equality in (3). From (3), it can be shown that

\[
\Sigma = A^{-1}B(A^{-1}B)' 
\]

As discussed in [TS] var svar, the estimates \(\hat{A}\) and \(\hat{B}\) are obtained by maximizing the concentrated log-likelihood function on the basis of the \(\hat{\Sigma}\) obtained from the underlying VAR. The short-run SVAR approach chooses \(P = \hat{A}^{-1}\hat{B}\) to identify the causal IRFs. The long-run SVAR approach works similarly, with \(P = \hat{C} = \hat{A}^{-1}\hat{B}\), where \(\hat{A}^{-1}\) is the matrix of estimated long-run or accumulated effects of the reduced-form VAR shocks.

There is one important difference between long-run and short-run SVAR models. As discussed by Amisano and Giannini (1997, chap. 6), in the short-run model the constraints are applied directly to the parameters in \(A\) and \(B\). Then \(A\) and \(B\) interact with the estimated parameters of the underlying VAR. In contrast, in a long-run model, the constraints are placed on functions of the estimated VAR parameters. Although estimation and inference of the parameters in \(C\) is straightforward, obtaining the asymptotic standard errors of the structural IRFs requires untenable assumptions. For this reason, irf create does not estimate the asymptotic standard errors of the structural IRFs generated by long-run SVAR models. However, bootstrap standard errors are still available.

**An introduction to dynamic-multiplier functions for VARs**

A dynamic-multiplier function measures the effect of a unit change in an exogenous variable on the endogenous variables over time. Per Lütkepohl (2005, chap. 10), if the VAR with exogenous variables is stable, it can be rewritten as

\[
y_t = \sum_{i=0}^{\infty} D_i x_{t-i} + \sum_{i=0}^{\infty} \Phi_i u_{t-i} 
\]

where the \(D_i\) are the dynamic-multiplier functions. (See Methods and formulas for details.) Some authors refer to the dynamic-multiplier functions as transfer functions because they specify how a unit change in an exogenous variable is “transferred” to the endogenous variables.

**Technical note**

irf create computes dynamic-multiplier functions only after var. After short-run SVAR models, the dynamic multipliers from the VAR are the same as those from the SVAR. The dynamic multipliers for long-run SVARs have not yet been worked out.

**An introduction to forecast-error variance decompositions for VARs**

Another measure of the effect of the innovations in variable \(k\) on variable \(j\) is the FEVD. This method, which is also known as innovation accounting, measures the fraction of the error in forecasting variable \(j\) after \(h\) periods that is attributable to the orthogonalized innovations in variable \(k\). Because deriving the FEVD requires orthogonalizing the \(u_t\) innovations, the FEVD is always predicated upon a choice of \(P\).
Lütkepohl (2005, sec. 2.2.2) shows that the \( h \)-step forecast error can be written as

\[
y_{t+h} - \hat{y}_t(h) = \sum_{i=0}^{h-1} \Phi_i u_{t+h-i}
\]

(4)

where \( y_{t+h} \) is the value observed at time \( t + h \) and \( \hat{y}_t(h) \) is the \( h \)-step-ahead predicted value for \( y_{t+h} \) that was made at time \( t \).

Because the \( u_t \) are contemporaneously correlated, their distinct contributions to the forecast error cannot be ascertained. However, if we choose a \( P \) such that \( \Sigma = PP' \), as above, we can orthogonalize the \( u_t \) into \( w_t = P^{-1} u_t \). We can then ascertain the relative contribution of the distinct elements of \( w_t \). Thus we can rewrite (4) as

\[
y_{t+h} - \hat{y}_t(h) = \sum_{i=0}^{h-1} \Phi_i PP^{-1} u_{t+h-i}
\]

\[
= \sum_{i=0}^{h-1} \Theta_i w_{t+h-i}
\]

Because the forecast errors can be written in terms of the orthogonalized errors, the forecast-error variance can be written in terms of the orthogonalized error variances. Forecast-error variance decompositions measure the fraction of the total forecast-error variance that is attributable to each orthogonalized shock.

**Technical note**

The details in this note are not critical to the discussion that follows. A forecast-error variance decomposition is derived for a given \( P \). Per Lütkepohl (2005, sec. 2.3.3), letting \( \theta_{mn,i} \) be the \( m, n \)th element of \( \Theta_i \), we can express the \( h \)-step forecast error of the \( j \)th component of \( y_t \) as

\[
y_{j,t+h} - \hat{y}_j(h) = \sum_{i=0}^{h-1} \theta_{j1,1} w_{1,t+h-i} + \cdots + \theta_{jK,i} w_{K,t+h-i}
\]

\[
= \sum_{k=1}^{K} \theta_{jk,0} w_{k,t+h} + \cdots + \theta_{jk,h-1} w_{k,t+1}
\]

The \( w_t \), which were constructed using \( P \), are mutually orthogonal with unit variance. This allows us to compute easily the mean squared error (MSE) of the forecast of variable \( j \) at horizon \( h \) in terms of the contributions of the components of \( w_t \). Specifically,

\[
E[\{y_{j,t+h} - y_{j,t}(h)\}^2] = \sum_{k=1}^{K} (\theta_{jk,0}^2 + \cdots + \theta_{jk,h-1}^2)
\]

The \( k \)th term in the sum above is interpreted as the contribution of the orthogonalized innovations in variable \( k \) to the \( h \)-step forecast error of variable \( j \). Note that the \( k \)th element in the sum above can be rewritten as

\[
(\theta_{jk,0}^2 + \cdots + \theta_{jk,h-1}^2) = \sum_{i=0}^{h-1} (e'_j \Theta_k e_k)^2
\]
where \( e_i \) is the \( i \)th column of \( I_k \). Normalizing by the forecast error for variable \( j \) at horizon \( h \) yields

\[
\omega_{jk,h} = \frac{\sum_{i=0}^{h-1} (e_j^\prime \Theta_k e_k)^2}{\text{MSE}\{y_{j,t}(h)\}}
\]

where \( \text{MSE}\{y_{j,t}(h)\} = \sum_{i=0}^{h-1} \sum_{k=1}^{K} \theta_{jk,i}^2 \).

Because the FEVD depends on the choice of \( P \), there are different forecast-error variance decompositions associated with each distinct \( P \). \texttt{irf create} can estimate the FEVD for a VAR or an SVAR. For a VAR, \( P \) is the Cholesky decomposition of \( \hat{\Sigma} \). For an SVAR, \( P \) is the estimated structural decomposition, \( P = \hat{A}^{-1} \hat{B} \) for short-run models and \( P = \hat{C} \) for long-run SVAR models. Due to the same complications that arose with the structural impulse–response functions, the asymptotic standard errors of the structural FEVD are not available after long-run SVAR models, but bootstrap standard errors are still available.

**IRF results for VECMs**

### An introduction to impulse–response functions for VECMs

As discussed in [TS vec intro], the VECM is a reparameterization of the VAR that is especially useful for fitting VARS with cointegrating variables. This implies that the estimated parameters for the corresponding VAR model can be backed out from the estimated parameters of the VECM model. This relationship means we can use the VAR form of the cointegrating VECM to discuss the IRFs for VECMs.

Consider a cointegrating VAR with one lag with no constant or trend,

\[
y_t = A y_{t-1} + u_t
\]

(5)

where \( y_t \) is a \( K \times 1 \) vector of endogenous, first-difference stationary variables among which there are \( 1 \leq r < K \) cointegration equations; \( A \) is \( K \times K \) matrix of parameters; and \( u_t \) is a \( K \times 1 \) vector of i.i.d. disturbances.

We developed intuition for the IRFs from a stationary VAR by rewriting the VAR as an infinite-order vector moving-average (VMA) process. While the Granger representation theorem establishes the existence of a VMA formulation of this model, because the cointegrating VAR is not stable, the inversion is not nearly so intuitive. (See Johansen [1995, chapters 3 and 4] for more details.) For this reason, we use (5) to develop intuition for the IRFs from a cointegrating VAR.

Suppose that \( K \) is 3, that \( u_1 = (1, 0, 0) \), and that we want to analyze the time paths of the variables in \( y \) conditional on the initial values \( y_0 = 0 \), \( A \), and the condition that there are no more shocks to the system, that is, \( 0 = u_2 = u_3 = \cdots \). These assumptions and (5) imply that

\[
y_1 = u_1
\]

\[
y_2 = A y_1 = A u_1
\]

\[
y_3 = A y_2 = A^2 u_1
\]
and so on. The \( i \)th-row element of the first column of \( \mathbf{A}^s \) contains the effect of the unit shock to the first variable after \( s \) periods. The first column of \( \mathbf{A}^s \) contains the IRF of a unit impulse to the first variable after \( s \) periods. We could deduce the IRFs of a unit impulse to any of the other variables by administering the unit shock to one of them instead of to the first variable. Thus we can see that the \((i, j)\)th element of \( \mathbf{A}^s \) contains the unit IRF from variable \( j \) to variable \( i \) after \( s \) periods. By starting with orthogonalized shocks of the form \( \mathbf{P}^{-1}\mathbf{u}_t \), we can use the same logic to derive the OIRFs to be \( \mathbf{A}^s \mathbf{P} \).

For the stationary VAR, stability implies that all the eigenvalues of \( \mathbf{A} \) have moduli strictly less than one, which in turn implies that all the elements of \( \mathbf{A}^s \to 0 \) as \( s \to \infty \). This implies that all the IRFs from a stationary VAR taper off to zero as \( s \to \infty \). In contrast, in a cointegrating VAR, some of the eigenvalues of \( \mathbf{A} \) are 1, while the remaining eigenvalues have moduli strictly less than 1. This implies that in cointegrating VARS some of the elements of \( \mathbf{A}^s \) are not going to zero as \( s \to \infty \), which in turn implies that some of the IRFs and OIRFs are not going to zero as \( s \to \infty \). The fact that the IRFs and OIRFs taper off to zero for stationary VARS but not for cointegrating VARS is one of the key differences between the two models.

When the IRF or OIRF from the innovation in one variable to another tapers off to zero as time goes on, the innovation to the first variable is said to have a transitory effect on the second variable. When the IRF or OIRF does not go to zero, the effect is said to be permanent.

Note that, because some of the IRFs and OIRFs do not taper off to zero, some of the cumulative IRFs and OIRFs diverge over time.

**An introduction to forecast-error variance decompositions for VECMs**

The results from *An introduction to impulse–response functions for VECMs* can be used to show that the interpretation of FEVDs for a finite number of steps in cointegrating VECMs is essentially the same as in the stationary case. Because the MSE of the forecast is diverging, this interpretation is valid only for a finite number of steps. (See [TS] vec intro and [TS] fcast compute for more information on this point.)

**IRF results for ARIMA and ARFIMA**

A covariance-stationary additive ARMA\((p, q)\) model can be written as

\[
\rho(L^p)(y_t - x_t\beta) = \theta(L^q)\epsilon_t
\]

where

\[
\rho(L^p) = 1 - \rho_1 L - \rho_2 L^2 - \cdots - \rho_p L^p
\]
\[
\theta(L^q) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q
\]

and \( L^j y_t = y_{t-j} \).

We can rewrite the above model as an infinite-order moving-average process

\[
y_t = x_t\beta + \psi(L)\epsilon_t
\]

where

\[
\psi(L) = \frac{\theta(L)}{\rho(L)} = 1 + \psi_1 L + \psi_2 L^2 + \cdots
\] (6)
This representation shows the impact of the past innovations on the current $y_t$. The $i$th coefficient describes the response of $y_t$ to a one-time impulse in $\epsilon_{t-i}$, holding everything else constant. The $\psi_i$ coefficients are collectively referred to as the impulse–response function of the ARMA model. For a covariance-stationary series, the $\psi_i$ coefficients decay exponentially.

A covariance-stationary multiplicative seasonal ARMA model, often abbreviated SARMA, of order $(p,q) \times (P,Q)$, can be written as

$$\rho(L^p)\rho_s(L^P)(y_t - \mathbf{x}_t\beta) = \theta(L^q)\theta_s(L^Q)\epsilon_t$$

where

$$\rho_s(L^P) = (1 - \rho_{s,1}L^s - \cdots - \rho_{s,p}L^{Ps})$$

$$\theta_s(L^Q) = (1 + \theta_{s,1}L^s + \cdots + \theta_{s,Q}L^{Qs})$$

with $\rho(L^p)$ and $\theta(L^q)$ defined as above.

We can express this model as an additive ARMA model by multiplying the terms and imposing nonlinear constraints on multiplied coefficients. For example, consider the SARMA model given by

$$(1 - \rho_1 L)(1 - \rho_{4,1} L^4)y_t = \epsilon_t$$

Expanding the above equation and solving for $y_t$ yields

$$y_t = \rho_1 y_{t-1} + \rho_{4,1} y_{t-4} - \rho_1 \rho_{4,1} y_{t-5} + \epsilon_t$$

or, in ARMA terms,

$$y_t = \rho_1 y_{t-1} + \rho_{4,1} y_{t-4} + \rho_5 y_{t-5} + \epsilon_t$$

subject to the constraint $\rho_5 = -\rho_1 \rho_{4,1}$.

Once we have obtained an ARMA representation of a SARMA process, we obtain the IRFs from (6).

An ARFIMA($p$, $d$, $q$) model can be written as

$$\rho(L^p)(1 - L)^d(y_t - \mathbf{x}_t\beta) = \theta(L^q)\epsilon_t$$

with $(1 - L)^d$ denoting a fractional integration operation.

Solving for $y_t$, we obtain

$$y_t = \mathbf{x}_t\beta + (1 - L)^{-d}\psi(L)\epsilon_t$$

This makes it clear that the impulse–response function for an ARFIMA model corresponds to a fractionally differenced impulse–response function for an ARIMA model. Because of the fractional differentiation, the $\psi_i$ coefficients decay very slowly; see Remarks and examples in [TS] arfima.

### Methods and formulas

Methods and formulas are presented under the following headings:

- Impulse–response function formulas for VARs
- Dynamic-multiplier function formulas for VARs
- Forecast-error variance decomposition formulas for VARs
- Impulse–response function formulas for VECMs
- Algorithms for bootstrapping the VAR IRF and FEVD standard errors
- Impulse–response function formulas for ARIMA and ARFIMA
Impulse–response function formulas for VARs

The previous discussion implies that there are three different choices of $P$ that can be used to obtain distinct $\Theta_i$. $P$ is the Cholesky decomposition of $\Sigma$ for the OIRFs. For the structural IRFs, $P = A^{-1}B$ for short-run models, and $P = C$ for long-run models. We will distinguish between the three by defining $\Theta^o_i$ to be the OIRFs, $\Theta^sr_i$ to be the short-run structural IRFs, and $\Theta^lr_i$ to be the long-run structural IRFs.

We also define $\hat{P}_c$ to be the Cholesky decomposition of $\hat{\Sigma}$, $\hat{P}^sr = \hat{A}^{-1}\hat{B}$ to be the short-run structural decomposition, and $\hat{P}^lr = \hat{C}$ to be the long-run structural decomposition.

Given estimates of the $\hat{A}_i$ and $\hat{\Sigma}$ from var or svar, the estimates of the simple IRFs and the OIRFs are, respectively,

$$\hat{\Phi}_i = \sum_{j=1}^{i} \hat{\Phi}_{i-j} \hat{A}_j$$

and

$$\hat{\Theta}^o_i = \hat{\Phi}_i \hat{P}_c$$

where $\hat{A}_j = 0_K$ for $j > p$.

Given the estimates $\hat{A}$ and $\hat{B}$, or $\hat{C}$, from svar, the estimates of the structural IRFs are either

$$\hat{\Theta}^sr_i = \hat{\Phi}_i \hat{P}^sr$$

or

$$\hat{\Theta}^lr_i = \hat{\Phi}_i \hat{P}^lr$$

The estimated structural IRFs stored in an IRF file with the variable name sirf may be from either a short-run model or a long-run SVAR model, depending on the estimation results used to create the IRFs. As discussed in [TS] irf describe, you can easily determine whether the structural IRFs were generated from a short-run or a long-run SVAR model using irf describe.

Following Lütkepohl (2005, sec. 3.7), estimates of the cumulative IRFs and the cumulative orthogonalized impulse–response functions (COIRFs) at period $n$ are, respectively,

$$\hat{\Psi}_n = \sum_{i=0}^{n} \hat{\Phi}_i$$

and

$$\hat{\Xi}_n = \sum_{i=0}^{n} \hat{\Theta}_i$$

The asymptotic standard errors of the different impulse–response functions are obtained by applications of the delta method. See Lütkepohl (2005, sec. 3.7) and Amisano and Giannini (1997, chap. 4) for the derivations. See Serfling (1980, sec. 3.3) for a discussion of the delta method. In presenting the variance–covariance matrix estimators, we make extensive use of the vec() operator, where vec$(X)$ is the vector obtained by stacking the columns of $X$.

Lütkepohl (2005, sec. 3.7) derives the asymptotic VCEs of vec$(\hat{\Phi}_i)$, vec$(\hat{\Theta}^o_i)$, vec$(\hat{\Psi}_n)$, and vec$(\hat{\Xi}_n)$. Because vec$(\hat{\Phi}_i)$ is $K^2 \times 1$, the asymptotic VCE of vec$(\hat{\Phi}_i)$ is $K^2 \times K^2$, and it is given by

$$G_i \hat{\Sigma}_a G_i'$$
where
\[
G_i = \sum_{m=0}^{i-1} J(M'(i-1-m) \otimes \Phi_m)
\]
\(G_i\) is \(K^2 \times K^2 p\)

\[
J = (I_K, 0_K, \ldots, 0_K)
\]
\(J\) is \(K \times K p\)

\[
\hat{M} = \begin{bmatrix}
\hat{A}_1 & \hat{A}_2 & \ldots & \hat{A}_{p-1} & \hat{A}_p \\
I_K & 0_K & \ldots & 0_K & 0_K \\
0_K & I_K & 0_K & 0_K & \ldots & \ldots & \ldots & \ldots & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_K & 0_K & \ldots & I_K & 0_K & \ldots & \ldots & \ldots & \ldots & \ldots
\end{bmatrix}
\]
\(\hat{M}\) is \(K p \times K p\)

The \(\hat{A}_i\) are the estimates of the coefficients on the lagged variables in the VAR, and \(\hat{\Sigma}_\alpha\) is the VCE matrix of \(\hat{\alpha} = \text{vec}(\hat{A}_1, \ldots, \hat{A}_p)\). \(\hat{\Sigma}_\alpha\) is a \(K^2 p \times K^2 p\) matrix whose elements come from the VCE of the VAR coefficient estimator. As such, this VCE is the VCE of the constrained estimator if there are any constraints placed on the VAR coefficients.

The \(K^2 \times K^2\) asymptotic VCE matrix for \(\text{vec}(\hat{\Psi}_n)\) after \(n\) periods is given by

\[
F_n \hat{\Sigma}_\alpha F_n'
\]

where

\[
F_n = \sum_{i=1}^{n} G_i
\]

The \(K^2 \times K^2\) asymptotic VCE matrix of the vectorized, orthogonalized, IRFs at horizon \(i\), \(\text{vec}(\Theta_i^o)\), is

\[
C_i \hat{\Sigma}_\alpha C_i' + C_i \hat{\Sigma}_\sigma C_i'
\]
where

\[
C_0 = 0
\]
\[
C_i = (\hat{P}_c \otimes I_K)G_i, \quad i = 1, 2, \ldots
\]
\[
\bar{C}_i = (I_K \otimes \Phi_i)H, \quad i = 0, 1, \ldots
\]
\[
H = L'_K \left\{ L_K N_K (\hat{P}_c \otimes I_K) L'_K \right\}^{-1}
\]

\[L_K\] solves \(\text{vech}(F) = L_K \text{vec}(F)\) for \(F \times K \times K \) and symmetric

\[K_K\] solves \(K_K \text{vec}(G) = \text{vec}(G')\) for any \(K \times K\) matrix \(G\)

\[
N_K = \frac{1}{2} (I_{K^2} + K_K)
\]
\[
\hat{\Sigma}_\sigma = 2D_K^+(\hat{\Sigma} \otimes \hat{\Sigma})D_K^+
\]
\[
D_K^+ = (D_K' D_K)^{-1} D_K'
\]

\[D_K\] solves \(D_K \text{vech}(F) = \text{vec}(F)\) for \(F \times K \times K\) and symmetric

\[
\text{vech}(X) = \begin{bmatrix}
x_{11} \\
x_{21} \\
\vdots \\
x_{K1} \\
x_{12} \\
\vdots \\
x_{K2} \\
\vdots \\
x_{KK}
\end{bmatrix}
\]

for \(X \times K \times K\)

\[
\text{vech}(X) \text{ is } K \frac{(K+1)}{2} \times 1
\]

Note that \(\hat{\Sigma}_\sigma\) is the VCE of \(\text{vech}(\hat{\Sigma})\). More details about \(L_K, K_K, D_K\) and \(\text{vech}\) are available in Lütkepohl (2005, sec. A.12). Finally, as Lütkepohl (2005, 113–114) discusses, \(D_K^+\) is the Moore–Penrose inverse of \(D_K\).

As discussed in Amisano and Giannini (1997, chap. 6), the asymptotic standard errors of the structural IRFs are available for short-run SVAR models but not for long-run SVAR models. Following Amisano and Giannini (1997, chap. 5), the asymptotic \(K^2 \times K^2\) VCE of the short-run structural IRFs after \(i\) periods, when a maximum of \(h\) periods are estimated, is the \(i, i\) block of

\[
\hat{\Sigma}(h)_{ij} = \tilde{G}_i \hat{\Sigma}_\alpha \tilde{G}_j' + \left\{ I_K \otimes (J\hat{M}' J') \right\} \Sigma(0) \left\{ I_K \otimes (J\hat{M}' J') \right\}'
\]
where

\[ \tilde{G}_0 = 0_K \]
\[ \tilde{G}_i = \sum_{k=0}^{i-1} \left\{ \hat{P}_s'J(\widetilde{M}')^{i-1-k} \otimes (J\widetilde{M}_kJ') \right\} \]

\[ \tilde{\Sigma}(0) = Q_2\tilde{\Sigma}_W Q_2' \]
\[ \tilde{\Sigma}_W = Q_1\tilde{\Sigma}_{AB} Q_1' \]
\[ Q_2 = \hat{P}_s' \otimes \hat{P}_s \]
\[ Q_1 = \left\{ (I_K \otimes \hat{B}^{-1}), (-\hat{P}_s'^{-1} \otimes \hat{B}^{-1}) \right\} \]

and \( \tilde{\Sigma}_{AB} \) is the \( 2K^2 \times 2K^2 \) VCE of the estimator of vec(\( A, B \)).

### Dynamic-multiplier function formulas for VARs

This section provides the details of how `irf create` estimates the dynamic-multiplier functions and their asymptotic standard errors.

A \( p \)th order vector autoregressive model (VAR) with exogenous variables may be written as

\[ y_t = v + A_1 y_{t-1} + \cdots + A_p y_{t-p} + B_0 x_t + B_1 x_{t-1} + \cdots + B_s x_{t-s} + u_t \]

where all the notation is the same as above except that the \( s K \times R \) matrices \( B_1, B_2, \ldots, B_s \) are explicitly included and \( s \) is the number of lags of the \( R \) exogenous variables in the model.

Lütkepohl (2005) shows that the dynamic-multipliers \( D_i \) are consistently estimated by

\[ \hat{D}_i = J_x \tilde{A}_x \hat{B}_x \quad i \in \{0, 1, \ldots\} \]

where

\[ J_x = (I_K, 0_K, \ldots, 0_K) \]
\[ \tilde{A}_x = \begin{bmatrix} \tilde{M} & \hat{B} \\ \hat{\bar{0}} & \hat{I} \end{bmatrix} \]
\[ \hat{B} = \begin{bmatrix} \hat{B}_1 & \hat{B}_2 & \cdots & \hat{B}_s \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \]
\[ \tilde{I} = \begin{bmatrix} 0_R & 0_R & \cdots & 0_R & 0_R \\ I_R & 0_R & \cdots & 0_R & 0_R \\ 0_R & I_R & \cdots & 0_R & 0_R \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0_R & 0_R & \cdots & I_R & 0_R \end{bmatrix} \]
\[ \hat{B}_x' = \begin{bmatrix} \hat{B}' & \hat{\bar{0}}' \end{bmatrix} \]
\[ \hat{\bar{0}}' = \begin{bmatrix} \hat{B}_0' & \hat{0}' & \cdots & \hat{0}' \end{bmatrix} \]
\[ \hat{\bar{I}}' = \begin{bmatrix} I_R & 0_R & \cdots & 0_R \end{bmatrix} \]

and \( \hat{\bar{0}} \) is a \( K \times R \) matrix of 0s and \( \hat{\bar{I}} \) is a \( Rs \times Kp \) matrix of 0s.
Consistent estimators of the cumulative dynamic-multiplier functions are given by

\[ \tilde{D}_i = \sum_{j=0}^{i} \hat{D}_j \]

Letting \( \beta_x = \text{vec}(A_1A_2 \cdots A_pB_1B_2 \cdots B_sB_0) \) and letting \( \Sigma \hat{\beta}_x \) be the asymptotic variance-covariance estimator (VCE) of \( \hat{\beta}_x \), Lütkepohl shows that an asymptotic VCE of \( \hat{D}_i \) is

\[ \tilde{G}_i \Sigma \hat{\beta}_x \tilde{G}_i' \]

where

\[ \tilde{G}_i = \left( \sum_{j=0}^{i-1} B'_x \tilde{A}_x^{-1-j} \otimes J_x \tilde{A}_x J'_x, I_R \otimes J_x \tilde{A}_x J_x \right) \]

Similarly, an asymptotic VCE of \( \hat{D}_i \) is

\[ \left( \sum_{j=0}^{i} \tilde{G}_j \right) \Sigma \hat{\beta}_x \left( \sum_{j=0}^{i} \tilde{G}_j' \right) \]

**Forecast-error variance decomposition formulas for VARs**

This section provides details of how \texttt{irf create} estimates the Cholesky FEVD, the structural FEVD, and their standard errors. Beginning with the Cholesky-based forecast-error decompositions, the fraction of the \( h \)-step-ahead forecast-error variance of variable \( j \) that is attributable to the Cholesky orthogonalized innovations in variable \( k \) can be estimated as

\[ \hat{\omega}_{jk,h} = \frac{\sum_{i=0}^{h-1} (e'_j \hat{\Theta}_i e_k)^2}{\text{MSE}_j(h)} \]

where \( \text{MSE}_j(h) \) is the \( j \)th diagonal element of

\[ \sum_{i=0}^{h-1} \hat{\Phi}_i \Sigma \hat{\Phi}_i' \]

(See Lütkepohl [2005, 109] for a discussion of this result.) \( \hat{\omega}_{jk,h} \) and \( \text{MSE}_j(h) \) are scalars. The square of the standard error of \( \hat{\omega}_{jk,h} \) is

\[ d_{jk,h} \Sigma_{\alpha} d_{jk,h} + \bar{d}_{jk,h} \Sigma_{\alpha} \bar{d}_{jk,h} \]

where

\[ d_{jk,h} = \frac{2}{\text{MSE}_j(h)^2} \sum_{i=0}^{h-1} \left\{ \text{MSE}_j(h)(e'_j \hat{\Phi}_i \hat{P}_c e_k)(e'_k \hat{P}'_c \otimes e'_j)G_i \right. \]

\[ \left. -(e'_j \hat{\Phi}_i \hat{P}_c e_k)^2 \sum_{m=0}^{h-1} (e'_j \hat{\Phi}_m \hat{\Theta} \otimes e'_j)G_m \right\} \]

\[ \text{d}_{jk,h} \text{ is } 1 \times K^2 p \]

\[ \bar{d}_{jk,h} = \sum_{i=0}^{h-1} \left\{ \text{MSE}_j(h)(e'_j \hat{\Phi}_i \hat{P}_c e_k)(e'_k \otimes e'_j \hat{\Phi}_i)H \right. \]

\[ \left. -(e'_j \hat{\Phi}_i \hat{P}_c e_k)^2 \sum_{m=0}^{h-1} (e'_j \hat{\Phi}_m \otimes e'_j \hat{\Phi}_m)D_K \right\} \frac{1}{\text{MSE}_j(h)^2} \]

\[ \text{\bar{d}}_{jk,h} \text{ is } 1 \times K \frac{(K+1)}{2} \]

\[ G_0 = 0 \]

\[ G_0 \text{ is } K^2 \times K^2 p \]

and \( D_K \) is the \( K^2 \times K \{ (K + 1)/2 \} \) duplication matrix defined previously.
For the structural forecast-error decompositions, we follow Amisano and Giannini (1997, sec. 5.2). They define the matrix of structural forecast-error decompositions at horizon $s$, when a maximum of $h$ periods are estimated, as

$$
\hat{W}_s = \hat{F}_s^{-1} \hat{M}_s \quad \text{for } s = 1, \ldots, h + 1
$$

$$
\hat{F}_s = \left( \sum_{i=0}^{s-1} \hat{\Theta}_i \hat{\Theta}_i' \right) \odot I_K
$$

$$
\hat{M}_s = \sum_{i=0}^{s-1} \hat{\Theta}_i \odot \hat{\Theta}_i
$$

where $\odot$ is the Hadamard, or element-by-element, product.

The $K^2 \times K^2$ asymptotic VCE of vec$(\hat{W}_s)$ is given by

$$
\tilde{Z}_s \Sigma(h) \tilde{Z}_s'
$$

where $\Sigma(h)$ is as derived previously, and

$$
\tilde{Z}_s = \left\{ \frac{\partial \text{vec}(\hat{W}_s)}{\partial \text{vec}(\hat{\Theta}_0^s)}, \frac{\partial \text{vec}(\hat{W}_s)}{\partial \text{vec}(\hat{\Theta}_1^s)}, \ldots, \frac{\partial \text{vec}(\hat{W}_s)}{\partial \text{vec}(\hat{\Theta}_h^s)} \right\}
$$

$$
\frac{\partial \text{vec}(\hat{W}_s)}{\partial \text{vec}(\hat{\Theta}_j^s)} = 2 \left\{ (I_K \otimes \hat{F}_s^{-1}) \hat{D}(\hat{\Theta}_j^s) - (\hat{W}_s' \otimes \hat{F}_s^{-1}) \hat{D}(I_K) N_K (\hat{\Theta}_j^s \otimes I_K) \right\}
$$

If $X$ is an $n \times n$ matrix, then $\hat{D}(X)$ is the $n^2 \times n^2$ matrix with vec$(X)$ on the diagonal and zeros in all the off-diagonal elements, and $N_K$ is as defined previously.

**Impulse–response function formulas for VECMs**

We begin by providing the formulas for backing out the estimates of the $A_i$ from the $\Gamma_i$ estimated by vec. As discussed in [TS] vec intro, the VAR in (1) can be rewritten as a VECM:

$$
\Delta y_t = v + \Pi y_{t-1} + \Gamma_1 \Delta y_{t-1} + \Gamma_{p-1} \Delta y_{p-2} + \epsilon_t
$$

vec estimates $\Pi$ and the $\Gamma_i$. Johansen (1995, 25) notes that

$$
\Pi = \sum_{i=1}^{p} A_i - I_K
$$

where $I_K$ is the $K$-dimensional identity matrix, and

$$
\Gamma_i = - \sum_{j=i+1}^{p} A_j
$$
Defining
\[ \Gamma = I_K - \sum_{i=1}^{p-1} \Gamma_i \]
and using (6) and (7) allow us to solve for the \( A_i \) as
\[ A_1 = \Pi + \Gamma_1 + I_K \]
\[ A_i = \Gamma_i - \Gamma_{i-1} \quad \text{for} \quad i = \{2, \ldots, p - 1\} \]
and
\[ A_p = -\Gamma_{p-1} \]

Using these formulas, we can back out estimates of \( A_i \) from the estimates of the \( \Gamma_i \) and \( \Pi \) produced by vec. Then we simply use the formulas for the IRFs and OIRFs presented in Impulse–response function formulas for VARs.

The running sums of the IRFs and OIRFs over the steps within each impulse–response pair are the cumulative IRFs and OIRFs.

### Algorithms for bootstrapping the VAR IRF and FEVD standard errors

`irf create` offers two bootstrap algorithms for estimating the standard errors of the various IRFs and FEVDs. Both `var` and `svar` contain estimators for the coefficients in a VAR that are conditional on the first \( p \) observations. The two bootstrap algorithms are also conditional on the first \( p \) observations.

Specifying the `bs` option calculates the standard errors by bootstrapping the residuals. For a bootstrap with \( R \) repetitions, this method uses the following algorithm:

1. Fit the model and save the estimated parameters.
2. Use the estimated coefficients to calculate the residuals.
3. Repeat steps 3a to 3c \( R \) times.
   3a. Draw a simple random sample of size \( T \) with replacement from the residuals. The random samples are drawn over the \( K \times 1 \) vectors of residuals. When the \( t \)th vector is drawn, all \( K \) residuals are selected. This preserves the contemporaneous correlations among the residuals.
   3b. Use the \( p \) initial observations, the sampled residuals, and the estimated coefficients to construct a new sample dataset.
   3c. Fit the model and calculate the different IRFs and FEVDs.
   3d. Save these estimates as observation \( r \) in the bootstrapped dataset.
4. For each IRF and FEVD, the estimated standard deviation from the \( R \) bootstrapped estimates is the estimated standard error of that impulse–response function or forecast-error variance decomposition.

Specifying the `bsp` option estimates the standard errors by a multivariate normal parametric bootstrap. The algorithm for the multivariate normal parametric bootstrap is identical to the one above, with the exception that 3a is replaced by 3a(bsp):

3a(bsp). Draw \( T \) pseudovariates from a multivariate normal distribution with covariance matrix \( \hat{\Sigma} \).
Impulse–response function formulas for ARIMA and ARFIMA

The previous discussion showed that a SARMA process can be rewritten as an ARMA process and that for an ARMA process, we can express $\psi(L)$ in terms of $\theta(L)$ and $\rho(L)$,

$$\psi(L) = \frac{\theta(L)}{\rho(L)}$$

Expanding the above, we obtain

$$\psi_0 + \psi_1 L + \psi_2 L^2 + \cdots = \frac{1 + \theta_1 L + \theta_2 L^2 + \cdots}{1 - \rho_1 L - \rho_2 L^2 - \cdots}$$

Given the estimate of the autoregressive terms $\hat{\rho}$ and the moving-average terms $\hat{\theta}$, the IRF is obtained by solving the above equation for the $\psi$ weights. The $\psi_i$ are calculated using the recursion

$$\hat{\psi}_i = \hat{\theta}_i + \sum_{j=1}^{p} \hat{\phi}_j \hat{\psi}_{i-j}$$

with $\psi_0 = 1$ and $\theta_i = 0$ for $i > \max(p, q + 1)$.

The asymptotic standard errors for the IRF for ARMA are calculated using the delta method; see Serfling (1980, sec. 3.3) for a discussion of the delta method. Let $\hat{\Sigma}$ be the estimate of the variance–covariance matrix for $\hat{\rho}$ and $\hat{\theta}$, and let $\hat{\Psi}$ be a matrix of derivatives of $\psi_i$ with respect to $\hat{\rho}$ and $\hat{\theta}$. Then the standard errors for $\hat{\psi}_i$ are calculated as

$$\hat{\Psi}_i \hat{\Sigma} \hat{\Psi}_i'$$

The IRF for the ARFIMA($p, d, q$) model is obtained by applying the filter $(1 - L)^{-d}$ to $\psi(L)$. The filter is given by Hassler and Kokoszka (2010) as

$$(1 - L)^{-d} = \sum_{i=0}^{\infty} b_i L^i$$

with $b_0 = 1$ and subsequent $b_i$ calculated by the recursion

$$\hat{b}_i = \frac{\hat{d} + i - 1}{i} \hat{b}_{i-1}$$

The resulting IRF is then given by

$$\hat{\phi}_i = \sum_{j=0}^{i} \hat{\psi}_j \hat{b}_{i-j}$$

The asymptotic standard errors for the IRF for ARFIMA are calculated using the delta method. Let $\hat{\Sigma}$ be the estimate of the variance–covariance matrix for $\hat{\rho}$, $\hat{\theta}$, and $\hat{d}$, and let $\hat{\Phi}$ be a matrix of derivatives of $\phi_i$ with respect to $\hat{\rho}$, $\hat{\theta}$, and $\hat{d}$. Then the standard errors for $\hat{\phi}_i$ are calculated as

$$\hat{\Phi}_i \hat{\Sigma} \hat{\Phi}_i'$$
References


Also see

[TS] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] `var intro` — Introduction to vector autoregressive models

[TS] `vec intro` — Introduction to vector error-correction models
**Description**

`irf ctable` makes a table or a combined table of IRF results. A table is made for specified combinations of named IRF results, impulse variables, response variables, and statistics. `irf ctable` combines these tables into one table, unless separate tables are requested.

`irf ctable` operates on the active IRF file; see [TS] `irf set`.

**Quick start**

Combine tables of an orthogonalized IRF `myirf` and cumulative IRF `mycirf` for dependent variables `y1` and `y2`

```
irf ctable (myirf y1 y2 oirf) (mycirf y1 y2 cirf)
```

As above, but suppress confidence intervals and add a title

```
irf ctable (myirf y1 y2 oirf) (mycirf y1 y2 cirf), noci ///
        title("My Title")
```

Note: `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see [TS] `var`, [TS] `var svar`, [TS] `vec`, [TS] `arima`, [TS] `arfima`, [DSGE] `dsge`, or [DSGE] `dsgenl`.

**Menu**

Statistics > Multivariate time series > IRF and FEVD analysis > Combined tables
Syntax

```
irf ctable (spec1) [ (spec2) ... [ (spec_N) ] ] [, options]
```

where (spec_k) is

```
(irfname impulsevar responsevar stat [ , spec_options ])
```

`irfname` is the name of a set of IRF results in the active IRF file. `impulsevar` should be specified as an endogenous variable for all statistics except `dm` and `cdm`; for those, specify as an exogenous variable. `responsevar` is an endogenous variable name. `stat` is one or more statistics from the list below:

<table>
<thead>
<tr>
<th>stat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irf</td>
<td>impulse–response function</td>
</tr>
<tr>
<td>oirf</td>
<td>orthogonalized impulse–response function</td>
</tr>
<tr>
<td>dm</td>
<td>dynamic-multiplier function</td>
</tr>
<tr>
<td>cirf</td>
<td>cumulative impulse–response function</td>
</tr>
<tr>
<td>coirf</td>
<td>cumulative orthogonalized impulse–response function</td>
</tr>
<tr>
<td>cdm</td>
<td>cumulative dynamic-multiplier function</td>
</tr>
<tr>
<td>fevd</td>
<td>Cholesky forecast-error variance decomposition</td>
</tr>
<tr>
<td>sirf</td>
<td>structural impulse–response function</td>
</tr>
<tr>
<td>sfevd</td>
<td>structural forecast-error variance decomposition</td>
</tr>
</tbody>
</table>

`options` Description

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set(filename)</td>
<td>make filename active</td>
</tr>
<tr>
<td>noci</td>
<td>do not report confidence intervals</td>
</tr>
<tr>
<td>stderror</td>
<td>include standard errors for each statistic</td>
</tr>
<tr>
<td>individual</td>
<td>make an individual table for each combination</td>
</tr>
<tr>
<td>title(&quot;text&quot;)</td>
<td>use text as overall table title</td>
</tr>
<tr>
<td>step(#)</td>
<td>set common maximum step</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
</tbody>
</table>

`spec_options` Description

<table>
<thead>
<tr>
<th>spec_options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noci</td>
<td>do not report confidence intervals</td>
</tr>
<tr>
<td>stderror</td>
<td>include standard errors for each statistic</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>ititle(&quot;text&quot;)</td>
<td>use text as individual subtitle for specific table</td>
</tr>
</tbody>
</table>

`spec_options` may be specified within a table specification, globally, or both. When specified in a table specification, the `spec_options` affect only the specification in which they are used. When supplied globally, the `spec_options` affect all table specifications. When specified in both places, options for the table specification take precedence. `ititle()` does not appear in the dialog box.
Options

`set(filename)` specifies the file to be made active; see [TS] irf set. If `set()` is not specified, the active file is used.

`noci` suppresses reporting of the confidence intervals for each statistic. `noci` is assumed when the model was fit by vec because no confidence intervals were estimated.

`stderr` specifies that standard errors for each statistic also be included in the table.

`individual` places each block, or `(spec_k)`, in its own table. By default, `irf ctable` combines all the blocks into one table.

`title("text")` specifies a title for the table or the set of tables.

`step(#)` specifies the maximum number of steps to use for all tables. By default, each table is constructed using all steps available.

`level(#)` specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is `level(95)` or as set by `set level`; see [U] 20.8 Specifying the width of confidence intervals.

The following option is available with `irf ctable` but is not shown in the dialog box:

`ititle("text")` specifies an individual subtitle for a specific table. `ititle()` may be specified only when the `individual` option is also specified.

Remarks and examples

If you have not read [TS] irf, please do so.

Also see [TS] irf table for a slightly easier to use, but less powerful, table command.

`irf ctable` creates a series of tables from IRF results. The information enclosed within each set of parentheses,

```
(irfname impulsevar responsevar stat [, spec_options])
```

forms a request for a specific table.

The first part—`irfname impulsevar responsevar`—identifies a set of IRF estimates or a set of variance decomposition estimates. The next part—`stat`—specifies which statistics are to be included in the table. The last part—`spec_options`—includes the `noci`, `level()`, and `stderr` options, and places (or suppresses) additional columns in the table.

Each specific table displays the requested statistics corresponding to the specified combination of `irfname`, `impulsevar`, and `responsevar` over the step horizon. By default, all the individual tables are combined into one table. Also by default, all the steps, or periods, available are included in the table. You can use the `step()` option to impose a common maximum for all tables.

Example 1

In example 1 of [TS] irf table, we fit a model using `var` and we saved the IRFs for two different orderings. The commands we used were

```
. use https://www.stata-press.com/data/r16/lutkepohl2
. var dln_inv dln_inc dln_consump
. irf set results4
. irf create ordera, step(8)
. irf create orderb, order(dln_inc dln_inv dln_consump) step(8)
```
We then formed the desired table by typing

```
.irf table oirf fevd, impulse(dln_inc) response(dln_consump) noci std
> title("Ordera versus orderb")
```

Using `irf ctable`, we can form the equivalent table by typing

```
.irf ctable (ordera dln_inc dln_consump oirf fevd)
> (orderb dln_inc dln_consump oirf fevd),
> noci std title("Ordera versus orderb")
```

<table>
<thead>
<tr>
<th>step</th>
<th>(1) oirf</th>
<th>S.E.</th>
<th>(1) fevd</th>
<th>S.E.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.005123</td>
<td>0.000878</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.001635</td>
<td>0.000984</td>
<td>0.288494</td>
<td>0.077483</td>
</tr>
<tr>
<td>2</td>
<td>0.002948</td>
<td>0.000993</td>
<td>0.294288</td>
<td>0.073722</td>
</tr>
<tr>
<td>3</td>
<td>-0.000221</td>
<td>0.000662</td>
<td>0.322454</td>
<td>0.075562</td>
</tr>
<tr>
<td>4</td>
<td>0.000811</td>
<td>0.000586</td>
<td>0.319227</td>
<td>0.074063</td>
</tr>
<tr>
<td>5</td>
<td>0.000462</td>
<td>0.000333</td>
<td>0.322579</td>
<td>0.075019</td>
</tr>
<tr>
<td>6</td>
<td>0.000444</td>
<td>0.000275</td>
<td>0.323552</td>
<td>0.075371</td>
</tr>
<tr>
<td>7</td>
<td>0.000151</td>
<td>0.000162</td>
<td>0.323383</td>
<td>0.075314</td>
</tr>
<tr>
<td>8</td>
<td>0.000091</td>
<td>0.000114</td>
<td>0.323499</td>
<td>0.075386</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>step</th>
<th>(2) oirf</th>
<th>S.E.</th>
<th>(2) fevd</th>
<th>S.E.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.005461</td>
<td>0.000925</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.001578</td>
<td>0.000988</td>
<td>0.327807</td>
<td>0.08159</td>
</tr>
<tr>
<td>2</td>
<td>0.003307</td>
<td>0.001042</td>
<td>0.328795</td>
<td>0.077519</td>
</tr>
<tr>
<td>3</td>
<td>-0.00019</td>
<td>0.000676</td>
<td>0.370775</td>
<td>0.080604</td>
</tr>
<tr>
<td>4</td>
<td>0.000846</td>
<td>0.000617</td>
<td>0.366896</td>
<td>0.079019</td>
</tr>
<tr>
<td>5</td>
<td>0.000491</td>
<td>0.000349</td>
<td>0.370399</td>
<td>0.079941</td>
</tr>
<tr>
<td>6</td>
<td>0.000069</td>
<td>0.000292</td>
<td>0.371487</td>
<td>0.080323</td>
</tr>
<tr>
<td>7</td>
<td>0.000158</td>
<td>0.000172</td>
<td>0.371315</td>
<td>0.080287</td>
</tr>
<tr>
<td>8</td>
<td>0.000096</td>
<td>0.000122</td>
<td>0.371438</td>
<td>0.080366</td>
</tr>
</tbody>
</table>

(1) `irfname = ordera`, `impulse = dln_inc`, and `response = dln_consump`
(2) `irfname = orderb`, `impulse = dln_inc`, and `response = dln_consump`

The output is displayed in one table. Because the table did not fit horizontally, it automatically wrapped. At the bottom of the table is a list of keys that appear at the top of each column. The results in the table above indicate that the orthogonalized IRFs do not change by much. Because the estimated forecast-error variances do change, we might want to produce two tables that contain the estimated forecast-error variance decompositions and their 95% confidence intervals:
. irf ctable (ordera dln_inc dln_consump fevd)
> (orderb dln_inc dln_consump fevd), individual

Table 1

<table>
<thead>
<tr>
<th>step</th>
<th>fevd</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.288494</td>
<td>0.13663</td>
<td>0.440357</td>
</tr>
<tr>
<td>2</td>
<td>0.294288</td>
<td>0.149797</td>
<td>0.43878</td>
</tr>
<tr>
<td>3</td>
<td>0.322454</td>
<td>0.174356</td>
<td>0.470552</td>
</tr>
<tr>
<td>4</td>
<td>0.319227</td>
<td>0.174066</td>
<td>0.464389</td>
</tr>
<tr>
<td>5</td>
<td>0.322579</td>
<td>0.175544</td>
<td>0.469613</td>
</tr>
<tr>
<td>6</td>
<td>0.323552</td>
<td>0.175826</td>
<td>0.471277</td>
</tr>
<tr>
<td>7</td>
<td>0.323383</td>
<td>0.17577</td>
<td>0.470996</td>
</tr>
<tr>
<td>8</td>
<td>0.323499</td>
<td>0.175744</td>
<td>0.471253</td>
</tr>
</tbody>
</table>

95% lower and upper bounds reported
(1) irfname = ordera, impulse = dln_inc, and response = dln_consump

Table 2

<table>
<thead>
<tr>
<th>step</th>
<th>fevd</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.327807</td>
<td>0.167893</td>
<td>0.487721</td>
</tr>
<tr>
<td>2</td>
<td>0.328795</td>
<td>0.176866</td>
<td>0.48073</td>
</tr>
<tr>
<td>3</td>
<td>0.370775</td>
<td>0.212794</td>
<td>0.528757</td>
</tr>
<tr>
<td>4</td>
<td>0.366896</td>
<td>0.212022</td>
<td>0.52177</td>
</tr>
<tr>
<td>5</td>
<td>0.370399</td>
<td>0.213718</td>
<td>0.52708</td>
</tr>
<tr>
<td>6</td>
<td>0.371487</td>
<td>0.214058</td>
<td>0.528917</td>
</tr>
<tr>
<td>7</td>
<td>0.371315</td>
<td>0.213956</td>
<td>0.528674</td>
</tr>
<tr>
<td>8</td>
<td>0.371438</td>
<td>0.213923</td>
<td>0.528953</td>
</tr>
</tbody>
</table>

95% lower and upper bounds reported
(2) irfname = orderb, impulse = dln_inc, and response = dln_consump

Because we specified the `individual` option, the output contains two tables, one for each specific table command. At the bottom of each table is a list of the keys used in that table and a note indicating the level of the confidence intervals that we requested. The results from table 1 and table 2 indicate that each estimated function is well within the confidence interval of the other, so we conclude that the functions are not significantly different.
Stored results

`irf ctable` stores the following in `r()`:  

Scalars  
- `r(ncols)`  
  number of columns in all tables  
- `r(k_uma)`  
  number of distinct keys  
- `r(k)`  
  number of specific table commands  

Macros  
- `r(key#)`  
  #th key  
- `r(tnotes)`  
  list of keys applied to each column

Also see

[TS] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs  
[TS] `var intro` — Introduction to vector autoregressive models  
[TS] `vec intro` — Introduction to vector error-correction models
**irf describe — Describe an IRF file**

### Description

`irf describe` describes the specification of the estimation command and the specification of the IRF used to create the IRF results that are saved in an IRF file.

### Quick start

Short summary of all IRF results in the active IRF file

```
irf describe
```

Summary of model and IRF specification for `irf1` in the active IRF file

```
irf describe irf1
```

As above, but for `irf1` in IRF file `myirf.irf`

```
irf describe irf1, using(myirf)
```

As above, and also set `myirf.irf` as the active IRF file

```
irf describe irf1, set(myirf)
```

**Note:** `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see


### Menu

Statistics > Multivariate time series > Manage IRF results and files > Describe IRF file
Syntax

```
irf describe [irf_resultslist] [, options]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set(filename)</td>
<td>make filename active</td>
</tr>
<tr>
<td>using(irf_filename)</td>
<td>describe irf_filename without making active</td>
</tr>
<tr>
<td>detail</td>
<td>show additional details of IRF results</td>
</tr>
<tr>
<td>variables</td>
<td>show underlying structure of the IRF dataset</td>
</tr>
</tbody>
</table>

Options

- **set(filename)** specifies the IRF file to be described and set; see [TS] irf set. If filename is specified without an extension, .irf is assumed.
- **using(irf_filename)** specifies the IRF file to be described. The active IRF file, if any, remains unchanged. If irf_filename is specified without an extension, .irf is assumed.
- **detail** specifies that irf describe display detailed information about each set of IRF results. detail is implied when irf_resultslist is specified.
- **variables** is a programmer’s option; additionally displays the output produced by the describe command.

Remarks and examples

If you have not read [TS] irf, please do so.

```
irf describe specified without irf_resultslist provides a short summary of the model used to create each set of results in an IRF file. If irf_resultslist is specified, then irf describe provides details of the model specification and the IRF specification used to create each set of IRF results. If set() or using() is not specified, the IRF results of the active IRF file are described.
```

Example 1

```
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
```
We create three sets of IRF results:

```
. irf create order1, set(myirfs, replace)
   (file myirfs.irf created)
   (file myirfs.irf now active)
   (file myirfs.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump)
   (file myirfs.irf updated)
. irf create order3, order(dln_inc dln_consump dln_inv)
   (file myirfs.irf updated)
. irf describe
```

Contains irf results from myirfs.irf (dated 22 Mar 2019 14:38)

<table>
<thead>
<tr>
<th>irfname</th>
<th>model</th>
<th>endogenous variables and order (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>order1</td>
<td>var</td>
<td>dln_inv dln_inc dln_consump</td>
</tr>
<tr>
<td>order2</td>
<td>var</td>
<td>dln_inc dln_inv dln_consump</td>
</tr>
<tr>
<td>order3</td>
<td>var</td>
<td>dln_inc dln_consump dln_inv</td>
</tr>
</tbody>
</table>

(*) order is relevant only when model is var

The output reveals the order in which we specified the variables.

```
. irf describe order1
```

### IRF results for order1

Estimation specification

- model: var
- endog: dln_inv dln_inc dln_consump
- sample: quarterly data from 1960q4 to 1978q4
- lags: 1 2
- constant: constant
- exog: none
- exogvars: none
- exlags: none
- varcns: unconstrained

IRF specification

- step: 8
- order: dln_inv dln_inc dln_consump
- std error: asymptotic
- reps: none

Here we see a summary of the model we fit as well as the specification of the IRFs.
Stored results

`irf describe` stores the following in `r()`:

Scalars
- `r(N)` number of observations in the IRF file
- `r(k)` number of variables in the IRF file
- `r(width)` width of dataset in the IRF file
- `r(changed)` flag indicating that data have changed since last saved

Macros
- `r(_version)` version of IRF results file
- `r(irfnames)` names of IRF results in the IRF file
- `r(irfname_model)` `var`, `sr var`, `lr var`, or `vec`
- `r(irfname_order)` Cholesky order assumed in IRF estimates
- `r(irfname_exog)` exogenous variables, and their lags, in VAR or underlying VAR
- `r(irfname_exogvars)` exogenous variables in VAR or underlying VAR
- `r(irfname_constant)` `constant` or `noconstant`
- `r(irfname_lags)` lags in model
- `r(irfname_exlags)` lags of exogenous variables in model
- `r(irfname_tmin)` minimum value of `timevar` in the estimation sample
- `r(irfname_tmax)` maximum value of `timevar` in the estimation sample
- `r(irfname_timevar)` name of `tsset` `timevar`
- `r(irfname_tsfmt)` format of `timevar` in the estimation sample
- `r(irfname_varcns)` unconstrained or colon-separated list of constraints placed on VAR coefficients
- `r(irfname_svarcns)` "." or colon-separated list of constraints placed on SVAR coefficients
- `r(irfname_step)` maximum step in IRF estimates
- `r(irfname_stderror)` asymptotic, `bs`, `bsp`, or `none`, depending on type of standard errors specified to `irf create`
- `r(irfname_reps)` "." or number of bootstrap replications performed
- `r(irfname_version)` version of IRF file that originally held `irfname` IRF results
- `r(irfname_rank)` "." or number of cointegrating equations
- `r(irfname_trend)` "." or `trend()` specified in `vec`
- `r(irfname_veccns)` "." or constraints placed on VECM parameters
- `r(irfname_sind)` "." or normalized seasonal indicators included in `vec`

Also see

[TS] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] `var intro` — Introduction to vector autoregressive models
[TS] `vec intro` — Introduction to vector error-correction models
**Description**

`irf drop` removes IRF results from the active IRF file.

**Quick start**

Drop impulse–response functions `irf1` and `irf2` from the active IRF file

```
irf drop irf1 irf2
```

Drop `irf1` and `irf2` from the IRF file `myirfs.irf`

```
irf drop irf1 irf2, set(myirfs)
```

Note: `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see [TS] `var`, [TS] `var svar`, [TS] `vec`, [TS] `arima`, [TS] `arfima`, [DSGE] `dsge`, or [DSGE] `dsgenl`.

**Menu**

Statistics > Multivariate time series > Manage IRF results and files > Drop IRF results
Syntax

```
irf drop irf_resultslist [, set(filename)]
```

Option

```
set(filename) specifies the file to be made active; see [TS] irf set. If set() is not specified, the active file is used.
```

Remarks and examples

If you have not read [TS] irf, please do so.

➢ Example 1

```
   use https://www.stata-press.com/data/r16/lutkepohl2
   (Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
   . var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
      (output omitted)

   We create three sets of IRF results:

   . irf create order1, set(myirfs, replace)
      (file myirfs.irf created)
      (file myirfs.irf now active)
      (file myirfs.irf updated)
   . irf create order2, order(dln_inc dln_inv dln_consump)
      (file myirfs.irf updated)
   . irf create order3, order(dln_inc dln_consump dln_inv)
      (file myirfs.irf updated)
   . irf describe
      Contains irf results from myirfs.irf (dated 22 Mar 2019 14:38)
      irfname model endogenous variables and order (*)
      +---------------------------+---------------------------+
      | irfname | model | endogenous variables and order (*) |
      |---------|-------|----------------------------------|
      | order1  | var   | dln_inv dln_inc dln_consump      |
      | order2  | var   | dln_inc dln_inv dln_consump     |
      | order3  | var   | dln_inc dln_consump dln_inv     |
      +---------------------------+---------------------------+
      (*) order is relevant only when model is var
```
Now let’s remove `order1` and `order2` from `myirfs.irf`.

```bash
. irf drop order1 order2  
(order1 dropped)  
(order2 dropped)  
file myirfs.irf updated  
. irf describe  
Contains irf results from myirfs.irf (dated 22 Mar 2019 14:38)  

<table>
<thead>
<tr>
<th>irfname</th>
<th>model</th>
<th>endogenous variables and order (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>order3</td>
<td>var</td>
<td>dln_inc  dln_consump dln_inv</td>
</tr>
</tbody>
</table>

(*) order is relevant only when model is var

order1 and order2 have been dropped.

Also see

[T] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[T] `var intro` — Introduction to vector autoregressive models
[T] `vec intro` — Introduction to vector error-correction models
Title

irf graph — Graphs of IRFs, dynamic-multiplier functions, and FEVDs

Description

`irf graph` graphs impulse–response functions (IRFs), dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs) over time.

Quick start

Graph impulse–response function for dependent variables `y1` and `y2` given an unexpected shock to `y1`

```
irf graph irf, impulse(y1) response(y2)
```

As above, but for orthogonalized shocks

```
irf graph oirf, impulse(y1) response(y2)
```

As above, but begin the plot with the third forecast period

```
irf graph oirf, impulse(y1) response(y2) lstep(3)
```

As above, but with a separate graph for each IRF in the current IRF file

```
irf graph oirf, impulse(y1) response(y2) lstep(3) individual
```

Note: `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see \[TS\] `var`, \[TS\] `var svar`, \[TS\] `vec`, \[TS\] `arima`, \[TS\] `arfima`, \[DSGE\] `dsge`, or \[DSGE\] `dsgenl`.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Graphs by impulse or response
**Syntax**

```bash
irf graph stat [ , options ]
```

<table>
<thead>
<tr>
<th>stat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irf</td>
<td>impulse–response function</td>
</tr>
<tr>
<td>oirf</td>
<td>orthogonalized impulse–response function</td>
</tr>
<tr>
<td>dm</td>
<td>dynamic-multiplier function</td>
</tr>
<tr>
<td>cirf</td>
<td>cumulative impulse–response function</td>
</tr>
<tr>
<td>coirf</td>
<td>cumulative orthogonalized impulse–response function</td>
</tr>
<tr>
<td>cdm</td>
<td>cumulative dynamic-multiplier function</td>
</tr>
<tr>
<td>fevd</td>
<td>Cholesky forecast-error variance decomposition</td>
</tr>
<tr>
<td>sirf</td>
<td>structural impulse–response function</td>
</tr>
<tr>
<td>sfevd</td>
<td>structural forecast-error variance decomposition</td>
</tr>
</tbody>
</table>

**Notes:**
1. No statistic may appear more than once.
2. If confidence intervals are included (the default), only two statistics may be included.
3. If confidence intervals are suppressed (option `noci`), up to four statistics may be included.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set(filename)</td>
<td>make <code>filename</code> active</td>
</tr>
<tr>
<td>irf( irfnames )</td>
<td>use <code>irfnames</code> IRF result sets</td>
</tr>
<tr>
<td>impulse( impulsevar )</td>
<td>use <code>impulsevar</code> as impulse variables</td>
</tr>
<tr>
<td>response( endogvars )</td>
<td>use endogenous variables as response variables</td>
</tr>
<tr>
<td>noci</td>
<td>suppress confidence bands</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td>lstep(#)</td>
<td>use # for first step</td>
</tr>
<tr>
<td>ustep(#)</td>
<td>use # for maximum step</td>
</tr>
<tr>
<td>individual</td>
<td>graph each combination individually</td>
</tr>
<tr>
<td>iname(namestub [ , replace ] )</td>
<td>stub for naming the individual graphs</td>
</tr>
<tr>
<td>isaving(filenamestub [ , replace ] )</td>
<td>stub for saving the individual graphs to files</td>
</tr>
<tr>
<td>plot#opts(cline_options)</td>
<td>affect rendition of the line plotting the # stat</td>
</tr>
<tr>
<td>ci#opts(area_options)</td>
<td>affect rendition of the confidence interval for the # stat</td>
</tr>
</tbody>
</table>

**Plots**

**CI plots**

**Y axis, X axis, Titles, Legend, Overall**

**twoway_options**

**byopts(by_option)**

how subgraphs are combined, labeled, etc.
**Options**

**Main**

*set*(`filename`) specifies the file to be made active; see [TS] `irf set`. If `set()` is not specified, the active file is used.

*irf*(`irfnames`) specifies the IRF result sets to be used. If `irf()` is not specified, each of the results in the active IRF file is used. (Files often contain just one set of IRF results saved under one `irfname`; in that case, those results are used.)

*impulse*(`impulsevar`) and *response*(`endogvars`) specify the impulse and response variables. Usually one of each is specified, and one graph is drawn. If multiple variables are specified, a separate subgraph is drawn for each impulse–response combination. If `impulse()` and `response()` are not specified, subgraphs are drawn for all combinations of impulse and response variables.

*impulsevar* should be specified as an endogenous variable for all statistics except `dm` or `cdm`; for those, specify as an exogenous variable.

*noci* suppresses graphing the confidence interval for each statistic. *noci* is assumed when the model was fit by `vec` because no confidence intervals were estimated.

*level(#)* specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is `level(95)` or as set by `set level`; see [U] 20.8 Specifying the width of confidence intervals. Also see [TS] `irf cgraph` for a graph command that allows the confidence level to vary over the graphs.

*lstep(#)* specifies the first step, or period, to be included in the graphs. `lstep(0)` is the default. *ustep(#), # ≥ 1,* specifies the maximum step, or period, to be included in the graphs.

**Advanced**

*individual* specifies that each graph be displayed individually. By default, `irf graph` combines the subgraphs into one image. When `individual` is specified, `byopts()` may not be specified, but the `isaving()` and `iname()` options may be specified.

*iname*(`namestub` [, replace]) specifies that the *i*th individual graph be stored in memory under the name `namestubi`, which must be a valid Stata name of 24 characters or fewer. `iname()` may be specified only with the `individual` option.

*isaving*(`filenamestub` [, replace]) specifies that the *i*th individual graph should be saved to disk in the current working directory under the name `filenamestubi.gph`. `isaving()` may be specified only when the `individual` option is also specified.

**Plots**

*plot1opts*(`cline_options`), ..., *plot4opts*(`cline_options`) affect the rendition of the plotted statistics (the `stat`). `plot1opts()` affects the rendition of the first statistic; `plot2opts()`, the second; and so on. `cline_options` are as described in [G-3] `cline_options`.

**CI plots**

*ci1opts*(`area_options`) and *ci2opts*(`area_options`) affect the rendition of the confidence intervals for the first (`ci1opts()`) and second (`ci2opts()`) statistics in `stat`. `area_options` are as described in [G-3] `area_options`. 
twoway_options are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option). The saving() and name() options may not be combined with the individual option.

byopts(by_option) is as documented in [G-3] by_option and may not be specified when individual is specified. byopts() affects how the subgraphs are combined, labeled, etc.

Remarks and examples

If you have not read [TS] irf, please do so.

Also see [TS] irf cgraph, which produces combined graphs; [TS] irf ograph, which produces overlaid graphs; and [TS] irf table, which displays results in tabular form.

irf graph produces one or more graphs and displays them arrayed into one image unless the individual option is specified, in which case the individual graphs are displayed separately. Each individual graph consists of all the specified stat and represents one impulse–response combination.

Because all the specified stat appear on the same graph, putting together statistics with very different scales is not recommended. For instance, sometimes sirf and oirf are on similar scales while irf is on a different scale. In such cases, combining sirf and oirf on the same graph looks fine, but combining either with irf produces an uninformative graph.

Example 1

Suppose that we have results generated from two different SVAR models. We want to know whether the shapes of the structural IRFs and the structural FEVDs are similar in the two models. We are also interested in knowing whether the structural IRFs and the structural FEVDs differ significantly from their Cholesky counterparts.

Filling in the background, we have previously issued the commands:

```
. use https://www.stata-press.com/data/r16/lutkepohl2
. mat a = (., 0, 0\0,.,0\.,.,.)
. mat b = I(3)
. svar dln_inv dln_inc dln_consump, aeq(a) beq(b)
. irf create modela, set(results3) step(8)
. svar dln_inc dln_inv dln_consump, aeq(a) beq(b)
. irf create modelb, step(8)
```
To see whether the shapes of the structural IRFs and the structural FEVDs are similar in the two models, we type

```
.irf graph oirf sirf, impulse(dln_inc) response(dln_consump)
```

![Graphs of IRFs and FEVDs](image)

The graph reveals that the `oirf` and the `sirf` estimates are essentially the same for both models and that the shapes of the functions are very similar for the two models.

To see whether the structural IRFs and the structural FEVDs differ significantly from their Cholesky counterparts, we type

```
.irf graph fevd sfevd, impulse(dln_inc) response(dln_consump) lstep(1)
> legend(cols(1))
```

![Graphs of FEVDs and Structural FEVDs](image)

Graphs by irfname, impulse variable, and response variable

The graph shows that the `fevd` and the `sfevd` estimates are similar, with slight differences in the shapes of the functions. This indicates that the structural IRFs and FEVDs are similar to their Cholesky counterparts.

---

**Note:** The above text is a natural language representation of the content of the image. The diagrams and graphs are not fully transcribed here due to the limitations of the text-based format. The commands and their outputs are intended to convey the steps and results of the analysis rather than the visual representation.
This combined graph reveals that the shapes of these functions are also similar for the two models. However, the graph illuminates one minor difference between them: In model a, the estimated structural FEVD is slightly larger than the Cholesky-based estimates, whereas in model b the Cholesky-based estimates are slightly larger than the structural estimates. For both models, however, the structural estimates are close to the center of the wide confidence intervals for the two estimates.

Example 2

Let’s focus on the results from model a. Suppose that we were interested in examining how \( \text{dln}_{\text{consump}} \) responded to impulses in its own structural innovations, structural innovations to \( \text{dln}_{\text{inc}} \), and structural innovations to \( \text{dln}_{\text{inv}} \). We type

\[
. \text{irf graph sirf, irf(modela) response(dln_consump)}
\]

![Graphs by irfname, impulse variable, and response variable](image)

The upper-left graph shows the structural IRF of an innovation in \( \text{dln}_{\text{consump}} \) on \( \text{dln}_{\text{consump}} \). It indicates that the identification restrictions used in model a imply that a positive shock to \( \text{dln}_{\text{consump}} \) causes an increase in \( \text{dln}_{\text{consump}} \), followed by a decrease, followed by an increase, and so on, until the effect dies out after roughly 5 periods.

The upper-right graph shows the structural IRF of an innovation in \( \text{dln}_{\text{inc}} \) on \( \text{dln}_{\text{consump}} \), indicating that a positive shock to \( \text{dln}_{\text{inc}} \) causes an increase in \( \text{dln}_{\text{consump}} \), which dies out after 4 or 5 periods.

Technical note

The IRF table contains a technical note warning you to be careful in naming variables when you fit models. What is said there applies equally here.
Stored results

`irf graph` stores the following in `r()`:

**Scalars**
- `r(k)`
  number of graphs

**Macros**
- `r(stats)`
  statlist
- `r(irfname)`
  resultslist
- `r(impulse)`
  impulselist
- `r(response)`
  responselist
- `r(plot#)`
  contents of plot#opts()
- `r(ci)`
  level applied to confidence intervals or noci
- `r(ciopts#)`
  contents of ci#opts()
- `r(byopts)`
  contents of byopts()
- `r(saving)`
  supplied saving() option
- `r(byopts)`
  contents of byopts()
- `r(name)`
  supplied name() option
- `r(individual)`
  individual or blank
- `r(isaving)`
  contents of isaving()
- `r(iname)`
  contents of name()
- `r(subtitle#)`
  subtitle for individual graph #

Also see

[Ts] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[Ts] `var intro` — Introduction to vector autoregressive models
[Ts] `vec intro` — Introduction to vector error-correction models
irf ograph — Overlaid graphs of IRFs, dynamic-multiplier functions, and FEVDs

Description

irf ograph displays plots of irf results on one graph (one pair of axes).

To become familiar with this command, type db irf ograph.

Quick start

Graph of an orthogonalized IRF myirf overlayed on cumulative IRF mycirf for dependent variables y1 and y2

```
irf ograph (myirf y1 y2 oirf) (mycirf y1 y2 cirf)
```

As above, and include confidence bands and add a title

```
irf cgraph (myirf y1 y2 oirf) (mycirf y1 y2 cirf), ci ///
    title("My Title")
```

Note: irf commands can be used after var, svar, vec, arima, arfima, dsge, or dsgenl; see [TS] var, [TS] var svar, [TS] vec, [TS] arima, [TS] arfima, [DSGE] dsge, or [DSGE] dsgenl.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Overlaid graph
Syntax

```
irf ograph (spec_1) [(spec_2) ... (spec_15)] [, options]
```

where \((\text{spec}_k)\) is

```
(irfname impulsevar responsevar stat [, spec_options])
```

\(irfname\) is the name of a set of IRF results in the active IRF file or “.”, which means the first named result in the active IRF file. \(impulsevar\) should be specified as an endogenous variable for all statistics except \(\text{dm}\) and \(\text{cdm}\); for those, specify as an exogenous variable. \(responsevar\) is an endogenous variable name. \(stat\) is one or more statistics from the list below:

<table>
<thead>
<tr>
<th>(stat)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(irf)</td>
<td>impulse–response function</td>
</tr>
<tr>
<td>(oirf)</td>
<td>orthogonalized impulse–response function</td>
</tr>
<tr>
<td>(dm)</td>
<td>dynamic-multiplier function</td>
</tr>
<tr>
<td>(cirf)</td>
<td>cumulative impulse–response function</td>
</tr>
<tr>
<td>(coirf)</td>
<td>cumulative orthogonalized impulse–response function</td>
</tr>
<tr>
<td>(cdm)</td>
<td>cumulative dynamic-multiplier function</td>
</tr>
<tr>
<td>(fevd)</td>
<td>Cholesky forecast-error variance decomposition</td>
</tr>
<tr>
<td>(sirf)</td>
<td>structural impulse–response function</td>
</tr>
<tr>
<td>(sfevd)</td>
<td>structural forecast-error variance decomposition</td>
</tr>
</tbody>
</table>

\(options\)

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Plots</strong></td>
</tr>
<tr>
<td>(plot_options) define the IRF plots</td>
</tr>
<tr>
<td>(set(filename)) make (filename) active</td>
</tr>
<tr>
<td><strong>Options</strong></td>
</tr>
<tr>
<td>(common_options) level and steps</td>
</tr>
<tr>
<td><strong>Y axis, X axis, Titles, Legend, Overall</strong></td>
</tr>
<tr>
<td>(twoway_options) any options other than (by()) documented in [G-3] (twoway_options)</td>
</tr>
</tbody>
</table>

\(plot_options\)

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(set(filename)) make (filename) active</td>
</tr>
<tr>
<td>(irf(irfnames)) use (irf\names) IRF result sets</td>
</tr>
<tr>
<td>(impulse(impulsevar)) use (impulsevar) as impulse variables</td>
</tr>
<tr>
<td>(response(endogvars)) use endogenous variables as response variables</td>
</tr>
<tr>
<td>(ci) add confidence bands to the graph</td>
</tr>
</tbody>
</table>
Overlaid graphs of IRFs, dynamic-multiplier functions, and FEVDs

**spec_options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>common_options</strong></td>
<td>level and steps</td>
</tr>
<tr>
<td><strong>Plot</strong></td>
<td></td>
</tr>
<tr>
<td><strong>cline_options</strong></td>
<td>affect rendition of the plotted lines</td>
</tr>
<tr>
<td><strong>CI plot</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ciopts(area_options)</strong></td>
<td>affect rendition of the confidence intervals</td>
</tr>
</tbody>
</table>

**common_options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>lstep(#)</td>
<td>use # for first step</td>
</tr>
<tr>
<td>ustep(#)</td>
<td>use # for maximum step</td>
</tr>
</tbody>
</table>

*common_options* may be specified within a plot specification, globally, or in both. When specified in a plot specification, the *common_options* affect only the specification in which they are used. When supplied globally, the *common_options* affect all plot specifications. When supplied in both places, options in the plot specification take precedence.

### Options

#### Plots

**plot_options** defines the IRF plots and are found under the **Main**, **Plot**, and **CI plot** tabs.

- **set(filename)** specifies the file to be made active; see [TS] `irf set`. If `set()` is not specified, the active file is used.

#### Main

- **set(filename)** specifies the file to be made active; see [TS] `irf set`. If `set()` is not specified, the active file is used.

- **irf(irfnames)** specifies the IRF result sets to be used. If `irf()` is not specified, each of the results in the active IRF file is used. (Files often contain just one set of IRF results saved under one *irfname*; in that case, those results are used.)

- **impulse(varlist)** and **response(endogvars)** specify the impulse and response variables. Usually one of each is specified, and one graph is drawn. If multiple variables are specified, a separate subgraph is drawn for each impulse–response combination. If `impulse()` and `response()` are not specified, subgraphs are drawn for all combinations of impulse and response variables.

- **ci** adds confidence bands to the graph. The **noci** option may be used within a plot specification to suppress its confidence bands when the **ci** option is supplied globally.

#### Plot

**cline_options** affect the rendition of the plotted lines; see [G-3] `cline_options`. 
ciplot

.ciplot(area_options) affects the rendition of the confidence bands for the plotted statistic; see [G-3] area_options. ciplot() implies ci.

.Options

level(#) specifies the confidence level, as a percentage, for confidence bands; see [U] 20.8 Specifying the width of confidence intervals.

lstep(#) specifies the first step, or period, to be included in the graph. lstep(0) is the default.

ustep(#), # ≥ 1, specifies the maximum step, or period, to be included.

twoway_options are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option).

Remarks and examples

If you have not read [TS] irf, please do so.

irf ograph overlays plots of IRFs and FEVDs on one graph.

Example 1

We have previously issued the commands:

. use https://www.stata-press.com/data/r16/lutkepohl2
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
. irf create order1, step(10) set(myirf1, new)
. irf create order2, step(10) order(dln_inc dln_inv dln_consump)
We now wish to compare the `oirf` for impulse `dln_inc` and response `dln_consump` for two different Cholesky orderings:

```
. irf ograph (order1 dln_inc dln_consump oirf)
> (order2 dln_inc dln_consump oirf)
```

- Technical note

Graph options allow you to change the appearance of each plot. The following graph contains the plots of the FEVDs (FEVDs) for impulse `dln_inc` and each response using the results from the first collection of results in the active IRF file (using the "." shortcut). In the second plot, we supply the `clpat(dash)` option (an abbreviation for `clpattern(dash)`) to give the line a dashed pattern. In the third plot, we supply the `m(o) clpat(dash_dot) recast(connected)` options to get small circles connected by a line with a dash–dot pattern; the `cilines` option plots the confidence bands by using lines instead of areas. We use the `title()` option to add a descriptive title to the graph and supply the `ci` option globally to add confidence bands to all the plots.
. irf ograph (. dln_inc dln_inc fevd)
> (. dln_inc dln_consump fevd, clpat(dash))
> (. dln_inc dln_inv fevd, cilines m(o) clpat(dash_dot)
> recast(connected))
> , ci title("Comparison of forecast-error variance decomposition")

The clpattern() option is described in [G-3] connect_options, msymbol() is described in [G-3] marker_options, title() is described in [G-3] title_options, and recast() is described in [G-3] advanced_options.

Stored results

irf ograph stores the following in r():

Scalars
- r(plots) number of plot specifications
- r(ciplots) number of plotted confidence bands

Macros
- r(irfname#) irfname from (spec#)
- r(impulse#) impulse from (spec#)
- r(response#) response from (spec#)
- r(stat#) statistics from (spec#)
- r(ci#) level from (spec#) or noci

Also see

[TS] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] var intro — Introduction to vector autoregressive models
[TS] vec intro — Introduction to vector error-correction models
Title

**irf rename** — Rename an IRF result in an IRF file

### Description

`irf rename` changes the name of a set of IRF results saved in the active IRF file.

### Quick start

Rename impulse–response function `oldirf` in the current file to `newirf`

```
irf rename oldirf newirf
```

As above, but for IRF file `myirfs.irf`

```
irf rename oldirf newirf, set(myirfs)
```

Note: `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see [TS] `var`, [TS] `var svar`, [TS] `vec`, [TS] `arima`, [TS] `arfima`, [DSGE] `dsge`, or [DSGE] `dsgenl`.

### Menu

Statistics > Multivariate time series > Manage IRF results and files > Rename IRF results

### Syntax

```
irf rename oldname newname [, set(filename)]
```

### Option

`set(filename)` specifies the file to be made active; see [TS] `irf set`. If `set()` is not specified, the active file is used.

### Remarks and examples

If you have not read [TS] `irf`, please do so.

#### Example 1

```
. use https://www.stata-press.com/data/r16/lutkepohl2
   (Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
   (output omitted)
```

338
We create three sets of IRF results:

```
. irf create original, set(myirfs, replace)
(file myirfs.irf created)
(file myirfs.irf now active)
(file myirfs.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump)
(file myirfs.irf updated)
. irf create order3, order(dln_inc dln_consump dln_inv)
(file myirfs.irf updated)
. irf describe
Contains irf results from myirfs.irf (dated 22 Mar 2019 14:39)

irfname    model     endogenous variables and order (*)
original    var       dln_inv dln_inc dln_consump
order2      var       dln_inc dln_inv dln_consump
order3      var       dln_inc dln_consump dln_inv
```

(*) order is relevant only when model is var

Now let’s rename IRF result original to order1.

```
. irf rename original order1
(81 real changes made)
original renamed to order1
. irf describe
Contains irf results from myirfs.irf (dated 22 Mar 2019 14:39)

irfname    model     endogenous variables and order (*)
order1      var       dln_inv dln_inc dln_consump
order2      var       dln_inc dln_inv dln_consump
order3      var       dln_inc dln_consump dln_inv
```

(*) order is relevant only when model is var

original has been renamed to order1.

**Stored results**

`irf rename` stores the following in `r()`:

Macros

```
(r(irfnames)   irfnames after rename
r(oldnew)      oldname newname
```

**Also see**

[TS] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] `var intro` — Introduction to vector autoregressive models
[TS] `vec intro` — Introduction to vector error-correction models
**Description**

`irf set` without arguments reports the identity of the active IRF file, if there is one. `irf set` with a filename specifies that the file be created and set as the active file. `irf set, clear` specifies that, if any IRF file is set, it be unset and that there be no active IRF file.

**Quick start**

Display filename of active IRF file

```
irf set
```

Set file `myirfs.irf` as the active file and create it if it does not exist

```
irf set myirfs
```

Set file `myirfs.irf` as the active file, but replace `myirfs.irf` if it exists

```
irf set myirfs, replace
```

Clear the active IRF file so that no files are active

```
irf set, clear
```

Note: `irf` commands can be used after `var`, `svar`, `vec`, `arima`, `arfima`, `dsge`, or `dsgenl`; see `[TS] var`, `[TS] var svar`, `[TS] vec`, `[TS] arima`, `[TS] arfima`, `[DSGE] dsge`, or `[DSGE] dsgenl.

**Menu**

Statistics > Multivariate time series > Manage IRF results and files > Set active IRF file
Syntax

Report identity of active file

    irf set

Set, and if necessary create, active file

    irf set irf_filename [, replace]

Clear any active IRF file

    irf set, clear

If `irf_filename` is specified without an extension, `.irf` is assumed.

Options

`replace` specifies that if `irf_filename` already exists, the file is to be erased and a new, empty IRF file is to be created in its place. If it does not already exist, a new, empty file is created.

`clear` unsets the active IRF file.

Remarks and examples

If you have not read [TS] `irf`, please do so.

`irf set` reports the identity of the active IRF file:

    . irf set
     no irf file active

`irf set irf_filename` creates and sets an IRF file:

    . irf set results1
     (file results1.irf now active)

We specified the name `results1`, and `results1.irf` became the active file.

`irf set irf_filename` can also be used to create a new file:

    . use https://www.stata-press.com/data/r16/lutkepohl2
     (Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
    . var dln_inc dln_consump, exog(1.dln_inv)
     (output omitted)
    . irf create order1
     (file results2.irf created)
     (file results2.irf now active)
    . irf set results2
     (file results2.irf created)
     (file results2.irf now active)
    . irf create order1
     (file results2.irf updated)
Stored results

`irf set` stores the following in `r()`:

Macros

- `r(irffile)` — name of active IRF file, if there is an active IRF

Also see

[TS] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] `irf describe` — Describe an IRF file

[TS] `var intro` — Introduction to vector autoregressive models

[TS] `vec intro` — Introduction to vector error-correction models
Description

irf table makes a table of the values of the requested statistics at each time since impulse. Each column represents a combination of an impulse variable and a response variable for each statistic from the named IRF results.

Quick start

Table of impulse–response function for dependent variables \( y_1 \) and \( y_2 \) given an unexpected shock to \( y_1 \)

\[
\text{irf table irf, impulse(y1) response(y2)}
\]

As above, but for orthogonalized shocks

\[
\text{irf table oirf, impulse(y1) response(y2)}
\]

As above, but with 3 as the common maximum step horizon for all tables

\[
\text{irf table oirf, impulse(y1) response(y2) step(3)}
\]

As above, but with a separate table for each IRF in the active IRF file

\[
\text{irf table oirf, impulse(y1) response(y2) step(3) individual}
\]

Note: irf commands can be used after var, svar, vec, arima, arfima, dsge, or dsgenl; see \[TS\] var, \[TS\] var svar, \[TS\] vec, \[TS\] arima, \[TS\] arfima, \[DSGE\] dsge, or \[DSGE\] dsgenl.

Menu

Statistics \> Multivariate time series \> IRF and FEVD analysis \> Tables by impulse or response
**Syntax**

`irf table [stat] [, options]`

<table>
<thead>
<tr>
<th>stat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>irf</code></td>
<td>impulse–response function</td>
</tr>
<tr>
<td><code>oirf</code></td>
<td>orthogonalized impulse–response function</td>
</tr>
<tr>
<td><code>dm</code></td>
<td>dynamic-multiplier function</td>
</tr>
<tr>
<td><code>cirf</code></td>
<td>cumulative impulse–response function</td>
</tr>
<tr>
<td><code>coirf</code></td>
<td>cumulative orthogonalized impulse–response function</td>
</tr>
<tr>
<td><code>cdm</code></td>
<td>cumulative dynamic-multiplier function</td>
</tr>
<tr>
<td><code>fevd</code></td>
<td>Cholesky forecast-error variance decomposition</td>
</tr>
<tr>
<td><code>sirf</code></td>
<td>structural impulse–response function</td>
</tr>
<tr>
<td><code>sfevd</code></td>
<td>structural forecast-error variance decomposition</td>
</tr>
</tbody>
</table>

If `stat` is not specified, all statistics are included, unless option `nostructural` is also specified, in which case `sirf` and `sfevd` are excluded. You may specify more than one `stat`.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set(filename)</code></td>
<td>make <code>filename</code> active</td>
</tr>
<tr>
<td><code>irf(irfnames)</code></td>
<td>use <code>irfnames</code> IRF result sets</td>
</tr>
<tr>
<td><code>impulse(impulsevar)</code></td>
<td>use <code>impulsevar</code> as impulse variables</td>
</tr>
<tr>
<td><code>response(endogvars)</code></td>
<td>use endogenous variables as response variables</td>
</tr>
<tr>
<td><code>individual</code></td>
<td>make an individual table for each result set</td>
</tr>
<tr>
<td><code>title(”text”)</code></td>
<td>use <code>text</code> for overall table title</td>
</tr>
</tbody>
</table>

**Options**

- `level(#)` set confidence level; default is `level(95)`
- `noci` suppress confidence intervals
- `stderr` include standard errors in the tables
- `nostructural` suppress `sirf` and `sfevd` from the default list of statistics
- `step(#)` use common maximum step horizon # for all tables

---

**Options**

- `set(filename)` specifies the file to be made active; see [TS] `irf set`. If `set()` is not specified, the active file is used.

  All results are obtained from one IRF file. If you have results in different files that you want in one table, use `irf add` to copy results into one file; see [TS] `irf add`.

- `irf(irfnames)` specifies the IRF result sets to be used. If `irf()` is not specified, all the results in the active IRF file are used. (Files often contain just one set of IRF results, saved under one `irfname`; in that case, those results are used. When there are multiple IRF results, you may also wish to specify the individual option.)
impulse(impulsevar) specifies the impulse variables for which the statistics are to be reported. If impulse() is not specified, each model variable, in turn, is used. impulsevar should be specified as an endogenous variable for all statistics except dm or cdm; for those, specify as an exogenous variable.

response(endogvars) specifies the response variables for which the statistics are to be reported. If response() is not specified, each endogenous variable, in turn, is used.

individual specifies that each set of IRF results be placed in its own table, with its own title and footer. By default, irf table places all the IRF results in one table with one title and one footer. individual may not be combined with title().

title("text") specifies a title for the overall table.

Options

level(#) specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is level(95) or as set by set level; see [U] 20.8 Specifying the width of confidence intervals.

noci suppresses reporting of the confidence intervals for each statistic. noci is assumed when the model was fit by vec because no confidence intervals were estimated.

stderr specifies that standard errors for each statistic also be included in the table.

nostructural specifies that stat, when not specified, exclude sirf and sfevd.

step(#) specifies the maximum step horizon for all tables. If step() is not specified, each table is constructed using all steps available.

Remarks and examples

If you have not read [TS] irf, please do so.

Also see [TS] irf graph, which produces output in graphical form, and see [TS] irf ctable, which also produces tabular output. irf ctable is more difficult to use but provides more control over how tables are formed.

Example 1

We have fit a model with var, and we saved the IRFs from two different orderings. The commands we previously used were

```
.use https://www.stata-press.com/data/r16/lutkepohl2
. var dln_inv dln_inc dln_consump
. irf set results4
. irf create ordera, step(8)
. irf create orderb, order(dln_inc dln_inv dln_consump) step(8)
```
We now wish to compare the two orderings:

```
. irf table oirf fevd, impulse(dln_inc) response(dln_consump) noci std
> title("Ordera versus orderb")
```

### Ordera versus orderb

<table>
<thead>
<tr>
<th>step</th>
<th>(1)</th>
<th>(1)</th>
<th>(1)</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oirf</td>
<td>S.E.</td>
<td>fevd</td>
<td>S.E.</td>
</tr>
<tr>
<td>0</td>
<td>0.005123</td>
<td>0.000878</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.001635</td>
<td>0.000984</td>
<td>0.288494</td>
<td>0.077483</td>
</tr>
<tr>
<td>2</td>
<td>0.002948</td>
<td>0.000993</td>
<td>0.294288</td>
<td>0.073722</td>
</tr>
<tr>
<td>3</td>
<td>0.002221</td>
<td>0.000662</td>
<td>0.322454</td>
<td>0.075562</td>
</tr>
<tr>
<td>4</td>
<td>0.000811</td>
<td>0.000586</td>
<td>0.319227</td>
<td>0.074063</td>
</tr>
<tr>
<td>5</td>
<td>0.000462</td>
<td>0.000333</td>
<td>0.322579</td>
<td>0.075019</td>
</tr>
<tr>
<td>6</td>
<td>0.000091</td>
<td>0.000114</td>
<td>0.323999</td>
<td>0.075386</td>
</tr>
<tr>
<td>7</td>
<td>0.000151</td>
<td>0.000162</td>
<td>0.323383</td>
<td>0.075314</td>
</tr>
<tr>
<td>8</td>
<td>0.000044</td>
<td>0.000085</td>
<td>0.323383</td>
<td>0.075314</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>step</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oirf</td>
<td>S.E.</td>
<td>fevd</td>
<td>S.E.</td>
</tr>
<tr>
<td>0</td>
<td>0.005461</td>
<td>0.000925</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.001578</td>
<td>0.000988</td>
<td>0.327807</td>
<td>0.08159</td>
</tr>
<tr>
<td>2</td>
<td>0.003307</td>
<td>0.001042</td>
<td>0.328795</td>
<td>0.077519</td>
</tr>
<tr>
<td>3</td>
<td>0.000191</td>
<td>0.000676</td>
<td>0.370775</td>
<td>0.080604</td>
</tr>
<tr>
<td>4</td>
<td>0.000846</td>
<td>0.000617</td>
<td>0.366896</td>
<td>0.079019</td>
</tr>
<tr>
<td>5</td>
<td>0.000491</td>
<td>0.000349</td>
<td>0.370399</td>
<td>0.079941</td>
</tr>
<tr>
<td>6</td>
<td>0.000096</td>
<td>0.000122</td>
<td>0.371438</td>
<td>0.080366</td>
</tr>
<tr>
<td>7</td>
<td>0.000158</td>
<td>0.000172</td>
<td>0.371315</td>
<td>0.080287</td>
</tr>
<tr>
<td>8</td>
<td>0.000096</td>
<td>0.000122</td>
<td>0.371438</td>
<td>0.080366</td>
</tr>
</tbody>
</table>

(1) irfname = ordera, impulse = dln_inc, and response = dln_consump
(2) irfname = orderb, impulse = dln_inc, and response = dln_consump

The output is displayed as a “single” table; because the table did not fit horizontally, it wrapped automatically. At the bottom of the table is a definition of the keys that appear at the top of each column. The results in the table above indicate that the orthogonalized IRFs do not change by much.
Example 2

Because the estimated FEVDs do change significantly, we might want to produce two tables that contain the estimated FEVDs and their 95% confidence intervals:

```
. irf table fevd, impulse(dln_inc) response(dln_consump) individual
```

Results from ordera

<table>
<thead>
<tr>
<th>step</th>
<th>fevd</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>.288494</td>
<td>.13663</td>
<td>.440357</td>
</tr>
<tr>
<td>2</td>
<td>.294288</td>
<td>.149797</td>
<td>.43878</td>
</tr>
<tr>
<td>3</td>
<td>.322454</td>
<td>.174356</td>
<td>.470552</td>
</tr>
<tr>
<td>4</td>
<td>.319227</td>
<td>.174066</td>
<td>.464389</td>
</tr>
<tr>
<td>5</td>
<td>.322579</td>
<td>.175544</td>
<td>.469613</td>
</tr>
<tr>
<td>6</td>
<td>.323552</td>
<td>.175826</td>
<td>.471277</td>
</tr>
<tr>
<td>7</td>
<td>.323383</td>
<td>.17577</td>
<td>.470995</td>
</tr>
<tr>
<td>8</td>
<td>.323499</td>
<td>.175744</td>
<td>.471253</td>
</tr>
</tbody>
</table>

95% lower and upper bounds reported
(1) irfname = ordera, impulse = dln_inc, and response = dln_consump

Results from orderb

<table>
<thead>
<tr>
<th>step</th>
<th>fevd</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>.327807</td>
<td>.167893</td>
<td>.487721</td>
</tr>
<tr>
<td>2</td>
<td>.328795</td>
<td>.17686</td>
<td>.48073</td>
</tr>
<tr>
<td>3</td>
<td>.370775</td>
<td>.212794</td>
<td>.528757</td>
</tr>
<tr>
<td>4</td>
<td>.366896</td>
<td>.212022</td>
<td>.52177</td>
</tr>
<tr>
<td>5</td>
<td>.370399</td>
<td>.213718</td>
<td>.52708</td>
</tr>
<tr>
<td>6</td>
<td>.371487</td>
<td>.214058</td>
<td>.528917</td>
</tr>
<tr>
<td>7</td>
<td>.371315</td>
<td>.213956</td>
<td>.528674</td>
</tr>
<tr>
<td>8</td>
<td>.371438</td>
<td>.213923</td>
<td>.528953</td>
</tr>
</tbody>
</table>

95% lower and upper bounds reported
(1) irfname = orderb, impulse = dln_inc, and response = dln_consump

Because we specified the `individual` option, the output contains two tables, one for each set of IRF results. Examining the results in the tables indicates that each of the estimated functions is well within the confidence interval of the other, so we conclude that the functions are not significantly different.

Technical note

Be careful in how you name variables when you fit models. Say that you fit one model with `var` and used time-series operators to form one of the endogenous variables

```
. var d.ln_inv ...
```

and in another model, you created a new variable:

```
. generate dln_inv = d.ln_inv
. var dln_inv ...
```
Say that you saved IRF results from both (perhaps they differ in the number of lags). Now you wish to use \texttt{irf table} to compare them. You would not be able to specify \texttt{response(d.ln_inv)} or \texttt{response(dln_inv)} because neither variable is in both models. Similarly, you could not specify \texttt{impulse(d.ln_inv)} or \texttt{impulse(dln_inv)} for the same reason.

All is not lost; if \texttt{impulse()} is not specified, all endogenous variables are used, and similarly if \texttt{response()} is not specified, so you could obtain the result you desired by simply not specifying the options, but you will also obtain a lot more, besides. If you want to specify the \texttt{impulse()} or \texttt{response()} options, be sure to name variables consistently.

Also, you may forget how the endogenous variables were named. If so, \texttt{irf describe, detail} can provide the answer. In \texttt{irf describe'}s output, the endogenous variables are listed next to \texttt{endog}.

\textbf{Stored results}

If the \texttt{individual} option is not specified, \texttt{irf table} stores the following in \texttt{r()}:

Scalars
- \texttt{r(ncols)}: number of columns in table
- \texttt{r(k_umax)}: number of distinct keys
- \texttt{r(k)}: number of specific table commands

Macros
- \texttt{r(key#)}: \#th key
- \texttt{r(tnotes)}: list of keys applied to each column

If the \texttt{individual} option is specified, then for each \texttt{irfname}, \texttt{irf table} stores the following in \texttt{r()}:

Scalars
- \texttt{r(irfname_ncols)}: number of columns in table for \texttt{irfname}
- \texttt{r(irfname_k_umax)}: number of distinct keys in table for \texttt{irfname}
- \texttt{r(irfname_k)}: number of specific table commands used to create table for \texttt{irfname}

Macros
- \texttt{r(irfname_key#)}: \#th key for \texttt{irfname} table
- \texttt{r(irfname_tnotes)}: list of keys applied to each column in table for \texttt{irfname}

\textbf{Also see}

[TS] \texttt{irf} — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] \texttt{var intro} — Introduction to vector autoregressive models
[TS] \texttt{vec intro} — Introduction to vector error-correction models
Description

`mgarch` estimates the parameters of multivariate generalized autoregressive conditional-heteroskedasticity (MGARCH) models. MGARCH models allow both the conditional mean and the conditional covariance to be dynamic.

The general MGARCH model is so flexible that not all the parameters can be estimated. For this reason, there are many MGARCH models that parameterize the problem more parsimoniously.

`mgarch` implements four commonly used parameterizations: the diagonal vech model, the constant conditional correlation model, the dynamic conditional correlation model, and the time-varying conditional correlation model.

Syntax

```
mgarch model eq [ eq ... eq ] [ if ] [ in ] [ , ... ]
```

Family

<table>
<thead>
<tr>
<th>Vech</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagonal vech</td>
<td>dvech</td>
</tr>
</tbody>
</table>

Conditional correlation

| constant conditional correlation | ccc |
| dynamic conditional correlation | dcc |
| varying conditional correlation | vcc |

Remarks and examples

Remarks are presented under the following headings:

- An introduction to MGARCH models
- Diagonal vech MGARCH models
- Conditional correlation MGARCH models
  - Constant conditional correlation MGARCH model
  - Dynamic conditional correlation MGARCH model
  - Varying conditional correlation MGARCH model
- Error distributions and quasimaximum likelihood
- Treatment of missing data
An introduction to MGARCH models

Multivariate GARCH models allow the conditional covariance matrix of the dependent variables to follow a flexible dynamic structure and allow the conditional mean to follow a vector-autoregressive (VAR) structure.

The general MGARCH model is too flexible for most problems. There are many restricted MGARCH models in the literature because there is no parameterization that always provides an optimal trade-off between flexibility and parsimony.

`mgarch` implements four commonly used parameterizations: the diagonal vech (DVECH) model, the constant conditional correlation (CCC) model, the dynamic conditional correlation (DCC) model, and the time-varying conditional correlation (VCC) model.

Bollerslev, Engle, and Wooldridge (1988); Bollerslev, Engle, and Nelson (1994); Bauwens, Laurent, and Rombouts (2006); Silvennoinen and Teräsvirta (2009); and Engle (2009) provide general introductions to MGARCH models. We provide a quick introduction organized around the models implemented in `mgarch`.

We give a formal definition of the general MGARCH model to establish notation that facilitates comparisons of the models. The general MGARCH model is given by

\[
y_t = Cx_t + \epsilon_t
\]

\[
\epsilon_t = H_t^{1/2} \nu_t
\]

where

- \( y_t \) is an \( m \times 1 \) vector of dependent variables;
- \( C \) is an \( m \times k \) matrix of parameters;
- \( x_t \) is a \( k \times 1 \) vector of independent variables, which may contain lags of \( y_t \);
- \( H_t^{1/2} \) is the Cholesky factor of the time-varying conditional covariance matrix \( H_t \); and
- \( \nu_t \) is an \( m \times 1 \) vector of zero-mean, unit-variance, and independent and identically distributed innovations.

In the general MGARCH model, \( H_t \) is a matrix generalization of univariate GARCH models. For example, in a general MGARCH model with one autoregressive conditional heteroskedastic (ARCH) term and one GARCH term,

\[
\text{vech} (H_t) = s + A \text{vech} (\epsilon_{t-1} \epsilon_{t-1}') + B \text{vech} (H_{t-1})
\]  

where the \text{vech}() function stacks the unique elements that lie on or below the main diagonal in a symmetric matrix into a vector, \( s \) is a vector of parameters, and \( A \) and \( B \) are conformable matrices of parameters. Because this model uses the \text{vech}() function to extract and model the unique elements of \( H_t \), it is also known as the VECH model.

Because it is a conditional covariance matrix, \( H_t \) must be positive definite. Equation (1) can be used to show that the parameters in \( s \), \( A \), and \( B \) are not uniquely identified and that further restrictions must be placed on \( s \), \( A \), and \( B \) to ensure that \( H_t \) is positive definite for all \( t \).
The various MGARCH models proposed in the literature differ in how they trade off flexibility and parsimony in their specifications for $H_t$. Increased flexibility allows a model to capture more complex $H_t$ processes. Increased parsimony makes parameter estimation feasible for more datasets. An important measure of the flexibility–parsimony trade-off is how fast the number of model parameters increases with the number of time series $m$, because many applied models use multiple time series.

**Diagonal vech MGARCH models**

Bollerslev, Engle, and Wooldridge (1988) derived the diagonal vech (DVECH) model by restricting $A$ and $B$ to be diagonal. Although the DVECH model is much more parsimonious than the general model, it can only handle a few series because the number of parameters grows quadratically with the number of series. For example, there are $3m(m + 1)/2$ parameters in a DVECH(1,1) model for $H_t$.

Despite the large number of parameters, the diagonal structure implies that each conditional variance and each conditional covariance depends on its own past but not on the past of the other conditional variances and covariances. Formally, in the DVECH(1,1) model each element of $H_t$ is modeled by

$$h_{ij,t} = s_{ij} + a_{ij} \epsilon_{i,(t-1)} \epsilon_{j,(t-1)} + b_{ij} h_{ij,(t-1)}$$

Parameter estimation can be difficult because it requires that $H_t$ be positive definite for each $t$. The requirement that $H_t$ be positive definite for each $t$ imposes complicated restrictions on the off-diagonal elements.

See [TS] mgarch dvech for more details about this model.

**Conditional correlation MGARCH models**

Conditional correlation (CC) models use nonlinear combinations of univariate GARCH models to represent the conditional covariances. In each of the conditional correlation models, the conditional covariance matrix is positive definite by construction and has a simple structure, which facilitates parameter estimation. CC models have a slower parameter growth rate than DVECH models as the number of time series increases.

In CC models, $H_t$ is decomposed into a matrix of conditional correlations $R_t$ and a diagonal matrix of conditional variances $D_t$:

$$H_t = D_t^{1/2} R_t D_t^{1/2}$$

where each conditional variance follows a univariate GARCH process and the parameterizations of $R_t$ vary across models.

Equation (2) implies that

$$h_{ij,t} = \rho_{ij,t} \sigma_{i,t} \sigma_{j,t}$$

where $\sigma_{i,t}^2$ is modeled by a univariate GARCH process. Equation (3) highlights that CC models use nonlinear combinations of univariate GARCH models to represent the conditional covariances and that the parameters in the model for $\rho_{ij,t}$ describe the extent to which the errors from equations $i$ and $j$ move together.
Comparing (1) and (2) shows that the number of parameters increases more slowly with the number of time series in a CC model than in a DVECH model.

The three CC models implemented in mgarch differ in how they parameterize $R_t$.

**Constant conditional correlation MGARCH model**

Bollerslev (1990) proposed a CC MGARCH model in which the correlation matrix is time invariant. It is for this reason that the model is known as a constant conditional correlation (CCC) MGARCH model. Restricting $R_t$ to a constant matrix reduces the number of parameters and simplifies the estimation but may be too strict in many empirical applications.

See [TS] mgarch ccc for more details about this model.

**Dynamic conditional correlation MGARCH model**

Engle (2002) introduced a dynamic conditional correlation (DCC) MGARCH model in which the conditional quasicorrelations $R_t$ follow a GARCH(1,1)-like process. (As described by Engle [2009] and Aielli [2009], the parameters in $R_t$ are not standardized to be correlations and are thus known as quasicorrelations.) To preserve parsimony, all the conditional quasicorrelations are restricted to follow the same dynamics. The DCC model is significantly more flexible than the CCC model without introducing an unestimable number of parameters for a reasonable number of series.

See [TS] mgarch dcc for more details about this model.

**Varying conditional correlation MGARCH model**

Tse and Tsui (2002) derived the varying conditional correlation (VCC) MGARCH model in which the conditional correlations at each period are a weighted sum of a time-invariant component, a measure of recent correlations among the residuals, and last period’s conditional correlations. For parsimony, all the conditional correlations are restricted to follow the same dynamics.

See [TS] mgarch vcc for more details about this model.

**Error distributions and quasimaximum likelihood**

By default, mgarch dvech, mgarch ccc, mgarch dcc, and mgarch vcc estimate the parameters of MGARCH models by maximum likelihood (ML), assuming that the errors come from a multivariate normal distribution. Both the ML estimator and the quasi–maximum likelihood (QML) estimator, which drops the normality assumption, are assumed to be consistent and normally distributed in large samples; see Jeantheau (1998), Berkes and Horváth (2003), Comte and Lieberman (2003), Ling and McAleer (2003), and Fiorentini and Sentana (2007). Specify vce(robust) to estimate the parameters by QML. The QML parameter estimates are the same as the ML estimates, but the VCEs are different.

Based on low-level assumptions, Jeantheau (1998), Comte and Lieberman (2003), and Ling and McAleer (2003) prove that some of the ML and QML estimators implemented in mgarch are consistent and asymptotically normal. Based on higher-level assumptions, Fiorentini and Sentana (2007) prove that all the ML and QML estimators implemented in mgarch are consistent and asymptotically normal. The low-level assumption proofs specify the technical restrictions on the data-generating processes more precisely than the high-level proofs, but they do not cover as many models or cases as the high-level proofs.
It is generally accepted that there could be more low-level theoretical work done to substantiate the claims that the ML and QML estimators are consistent and asymptotically normally distributed. These widely applied estimators have been subjected to many Monte Carlo studies that show that the large-sample theory performs well in finite samples.

The `distribution(t)` option causes the `mgarch` commands to estimate the parameters of the corresponding model by ML assuming that the errors come from a multivariate Student $t$ distribution.

The choice between the multivariate normal and the multivariate $t$ distributions is one between robustness and efficiency. If the disturbances come from a multivariate Student $t$, then the ML estimates based on the multivariate Student $t$ assumption will be consistent and efficient, while the QML estimates based on the multivariate normal assumption will be consistent but not efficient. In contrast, if the disturbances come from a well-behaved distribution that is neither multivariate Student $t$ nor multivariate normal, then the ML estimates based on the multivariate Student $t$ assumption will not be consistent, while the QML estimates based on the multivariate normal assumption will be consistent but not efficient.

Fiorentini and Sentana (2007) compare the ML and QML estimators implemented in `mgarch` and provide many useful technical results pertaining to the estimators.

### Treatment of missing data

`mgarch` allows for gaps due to missing data. The unconditional expectations are substituted for the dynamic components that cannot be computed because of gaps. This method of handling gaps can only handle the case in which $g/T$ goes to zero as $T$ goes to infinity, where $g$ is the number of observations lost to gaps in the data and $T$ is the number of nonmissing observations.

### References


Also see

[TS] arch — Autoregressive conditional heteroskedasticity (ARCH) family of estimators

[TS] var — Vector autoregressive models

[U] 20 Estimation and postestimation commands
**mgarch ccc** — Constant conditional correlation multivariate GARCH models

### Description

`mgarch ccc` estimates the parameters of constant conditional correlation (CCC) multivariate generalized autoregressive conditionally heteroskedastic (MGARCH) models in which the conditional variances are modeled as univariate generalized autoregressive conditionally heteroskedastic (GARCH) models and the conditional covariances are modeled as nonlinear functions of the conditional variances. The conditional correlation parameters that weight the nonlinear combinations of the conditional variance are constant in the CCC MGARCH model.

The CCC MGARCH model is less flexible than the dynamic conditional correlation MGARCH model (see [TS] `mgarch dcc`) and varying conditional correlation MGARCH model (see [TS] `mgarch vcc`), which specify GARCH-like processes for the conditional correlations. The conditional correlation MGARCH models are more parsimonious than the diagonal vech MGARCH model (see [TS] `mgarch dvech`).

### Quick start

Fit constant conditional correlation multivariate GARCH with first- and second-order ARCH components for dependent variables `y1` and `y2` using `tsset` data

```
mgarch ccc (y1 y2), arch(1 2)
```

Add regressors `x1` and `x2` and first-order GARCH component

```
mgarch ccc (y1 y2 = x1 x2), arch(1 2) garch(1)
```

Add `z1` to the model for the conditional heteroskedasticity

```
mgarch ccc (y1 y2 = x1 x2), arch(1 2) garch(1) het(z1)
```

### Menu

Statistics > Multivariate time series > Multivariate GARCH
Syntax

```
mgarch ccc eq [ eq ... eq ] [ if ] [ in ] [ , options ]
```

where each `eq` has the form

```
(depvars = [ indepvars ] [ , eqoptions ])
```

### options Description

**Model**

- `arch(numlist)` ARCH terms for all equations
- `garch(numlist)` GARCH terms for all equations
- `het(varlist)` include `varlist` in the specification of the conditional variance for all equations
- `distribution(dist [#])` use `dist` distribution for errors [may be `gaussian` (synonym `normal`) or `t`; default is `gaussian`]
- `unconcentrated` perform optimization on unconcentrated log likelihood
- `constraints(numlist)` apply linear constraints

**SE/Robust**

- `vce(vcetype)` `vcetype` may be `oim` or `robust`

**Reporting**

- `level(#)` set confidence level; default is `level(95)`
- `nocnsreport` do not display constraints
- `display_options` control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling

**Maximization**

- `maximize_options` control the maximization process; seldom used
- `from(matname)` initial values for the coefficients; seldom used
- `coeflegend` display legend instead of statistics

**eqoptions** Description

- `noconstant` suppress constant term in the mean equation
- `arch(numlist)` ARCH terms
- `garch(numlist)` GARCH terms
- `het(varlist)` include `varlist` in the specification of the conditional variance

You must `tsset` your data before using `mgarch ccc`; see [TS] `tsset`.

`indepvars` and `varlist` may contain factor variables; see [U] 11.4.3 Factor variables.

`depvars`, `indepvars`, and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, fp, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.

`coeflegend` does not appear in the dialog box.

See [U] 20 *Estimation and postestimation commands* for more capabilities of estimation commands.
Options

**Model**

- `arch(numlist)` specifies the ARCH terms for all equations in the model. By default, no ARCH terms are specified.
- `garch(numlist)` specifies the GARCH terms for all equations in the model. By default, no GARCH terms are specified.
- `het(varlist)` specifies that varlist be included in the specification of the conditional variance for all equations. This varlist enters the variance specification collectively as multiplicative heteroskedasticity.
- `distribution(dist [#])` specifies the assumed distribution for the errors. `dist` may be `gaussian`, `normal`, or `t`. `gaussian` and `normal` are synonyms; each causes `mgarch ccc` to assume that the errors come from a multivariate normal distribution. `#` cannot be specified with either of them.
- `t` causes `mgarch ccc` to assume that the errors follow a multivariate Student t distribution, and the degree-of-freedom parameter is estimated along with the other parameters of the model. If `distribution(t #)` is specified, then `mgarch ccc` uses a multivariate Student t distribution with `#` degrees of freedom. `#` must be greater than 2.
- `unconcentrated` specifies that optimization be performed on the unconcentrated log likelihood. The default is to start with the concentrated log likelihood.
- `constraints(numlist)` specifies linear constraints to apply to the parameter estimates.

**SE/Robust**

- `vce(vcetype)` specifies the estimator for the variance–covariance matrix of the estimator.
  - `vce(oim)`, the default, specifies to use the observed information matrix (OIM) estimator.
  - `vce(robust)` specifies to use the Huber/White/sandwich estimator.

**Reporting**

- `level(#)`, `nocnsreport`; see [R] Estimation options.
- `display_options`: `noci`, `noplvalues`, `noolitted`, `v squash`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `f vwrap(#)`, `f vrap on(style)`, `cformat(%, fmt)`, `pformat(%, fmt)`, `sformat(%, fmt)`, and `nolstretch`; see [R] Estimation options.

**Maximization**

- `maximize_options`: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no] log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `nonrtolerance`, and `from(matname)`; see [R] Maximize for all options except `from()`, and see below for information on `from()`. These options are seldom used.
- `from(matname)` specifies initial values for the coefficients. `from(b0)` causes `mgarch ccc` to begin the optimization algorithm with the values in `b0`. `b0` must be a row vector, and the number of columns must equal the number of parameters in the model.

The following option is available with `mgarch ccc` but is not shown in the dialog box:

- `coeflegend`; see [R] Estimation options.
Eqoptions

noconstant suppresses the constant term in the mean equation.
arch(numlist) specifies the ARCH terms in the equation. By default, no ARCH terms are specified.
   This option may not be specified with model-level arch().
garch(numlist) specifies the GARCH terms in the equation. By default, no GARCH terms are specified.
   This option may not be specified with model-level garch().
het(varlist) specifies that varlist be included in the specification of the conditional variance. This
   varlist enters the variance specification collectively as multiplicative heteroskedasticity. This option
   may not be specified with model-level het().

Remarks and examples

We assume that you have already read [TS] mgarch, which provides an introduction to MGARCH
models and the methods implemented in mgarch ccc.

MGARCH models are dynamic multivariate regression models in which the conditional variances
and covariances of the errors follow an autoregressive-moving-average structure. The CCC MGARCH
model uses a nonlinear combination of univariate GARCH models in which the cross-equation weights
are time invariant to model the conditional covariance matrix of the disturbances.

As discussed in [TS] mgarch, MGARCH models differ in the parsimony and flexibility of their
specifications for a time-varying conditional covariance matrix of the disturbances, denoted by $H_t$.
In the conditional correlation family of MGARCH models, the diagonal elements of $H_t$ are modeled
as univariate GARCH models, whereas the off-diagonal elements are modeled as nonlinear functions
of the diagonal terms. In the CCC MGARCH model,

$$h_{ij,t} = \rho_{ij} \sqrt{h_{ii,t} h_{jj,t}}$$

where the diagonal elements $h_{ii,t}$ and $h_{jj,t}$ follow univariate GARCH processes and $\rho_{ij}$ is a time-
invariant weight interpreted as a conditional correlation.

In the dynamic conditional correlation (DCC) and varying conditional correlation (VCC) MGARCH
models discussed in [TS] mgarch dcc and [TS] mgarch vcc, the $\rho_{ij}$ are allowed to vary over time.
Although the conditional-correlation structure provides a useful trade-off between parsimony
and flexibility in the DCC MGARCH and VCC MGARCH models, the time-invariant parameterization
used in the CCC MGARCH model is generally viewed as too restrictive for many applications; see
Silvennoinen and Teräsvirta (2009). The baseline CCC MGARCH estimates are frequently compared
with DCC MGARCH and VCC MGARCH estimates.

Technical note

Formally, the CCC MGARCH model derived by Bollerslev (1990) can be written as

$$y_t = Cx_t + \epsilon_t$$
$$\epsilon_t = H_t^{1/2} \nu_t$$
$$H_t = D_t^{1/2} R D_t^{1/2}$$
where

\[ y_t \] is an \( m \times 1 \) vector of dependent variables;

\[ C \] is an \( m \times k \) matrix of parameters;

\[ x_t \] is a \( k \times 1 \) vector of independent variables, which may contain lags of \( y_t \);

\[ H_{t/2} \] is the Cholesky factor of the time-varying conditional covariance matrix \( H_t \);

\[ \nu_t \] is an \( m \times 1 \) vector of normal, independent, and identically distributed innovations;

\[ D_t \] is a diagonal matrix of conditional variances,

\[
D_t = \begin{pmatrix}
\sigma_{1,t}^2 & 0 & \cdots & 0 \\
0 & \sigma_{2,t}^2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_{m,t}^2
\end{pmatrix}
\]

in which each \( \sigma_{i,t}^2 \) evolves according to a univariate GARCH model of the form

\[
\sigma_{i,t}^2 = s_i + \sum_{j=1}^{p_i} \alpha_j \epsilon_{i,t-j}^2 + \sum_{j=1}^{q_i} \beta_j \sigma_{i,t-j}^2
\]

by default, or

\[
\sigma_{i,t}^2 = \exp(\gamma_t z_i,t) + \sum_{j=1}^{p_i} \alpha_j \epsilon_{i,t-j}^2 + \sum_{j=1}^{q_i} \beta_j \sigma_{i,t-j}^2
\]

when the \texttt{het()} option is specified, where \( \gamma_t \) is a \( 1 \times p \) vector of parameters, \( z_i \) is a \( p \times 1 \) vector of independent variables including a constant term, the \( \alpha_j \)'s are ARCH parameters, and the \( \beta_j \)'s are GARCH parameters; and

\[ R \] is a matrix of time-invariant unconditional correlations of the standardized residuals \( D_t^{-1/2} \epsilon_t \),

\[
R = \begin{pmatrix}
1 & \rho_{12} & \cdots & \rho_{1m} \\
\rho_{12} & 1 & \cdots & \rho_{2m} \\
\vdots & \vdots & \ddots & \vdots \\
\rho_{1m} & \rho_{2m} & \cdots & 1
\end{pmatrix}
\]

This model is known as the constant conditional correlation MGARCH model because \( R \) is time invariant.

Some examples

Example 1: Model with common covariates

We have daily data on the stock returns of three car manufacturers—Toyota, Nissan, and Honda, from January 2, 2003, to December 31, 2010—in the variables \texttt{toyota}, \texttt{nissan}, and \texttt{honda}. We model the conditional means of the returns as a first-order vector autoregressive process and the conditional covariances as a CCC MGARCH process in which the variance of each disturbance term follows a GARCH(1,1) process. We specify the \texttt{noconstant} option, because the returns have mean zero. The estimated constants in the variance equations are near zero in this example because of how the data are scaled.
. use https://www.stata-press.com/data/r16/stocks
    (Data from Yahoo! Finance)
. mgarch ccc (toyota nissan honda = L.toyota L.nissan L.honda, noconstant),
    > arch(1) garch(1)
Calculating starting values....
Optimizing concentrated log likelihood

(setting technique to bhhh)
Iteration 0:  log likelihood =  16898.994
Iteration 1:  log likelihood =  17008.914
Iteration 2:  log likelihood =  17156.946
Iteration 3:  log likelihood =  17249.527
Iteration 4:  log likelihood =  17287.251
Iteration 5:  log likelihood =  17313.5
Iteration 6:  log likelihood =  17335.087
Iteration 7:  log likelihood =  17356.534
Iteration 8:  log likelihood =  17376.051
Iteration 9:  log likelihood =  17400.035

(switching technique to nr)
Iteration 10: log likelihood =  17423.634
Iteration 11: log likelihood =  17440.261
Iteration 12: log likelihood =  17446.381
Iteration 13: log likelihood =  17447.614
Iteration 14: log likelihood =  17447.645
Iteration 15: log likelihood =  17447.645
Optimizing unconcentrated log likelihood
Iteration 0:  log likelihood =  17447.645
Iteration 1:  log likelihood =  17447.651
Iteration 2:  log likelihood =  17447.651

Constant conditional correlation MGARCH model
Sample: 2 - 2015          Number of obs  =        2,014
Distribution: Gaussian    Wald chi2(9)  =      17.46
Log likelihood =  17447.65  Prob > chi2   = 0.0420

                      Coef.  Std. Err.     z  P>|z|     [95% Conf. Interval]
  toyota
    toyota
L1.    -.0537817   .0353211   -1.52   0.128    -.1230098    .0154463
  nissan
L1.     .026686    .024841   1.07   0.283   -.0220015    .0753734
  honda
L1.    -.0043073   .0302761   -0.14   0.887    -.0636473    .0550327
ARCH_toyota
arch
L1.     .0615321   .0087313    7.05   0.000     .0444191    .0786452
garch
L1.     .9213798   .0110412   83.45   0.000     .8997395    .9430201
   _cons     4.42e-06   1.12e-06    3.93   0.000       2.21e-06    6.62e-06
The iteration log has three parts: the dots from the search for initial values, the iteration log from optimizing the concentrated log likelihood, and the iteration log from maximizing the unconcentrated log likelihood. A detailed discussion of the optimization methods can be found in Methods and formulas.

The header describes the estimation sample and reports a Wald test against the null hypothesis that all the coefficients on the independent variables in the mean equations are zero. Here the null hypothesis is rejected at the 5% level.

The output table first presents results for the mean or variance parameters used to model each dependent variable. Subsequently, the output table presents results for the conditional correlation parameters. For example, the conditional correlation between the standardized residuals for Toyota and Nissan is estimated to be 0.65.
The output above indicates that we may not need all the vector autoregressive parameters, but that each of the univariate ARCH, univariate GARCH, and conditional correlation parameters are statistically significant. That the estimated conditional correlation parameters are positive and significant indicates that the returns on these stocks rise or fall together.

That the conditional correlations are time invariant is a restrictive assumption. The DCC MGARCH model and the VCC MGARCH model nest the CCC MGARCH model. When we test the time-invariance assumption with Wald tests on the parameters of these more general models in [TS] mgarch dcc and [TS] mgarch vcc, we reject the null hypothesis that these conditional correlations are time invariant.

Example 2: Model with covariates that differ by equation

We improve the previous example by removing the insignificant parameters from the model. To remove these parameters, we specify the honda equation separately from the toyota and nissan equations:

```
. mgarch ccc (toyota nissan = , noconstant) (honda = L.nissan, noconstant),
> arch(1) garch(1)
```

```
Calculating starting values....
Optimizing concentrated log likelihood
(setting technique to bhhh)
Iteration 0:  log likelihood =  16886.88
Iteration 1:  log likelihood =  16974.779
Iteration 2:  log likelihood =  17147.893
Iteration 3:  log likelihood =  17247.473
Iteration 4:  log likelihood =  17285.549
Iteration 5:  log likelihood =  17311.153
Iteration 6:  log likelihood =  17333.588
Iteration 7:  log likelihood =  17353.717
Iteration 8:  log likelihood =  17374.895
Iteration 9:  log likelihood =  17400.669
(switching technique to nr)
Iteration 10: log likelihood =  17425.661
Iteration 11: log likelihood =  17436.789
Iteration 12: log likelihood =  17439.74
Iteration 13: log likelihood =  17439.865
Iteration 14: log likelihood =  17439.866
Optimizing unconcentrated log likelihood
Iteration 0:  log likelihood =  17439.865
Iteration 1:  log likelihood =  17439.872
Iteration 2:  log likelihood =  17439.872
```
Constant conditional correlation MGARCH model

|                | Coef.  | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|----------------|--------|-----------|------|------|---------------------|
| ARCH_toyota    |        |           |      |      |                     |
| arch L1.       | .0619604 | .0087942  | 7.05 | 0.000 | .044724 -.0791968  |
| garch L1.      | .9208961 | .0110995  | 82.97| 0.000 | .8991414 -.9426508 |
| _cons          | 4.43e-06 | 1.13e-06  | 3.94 | 0.000 | 2.23e-06 6.64e-06  |
| ARCH_nissan    |        |           |      |      |                     |
| arch L1.       | .0773095 | .012328   | 6.27 | 0.000 | .0531471 1.014719  |
| garch L1.      | .906088  | .0147303  | 61.51| 0.000 | .8772171  .9349589 |
| _cons          | 6.77e-06 | 1.85e-06  | 3.66 | 0.000 | 3.14e-06  .0000104 |
| honda          |        |           |      |      |                     |
| nissan L1.     | .0186628 | .0138575  | 1.35 | 0.178 | -.0084975 .0458231 |
| ARCH_honda     |        |           |      |      |                     |
| arch L1.       | .0433741 | .006996   | 6.20 | 0.000 | .0296622  .0570861 |
| garch L1.      | .9391094 | .0100707  | 93.25| 0.000 | .9193712  .9588476 |
| _cons          | 5.02e-06 | 1.31e-06  | 3.83 | 0.000 | 2.45e-06  .760e-06 |
| corr(toyota,   |        |           |      |      |                     |
| nissan)       | .652299  | .0128271  | 50.85| 0.000 | .6271583  .6774396 |
| corr(toyota,   |        |           |      |      |                     |
| honda)        | .7189531 | .0108005  | 66.57| 0.000 | .6977845  .7401218 |
| corr(nissan,   |        |           |      |      |                     |
| honda)        | .628435  | .0135653  | 46.33| 0.000 | .6018475  .6550225 |

It turns out that the coefficient on L1.nissan in the honda equation is now statistically insignificant. We could further improve the model by removing L1.nissan from the model.

As expected, removing the insignificant parameters from conditional mean equations had almost no effect on the estimated conditional variance parameters.

There is no mean equation for Toyota or Nissan. In [TS] mgarch ccc postestimation, we discuss prediction from models without covariates.
Example 3: Model with constraints

Here we fit a bivariate CCC MGARCH model for the Toyota and Nissan shares. We believe that the shares of these car manufacturers follow the same process, so we impose the constraints that the ARCH and the GARCH coefficients are the same for the two companies.

```
constraint 1 _b[ARCH_toyota:L.arch] = _b[ARCH_nissan:L.arch]
constraint 2 _b[ARCH_toyota:L.garch] = _b[ARCH_nissan:L.garch]
mgarch ccc (toyota nissan = , noconstant), arch(1) garch(1) constraints(1 2)
```

Calculating starting values....

Optimizing concentrated log likelihood

(setting technique to bhhh)
Iteration 0: log likelihood = 10317.225
Iteration 1: log likelihood = 10630.464
Iteration 2: log likelihood = 10865.964
Iteration 3: log likelihood = 11063.329
(output omitted)
Iteration 8: log likelihood = 11273.962
Iteration 9: log likelihood = 11274.409
(switching technique to nr)
Iteration 10: log likelihood = 11274.494
Iteration 11: log likelihood = 11274.499
Iteration 12: log likelihood = 11274.499

Optimizing unconcentrated log likelihood
Iteration 0: log likelihood = 11274.499
Iteration 1: log likelihood = 11274.501
Iteration 2: log likelihood = 11274.501

Constant conditional correlation MGARCH model

Sample: 1 - 2015                Number of obs = 2,015
Distribution: Gaussian          Wald chi2(.) = .
Log likelihood = 11274.5        Prob > chi2 = .
( 1) [ARCH_toyota]L.arch - [ARCH_nissan]L.arch = 0
( 2) [ARCH_toyota]L.garch - [ARCH_nissan]L.garch = 0

|                | Coef. | Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|----------------|-------|-----------|-------|-------|----------------------|
| ARCH_toyota    |       |           |       |       |                      |
| arch L1.       | .0742678 | .0095464 | 7.78  | 0.000 | .0555572 - .0929785 |
| garch L1.      | .9131674 | .0111558 | 81.86 | 0.000 | .8913024 - .9350323 |
| _cons          | 3.77e-06 | 1.02e-06 | 3.71  | 0.000 | 1.78e-06 - 5.77e-06 |
| ARCH_nissan    |       |           |       |       |                      |
| arch L1.       | .0742678 | .0095464 | 7.78  | 0.000 | .0555572 - .0929785 |
| garch L1.      | .9131674 | .0111558 | 81.86 | 0.000 | .8913024 - .9350323 |
| _cons          | 5.30e-06 | 1.36e-06 | 3.89  | 0.000 | 2.63e-06 - 7.97e-06 |
| corr(toyota,   |       |           |       |       |                      |
| nissan)        | .651389 | .0128482  | 50.70 | 0.000 | .6262071 - .6765709 |
We could test our constraints by fitting the unconstrained model and performing a likelihood-ratio test. The results indicate that the restricted model is preferable.

Example 4: Model with a GARCH term

In this example, we have data on fictional stock returns for the Acme and Anvil corporations and we believe that the movement of the two stocks is governed by different processes. We specify one ARCH and one GARCH term for the conditional variance equation for Acme and two ARCH terms for the conditional variance equation for Anvil. In addition, we include the lagged value of the stock return for Apex, the main subsidiary of Anvil corporation, in the variance equation of Anvil. For Acme, we have data on the changes in an index of futures prices of products related to those produced by Acme in a related. For Anvil, we have data on the changes in an index of futures prices of inputs used by Anvil in a related.
The results indicate that increases in the futures prices for related products lead to higher returns on the Acme stock, and increased input prices lead to lower returns on the Anvil stock. In the conditional variance equation for Anvil, the coefficient on \( L1.\\text{apex} \) is positive and significant, which indicates that an increase in the return on the Apex stock leads to more variability in the return on the Anvil stock. That the estimated conditional correlation between the two returns is \(-0.54\) indicates that these
returns tend to move in opposite directions; in other words, an increase in the return for the Acme stock tends to be associated with a decrease in the return for the Anvil stock, and vice versa.

Stored results

```
mgarch ccc stores the following in e():
```

Scalars

- `e(N)` number of observations
- `e(k)` number of parameters
- `e(k_extra)` number of extra estimates added to _b
- `e(k_eq)` number of equations in e(b)
- `e(k_dv)` number of dependent variables
- `e(dv_eq)` model degrees of freedom
- `e(ll)` log likelihood
- `e(chi2)` $\chi^2$
- `e(p)` $p$-value for model test
- `e(estdf)` 1 if distribution parameter was estimated, 0 otherwise
- `e(usrd)` user-provided distribution parameter
- `e(tmin)` minimum time in sample
- `e(tmax)` maximum time in sample
- `e(N_gaps)` number of gaps
- `e(rank)` rank of e(V)
- `e(rc)` return code
- `e(converged)` 1 if converged, 0 otherwise

Macros

- `e(cmd)` mgarch
- `e(model)` ccc
- `e(cmdline)` command as typed
- `e(depvar)` names of dependent variables
- `e(covariates)` list of covariates
- `e(dv_eqs)` dependent variables with mean equations
- `e(indeps)` independent variables in each equation
- `e(tvar)` time variable
- `e(hetvars)` variables included in the conditional variance equations
- `e(title)` title in estimation output
- `e(chi2type)` Wald; type of model $\chi^2$ test
- `e(vce)` vcetype specified in vce()
- `e(vcetype)` title used to label Std. Err.
- `e(tmins)` formatted minimum time
- `e(tmaxs)` formatted maximum time
- `e(dist)` distribution for error term: gaussian or t
- `e(arch)` specified ARCH terms
- `e(garch)` specified GARCH terms
- `e(technique)` maximization technique
- `e(properties)` b V
- `e(estat_cmd)` program used to implement estat
- `e(predict)` program used to implement predict
- `e(marginsok)` predictions allowed by margins
- `e(marginsnotok)` predictions disallowed by margins
- `e(marginsdefault)` default predict() specification for margins
- `e(asbalanced)` factor variables fvset as asbalanced
- `e(asobserved)` factor variables fvset as asobserved
Methods and formulas

`mgarch ccc` estimates the parameters of the CCC MGARCH model by maximum likelihood. The unconcentrated log-likelihood function based on the multivariate normal distribution for observation $t$ is

$$l_t = -0.5m \log(2\pi) - 0.5\log \{\det (R)\} - \log \left\{\det \left(D_t^{1/2}\right)\right\} - 0.5\tilde{\epsilon}_t R^{-1}\tilde{\epsilon}_t'$$  \hspace{1cm} (1)

where $\tilde{\epsilon}_t = D_t^{-1/2}\epsilon_t$ is an $m \times 1$ vector of standardized residuals, $\epsilon_t = y_t - Cx_t$. The log-likelihood function is $\sum_{t=1}^{T} l_t$.

If we assume that $\nu_t$ follow a multivariate $t$ distribution with degrees of freedom (df) greater than 2, then the unconcentrated log-likelihood function for observation $t$ is

$$l_t = \log \Gamma \left(\frac{df + m}{2}\right) - \log \Gamma \left(\frac{df}{2}\right) - \frac{m}{2} \log \{(df - 2)\pi\} - 0.5\log \{\det (R)\} - \log \left\{\det \left(D_t^{1/2}\right)\right\} - \frac{df + m}{2} \log \left(1 + \frac{\tilde{\epsilon}_t R^{-1}\tilde{\epsilon}_t'}{df - 2}\right)$$  \hspace{1cm} (2)

The correlation matrix $R$ can be concentrated out of (1) and (2) by defining the $(i, j)$th element of $R$ as

$$\hat{\rho}_{ij} = \left(\sum_{t=1}^{T} \tilde{\epsilon}_{it}\tilde{\epsilon}_{jt}\right) \left(\sum_{t=1}^{T} \tilde{\epsilon}_{it}^2\right)^{-\frac{1}{2}} \left(\sum_{t=1}^{T} \tilde{\epsilon}_{jt}^2\right)^{-\frac{1}{2}}$$

`mgarch ccc` starts the optimization process with the concentrated log-likelihood function.

The starting values for the parameters in the mean equations and the initial residuals $\hat{\epsilon}_t$ are obtained by least-squares regression. The starting values for the parameters in the variance equations are obtained by a procedure proposed by Gourieroux and Monfort (1997, sec. 6.2.2). If the optimization is started with the unconcentrated log likelihood, then the initial values for the parameters in $R$ are calculated from the standardized residuals $\tilde{\epsilon}_t$.
GARCH estimators require initial values that can be plugged in for $\epsilon_{t-i}\epsilon'_{t-i}$ and $H_{t-j}$ when $t-i < 1$ and $t-j < 1$. `mgarch ccc` substitutes an estimator of the unconditional covariance of the disturbances

$$\hat{\Sigma} = T^{-1}\sum_{t=1}^{T}\hat{\epsilon}_t\epsilon'_t$$  \hspace{1cm} (3)

for $\epsilon_{t-i}\epsilon'_{t-i}$ when $t-i < 1$ and for $H_{t-j}$ when $t-j < 1$, where $\hat{\epsilon}_t$ is the vector of residuals calculated using the estimated parameters.

`mgarch ccc` requires a sample size that at the minimum is equal to the number of parameters in the model plus twice the number of equations.

`mgarch ccc` uses numerical derivatives in maximizing the log-likelihood function.

**References**


**Also see**

[Ts] `mgarch ccc postestimation` — Postestimation tools for `mgarch ccc`

[Ts] `arch` — Autoregressive conditional heteroskedasticity (ARCH) family of estimators

[Ts] `mgarch` — Multivariate GARCH models

[Ts] `tsset` — Declare data to be time-series data

[Ts] `var` — Vector autoregressive models

[U] 20 Estimation and postestimation commands
Postestimation commands

The following standard postestimation commands are available after `mgarch ccc`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>contrast</strong></td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td><strong>estat ic</strong></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><strong>estat summarize</strong></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><strong>estat vce</strong></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><strong>estimates</strong></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><strong>forecast</strong></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><strong>lincom</strong></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><strong>lrtest</strong></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><strong>margins</strong></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><strong>marginsplot</strong></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><strong>nlcom</strong></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><strong>predict</strong></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><strong>predictnl</strong></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><strong>pwcompare</strong></td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td><strong>test</strong></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><strong>testnl</strong></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and conditional variances, covariances, and correlations. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] { stub* | newvarlist } [ if ] [ in ] [ , statistic options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
<tr>
<td>variance</td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td>correlation</td>
<td>conditional correlations</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equation(eqnames)</td>
<td>names of equations for which predictions are made</td>
</tr>
<tr>
<td>dynamic(time_constant)</td>
<td>begin dynamic forecast at specified time</td>
</tr>
</tbody>
</table>

Options for predict

### Main

- **xb**, the default, calculates the linear predictions of the dependent variables.
- **residuals** calculates the residuals.
- **variance** predicts the conditional variances and conditional covariances.
- **correlation** predicts the conditional correlations.

### Options

- **equation(eqnames)** specifies the equation for which the predictions are calculated. Use this option to predict a statistic for a particular equation. Equation names, such as `equation(income)`, are used to identify equations.
One equation name may be specified when predicting the dependent variable, the residuals, or the conditional variance. For example, specifying `equation(income)` causes `predict` to predict `income`, and specifying `variance equation(income)` causes `predict` to predict the conditional variance of `income`.

Two equations may be specified when predicting a conditional variance or covariance. For example, specifying `equation(income, consumption) variance` causes `predict` to predict the conditional covariance of `income` and `consumption`.

dynamic(`time_constant`) specifies when `predict` starts producing dynamic forecasts. The specified `time_constant` must be in the scale of the time variable specified in `tsset`, and the `time_constant` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. dynamic() may not be specified with residuals.

**margins**

**Description for margins**

`margins` estimates margins of response for linear predictions and conditional variances, covariances, and correlations. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

**Menu for margins**

Statistics > Postestimation

**Syntax for margins**

```plaintext
margins [marginlist] [, options]
margins [marginlist], predict(statistic ...) predict(statistic ...) [options]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td><code>xb</code></td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td><code>variance</code></td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td><code>correlation</code></td>
<td>conditional correlations</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>not allowed with <code>margins</code></td>
</tr>
</tbody>
</table>

`xb` defaults to the first equation.

Statistics not allowed with `margins` are functions of stochastic quantities other than `e(b)`.

For the full syntax, see [R] margins.
Remarks and examples

We assume that you have already read [TS] mgarch ccc. In this entry, we use predict after mgarch ccc to make in-sample and out-of-sample forecasts.

Example 1: Dynamic forecasts

In this example, we obtain dynamic forecasts for the Toyota, Nissan, and Honda stock returns modeled in example 2 of [TS] mgarch ccc. In the output below, we reestimate the parameters of the model, use tsappend (see [TS] tsappend) to extend the data, and use predict to obtain in-sample one-step-ahead forecasts and dynamic forecasts of the conditional variances of the returns. We graph the forecasts below.

```
. use https://www.stata-press.com/data/r16/stocks
   (Data from Yahoo! Finance)
. quietly mgarch ccc (toyota nissan = , noconstant)
   > (honda = L.nissan, noconstant), arch(1) garch(1)
. tsappend, add(50)
. predict H*, variance dynamic(2016)
```

Recent in-sample one-step-ahead forecasts are plotted to the left of the vertical line in the above graph, and the dynamic out-of-sample forecasts appear to the right of the vertical line. The graph shows the tail end of the huge increase in return volatility that took place in 2008 and 2009. It also shows that the dynamic forecasts quickly converge.

Methods and formulas

All one-step predictions are obtained by substituting the parameter estimates into the model. The estimated unconditional variance matrix of the disturbances, \( \hat{\Sigma} \), is the initial value for the ARCH and GARCH terms. The postestimation routines recompute \( \hat{\Sigma} \) using the prediction sample, the parameter estimates stored in \( e(b) \), and (3) in Methods and formulas of [TS] mgarch ccc.

For observations in which the residuals are missing, the estimated unconditional variance matrix of the disturbances is used in place of the outer product of the residuals.
Dynamic predictions of the dependent variables use previously predicted values beginning in the period specified by `dynamic()`.

Dynamic variance predictions are implemented by substituting $\hat{\Sigma}$ for the outer product of the residuals beginning in the period specified in `dynamic()`.

Also see

[TS] `mgarch ccc` — Constant conditional correlation multivariate GARCH models

[U] 20 `Estimation and postestimation commands`
**Description**

`mgarch dcc` estimates the parameters of dynamic conditional correlation (DCC) multivariate generalized autoregressive conditionally heteroskedastic (MGARCH) models in which the conditional variances are modeled as univariate generalized autoregressive conditionally heteroskedastic (GARCH) models and the conditional covariances are modeled as nonlinear functions of the conditional variances. The conditional quasicorrelation parameters that weight the nonlinear combinations of the conditional variances follow the GARCH-like process specified in Engle (2002).

The DCC MGARCH model is about as flexible as the closely related varying conditional correlation MGARCH model (see [TS] `mgarch vcc`), more flexible than the conditional correlation MGARCH model (see [TS] `mgarch ccc`), and more parsimonious than the diagonal vech MGARCH model (see [TS] `mgarch dvech`).

**Quick start**

Fit dynamic conditional correlation multivariate GARCH with first- and second-order ARCH components for dependent variables `y1` and `y2` using `tsset` data

```
mgarch dcc (y1 y2), arch(1 2)
```

Add regressors `x1` and `x2` and first-order GARCH component

```
mgarch dcc (y1 y2 = x1 x2), arch(1 2) garch(1)
```

Add `z1` to the model for the conditional heteroskedasticity

```
mgarch dcc (y1 y2 = x1 x2), arch(1 2) garch(1) het(z1)
```
Syntax

\texttt{mgarch dcc eq [ eq ... eq ] [ if ] [ in ] [ , options ]}

where each \textit{eq} has the form

\[(\textit{depvars} = [ \textit{indepvars} ] [ , \textit{eqoptions} ])\]

\begin{tabular}{ll}
\textit{options} & Description \\
\hline
\texttt{arch(numlist)} & ARCH terms for all equations \\
\texttt{garch(numlist)} & GARCH terms for all equations \\
\texttt{het(varlist)} & include \textit{varlist} in the specification of the conditional variance for all equations \\
\texttt{distribution(dist [#])} & use \textit{dist} distribution for errors [may be \texttt{gaussian} (synonym \texttt{normal}) or \texttt{t}; default is \texttt{gaussian}] \\
\texttt{constraints(numlist)} & apply linear constraints \\
\texttt{vce(vcetype)} & \textit{vcetype} may be \texttt{oim} or \texttt{robust} \\
\texttt{level(#)} & set confidence level; default is \texttt{level(95)} \\
\texttt{nocons} & suppress constant term in the mean equation \\
\texttt{arch(numlist)} & ARCH terms \\
\texttt{garch(numlist)} & GARCH terms \\
\texttt{het(varlist)} & include \textit{varlist} in the specification of the conditional variance \\
\end{tabular}

You must \texttt{tsset} your data before using \texttt{mgarch dcc}; see \cite{tsset}. \textit{depvars} and \textit{varlist} may contain factor variables; see \cite{factor}. \textit{depvars}, \textit{indepvars}, and \textit{varlist} may contain time-series operators; see \cite{time-series}. \textit{by}, \texttt{fp}, \texttt{rolling}, and \texttt{statsby} are allowed; see \cite{prefix}. \texttt{coeflegend} does not appear in the dialog box.

See \cite{estimation} for more capabilities of estimation commands.
Options

arch(numlist) specifies the ARCH terms for all equations in the model. By default, no ARCH terms are specified.

garch(numlist) specifies the GARCH terms for all equations in the model. By default, no GARCH terms are specified.

het(varlist) specifies that varlist be included in the specification of the conditional variance for all equations. This varlist enters the variance specification collectively as multiplicative heteroskedasticity.

distribution(dist [#]) specifies the assumed distribution for the errors. dist may be gaussian, normal, or t.

  gaussian and normal are synonyms; each causes mgarch dcc to assume that the errors come from a multivariate normal distribution. # may not be specified with either of them.

  t causes mgarch dcc to assume that the errors follow a multivariate Student t distribution, and the degree-of-freedom parameter is estimated along with the other parameters of the model. If distribution(t #) is specified, then mgarch dcc uses a multivariate Student t distribution with # degrees of freedom. # must be greater than 2.

constraints(numlist) specifies linear constraints to apply to the parameter estimates.

vce(vcetype) specifies the estimator for the variance–covariance matrix of the estimator.

  vce(oim), the default, specifies to use the observed information matrix (OIM) estimator.
  vce(robust) specifies to use the Huber/White/sandwich estimator.

level(#), nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels,nofvlabel, fvwrap(#), fvwrapon(style), cformat(%fmt), pformat(%fmt), sformat(%fmt), and nolstretch; see [R] Estimation options.

maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), nonrtolerance, and from(matname); see [R] Maximize for all options except from(), and see below for information on from(). These options are seldom used.

from(matname) specifies initial values for the coefficients. from(b0) causes mgarch dcc to begin the optimization algorithm with the values in b0. b0 must be a row vector, and the number of columns must equal the number of parameters in the model.

The following option is available with mgarch dcc but is not shown in the dialog box:

coefflegend; see [R] Estimation options.
Eq.options

noconstant suppresses the constant term in the mean equation.

arch(numlist) specifies the ARCH terms in the equation. By default, no ARCH terms are specified. This option may not be specified with model-level arch().

garch(numlist) specifies the GARCH terms in the equation. By default, no GARCH terms are specified. This option may not be specified with model-level garch().

het(varlist) specifies that varlist be included in the specification of the conditional variance. This varlist enters the variance specification collectively as multiplicative heteroskedasticity. This option may not be specified with model-level het().

Remarks and examples

We assume that you have already read [TS] mgarch, which provides an introduction to MGARCH models and the methods implemented in mgarch dcc.

MGARCH models are dynamic multivariate regression models in which the conditional variances and covariances of the errors follow an autoregressive-moving-average structure. The DCC MGARCH model uses a nonlinear combination of univariate GARCH models with time-varying cross-equation weights to model the conditional covariance matrix of the errors.

As discussed in [TS] mgarch, MGARCH models differ in the parsimony and flexibility of their specifications for a time-varying conditional covariance matrix of the disturbances, denoted by $H_t$. In the conditional correlation family of MGARCH models, the diagonal elements of $H_t$ are modeled as univariate GARCH models, whereas the off-diagonal elements are modeled as nonlinear functions of the diagonal terms. In the DCC MGARCH model,

$$h_{i,j,t} = \rho_{i,j,t} \sqrt{h_{i,i,t} h_{j,j,t}}$$

where the diagonal elements $h_{i,i,t}$ and $h_{j,j,t}$ follow univariate GARCH processes and $\rho_{i,j,t}$ follows the dynamic process specified in Engle (2002) and discussed below.

Because the $\rho_{i,j,t}$ varies with time, this model is known as the DCC GARCH model.

Technical note

The DCC GARCH model proposed by Engle (2002) can be written as

$$y_t = Cx_t + \epsilon_t$$
$$\epsilon_t = H_t^{1/2} \nu_t$$
$$H_t = D_t^{1/2} R_t D_t^{1/2}$$
$$R_t = \text{diag}(Q_t)^{-1/2} Q_t \text{diag}(Q_t)^{-1/2}$$
$$Q_t = (1 - \lambda_1 - \lambda_2) R + \lambda_1 \tilde{\epsilon}_{t-1} \tilde{\epsilon}_{t-1}' + \lambda_2 Q_{t-1}$$

(1)

where

$y_t$ is an $m \times 1$ vector of dependent variables;

$C$ is an $m \times k$ matrix of parameters;

$x_t$ is a $k \times 1$ vector of independent variables, which may contain lags of $y_t$;
\( H_t^{1/2} \) is the Cholesky factor of the time-varying conditional covariance matrix \( H_t \);

\( \nu_t \) is an \( m \times 1 \) vector of normal, independent, and identically distributed innovations;

\( D_t \) is a diagonal matrix of conditional variances,

\[
D_t = \begin{pmatrix}
\sigma_{1,t}^2 & 0 & \cdots & 0 \\
0 & \sigma_{2,t}^2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_{m,t}^2 \\
\end{pmatrix}
\]

in which each \( \sigma_{i,t}^2 \) evolves according to a univariate GARCH model of the form

\[
\sigma_{i,t}^2 = s_i + \sum_{j=1}^{p_i} \alpha_j \epsilon_{i,t-j}^2 + \sum_{j=1}^{q_i} \beta_j \sigma_{i,t-j}^2
\]

by default, or

\[
\sigma_{i,t}^2 = \exp(\gamma_t z_{i,t}) + \sum_{j=1}^{p_i} \alpha_j \epsilon_{i,t-j}^2 + \sum_{j=1}^{q_i} \beta_j \sigma_{i,t-j}^2
\]

when the \texttt{het()} option is specified, where \( \gamma_t \) is a \( 1 \times p \) vector of parameters, \( z_t \) is a \( p \times 1 \) vector of independent variables including a constant term, the \( \alpha_j \)'s are ARCH parameters, and the \( \beta_j \)'s are GARCH parameters;

\( R_t \) is a matrix of conditional quasicorrelations,

\[
R_t = \begin{pmatrix}
1 & \rho_{12,t} & \cdots & \rho_{1m,t} \\
\rho_{12,t} & 1 & \cdots & \rho_{2m,t} \\
\vdots & \vdots & \ddots & \vdots \\
\rho_{1m,t} & \rho_{2m,t} & \cdots & 1 \\
\end{pmatrix}
\]

\( \tilde{\epsilon}_t \) is an \( m \times 1 \) vector of standardized residuals, \( D_t^{-1/2} \epsilon_t \); and

\( \lambda_1 \) and \( \lambda_2 \) are parameters that govern the dynamics of conditional quasicorrelations. \( \lambda_1 \) and \( \lambda_2 \) are nonnegative and satisfy \( 0 \leq \lambda_1 + \lambda_2 < 1 \).

When \( Q_t \) is stationary, the \( R \) matrix in (1) is a weighted average of the unconditional covariance matrix of the standardized residuals \( \tilde{\epsilon}_t \), denoted by \( \overline{R} \), and the unconditional mean of \( Q_t \), denoted by \( \overline{Q} \). Because \( \overline{R} \neq \overline{Q} \), as shown by Aielli (2009), \( R \) is neither the unconditional correlation matrix nor the unconditional mean of \( Q_t \). For this reason, the parameters in \( R \) are known as quasicorrelations; see Aielli (2009) and Engle (2009) for discussions.

Some examples

Example 1: Model with common covariates

We have daily data on the stock returns of three car manufacturers—Toyota, Nissan, and Honda, from January 2, 2003, to December 31, 2010—in the variables \texttt{toyota}, \texttt{nissan} and \texttt{honda}. We model the conditional means of the returns as a first-order vector autoregressive process and the conditional covariances as a DCC MGARCH process in which the variance of each disturbance term follows a GARCH(1,1) process.
. use https://www.stata-press.com/data/r16/stocks
(Data from Yahoo! Finance)
. mgarch dcc (toyota nissan honda = L.toyota L.nissan L.honda, noconstant),>
> arch(1) garch(1)
Calculating starting values....
Optimizing log likelihood
(setting technique to bhhh)
Iteration 0:  log likelihood =  16902.435
Iteration 1:  log likelihood =  17005.448
Iteration 2:  log likelihood =  17157.958
Iteration 3:  log likelihood =  17267.363
Iteration 4:  log likelihood =  17318.29
Iteration 5:  log likelihood =  17353.029
Iteration 6:  log likelihood =  17369.115
Iteration 7:  log likelihood =  17388.035
Iteration 8:  log likelihood =  17401.254
Iteration 9:  log likelihood =  17435.556
(switching technique to nr)
Iteration 10: log likelihood =  17451.739
Iteration 11: log likelihood =  17476.882
Iteration 12: log likelihood =  17478.382
Iteration 13: log likelihood =  17483.858
Iteration 14: log likelihood =  17484.886
Iteration 15: log likelihood =  17484.95
Iteration 16: log likelihood =  17484.95
Refining estimates
Iteration 0:  log likelihood =  17484.95
Iteration 1:  log likelihood =  17484.95
Dynamic conditional correlation MGARCH model
Sample:  2 - 2015  Number of obs      =       2,014
Distribution: Gaussian  Wald chi2(9)  =      19.54
Log likelihood =  17484.95  Prob > chi2   =    0.0210

<p>|                | Coef.  | Std. Err. |      z    |   P&gt;|z|   |   [95% Conf. Interval]   |
|----------------|--------|-----------|-----------|--------|-------------------------|
| toyota         |        |           |           |        |                         |
| toyota         | -.0510867 |  .0339825 |  -1.50    |  0.133 |  -.1176911 -.0155177   |
| L1.            |        |           |           |        |                         |
| nissan         | .0297829 |  .0247455 |  1.20     |  0.229 |  -.0187173 .0782832    |
| L1.            |        |           |           |        |                         |
| honda          | -.0162824 |  .0300323 |  -0.54    |  0.588 |  -.0751447 .0425799    |
| L1.            |        |           |           |        |                         |
| ARCH_toyota    |        |           |           |        |                         |
| arch           | .0608223 |  .0086687 |   7.02    |  0.000 |   .043832 .0778127     |
| L1.            |        |           |           |        |                         |
| garch          | .9222203 |  .0111055 |  83.04    |  0.000 |   .9004539 .9439866    |
| L1.            |        |           |           |        |                         |
| _cons          |  4.47e-06 |  1.15e-06 |    3.90   |  0.000 |   2.22e-06 6.72e-06    |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nissan</td>
<td>toyota</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td>-.0056722</td>
<td>.0389348</td>
<td>-.15</td>
<td>0.884</td>
<td>-.0819829</td>
<td>.0706386</td>
</tr>
<tr>
<td></td>
<td>nissan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td>-.0287097</td>
<td>.0309379</td>
<td>-.93</td>
<td>0.353</td>
<td>-.0893468</td>
<td>.0319275</td>
</tr>
<tr>
<td></td>
<td>honda</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td>.015498</td>
<td>.0358802</td>
<td>0.43</td>
<td>0.666</td>
<td>-.0548259</td>
<td>.0858218</td>
</tr>
<tr>
<td></td>
<td>ARCH_nissan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arch</td>
<td>L1.</td>
<td>.0844244</td>
<td>.0128192</td>
<td>6.59</td>
<td>0.000</td>
<td>.0592992</td>
</tr>
<tr>
<td>garch</td>
<td>L1.</td>
<td>.89942</td>
<td>.0151125</td>
<td>59.51</td>
<td>0.000</td>
<td>.8698</td>
</tr>
<tr>
<td>_cons</td>
<td></td>
<td>7.21e-06</td>
<td>1.93e-06</td>
<td>3.74</td>
<td>0.000</td>
<td>3.43e-06</td>
</tr>
<tr>
<td></td>
<td>honda</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>toyota</td>
<td>L1.</td>
<td>-.0272415</td>
<td>.0361819</td>
<td>-.75</td>
<td>0.452</td>
<td>-.0981566</td>
</tr>
<tr>
<td>nissan</td>
<td>L1.</td>
<td>.0617491</td>
<td>.0271378</td>
<td>2.28</td>
<td>0.023</td>
<td>.0085599</td>
</tr>
<tr>
<td></td>
<td>honda</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td>-.063507</td>
<td>.0332918</td>
<td>-1.91</td>
<td>0.056</td>
<td>-.1287578</td>
<td>.0017437</td>
</tr>
<tr>
<td></td>
<td>ARCH_honda</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arch</td>
<td>L1.</td>
<td>.0490134</td>
<td>.0073695</td>
<td>6.65</td>
<td>0.000</td>
<td>.0345693</td>
</tr>
<tr>
<td>garch</td>
<td>L1.</td>
<td>.9331125</td>
<td>.0103686</td>
<td>89.99</td>
<td>0.000</td>
<td>.9127905</td>
</tr>
<tr>
<td>_cons</td>
<td></td>
<td>5.35e-06</td>
<td>1.35e-06</td>
<td>3.95</td>
<td>0.000</td>
<td>2.69e-06</td>
</tr>
<tr>
<td>corr(toyota, nissan)</td>
<td></td>
<td>.6689537</td>
<td>.0168019</td>
<td>39.81</td>
<td>0.000</td>
<td>.6360226</td>
</tr>
<tr>
<td>corr(toyota, honda)</td>
<td></td>
<td>.7259623</td>
<td>.0140155</td>
<td>51.80</td>
<td>0.000</td>
<td>.6984925</td>
</tr>
<tr>
<td>corr(nissan, honda)</td>
<td></td>
<td>.6335651</td>
<td>.0180409</td>
<td>35.12</td>
<td>0.000</td>
<td>.5982056</td>
</tr>
<tr>
<td>/Adjustment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lambda1</td>
<td></td>
<td>.0315281</td>
<td>.0088382</td>
<td>3.57</td>
<td>0.000</td>
<td>.0142054</td>
</tr>
<tr>
<td>lambda2</td>
<td></td>
<td>.8704093</td>
<td>.0613336</td>
<td>14.19</td>
<td>0.000</td>
<td>.7501977</td>
</tr>
</tbody>
</table>

The iteration log has three parts: the dots from the search for initial values, the iteration log from optimizing the log likelihood, and the iteration log from the refining step. A detailed discussion of the optimization methods is in Methods and formulas.

The header describes the estimation sample and reports a Wald test against the null hypothesis that all the coefficients on the independent variables in the mean equations are zero. Here the null hypothesis is rejected at the 5% level.

The output table first presents results for the mean or variance parameters used to model each dependent variable. Subsequently, the output table presents results for the conditional quasicorrelations.
For example, the conditional quasicorrelation between the standardized residuals for Toyota and Nissan is estimated to be 0.67. Finally, the output table presents results for the adjustment parameters $\lambda_1$ and $\lambda_2$. In the example at hand, the estimates for both $\lambda_1$ and $\lambda_2$ are statistically significant.

The DCC MGARCH model reduces to the CCC MGARCH model when $\lambda_1 = \lambda_2 = 0$. The output below shows that a Wald test rejects the null hypothesis that $\lambda_1 = \lambda_2 = 0$ at all conventional levels.

```
. test _b[/Adjustment:lambda1] = _b[/Adjustment:lambda2] = 0
( 1) [Adjustment]lambda1 - [Adjustment]lambda2 = 0
( 2) [Adjustment]lambda1 = 0
   chi2(  2) = 1102.27
   Prob > chi2 =    0.0000
```

These results indicate that the assumption of time-invariant conditional correlations maintained in the CCC MGARCH model is too restrictive for these data.

### Example 2: Model with covariates that differ by equation

We improve the previous example by removing the insignificant parameters from the model. To remove these parameters, we specify the `honda` equation separately from the `toyota` and `nissan` equations:

```
. mgarch dcc (toyota nissan = , noconstant) (honda = L.nissan, noconstant),
    > arch(1) garch(1)
```

Calculating starting values....

Optimizing log likelihood

(setting technique to bhhh)

Iteration 0:  log likelihood =  16884.502
Iteration 1:  log likelihood =  16970.755
Iteration 2:  log likelihood =  17140.318
Iteration 3:  log likelihood =  17237.807
Iteration 4:  log likelihood =  17306.120
Iteration 5:  log likelihood =  17342.533
Iteration 6:  log likelihood =  17392.501
Iteration 7:  log likelihood =  17407.242
Iteration 8:  log likelihood =  17448.702
(switching technique to nr)
Iteration 10: log likelihood =  17472.199
Iteration 11: log likelihood =  17475.842
Iteration 12: log likelihood =  17476.345
Iteration 13: log likelihood =  17476.350
Iteration 14: log likelihood =  17476.350

Refining estimates
Iteration 0: log likelihood =  17476.350
Iteration 1: log likelihood =  17476.350
Dynamic conditional correlation MGARCH model

Sample: 2 - 2015    Number of obs = 2,014
Distribution: Gaussian   Wald chi2(1) = 2.21
Log likelihood = 17476.35  Prob > chi2 = 0.1374

|                      | Coef.  | Std. Err. |   z  | P>|z|  | [95% Conf. Interval] |
|----------------------|--------|-----------|------|------|----------------------|
| ARCH_toyota          |        |           |      |      |                      |
| arch L1.             | .0608188 | .0086675 | 7.02 | 0.000 | .0438308 - .0778067  |
| garch L1.            | .9219957 | .0111066 | 83.01| 0.000 | .9002271 - .9437643  |
| _cons                | 4.49e-06 | 1.14e-06 | 3.95 | 0.000 | 2.27e-06 - 6.72e-06  |
| ARCH_nissan          |        |           |      |      |                      |
| arch L1.             | .0876161 | .01302   | 6.73 | 0.000 | .0620974 - .1131349  |
| garch L1.            | .8950964 | .0152908 | 58.54| 0.000 | .865127 - .9250658   |
| _cons                | 7.69e-06 | 1.99e-06 | 3.86 | 0.000 | 3.79e-06 - .0000116  |
| honda                |        |           |      |      |                      |
| nissan L1.           | .019978  | .0134488 | 1.49 | 0.137 | -.0063811 - .0463371 |
| ARCH_honda           |        |           |      |      |                      |
| arch L1.             | .0488799 | .0073767 | 6.63 | 0.000 | .0344218 - .063338   |
| garch L1.            | .9330047 | .0103944 | 89.76| 0.000 | .912632 - .9533774   |
| _cons                | 5.42e-06 | 1.36e-06 | 3.98 | 0.000 | 2.75e-06 - 8.08e-06  |
| corr(toyota, nissan) | .6668433 | .0163209 | 40.86| 0.000 | .6348548 - .6988317  |
| corr(toyota, honda)  | .7258101 | .0137072 | 52.95| 0.000 | .6989446 - .7526757  |
| corr(nissan, honda)  | .6313515 | .0175454 | 35.98| 0.000 | .5969631 - .6657399  |
| /Adjustment          |        |           |      |      |                      |
| lambda1              | .0324493 | .0074013 | 4.38 | 0.000 | .0179429 - .0469556  |
| lambda2              | .8574681 | .0476274 | 18.00| 0.000 | .7641202 - .950816   |

It turns out that the coefficient on L1.nissan in the honda equation is now statistically insignificant. We could further improve the model by removing L1.nissan from the model.

There is no mean equation for Toyota or Nissan. In [TS] mgarch dcc postestimation, we discuss prediction from models without covariates.
Example 3: Model with constraints

Here we fit a bivariate DCC MGARCH model for the Toyota and Nissan shares. We believe that the shares of these car manufacturers follow the same process, so we impose the constraints that the ARCH coefficients are the same for the two companies and that the GARCH coefficients are also the same.

\. constraint 1 _b[ARCH_toyota:L.arch] = _b[ARCH_nissan:L.arch]
\. constraint 2 _b[ARCH_toyota:L.garch] = _b[ARCH_nissan:L.garch]
\. mgarch dcc (toyota nissan = , noconstant), arch(1) garch(1) constraints(1 2)
Calculating starting values....
Optimizing log likelihood
(setting technique to bhhh)
Iteration 0:  log likelihood =  10307.609
Iteration 1:  log likelihood =  10656.153
Iteration 2:  log likelihood =  10862.137
Iteration 3:  log likelihood =  10987.457
Iteration 4:  log likelihood =  11062.347
Iteration 5:  log likelihood =  11135.207
Iteration 6:  log likelihood =  11245.619
Iteration 7:  log likelihood =  11253.56
Iteration 8:  log likelihood =  11294
Iteration 9:  log likelihood =  11296.364
(switching technique to nr)
Iteration 10: log likelihood =  11296.76
Iteration 11: log likelihood =  11297.087
Iteration 12: log likelihood =  11297.091
Iteration 13: log likelihood =  11297.091
Refining estimates
Iteration 0:  log likelihood =  11297.091
Iteration 1:  log likelihood =  11297.091
Dynamic conditional correlation MGARCH model

Sample: 1 - 2015  
Number of obs = 2,015

Distribution: Gaussian  
Wald chi2(.) = .

Log likelihood = 11297.09  
Prob > chi2 = .

( 1) [ARCH_toyota]L.arch - [ARCH_nissan]L.arch = 0
( 2) [ARCH_toyota]L.garch - [ARCH_nissan]L.garch = 0

| Coef.  | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|--------|-----------|-----|------|----------------------|
| ARCH_toyota  |           |     |      |                      |
| arch L1.    | 0.080889  | 0.0103227 | 7.84 | 0.000 | 0.060657 - 0.1011211 |
| garch L1.   | 0.9060711 | 0.0119107 | 76.07 | 0.000 | 0.8827267 - 0.9294156 |
| _cons       | 4.21e-06  | 1.10e-06 | 3.83 | 0.000 | 2.05e-06 - 6.36e-06   |
| ARCH_nissan |           |     |      |                      |
| arch L1.    | 0.080889  | 0.0103227 | 7.84 | 0.000 | 0.060657 - 0.1011211 |
| garch L1.   | 0.9060711 | 0.0119107 | 76.07 | 0.000 | 0.8827267 - 0.9294156 |
| _cons       | 5.92e-06  | 1.47e-06 | 4.03 | 0.000 | 3.04e-06 - 8.80e-06   |
| corr(toyota, nissan) | 0.6646283 | 0.0187793 | 35.39 | 0.000 | 0.6278215 - 0.7014351 |
| /Adjustment |           |     |      |                      |
| lambda1    | 0.0446559 | 0.0123017 | 3.63 | 0.000 | 0.020545 - 0.0687668 |
| lambda2    | 0.8686054 | 0.0510884 | 17.00 | 0.000 | 0.7684739 - 0.968737 |

We could test our constraints by fitting the unconstrained model and performing a likelihood-ratio test. The results indicate that the restricted model is preferable.

Example 4: Model with a GARCH term

In this example, we have data on fictional stock returns for the Acme and Anvil corporations, and we believe that the movement of the two stocks is governed by different processes. We specify one ARCH and one GARCH term for the conditional variance equation for Acme and two ARCH terms for the conditional variance equation for Anvil. In addition, we include the lagged value of the stock return for Apex, the main subsidiary of Anvil corporation, in the variance equation of Anvil. For Acme, we have data on the changes in an index of futures prices of products related to those produced by Acme in afrelated. For Anvil, we have data on the changes in an index of futures prices of inputs used by Anvil in afinputs.
. use https://www.stata-press.com/data/r16/acmeh
. mgarch dcc (acme = afrelated, noconstant arch(1) garch(1))
> (anvil = afinputs, arch(1/2) het(L.apex))
Calculating starting values....
Optimizing log likelihood
(setting technique to bhhh)
Iteration 0: log likelihood = -13260.522
(output omitted)
Iteration 9: log likelihood = -12362.876
(switching technique to nr)
Iteration 10: log likelihood = -12362.876
Refining estimates
Iteration 0: log likelihood = -12362.876
Iteration 1: log likelihood = -12362.876
Dynamic conditional correlation MGARCH model
Sample: 2 - 2500  Number of obs = 2,499
Distribution: Gaussian  Wald chi²(2) = 2596.18
Log likelihood = -12362.88  Prob > chi² = 0.0000

|                | Coef. | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|----------------|-------|-----------|-------|------|----------------------|
| acme           |       |           |       |      |                      |
| afrelated      | .950805 | .0557082 | 17.07 | 0.000 | .841619 - 1.059991   |
| arch           |       |           |       |      |                      |
| L1.            | .1063295 | .015716  | 6.77  | 0.000 | .0755266 - .1371324  |
| garch          |       |           |       |      |                      |
| L1.            | .7556294 | .0391568 | 19.30 | 0.000 | .6788836 - .8323752  |
| _cons          | 2.197566 | .458343  | 4.79  | 0.000 | 1.29923 - 3.095901   |
| anvil          |       |           |       |      |                      |
| afinputs       | -1.015657 | .0209959 | -48.37 | 0.000 | -1.056808 - .9745054 |
| _cons          | .0808653 | .019445  | 4.16  | 0.000 | .0427538 - .1189767  |
| arch           |       |           |       |      |                      |
| L1.            | .5261675 | .0281586 | 18.69 | 0.000 | .4709777 - .5813572  |
| L2.            | .2866454 | .0196504 | 14.59 | 0.000 | .2481314 - .3251595  |
| apex           |       |           |       |      |                      |
| L1.            | 1.953173 | .0594862 | 32.83 | 0.000 | 1.836582 - 2.069763  |
| _cons          | -.0062964 | .0710842 | -0.09 | 0.929 | -.1456188 - .1330261 |
| corr(acme,     |       |           |       |      |                      |
| anvil)         | -.5600358 | .0326358 | -17.16 | 0.000 | -.6240008 - .4960708 |
| /Adjustment    |       |           |       |      |                      |
| lambda1        | .1904321 | .0154449 | 12.33 | 0.000 | .1601607 - .2207035  |
| lambda2        | .7147267 | .0226204 | 31.60 | 0.000 | .6703916 - .7590618  |
The results indicate that increases in the futures prices for related products lead to higher returns on the Acme stock, and increased input prices lead to lower returns on the Anvil stock. In the conditional variance equation for Anvil, the coefficient on L1.apex is positive and significant, which indicates that an increase in the return on the Apex stock leads to more variability in the return on the Anvil stock.

**Stored results**

`mgarch dcc` stores the following in `e()`:

**Scalars**
- `e(N)`: number of observations
- `e(k)`: number of parameters
- `e(k_extra)`: number of extra estimates added to `_b`
- `e(k_eq)`: number of equations in `e(b)`
- `e(k_dv)`: number of dependent variables
- `e(df_m)`: model degrees of freedom
- `e(ll)`: log likelihood
- `e(chi2)`: $\chi^2$ p-value for model test
- `e(p)`: p-value for model test
- `e(estdf)`: 1 if distribution parameter was estimated, 0 otherwise
- `e(usr)`: user-provided distribution parameter
- `e(tmin)`: minimum time in sample
- `e(tmax)`: maximum time in sample
- `e(N_gaps)`: number of gaps
- `e(rank)`: rank of `e(V)`
- `e(ic)`: number of iterations
- `e(rc)`: return code
- `e(converged)`: 1 if converged, 0 otherwise

**Macros**
- `e(cmd)`: `mgarch`
- `e(model)`: `dcc`
- `e(cmdline)`: command as typed
- `e(depvar)`: names of dependent variables
- `e(covariates)`: list of covariates
- `e(dv_eqs)`: dependent variables with mean equations
- `e(indepvars)`: independent variables in each equation
- `e(tvar)`: time variable
- `e(hetvars)`: variables included in the conditional variance equations
- `e(title)`: title in estimation output
- `e(chi2type)`: Wald; type of model $\chi^2$ test
- `e(vce)`: vcetype specified in `vce()`
- `e(vctype)`: title used to label Std. Err.
- `e(tmins)`: formatted minimum time
- `e(tmaxs)`: formatted maximum time
- `e(dist)`: distribution for error term: `gaussian` or `t`
- `e(arch)`: specified ARCH terms
- `e(garch)`: specified GARCH terms
- `e(technique)`: maximization technique
- `e(properties)`: `b V`
- `e(estat_cmd)`: program used to implement `estat`
- `e(predict)`: program used to implement `predict`
- `e(marginsok)`: predictions allowed by `margins`
- `e(marginsnotok)`: predictions disallowed by `margins`
- `e(marginsdefault)`: default `predict()` specification for `margins`
- `e(asbalanced)`: factor variables `fvset` as `asbalanced`
- `e(asobserved)`: factor variables `fvset` as `asobserved`
Methods and formulas

**mgarch dcc** estimates the parameters of the DCC MGARCH model by maximum likelihood. The log-likelihood function based on the multivariate normal distribution for observation $t$ is

$$l_t = -0.5m \log(2\pi) - 0.5 \log \{ \det (R_t) \} - \log \left\{ \det \left( D_t^{1/2} \right) \right\} - 0.5 \tilde{\epsilon}_t R_t^{-1} \tilde{\epsilon}_t'$$

where $\tilde{\epsilon}_t = D_t^{-1/2} \epsilon_t$ is an $m \times 1$ vector of standardized residuals, $\epsilon_t = y_t - Cx_t$. The log-likelihood function is $\sum_{t=1}^{T} l_t$.

If we assume that $\nu_t$ follow a multivariate $t$ distribution with degrees of freedom (df) greater than 2, then the log-likelihood function for observation $t$ is

$$l_t = \log \Gamma \left( \frac{df + m}{2} \right) - \log \Gamma \left( \frac{df}{2} \right) - \frac{m}{2} \log \{(df - 2)\pi\} - 0.5 \log \{ \det (R_t) \} - \log \left\{ \det \left( D_t^{1/2} \right) \right\} - \frac{df + m}{2} \log \left( 1 + \tilde{\epsilon}_t R_t^{-1} \tilde{\epsilon}_t' \right)$$

The starting values for the parameters in the mean equations and the initial residuals $\hat{\epsilon}_t$ are obtained by least-squares regression. The starting values for the parameters in the variance equations are obtained by a procedure proposed by Gourieroux and Monfort (1997, sec. 6.2.2). The starting values for the quasicorrelation parameters are calculated from the standardized residuals $\tilde{\epsilon}_t$. Given the starting values for the mean and variance equations, the starting values for the parameters $\lambda_1$ and $\lambda_2$ are obtained from a grid search performed on the log likelihood.

The initial optimization step is performed in the unconstrained space. Once the maximum is found, we impose the constraints $\lambda_1 \geq 0$, $\lambda_2 \geq 0$, and $0 \leq \lambda_1 + \lambda_2 < 1$, and maximize the log likelihood in the constrained space. This step is reported in the iteration log as the refining step.

GARCH estimators require initial values that can be plugged in for $\epsilon_{t-i} \epsilon_{t-i}'$ and $H_{t-j}$ when $t - i < 1$ and $t - j < 1$. **mgarch dcc** substitutes an estimator of the unconditional covariance of the disturbances

$$\hat{\Sigma} = T^{-1} \sum_{t=1}^{T} \hat{\epsilon}_t \hat{\epsilon}_t'$$

for $\epsilon_{t-i} \epsilon_{t-i}'$ when $t - i < 1$ and for $H_{t-j}$ when $t - j < 1$, where $\hat{\epsilon}_t$ is the vector of residuals calculated using the estimated parameters.

**mgarch dcc** uses numerical derivatives in maximizing the log-likelihood function.
References


Also see

[TS] mgarch dcc postestimation — Postestimation tools for mgarch dcc

[TS] arch — Autoregressive conditional heteroskedasticity (ARCH) family of estimators

[TS] mgarch — Multivariate GARCH models

[TS] tsset — Declare data to be time-series data

[TS] var — Vector autoregressive models

[U] 20 Estimation and postestimation commands
Postestimation commands

The following standard postestimation commands are available after `mgarch dcc`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contrast</code></td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>pwcompare</code></td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and conditional variances, covariances, and correlations. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] {stub* | newvarlist} [if] [in] [, statistic options]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>xb linear prediction; the default</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
<tr>
<td>variance</td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td>correlation</td>
<td>conditional correlations</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>equation(eqnames) names of equations for which predictions are made</td>
</tr>
<tr>
<td></td>
<td>dynamic(time_constant) begin dynamic forecast at specified time</td>
</tr>
</tbody>
</table>

Options for predict

- **Main**
  - `xb`, the default, calculates the linear predictions of the dependent variables.
  - `residuals` calculates the residuals.
  - `variance` predicts the conditional variances and conditional covariances.
  - `correlation` predicts the conditional correlations.

- **Options**
  - `equation(eqnames)` specifies the equation for which the predictions are calculated. Use this option to predict a statistic for a particular equation. Equation names, such as `equation(income)`, are used to identify equations.
One equation name may be specified when predicting the dependent variable, the residuals, or the conditional variance. For example, specifying `equation(income)` causes `predict` to predict `income`, and specifying `variance equation(income)` causes `predict` to predict the conditional variance of `income`.

Two equations may be specified when predicting a conditional variance or covariance. For example, specifying `equation(income, consumption)` `variance` causes `predict` to predict the conditional covariance of `income` and `consumption`.

dynamic(`time_constant`) specifies when `predict` starts producing dynamic forecasts. The specified `time_constant` must be in the scale of the time variable specified in `tsset`, and the `time_constant` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. dynamic() may not be specified with residuals.

margins

Description for margins

margins estimates margins of response for linear predictions and conditional variances, covariances, and correlations. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for margins

Statistics > Postestimation

Syntax for margins

```
margins [ `marginlist` ] [, `options` ]
margins [ `marginlist` ], `predict(statistic ...)` [ `predict(statistic ...)` ... ] [ `options` ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td>variance</td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td>correlation</td>
<td>conditional correlations</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with <code>margins</code></td>
</tr>
</tbody>
</table>

xb defaults to the first equation.

Statistics not allowed with `margins` are functions of stochastic quantities other than `e(b)`.

For the full syntax, see [R] margins.
Remarks and examples

We assume that you have already read [TS] mgarch dcc. In this entry, we use predict after mgarch dcc to make in-sample and out-of-sample forecasts.

Example 1: Dynamic forecasts

In this example, we obtain dynamic forecasts for the Toyota, Nissan, and Honda stock returns modeled in example 2 of [TS] mgarch dcc. In the output below, we reestimate the parameters of the model, use tsappend (see [TS] tsappend) to extend the data, and use predict to obtain in-sample one-step-ahead forecasts and dynamic forecasts of the conditional variances of the returns. We graph the forecasts below.

```
. use https://www.stata-press.com/data/r16/stocks
(Data from Yahoo! Finance)
. quietly mgarch dcc (toyota nissan = , noconstant)
> (honda = L.nissan, noconstant), arch(1) garch(1)
. tsappend, add(50)
. predict H*, variance dynamic(2016)

```

![Graph showing dynamic forecasts](image)

Recent in-sample one-step-ahead forecasts are plotted to the left of the vertical line in the above graph, and the dynamic out-of-sample forecasts appear to the right of the vertical line. The graph shows the tail end of the huge increase in return volatility that took place in 2008 and 2009. It also shows that the dynamic forecasts quickly converge.

Methods and formulas

All one-step predictions are obtained by substituting the parameter estimates into the model. The estimated unconditional variance matrix of the disturbances, \( \hat{\Sigma} \), is the initial value for the ARCH and GARCH terms. The postestimation routines recompute \( \hat{\Sigma} \) using the prediction sample, the parameter estimates stored in \( \mathbf{e}(b) \), and (2) in Methods and formulas of [TS] mgarch dcc.

For observations in which the residuals are missing, the estimated unconditional variance matrix of the disturbances is used in place of the outer product of the residuals.
Dynamic predictions of the dependent variables use previously predicted values beginning in the period specified by `dynamic()`.

Dynamic variance predictions are implemented by substituting \( \hat{\Sigma} \) for the outer product of the residuals beginning in the period specified in `dynamic()`.

**Also see**

[TS] `mgarch dcc` — Dynamic conditional correlation multivariate GARCH models

[U] 20 Estimation and postestimation commands
mgarch dvech estimates the parameters of diagonal vech (DVECH) multivariate generalized autoregressive conditionally heteroskedastic (MGARCH) models in which each element of the conditional correlation matrix is parameterized as a linear function of its own past and past shocks.

DVECH MGARCH models are less parsimonious than the conditional correlation models discussed in [TS] mgarch ccc, [TS] mgarch dcc, and [TS] mgarch vcc because the number of parameters in DVECH MGARCH models increases more rapidly with the number of series modeled.

Quick start

Fit diagonal vech multivariate GARCH with first- and second-order ARCH components for dependent variables \( y_1 \) and \( y_2 \) using \texttt{tsset} data

\[
\text{mgarch dvech (y1 y2), arch(1 2)}
\]

Add regressors \( x_1 \) and \( x_2 \) and first-order GARCH component

\[
\text{mgarch dvech (y1 y2 = x1 x2), arch(1 2) garch(1)}
\]

Menu

Statistics > Multivariate time series > Multivariate GARCH
Syntax

```
mgarch dvech eq [ eq ... eq ] [ if ] [ in ] [ , options ]
```

where each `eq` has the form

```
(depvars = [ indepvars ] [ , noconstant ])
```

**options**    Description

**Model**

- `arch(numlist)` ARCH terms
- `garch(numlist)` GARCH terms
- `distribution(dist [#])` use `dist` distribution for errors [may be `gaussian` (synonym `normal`) or `t`; default is `gaussian`]
- `constraints(numlist)` apply linear constraints

**SE/Robust**

- `vce(vcetype)` `vcetype` may be `oim` or `robust`

**Reporting**

- `level(#)` set confidence level; default is `level(95)`
- `nocnsreport` do not display constraints
- `display_options` control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling

**Maximization**

- `maximize_options` control the maximization process; seldom used
- `from(matname)` initial values for the coefficients; seldom used
- `svtechnique(algorithm_spec)` starting-value maximization algorithm
- `sviterate(#)` number of starting-value iterations; default is `sviterate(25)`
- `coeflegend` display legend instead of statistics

You must `tsset` your data before using `mgarch dvech`; see [TS] `tsset`. `indepvars` may contain factor variables; see [U] 11.4.3 Factor variables. `depvars` and `indepvars` may contain time-series operators; see [U] 11.4.4 Time-series varlists. by, fp, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands. `coeflegend` does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

**Options**

<table>
<thead>
<tr>
<th>Model</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>noconstant</td>
<td>suppresses the constant term(s).</td>
</tr>
<tr>
<td><code>arch(numlist)</code></td>
<td>specifies the ARCH terms in the model. By default, no ARCH terms are specified.</td>
</tr>
<tr>
<td><code>garch(numlist)</code></td>
<td>specifies the GARCH terms in the model. By default, no GARCH terms are specified.</td>
</tr>
<tr>
<td><code>distribution(dist [#])</code></td>
<td>specifies the assumed distribution for the errors. <code>dist</code> may be <code>gaussian</code>, <code>normal</code>, or <code>t</code>.</td>
</tr>
</tbody>
</table>
gaussian and normal are synonyms; each causes mgarch dvech to assume that the errors come from a multivariate normal distribution. # cannot be specified with either of them.

t causes mgarch dvech to assume that the errors follow a multivariate Student t distribution, and the degree-of-freedom parameter is estimated along with the other parameters of the model. If distribution(t #) is specified, then mgarch dvech uses a multivariate Student t distribution with # degrees of freedom. # must be greater than 2.

constraints(numlist) specifies linear constraints to apply to the parameter estimates.

vce(vcetype) specifies the estimator for the variance–covariance matrix of the estimator.

vce(oim), the default, specifies to use the observed information matrix (OIM) estimator.

vce(robust) specifies to use the Huber/White/sandwich estimator.

level(#), nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nofvlabel, fvwrap(#), fvwrapon(style), cformat(%fmt), pformat(%fmt), sformat(%fmt), and nolstretch; see [R] Estimation options.

Maximization

maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), nonrtolerance, and from(matname); see [R] Maximize for all options except from(), and see below for information on from(). These options are seldom used.

from(matname) specifies initial values for the coefficients. from(b0) causes mgarch dvech to begin the optimization algorithm with the values in b0. b0 must be a row vector, and the number of columns must equal the number of parameters in the model.

svtechnique(algorithm_spec) and sviterate(#) specify options for the starting-value search process.

svtechnique(algorithm_spec) specifies the algorithm used to search for initial values. The syntax for algorithm_spec is the same as for the technique() option; see [R] Maximize.

svtechnique(bhhh 5 nr 16000) is the default. This option may not be specified with from().

sviterate(#) specifies the maximum number of iterations that the search algorithm may perform. The default is sviterate(25). This option may not be specified with from().

The following option is available with mgarch dvech but is not shown in the dialog box: coeflegend; see [R] Estimation options.

Remarks and examples

We assume that you have already read [TS] mgarch, which provides an introduction to MGARCH models and the methods implemented in mgarch dvech.
MGARCH models are dynamic multivariate regression models in which the conditional variances and covariances of the errors follow an autoregressive-moving-average structure. The DVECH MGARCH model parameterizes each element of the current conditional covariance matrix as a linear function of its own past and past shocks.

As discussed in [TS] mgarch, MGARCH models differ in the parsimony and flexibility of their specifications for a time-varying conditional covariance matrix of the disturbances, denoted by $H_t$. In a DVECH MGARCH model with one ARCH term and one GARCH term, the $(i, j)$th element of conditional covariance matrix is modeled by

$$h_{ij,t} = s_{ij} + a_{ij} \epsilon_{i,t-1} \epsilon_{j,t-1} + b_{ij} h_{ij,t-1}$$

where $s_{ij}$, $a_{ij}$, and $b_{ij}$ are parameters and $\epsilon_{t-1}$ is the vector of errors from the previous period. This expression shows the linear form in which each element of the current conditional covariance matrix is a function of its own past and past shocks.

### Technical note

The general vech MGARCH model developed by Bollerslev, Engle, and Wooldridge (1988) can be written as

$$\begin{align*}
y_t &= Cx_t + \epsilon_t \\
\epsilon_t &= H_t^{1/2} \nu_t \\
h_t &= s + \sum_{i=1}^{p} A_i \text{vech}(\epsilon_{t-i} \epsilon_{t-i}') + \sum_{j=1}^{q} B_j h_{t-j}
\end{align*}$$

where

- $y_t$ is an $m \times 1$ vector of dependent variables;
- $C$ is an $m \times k$ matrix of parameters;
- $x_t$ is a $k \times 1$ vector of independent variables, which may contain lags of $y_t$;
- $H_t^{1/2}$ is the Cholesky factor of the time-varying conditional covariance matrix $H_t$;
- $\nu_t$ is an $m \times 1$ vector of independent and identically distributed innovations;
- $h_t = \text{vech}(H_t)$;
- the vech() function stacks the lower diagonal elements of a symmetric matrix into a column vector, for example,

$$\text{vech}\begin{pmatrix}1 & 2 \\ 2 & 3\end{pmatrix} = (1, 2, 3)'$$

- $s$ is an $m(m+1)/2 \times 1$ vector of parameters;
- each $A_i$ is an $\{m(m+1)/2\} \times \{m(m+1)/2\}$ matrix of parameters; and
- each $B_j$ is an $\{m(m+1)/2\} \times \{m(m+1)/2\}$ matrix of parameters.

Bollerslev, Engle, and Wooldridge (1988) argued that the general-vech MGARCH model in (1)–(3) was too flexible to fit to data, so they proposed restricting the matrices $A_i$ and $B_j$ to be diagonal matrices. It is for this restriction that the model is known as a diagonal vech MGARCH model. The diagonal vech MGARCH model can also be expressed by replacing (3) with
\[
H_t = S + \sum_{i=1}^{p} A_i \odot \epsilon_{t-i} \epsilon'_{t-i} + \sum_{j=1}^{q} B_j \odot H_{t-j}
\]

where \( S \) is an \( m \times m \) symmetric parameter matrix; each \( A_i \) is an \( m \times m \) symmetric parameter matrix; \( \odot \) is the elementwise or Hadamard product; and each \( B_j \) is an \( m \times m \) symmetric parameter matrix. In (3'), \( A \) and \( B \) are symmetric but not diagonal matrices because we used the Hadamard product. The matrices are diagonal in the vech representation of (3) but not in the Hadamard-product representation of (3').

The Hadamard-product representation in (3') clarifies that each element in \( H_t \) depends on its past values and the past values of the corresponding ARCH terms. Although this representation does not allow cross-covariance effects, it is still quite flexible. The rapid rate at which the number of parameters grows with \( m \), \( p \), or \( q \) is one aspect of the model’s flexibility.

Some examples

Example 1: Model with common covariates

We have data on a secondary market rate of a six-month U.S. Treasury bill, \( \text{tbill} \), and on Moody’s seasoned AAA corporate bond yield, \( \text{bond} \). We model the first differences of \( \text{tbill} \) and the first differences of \( \text{bond} \) as a VAR(1) with an ARCH(1) term.

```
use https://www.stata-press.com/data/r16/irates4
(St. Louis Fed (FRED) financial data)
.mgarch dvech (D.bond D.tbill = LD.bond LD.tbill), arch(1)
```

Getting starting values

(setting technique to bhhh)

Iteration 0: log likelihood = 3569.2723
Iteration 1: log likelihood = 3708.4561
(output omitted)
Iteration 6: log likelihood = 4183.8853
Iteration 7: log likelihood = 4184.2424
(switching technique to nr)
Iteration 8: log likelihood = 4184.4141
Iteration 9: log likelihood = 4184.5973
Iteration 10: log likelihood = 4184.5975

Estimating parameters

(setting technique to bhhh)

Iteration 0: log likelihood = 4184.5975
Iteration 1: log likelihood = 4200.6303
Iteration 2: log likelihood = 4208.5342
Iteration 3: log likelihood = 4212.426
Iteration 4: log likelihood = 4215.2373
(switching technique to nr)
Iteration 5: log likelihood = 4217.0676
Iteration 6: log likelihood = 4221.5706
Iteration 7: log likelihood = 4221.6576
Iteration 8: log likelihood = 4221.6577
The output has three parts: an iteration log, a header, and an output table. The iteration log has two parts: the first part reports the iterations from the process of searching for starting values, and the second part reports the iterations from maximizing the log-likelihood function.

The header describes the estimation sample and reports a Wald test against the null hypothesis that all the coefficients on the independent variables in each equation are zero. Here the null hypothesis is rejected at all conventional levels.

The output table reports point estimates, standard errors, tests against zero, and confidence intervals for the estimated coefficients, the estimated elements of $S$, and any estimated elements of $A$ or $B$. Here the output indicates that in the equation for $D.bond$, neither the coefficient on $LD.bond$ nor the constant are statistically significant. The elements of $S$ are reported in the $\Sigma_0$ equation. The estimate of $S[1,1]$ is 0.005, and the estimate of $S[2,1]$ is 0.004. The ARCH term results are reported in the $L.ARCH$ equation. In the $L.ARCH$ equation, $1_1$ is the coefficient on the ARCH term for the conditional variance of the first dependent variable, $2_1$ is the coefficient on the ARCH term for the conditional covariance between the first and second dependent variables, and $2_2$ is the coefficient on the ARCH term for the conditional variance of the second dependent variable.
Example 2: Model with covariates that differ by equation

We improve the previous example by removing the insignificant parameters from the model:

```
.mgarch dvech (D.bond = LD.bond LD.tbill, noconstant)
> (D.tbill = LD.tbill, noconstant), arch(1)
```

Getting starting values

(setting technique to bhhh)

Iteration 0:  log likelihood = 3566.8824
Iteration 1:  log likelihood = 3701.6181
Iteration 2:  log likelihood = 3952.8048
Iteration 3:  log likelihood = 4076.5164
Iteration 4:  log likelihood = 4166.6842
Iteration 5:  log likelihood = 4180.2998
Iteration 6:  log likelihood = 4182.4545
Iteration 7:  log likelihood = 4182.9563

(switching technique to nr)
Iteration 8:  log likelihood = 4183.0293
Iteration 9:  log likelihood = 4183.1112
Iteration 10: log likelihood = 4183.1113

Estimating parameters

(setting technique to bhhh)

Iteration 0:  log likelihood = 4183.1113
Iteration 1:  log likelihood = 4202.0304
Iteration 2:  log likelihood = 4210.2929
Iteration 3:  log likelihood = 4215.7798
Iteration 4:  log likelihood = 4217.7755

(switching technique to nr)
Iteration 5:  log likelihood = 4219.0078
Iteration 6:  log likelihood = 4221.4197
Iteration 7:  log likelihood = 4221.433
Iteration 8:  log likelihood = 4221.433

Diagonal vech MGARCH model

Sample: 3 - 2456  Number of obs = 2,454
Distribution: Gaussian  Wald chi2(3) = 1197.76
Log likelihood = 4221.433  Prob > chi2 = 0.0000

|                      | Coef.   | Std. Err. | z   | P>|z|  | [95% Conf. Interval] |
|----------------------|---------|-----------|-----|-------|----------------------|
| D.bond               |         |           |     |       |                      |
| bond LD.             | .2941649| .0234734  | 12.53| 0.000 | .2481579 .3401718    |
| tbill LD.            | .0953158| .0098077  | 9.72| 0.000 | .076093 .1145386     |
| D.tbill              |         |           |     |       |                      |
| tbill LD.            | .4385945| .0136672  | 32.09| 0.000 | .4118072 .4653817    |
| /Sigma0              |         |           |     |       |                      |
| 1_1                  | .0048922| .0002005  | 24.40| 0.000 | .0044993 .0052851    |
| 2_1                  | .0040949| .0002394  | 17.10| 0.000 | .0036256 .0045641    |
| 2_2                  | .0115043| .0005184  | 22.19| 0.000 | .0104883 .0125203    |
| L.ARCH               |         |           |     |       |                      |
| 1_1                  | .4519233| .045671   | 9.90| 0.000 | .3624099 .5414368    |
| 2_1                  | .2515474| .0366701  | 6.86| 0.000 | .1796752 .3234195    |
| 2_2                  | .8437212| .0600839  | 14.04| 0.000 | .7259589 .9614836    |
We specified each equation separately to remove the insignificant parameters. All the parameter estimates are statistically significant.

Example 3: Model with constraints

Here we analyze some fictional weekly data on the percentages of bad widgets found in the factories of Acme Inc. and Anvil Inc. We model the levels as a first-order autoregressive process. We believe that the adaptive management style in these companies causes the variances to follow a diagonal vech MGARCH process with one ARCH term and one GARCH term. Furthermore, these close competitors follow essentially the same process, so we impose the constraints that the ARCH coefficients are the same for the two companies and that the GARCH coefficients are also the same.

Imposing these constraints yields

```plaintext
use https://www.stata-press.com/data/r16/acme
constraint 1 [L.ARCH]1_1 = [L.ARCH]2_2
constraint 2 [L.GARCH]1_1 = [L.GARCH]2_2
mgarch dvech (acme = L.acme) (anvil = L.anvil), arch(1) garch(1) > constraints(1 2)
```

Getting starting values

(setting technique to bhhh)

Iteration 0: log likelihood = -6087.0665 (not concave)
Iteration 1: log likelihood = -6022.2046
Iteration 2: log likelihood = -5986.6152
Iteration 3: log likelihood = -5976.5739
Iteration 4: log likelihood = -5974.4342
Iteration 5: log likelihood = -5974.4046
Iteration 6: log likelihood = -5974.4036
Iteration 7: log likelihood = -5974.4035

Estimating parameters

(setting technique to bhhh)

Iteration 0: log likelihood = -5974.4035
Iteration 1: log likelihood = -5973.812
Iteration 2: log likelihood = -5973.8004
Iteration 3: log likelihood = -5973.7999
Iteration 4: log likelihood = -5973.7999
Diagonal vech MGARCH model

Sample: 1969w35 - 1998w25
Number of obs = 1,499
Distribution: Gaussian
Wald chi2(2) = 272.47
Log likelihood = -5973.8
Prob > chi2 = 0.0000

( 1) [L.ARCH]1_1 - [L.ARCH]2_2 = 0
( 2) [L.GARCH]1_1 - [L.GARCH]2_2 = 0

|            | Coef.  | Std. Err. | z     | P>|z|    | [95% Conf. Interval] |
|------------|--------|-----------|-------|--------|----------------------|
| acme       |        |           |       |        |                      |
| acme       |        |           |       |        |                      |
| L1.        | 0.3365278 | 0.0255134 | 13.19 | 0.000  | 0.2865225 - 0.3865331 |
| _cons      | 1.124611 | 0.060085  | 18.72 | 0.000  | 1.006847 - 1.242376  |
| anvil      |        |           |       |        |                      |
| anvil      |        |           |       |        |                      |
| L1.        | 0.3151955 | 0.0263287 | 11.97 | 0.000  | 0.2635922 - 0.3667988 |
| _cons      | 1.215786 | 0.0642052 | 18.94 | 0.000  | 1.089947 - 1.341626  |
| /Sigma0    |        |           |       |        |                      |
| 1_1        | 1.889237 | 0.2168733 | 8.71  | 0.000  | 1.464173 - 2.314301  |
| 2_1        | 0.4599576 | 0.1139843 | 4.04  | 0.000  | 0.2365525 - 0.6833626 |
| 2_2        | 2.063113 | 0.2454633 | 8.40  | 0.000  | 1.582014 - 2.544213  |
| L.ARCH     |        |           |       |        |                      |
| 1_1        | 0.2813443 | 0.0299124 | 9.41  | 0.000  | 0.222717 - 0.3399716 |
| 2_1        | 0.181877 | 0.0335393 | 5.42  | 0.000  | 0.1161412 - 0.2476128 |
| 2_2        | 0.2813443 | 0.0299124 | 9.41  | 0.000  | 0.222717 - 0.3399716 |
| L.GARCH    |        |           |       |        |                      |
| 1_1        | 0.1487581 | 0.0697531 | 2.13  | 0.033  | 0.0120445 - 0.2854716 |
| 2_1        | 0.085404 | 0.1446524 | 0.59  | 0.555  | -0.1981094 - 0.3689175 |
| 2_2        | 0.1487581 | 0.0697531 | 2.13  | 0.033  | 0.0120445 - 0.2854716 |

We could test our constraints by fitting the unconstrained model and performing either a Wald or a likelihood-ratio test. The results indicate that we might further restrict the time-invariant components of the conditional variances to be the same across companies.

Example 4: Model with a GARCH term

Some models of financial data include no covariates or constant terms. For example, in modeling fictional data on the stock returns of Acme Inc. and Anvil Inc., we found it best not to include any covariates or constant terms. We include two ARCH terms and one GARCH term to model the conditional variances.
. use https://www.stata-press.com/data/r16/aacmer
. mgarch dvech (acme anvil = , noconstant), arch(1/2) garch(1)

Getting starting values
(setting technique to bhhh)
Iteration 0: log likelihood = -18417.243 (not concave)
Iteration 1: log likelihood = -18215.005
Iteration 2: log likelihood = -18199.691
Iteration 3: log likelihood = -18136.699
Iteration 4: log likelihood = -18084.256
Iteration 5: log likelihood = -17993.662
Iteration 6: log likelihood = -17731.1
Iteration 7: log likelihood = -17629.505
(switching technique to nr)
Iteration 8: log likelihood = -17548.172
Iteration 9: log likelihood = -17544.987
Iteration 10: log likelihood = -17544.937
Iteration 11: log likelihood = -17544.937

Estimating parameters
(setting technique to bhhh)
Iteration 0: log likelihood = -17544.937
Iteration 1: log likelihood = -17544.937

Diagonal vech MGARCH model
Sample: 1 - 5000 Number of obs = 5,000
Distribution: Gaussian Wald chi2(.) = 
Log likelihood = -17544.94 Prob > chi2 = 

| Coef. Std. Err. z P>|z|   [95% Conf. Interval] |
|-------------------|-----------------|---------|--------|------------------|
| /Sigma0           |                 |         |        |                  |
| 1_1               | 1.026283        | 0.0823348 | 12.46  | 0.000            | 0.8649096 1.187656 |
| 2_1               | 0.4300997       | 0.0590294 | 7.29   | 0.000            | 0.3144042 0.5457952 |
| 2_2               | 1.019753        | 0.0837146 | 12.18  | 0.000            | 0.8556751 1.18383  |
| L.ARCH            |                 |         |        |                  |
| 1_1               | 0.2878739       | 0.02157  | 13.35  | 0.000            | 0.2455975 0.3301504 |
| 2_1               | 0.1036685       | 0.016446 | 6.42   | 0.000            | 0.0720256 0.1353114 |
| 2_2               | 0.2034196       | 0.019855 | 10.25  | 0.000            | 0.1645044 0.2423347 |
| L2.ARCH           |                 |         |        |                  |
| 1_1               | 0.1837825       | 0.0274555 | 6.69   | 0.000            | 0.1299706 0.2375943 |
| 2_1               | 0.0884425       | 0.02208  | 4.01   | 0.000            | 0.0451665 0.1317185 |
| 2_2               | 0.2025718       | 0.0272639 | 7.43   | 0.000            | 0.1491355 0.256008 |
| L.GARCH           |                 |         |        |                  |
| 1_1               | 0.0782467       | 0.053944 | 1.45   | 0.147            | -0.0274816 0.183975 |
| 2_1               | 0.2888104       | 0.081303 | 3.53   | 0.000            | 0.1284261 0.4491948 |
| 2_2               | 0.201618        | 0.0470584 | 4.28   | 0.000            | 0.1093853 0.2938508 |

The model test is omitted from the output, because there are no covariates in the model. The univariate tests indicate that the included parameters fit the data well. In [TS] mgarch dvech postestimation, we discuss prediction from models without covariates.
Stored results

`mgarch dvech` stores the following in `e()`:

Scalars
- `e(N)` number of observations
- `e(k)` number of parameters
- `e(k_extra)` number of extra estimates added to `_b`
- `e(k_eq)` number of equations in `e(b)`
- `e(k_dv)` number of dependent variables
- `e(df_m)` model degrees of freedom
- `e(ll)` log likelihood
- `e(chi2)` \( \chi^2 \) statistic
- `e(p)` p-value for model test
- `e(estdf)` 1 if distribution parameter was estimated, 0 otherwise
- `e(usr)` user-provided distribution parameter
- `e(tmin)` minimum time in sample
- `e(tmax)` maximum time in sample
- `e(N_gaps)` number of gaps
- `e(rank)` rank of `e(V)`
- `e(ic)` number of iterations
- `e(rc)` return code
- `e(converged)` 1 if converged, 0 otherwise

Macros
- `e(cmd)` `mgarch`
- `e(model)` `dvech`
- `e(cmdline)` command as typed
- `e(depvar)` names of dependent variables
- `e(covariates)` list of covariates
- `e(dv_eqs)` dependent variables with mean equations
- `e(indep)` independent variables in each equation
- `e(tvar)` time variable
- `e(title)` title in estimation output
- `e(chi2type)` Wald; type of model \( \chi^2 \) test
- `e(vce)` `vcetype` specified in `vce()`
- `e(vctype)` title used to label Std. Err.
- `e(tmins)` formatted minimum time
- `e(tmaxs)` formatted maximum time
- `e(dist)` distribution for error term: `gaussian` or `t`
- `e(arch)` specified ARCH terms
- `e(garch)` specified GARCH terms
- `e(svtechnique)` maximization technique(s) for starting values
- `e(technique)` maximization technique
- `e(properties)` `b V`
- `e(estat_cmd)` program used to implement `estat`
- `e(predict)` program used to implement `predict`
- `e(marginsok)` predictions allowed by `margins`
- `e(marginsnotok)` predictions disallowed by `margins`
- `e(marginsdefault)` default `predict()` specification for `margins`
- `e(asbalanced)` factor variables `fvset` as `asbalanced`
- `e(asobserved)` factor variables `fvset` as `asobserved`
Matrices
- \( e(b) \): coefficient vector
- \( e(Cns) \): constraints matrix
- \( e(ilog) \): iteration log (up to 20 iterations)
- \( e(gradient) \): gradient vector
- \( e(hessian) \): Hessian matrix
- \( e(A) \): estimates of \( A \) matrices
- \( e(B) \): estimates of \( B \) matrices
- \( e(S) \): estimates of \( \Sigma_0 \) matrix
- \( e(Sigma) \): \( \Sigma \) hat
- \( e(V) \): variance–covariance matrix of the estimators
- \( e(V\_modelbased) \): model-based variance
- \( e(pinfo) \): parameter information, used by \texttt{predict}

Functions
- \( e(sample) \): marks estimation sample

Methods and formulas
Recall that the diagonal vech MGARCH model can be written as

\[
y_t = Cx_t + \epsilon_t
\]
\[
\epsilon_t = H_t^{1/2} \nu_t
\]
\[
H_t = S + \sum_{i=1}^{p} A_i \odot \epsilon_{t-i} \epsilon_{t-i}' + \sum_{j=1}^{q} B_j \odot H_{t-j}
\]

where
- \( y_t \) is an \( m \times 1 \) vector of dependent variables;
- \( C \) is an \( m \times k \) matrix of parameters;
- \( x_t \) is a \( k \times 1 \) vector of independent variables, which may contain lags of \( y_t \);
- \( H_t^{1/2} \) is the Cholesky factor of the time-varying conditional covariance matrix \( H_t \);
- \( \nu_t \) is an \( m \times 1 \) vector of normal, independent, and identically distributed innovations;
- \( S \) is an \( m \times m \) symmetric matrix of parameters;
- each \( A_i \) is an \( m \times m \) symmetric matrix of parameters;
- \( \odot \) is the elementwise or Hadamard product; and
- each \( B_j \) is an \( m \times m \) symmetric matrix of parameters.

\texttt{mgarch dvech} estimates the parameters by maximum likelihood. The log-likelihood function based on the multivariate normal distribution for observation \( t \) is

\[
l_t = -0.5m \log(2\pi) - 0.5 \log \{ \det (H_t) \} - 0.5 \epsilon_t H_t^{-1} \epsilon_t'
\]

where \( \epsilon_t = y_t - Cx_t \). The log-likelihood function is \( \sum_{t=1}^{T} l_t \).
If we assume that $\nu_t$ follow a multivariate $t$ distribution with degrees of freedom (df) greater than 2, then the log-likelihood function for observation $t$ is

$$
\begin{equation}
\ln l_t = \log \Gamma \left( \frac{\text{df} + m}{2} \right) - \log \Gamma \left( \frac{\text{df}}{2} \right) - \frac{m}{2} \log \left\{ (\text{df} - 2)\pi \right\} \\
- 0.5 \log \left\{ \det (H_t) \right\} - \frac{df + m}{2} \log \left( 1 + \frac{\epsilon_t H_t^{-1} \epsilon_t'}{df - 2} \right)
\end{equation}
$$

$m\text{garch dvech}$ ensures that $H_t$ is positive definite for each $t$.

By default, $m\text{garch dvech}$ performs an iterative search for starting values. $m\text{garch dvech}$ estimates starting values for $C$ by seemingly unrelated regression, uses these estimates to compute residuals $\hat{\epsilon}_t$, plugs $\hat{\epsilon}_t$ into the above log-likelihood function, and optimizes this log-likelihood function over the parameters in $H_t$. This starting-value method plugs in consistent estimates of the parameters for the conditional means of the dependent variables and then iteratively searches for the variance parameters that maximize the log-likelihood function. Lütkepohl (2005, chap. 16) discusses this method as an estimator for the variance parameters.

GARCH estimators require initial values that can be plugged in for $\epsilon_{t-i} \epsilon_{t-i}'$ and $H_{t-j}$ when $t - i < 1$ and $t - j < 1$. $m\text{garch dvech}$ substitutes an estimator of the unconditional covariance of the disturbances,

$$
\hat{\Sigma} = T^{-1} \sum_{t=1}^{T} \hat{\epsilon}_t \hat{\epsilon}_t'
$$

for $\epsilon_{t-i} \epsilon_{t-i}'$ when $t - i < 1$ and for $H_{t-j}$ when $t - j < 1$, where $\hat{\epsilon}_t$ is the vector of residuals calculated using the estimated parameters.

$m\text{garch dvech}$ uses analytic first and second derivatives in maximizing the log-likelihood function based on the multivariate normal distribution. $m\text{garch dvech}$ uses numerical derivatives in maximizing the log-likelihood function based on the multivariate $t$ distribution.

References


Also see

[TS] *m\text{garch dvech postestimation}* — Postestimation tools for $m\text{garch dvech}$

[TS] *arch* — Autoregressive conditional heteroskedasticity (ARCH) family of estimators

[TS] *m\text{garch}* — Multivariate GARCH models

[TS] *tsset* — Declare data to be time-series data

[TS] *var* — Vector autoregressive models

[U] *20 Estimation and postestimation commands*
Postestimation commands

The following standard postestimation commands are available after \texttt{mgarch dvech}:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contrast</td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td>estat ic</td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td>estat summarize</td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td>estat vce</td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td>estimates</td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td>forecast</td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td>lincom</td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td>lrtest</td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td>margins</td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td>marginsplot</td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td>nlcom</td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td>predict</td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td>predictnl</td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td>pwcompare</td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td>test</td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td>testnl</td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and conditional variances and covariances. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

predict [type] {stub*|newvarlist} [if] [in] [, statistic options]

statistic Description

Main
xb linear prediction; the default
residuals residuals
variance conditional variances and covariances

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for the estimation sample.

options Description

Options
equation(eqnames) names of equations for which predictions are made
dynamic(time_constant) begin dynamic forecast at specified time

Options for predict

Main
xb, the default, calculates the linear predictions of the dependent variables.
residuals calculates the residuals.
variance predicts the conditional variances and conditional covariances.

Options
equation(eqnames) specifies the equation for which the predictions are calculated. Use this option to predict a statistic for a particular equation. Equation names, such as equation(income), are used to identify equations.

One equation name may be specified when predicting the dependent variable, the residuals, or the conditional variance. For example, specifying equation(income) causes predict to predict income, and specifying variance equation(income) causes predict to predict the conditional variance of income.
Two equations may be specified when predicting a conditional variance or covariance. For example, specifying `equation(income, consumption) variance` causes `predict` to predict the conditional covariance of income and consumption.

`dynamic(time_constant)` specifies when `predict` starts producing dynamic forecasts. The specified `time_constant` must be in the scale of the time variable specified in `tsset`, and the `time_constant` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may not be specified with `residuals`.

**margins**

**Description for margins**

`margins` estimates margins of response for linear predictions and conditional variances and covariances. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

**Menu for margins**

Statistics > Postestimation

**Syntax for margins**

```
margins [marginlist] [, options]
margins [marginlist], predict(statistic ...) [predict(statistic ...) ...] [options]
```

**Statistic Description**

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td>variance</td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

`xb` defaults to the first equation.

Statistics not allowed with `margins` are functions of stochastic quantities other than `e(b)`.

For the full syntax, see [R] margins.
Remarks and examples

We assume that you have already read [TS] mgarch dvech. In this entry, we illustrate some of the features of predict after using mgarch dvech to estimate the parameters of diagonal vech MGARCH models.

Example 1: Dynamic forecasts

In example 3 of [TS] mgarch dvech, we obtained dynamic predictions for the Acme Inc. and Anvil Inc. fictional widget data.

```
. use https://www.stata-press.com/data/r16/acme
. constraint 1 [L.ARCH]1_1 = [L.ARCH]2_2
. constraint 2 [L.GARCH]1_1 = [L.GARCH]2_2
. mgarch dvech (acme = L.acme) (anvil = L.anvil), arch(1) garch(1) > constraints(1 2)
```

Now we use tsappend (see [TS] tsappend) to extend the data, use predict to obtain the dynamic predictions, and graph the predictions.

```
. tsappend, add(12)
. predict H*, variance dynamic(tw(1998w26))
. tsline H_acme_acme H_anvil_anvil if t>=tw(1995w25), legend(rows(2))
```

The graph shows that the in-sample predictions are similar for the conditional variances of Acme Inc. and Anvil Inc. and that the dynamic forecasts converge to similar levels. It also shows that the ARCH and GARCH parameters cause substantial time-varying volatility. The predicted conditional variance of acme ranges from lows of just over 2 to highs above 10.
Example 2: Predicting in-sample conditional variances

In this example, we obtain the in-sample predicted conditional variances of the returns for the fictional Acme Inc., which we modeled in example 4 of [TS] mgarch dvech. First, we reestimate the parameters of the model.

```
use https://www.stata-press.com/data/r16/aacmer, clear
mgarch dvech (acme anvil = , noconstant), arch(1/2) garch(1)
```

Getting starting values
(setting technique to bhhh)
Iteration 0: log likelihood = -18417.243 (not concave)
Iteration 1: log likelihood = -18215.005
Iteration 2: log likelihood = -18199.691
Iteration 3: log likelihood = -18136.699
Iteration 4: log likelihood = -18084.256
Iteration 5: log likelihood = -17993.662
Iteration 6: log likelihood = -17731.1
Iteration 7: log likelihood = -17629.505

(switching technique to nr)
Iteration 8: log likelihood = -17548.172
Iteration 9: log likelihood = -17544.987
Iteration 10: log likelihood = -17544.937
Iteration 11: log likelihood = -17544.937

Estimating parameters
(setting technique to bhhh)
Iteration 0: log likelihood = -17544.937
Iteration 1: log likelihood = -17544.937

Diagonal vech MGARCH model

```
Sample: 1 - 5000
Number of obs = 5,000
Distribution: Gaussian
 Wald chi2(.) = .
 Log likelihood = -17544.94 Prob > chi2 = .

| Coef. | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|-------|-----------|-------|------|----------------------|
| /Sigma0 |
| 1_1   | 1.026283  | .0823348 | 12.46 | 0.000   | .8649096  | 1.187656 |
| 2_1   | .4300997  | .0590294 | 7.29  | 0.000   | .3144042  | .5457952 |
| 2_2   | 1.019753  | .0837146 | 12.18 | 0.000   | .8556751  | 1.18383  |
| L.ARCH |
| 1_1   | .2878739  | .02157  | 13.35 | 0.000   | .2455975  | .3301504 |
| 2_1   | .1036685  | .0161446 | 6.42  | 0.000   | .0720256  | .1353114 |
| 2_2   | .2034196  | .019855  | 10.25 | 0.000   | .1645044  | .2423347 |
| L2.ARCH |
| 1_1   | .1837825  | .0274555 | 6.69  | 0.000   | .1299706  | .2375943 |
| 2_1   | .0884425  | .02208   | 4.01  | 0.000   | .0451665  | .1317185 |
| 2_2   | .2025718  | .0272639 | 7.43  | 0.000   | .1491355  | .256008  |
| L.GARCH |
| 1_1   | .0782467  | .053944  | 1.45  | 0.147   | -.0274816 | .183975  |
| 2_1   | .2888104  | .0818303 | 3.53  | 0.000   | .1284261  | .4491948 |
| 2_2   | .201618   | .0470584 | 4.28  | 0.000   | .1093853  | .2938508 |
```
Now we use `predict` to obtain the in-sample conditional variances of `acme` and use `tsline` (see [TS] `tsline`) to graph the results.

```
. predict h_acme, variance eq(acme, acme)
. tsline h_acme
```

The graph shows that the predicted conditional variances vary substantially over time, as the parameter estimates indicated.

Because there are no covariates in the model for `acme`, specifying `xb` puts a prediction of 0 in each observation, and specifying `residuals` puts the value of the dependent variable into the prediction.

### Methods and formulas

All one-step predictions are obtained by substituting the parameter estimates into the model. The estimated unconditional variance matrix of the disturbances, $\hat{\Sigma}$, is the initial value for the ARCH and GARCH terms. The postestimation routines recompute $\hat{\Sigma}$ using the prediction sample, the parameter estimates stored in $\mathbf{e}(\mathbf{b})$, and (4) in `Methods and formulas` of [TS] `mgarch dvech`.

For observations in which the residuals are missing, the estimated unconditional variance matrix of the disturbances is used in place of the outer product of the residuals.

Dynamic predictions of the dependent variables use previously predicted values beginning in the period specified by `dynamic()`.

Dynamic variance predictions are implemented by substituting $\hat{\Sigma}$ for the outer product of the residuals beginning in the period specified by `dynamic()`.

### Also see

[TS] `mgarch dvech` — Diagonal vech multivariate GARCH models

[U] 20 Estimation and postestimation commands
mgarch vcc — Varying conditional correlation multivariate GARCH models

Description

mgarch vcc estimates the parameters of varying conditional correlation (VCC) multivariate generalized autoregressive conditionally heteroskedastic (MGARCH) models in which the conditional variances are modeled as univariate generalized autoregressive conditionally heteroskedastic (GARCH) models and the conditional covariances are modeled as nonlinear functions of the conditional variances. The conditional correlation parameters that weight the nonlinear combinations of the conditional variance follow the GARCH-like process specified in Tse and Tsui (2002).

The VCC MGARCH model is about as flexible as the closely related dynamic conditional correlation MGARCH model (see [TS] mgarch dcc), more flexible than the conditional correlation MGARCH model (see [TS] mgarch ccc), and more parsimonious than the diagonal vech model (see [TS] mgarch dvech).

Quick start

Fit varying conditional correlation multivariate GARCH with first- and second-order ARCH components for dependent variables y1 and y2 using tsset data

mgarch vcc (y1 y2), arch(1 2)

Add regressors x1 and x2 and first-order GARCH component

mgarch vcc (y1 y2 = x1 x2), arch(1 2) garch(1)

Add z1 to the model for the conditional heteroskedasticity

mgarch vcc (y1 y2 = x1 x2), arch(1 2) garch(1) het(z1)

Menu

Statistics > Multivariate time series > Multivariate GARCH
Syntax

\texttt{mgarch vcc eq [eq ... eq] [if] [in] [ , options]}

where each \textit{eq} has the form

\[
\text{(depvars = [indepvars] [ , eqoptions])}
\]

\noindent \textbf{options} \hspace{1cm} \textbf{Description}

\begin{tabular}{ll}
\hline
\textbf{Model} & \\
\texttt{arch(numlist)} & ARCH terms for all equations \\
\texttt{garch(numlist)} & GARCH terms for all equations \\
\texttt{het(varlist)} & include \textit{varlist} in the specification of the conditional variance for all equations \\
\texttt{distribution(dist [#])} & use \textit{dist} distribution for errors [may be \texttt{gaussian} (synonym \texttt{normal}) or \texttt{t}; default is \texttt{gaussian}] \\
\texttt{constraints(numlist)} & apply linear constraints \\
\hline
\textbf{SE/Robust} & \\
\texttt{vce(vcetype)} & \textit{vcetype} may be \texttt{oim} or \texttt{robust} \\
\hline
\textbf{Reporting} & \\
\texttt{level(#)} & set confidence level; default is level(95) \\
\texttt{nocnsreport} & do not display constraints \\
\texttt{display_options} & control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling \\
\hline
\textbf{Maximization} & \\
\texttt{maximize_options} & control the maximization process; seldom used \\
\texttt{from(matname)} & initial values for the coefficients; seldom used \\
\texttt{coeflegend} & display legend instead of statistics \\
\hline
\textbf{eqoptions} & \\
\textbf{Description} & \\
\hline
\textbf{Model} & \\
\texttt{noconstant} & suppress constant term in the mean equation \\
\texttt{arch(numlist)} & ARCH terms \\
\texttt{garch(numlist)} & GARCH terms \\
\texttt{het(varlist)} & include \textit{varlist} in the specification of the conditional variance \\
\hline
\end{tabular}

You must \texttt{tsset} your data before using \texttt{mgarch vcc}; see [TS] \texttt{tsset}.

\textit{indepvars} and \textit{varlist} may contain factor variables; see [U] 11.4.3 Factor variables.

\textit{depvars}, \textit{indepvars}, and \textit{varlist} may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, \texttt{fp}, \texttt{rolling}, and \texttt{statsby} are allowed; see [U] 11.1.10 Prefix commands.

\texttt{coeflegend} does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
Options

**Model**

- `arch(numlist)` specifies the ARCH terms for all equations in the model. By default, no ARCH terms are specified.
- `garch(numlist)` specifies the GARCH terms for all equations in the model. By default, no GARCH terms are specified.
- `het(varlist)` specifies that `varlist` be included in the model in the specification of the conditional variance for all equations. This varlist enters the variance specification collectively as multiplicative heteroskedasticity.
- `distribution(dist [#])` specifies the assumed distribution for the errors. `dist` may be `gaussian`, `normal`, or `t`. `gaussian` and `normal` are synonyms; each causes `mgarch vcc` to assume that the errors come from a multivariate normal distribution. `#` may not be specified with either of them.
- `t` causes `mgarch vcc` to assume that the errors follow a multivariate Student t distribution, and the degree-of-freedom parameter is estimated along with the other parameters of the model. If `distribution(t #)` is specified, then `mgarch vcc` uses a multivariate Student t distribution with `#` degrees of freedom. `#` must be greater than 2.
- `constraints(numlist)` specifies linear constraints to apply to the parameter estimates.

**SE/Robust**

- `vce(vcetype)` specifies the estimator for the variance–covariance matrix of the estimator.
  - `vce(oim)`, the default, specifies to use the observed information matrix (OIM) estimator.
  - `vce(robust)` specifies to use the Huber/White/sandwich estimator.

**Reporting**

- `level(#)`, `nocnsreport`; see [R] Estimation options.

**Maximization**

- `maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), nonrtolerance, and from(matname)`; see [R] Maximize for all options except from(), and see below for information on from(). These options are seldom used.
- `from(matname)` specifies initial values for the coefficients. `from(b0)` causes `mgarch vcc` to begin the optimization algorithm with the values in `b0`. `b0` must be a row vector, and the number of columns must equal the number of parameters in the model.

The following option is available with `mgarch vcc` but is not shown in the dialog box:
- `coeflegend`; see [R] Estimation options.
Eqoptions

noconstant suppresses the constant term in the mean equation.
arch(numlist) specifies the ARCH terms in the equation. By default, no ARCH terms are specified. This option may not be specified with model-level arch().
garch(numlist) specifies the GARCH terms in the equation. By default, no GARCH terms are specified. This option may not be specified with model-level garch().
het(varlist) specifies that varlist be included in the specification of the conditional variance. This varlist enters the variance specification collectively as multiplicative heteroskedasticity. This option may not be specified with model-level het().

Remarks and examples

We assume that you have already read [TS] mgarch, which provides an introduction to MGARCH models and the methods implemented in mgarch vcc.

MGARCH models are dynamic multivariate regression models in which the conditional variances and covariances of the errors follow an autoregressive-moving-average structure. The VCC MGARCH model uses a nonlinear combination of univariate GARCH models with time-varying cross-equation weights to model the conditional covariance matrix of the errors.

As discussed in [TS] mgarch, MGARCH models differ in the parsimony and flexibility of their specifications for a time-varying conditional covariance matrix of the disturbances, denoted by $H_t$. In the conditional correlation family of MGARCH models, the diagonal elements of $H_t$ are modeled as univariate GARCH models, whereas the off-diagonal elements are modeled as nonlinear functions of the diagonal terms. In the VCC MGARCH model,

$$ h_{ij,t} = \rho_{ij,t} \sqrt{h_{ii,t} h_{jj,t}} $$

where the diagonal elements $h_{ii,t}$ and $h_{jj,t}$ follow univariate GARCH processes and $\rho_{ij,t}$ follows the dynamic process specified in Tse and Tsui (2002) and discussed below.

Because the $\rho_{ij,t}$ varies with time, this model is known as the VCC GARCH model.

Technical note

The VCC GARCH model proposed by Tse and Tsui (2002) can be written as

$$ y_t = Cx_t + \epsilon_t $$
$$ \epsilon_t = H_t^{1/2} \nu_t $$
$$ H_t = D_t^{1/2} R_t D_t^{1/2} $$
$$ R_t = (1 - \lambda_1 - \lambda_2) R + \lambda_1 \Psi_{t-1} + \lambda_2 R_{t-1} $$

where

$y_t$ is an $m \times 1$ vector of dependent variables;
$C$ is an $m \times k$ matrix of parameters;
$x_t$ is a $k \times 1$ vector of independent variables, which may contain lags of $y_t$;
\( H_t^{1/2} \) is the Cholesky factor of the time-varying conditional covariance matrix \( H_t \); 
\( \nu_t \) is an \( m \times 1 \) vector of independent and identically distributed innovations; 
\( D_t \) is a diagonal matrix of conditional variances,
\[
D_t = \begin{pmatrix}
\sigma_{1,t}^2 & 0 & \cdots & 0 \\
0 & \sigma_{2,t}^2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_{m,t}^2
\end{pmatrix}
\]
in which each \( \sigma_{i,t}^2 \) evolves according to a univariate GARCH model of the form
\[
\sigma_{i,t}^2 = s_i + \sum_{j=1}^{p_i} \alpha_j \epsilon_{i,t-j}^2 + \sum_{j=1}^{q_i} \beta_j \sigma_{i,t-j}^2
\]
by default, or
\[
\sigma_{i,t}^2 = \exp(\gamma_t z_{i,t}) + \sum_{j=1}^{p_i} \alpha_j \epsilon_{i,t-j}^2 + \sum_{j=1}^{q_i} \beta_j \sigma_{i,t-j}^2
\]
when the \texttt{het()} option is specified, where \( \gamma_t \) is a \( 1 \times p \) vector of parameters, \( z_t \) is a \( p \times 1 \) vector of independent variables including a constant term, the \( \alpha_j \)'s are ARCH parameters, and the \( \beta_j \)'s are GARCH parameters;
\( R_t \) is a matrix of conditional correlations,
\[
R_t = \begin{pmatrix}
1 & \rho_{12,t} & \cdots & \rho_{1m,t} \\
\rho_{12,t} & 1 & \cdots & \rho_{2m,t} \\
\vdots & \vdots & \ddots & \vdots \\
\rho_{1m,t} & \rho_{2m,t} & \cdots & 1
\end{pmatrix}
\]
\( R \) is the matrix of means to which the dynamic process in (1) reverts;
\( \Psi_t \) is the rolling estimator of the correlation matrix of \( \tilde{\epsilon}_t \), which uses the previous \( m + 1 \) observations; and
\( \lambda_1 \) and \( \lambda_2 \) are parameters that govern the dynamics of conditional correlations. \( \lambda_1 \) and \( \lambda_2 \) are nonnegative and satisfy \( 0 \leq \lambda_1 + \lambda_2 < 1 \).

To differentiate this model from Engle (2002), Tse and Tsui (2002) call their model a VCC MGARCH model.

Some examples

\textbf{Example 1: Model with common covariates}

We have daily data on the stock returns of three car manufacturers—Toyota, Nissan, and Honda, from January 2, 2003, to December 31, 2010—in the variables \texttt{toyota}, \texttt{nissan}, and \texttt{honda}. We model the conditional means of the returns as a first-order vector autoregressive process and the conditional covariances as a VCC MGARCH process in which the variance of each disturbance term follows a GARCH(1,1) process.
. use https://www.stata-press.com/data/r16/stocks
(Data from Yahoo! Finance)
. mgarch vcc (toyota nissan honda = L.toyota L.nissan L.honda, noconstant),
> arch(1) garch(1)
Calculating starting values....
Optimizing log likelihood
(setting technique to bhhh)
Iteration 0:  log likelihood =  16901.2
Iteration 1:  log likelihood =  17028.644
Iteration 2:  log likelihood =  17145.905
Iteration 3:  log likelihood =  17251.485
Iteration 4:  log likelihood =  17306.115
Iteration 5:  log likelihood =  17332.59
Iteration 6:  log likelihood =  17353.617
Iteration 7:  log likelihood =  17374.86
Iteration 8:  log likelihood =  17398.526
Iteration 9:  log likelihood =  17418.748
(switching technique to nr)
Iteration 10: log likelihood =  17442.552
Iteration 11: log likelihood =  17455.64
Iteration 12: log likelihood =  17463.593
Iteration 13: log likelihood =  17463.922
Iteration 14: log likelihood =  17463.925
Iteration 15: log likelihood =  17463.925
Refining estimates
Iteration 0:  log likelihood =  17463.925
Iteration 1:  log likelihood =  17463.925
Varying conditional correlation MGARCH model
Sample:  2 - 2015  Number of obs =  2,014
Distribution:  Gaussian     Wald chi2(9) =   17.67
Log likelihood =  17463.92  Prob > chi2 =  0.0392

|          | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|----------|--------|-----------|-------|------|----------------------|
| toyota   |        |           |       |      |                      |
|      L1. | -.0565645 | .0335696 | -1.68 | 0.092 | -.1223597 -.0092307  |
| nissan   |        |           |       |      |                      |
|      L1. | .0248101 | .0252701  | 0.98  | 0.326 | -.0247184 .0743385  |
| honda    |        |           |       |      |                      |
|      L1. | .0035836 | .0298895  | 0.12  | 0.905 | -.0549986 .0621659  |
| ARCH_toyota |       |           |       |      |                      |
| arch     |        |           |       |      |                      |
|      L1. | .0602807 | .0086799  | 6.94  | 0.000 | .0432684 .0772929   |
| garch    |        |           |       |      |                      |
|      L1. | .922469 | .0110316  | 83.62 | 0.000 | .9008474 .9440906   |
| _cons   | 4.38e-06 | 1.12e-06  | 3.91  | 0.000 | 2.18e-06 6.58e-06  |
The output has three parts: an iteration log, a header, and an output table.

The iteration log has three parts: the dots from the search for initial values, the iteration log from optimizing the log likelihood, and the iteration log from the refining step. A detailed discussion of the optimization methods is in Methods and formulas.

The header describes the estimation sample and reports a Wald test against the null hypothesis that all the coefficients on the independent variables in the mean equations are zero. Here the null hypothesis is rejected at the 5% level.
The output table first presents results for the mean or variance parameters used to model each dependent variable. Subsequently, the output table presents results for the parameters in \( \mathbf{R} \). For example, the estimate of the mean of the process that associates Toyota and Nissan is 0.66. Finally, the output table presents results for the adjustment parameters \( \lambda_1 \) and \( \lambda_2 \). In the example at hand, the estimates for both \( \lambda_1 \) and \( \lambda_2 \) are statistically significant.

The VCC MGARCH model reduces to the CCC MGARCH model when \( \lambda_1 = \lambda_2 = 0 \). The output below shows that a Wald test rejects the null hypothesis that \( \lambda_1 = \lambda_2 = 0 \) at all conventional levels.

```plaintext
. test _b[/Adjustment:lambda1] = _b[/Adjustment:lambda2] = 0
  ( 1) [/Adjustment]lambda1 - [/Adjustment]lambda2 = 0
  ( 2) [/Adjustment]lambda1 = 0
    chi2( 2) = 482.80
    Prob > chi2 = 0.0000
```

These results indicate that the assumption of time-invariant conditional correlations maintained in the CCC MGARCH model is too restrictive for these data.

Example 2: Model with covariates that differ by equation

We improve the previous example by removing the insignificant parameters from the model. To accomplish that, we specify the `honda` equation separately from the `toyota` and `nissan` equations:

```plaintext
. mgarch vcc (toyota nissan = , noconstant) (honda = L.nissan, noconstant),
    > arch(1) garch(1)
Calculating starting values....
Optimizing log likelihood
  (setting technique to bhhh)
Iteration 0:  log likelihood =  16889.43
Iteration 1:  log likelihood =  17002.567
Iteration 2:  log likelihood =  17134.525
Iteration 3:  log likelihood =  17233.192
Iteration 4:  log likelihood =  17295.342
Iteration 5:  log likelihood =  17323.192
Iteration 6:  log likelihood =  17348.063
Iteration 7:  log likelihood =  17363.988
Iteration 8:  log likelihood =  17387.216
Iteration 9:  log likelihood =  17404.734
  (switching technique to nr)
Iteration 10: log likelihood =  17438.432  (not concave)
Iteration 11: log likelihood =  17450.002
Iteration 12: log likelihood =  17455.443
Iteration 13: log likelihood =  17455.971
Iteration 14: log likelihood =  17455.98
Iteration 15: log likelihood =  17455.98
Refining estimates
Iteration 0:  log likelihood =  17455.98
Iteration 1:  log likelihood =  17455.98  (backed up)
```
It turns out that the coefficient on L1.nissan in the honda equation is now statistically insignificant. We could further improve the model by removing L1.nissan from the model.

There is no mean equation for Toyota or Nissan. In [TS] mgarch vcc postestimation, we discuss prediction from models without covariates.
Example 3: Model with constraints

Here we fit a bivariate VCC MGARCH model for the Toyota and Nissan shares. We believe that the shares of these car manufacturers follow the same process, so we impose the constraints that the ARCH coefficients are the same for the two companies and that the GARCH coefficients are also the same.

\[
\begin{align*}
\text{constraint 1 } & \quad \_b[\text{ARCH}_\text{toyota}:L.\text{arch}] = \_b[\text{ARCH}_\text{nissan}:L.\text{arch}] \\
\text{constraint 2 } & \quad \_b[\text{ARCH}_\text{toyota}:L.\text{garch}] = \_b[\text{ARCH}_\text{nissan}:L.\text{garch}] \\
\text{. mgarch vcc (toyota nissan = , noconstant), arch(1) garch(1) constraints(1 2)} \\
\text{Calculating starting values....}
\end{align*}
\]

Optimizing log likelihood

(setting technique to bhhh)

Iteration 0:  log likelihood = 10326.298
Iteration 1:  log likelihood = 10680.73
Iteration 2:  log likelihood = 10881.388
Iteration 3:  log likelihood = 11043.345
Iteration 4:  log likelihood = 11122.459
Iteration 5:  log likelihood = 11202.411
Iteration 6:  log likelihood = 11253.657
Iteration 7:  log likelihood = 11276.325
Iteration 8:  log likelihood = 11279.823
Iteration 9:  log likelihood = 11281.704

(switching technique to nr)

Iteration 10: log likelihood = 11282.313
Iteration 11: log likelihood = 11282.46
Iteration 12: log likelihood = 11282.461
Refining estimates
Iteration 0:  log likelihood = 11282.461
Iteration 1:  log likelihood = 11282.461  (backed up)
Varying conditional correlation MGARCH model

Sample:  1 - 2015  Number of obs =  2,015
Distribution: Gaussian  Wald chi2(.) = .
Log likelihood = 11282.46  Prob > chi2 = .
( 1)  [ARCH_toyota]L.arch - [ARCH_nissan]L.arch = 0
( 2)  [ARCH_toyota]L.garch - [ARCH_nissan]L.garch = 0

|                     | Coef.    | Std. Err. |    z  |     P>|z|    |   [95% Conf. Interval] |
|---------------------|----------|-----------|-------|---------|------------------------|
| ARCH_toyota         |          |           |       |         |                        |
| arch L1.            | 0.0797459| 0.0101634 | 7.85  | 0.000   | 0.059826               |
|                     |          |           |       |         | 0.0996659              |
| garch L1.           | 0.9063808| 0.0118211 | 76.67 | 0.000   | 0.883212               |
|                     |          |           |       |         | 0.9295497              |
| _cons               | 4.24e-06 | 1.10e-06  | 3.85  | 0.000   | 2.08e-06               |
|                     |          |           |       |         | 6.40e-06               |
| ARCH_nissan         |          |           |       |         |                        |
| arch L1.            | 0.0797459| 0.0101634 | 7.85  | 0.000   | 0.059826               |
|                     |          |           |       |         | 0.0996659              |
| garch L1.           | 0.9063808| 0.0118211 | 76.67 | 0.000   | 0.883212               |
|                     |          |           |       |         | 0.9295497              |
| _cons               | 5.91e-06 | 1.47e-06  | 4.03  | 0.000   | 3.03e-06               |
|                     |          |           |       |         | 8.79e-06               |
| corr(toyota, nissan)| .6720056 | .0162585  | 41.33 | 0.000   | .6401394               |
|                     |          |           |       |         | .7038718               |
| /Adjustment         |          |           |       |         |                        |
| lambda1             | 0.0343012| 0.0128097 | 2.68  | 0.007   | 0.0091945              |
|                     |          |           |       |         | 0.0594078              |
| lambda2             | 0.7945548| 0.101067  | 7.86  | 0.000   | 0.5964671              |
|                     |          |           |       |         | 0.9926425              |

We could test our constraints by fitting the unconstrained model and performing a likelihood-ratio test. The results indicate that the restricted model is preferable.

Example 4: Model with a GARCH term

In this example, we have data on fictional stock returns for the Acme and Anvil corporations, and we believe that the movement of the two stocks is governed by different processes. We specify one ARCH and one GARCH term for the conditional variance equation for Acme and two ARCH terms for the conditional variance equation for Anvil. In addition, we include the lagged value of the stock return for Apex, the main subsidiary of Anvil corporation, in the variance equation of Anvil. For Acme, we have data on the changes in an index of futures prices of products related to those produced by Acme in afrelated. For Anvil, we have data on the changes in an index of futures prices of inputs used by Anvil in afinputs.

. use https://www.stata-press.com/data/r16/acmeh
. mgarch vcc (acme = afrelated, noconstant arch(1) garch(1))
   > (anvil = afinputs, arch(1/2) het(L.apex))
   Calculating starting values....
Optimizing log likelihood
(setting technique to bhhh)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>log likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-13252.793</td>
</tr>
<tr>
<td>1</td>
<td>-12859.124</td>
</tr>
<tr>
<td>2</td>
<td>-12522.14</td>
</tr>
<tr>
<td>3</td>
<td>-12406.487</td>
</tr>
<tr>
<td>4</td>
<td>-12304.275</td>
</tr>
<tr>
<td>5</td>
<td>-12273.103</td>
</tr>
<tr>
<td>6</td>
<td>-12256.104</td>
</tr>
<tr>
<td>7</td>
<td>-12254.55</td>
</tr>
<tr>
<td>8</td>
<td>-12254.482</td>
</tr>
<tr>
<td>9</td>
<td>-12254.478</td>
</tr>
<tr>
<td>10</td>
<td>-12254.478</td>
</tr>
<tr>
<td>11</td>
<td>-12254.478</td>
</tr>
</tbody>
</table>

(switching technique to nr)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>log likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-12254.478</td>
</tr>
<tr>
<td>11</td>
<td>-12254.478</td>
</tr>
</tbody>
</table>

Refining estimates

<table>
<thead>
<tr>
<th>Iteration</th>
<th>log likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-12254.478</td>
</tr>
<tr>
<td>1</td>
<td>-12254.478</td>
</tr>
</tbody>
</table>

Varying conditional correlation MGARCH model

Sample: 2 - 2500  Number of obs = 2,499
Distribution: Gaussian  Wald chi2(2) = 5226.19
Log likelihood = -12254.48  Prob > chi2 = 0.0000

|          | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|----------|--------|-----------|-------|------|----------------------|
| acme     |        |           |       |      |                      |
| afrelated| .9672465 | .0510066  | 18.96 | 0.000| .8672753 1.067218    |
| ARCH_acme|        |           |       |      |                      |
| arch     |        |           |       |      |                      |
| L1.      | .0949142 | .0147302  | 6.44  | 0.000| .0660435  .1237849   |
| garch    |        |           |       |      |                      |
| L1.      | .7689442 | .038885   | 19.77 | 0.000| .6927309  .8451574   |
| _cons    | 2.129468 | .464916   | 4.58  | 0.000| 1.218249 3.040687    |
| anvil    |        |           |       |      |                      |
| afinputs | -1.018629 | .0145027 | -70.24 | 0.000| -1.047053 -.9902037 |
| _cons    | .1015986 | .0177952  | 5.71  | 0.000| .0667205  .1364766   |
| ARCH_anvil|       |           |       |      |                      |
| arch     |        |           |       |      |                      |
| L1.      | .4990272 | .0243531  | 20.49 | 0.000| .4512959  .5467584   |
| L2.      | .2839812 | .0181966  | 15.61 | 0.000| .2483165  .3196459   |
| apex     |        |           |       |      |                      |
| L1.      | 1.897144 | .0558791  | 33.95 | 0.000| 1.787623 2.006665    |
| _cons    | .0682724 | .0662257  | 1.03  | 0.303| -.0615276 .1980724   |
| corr(acme, anvil) | -.6574256 | .0294259 | -22.34 | 0.000| -.7150994 -.5997518 |
| /Adjustment |        |           |       |      |                      |
| lambda1  | .2375029 | .0179114  | 13.26 | 0.000| .2023971  .2726086   |
| lambda2  | .6492072 | .0254493  | 25.51 | 0.000| .5993274  .6990869   |

Adjustment

|          | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|----------|--------|-----------|-------|------|----------------------|
| lambda1  | .2375029 | .0179114  | 13.26 | 0.000| .2023971  .2726086   |
| lambda2  | .6492072 | .0254493  | 25.51 | 0.000| .5993274  .6990869   |
The results indicate that increases in the futures prices for related products lead to higher returns on the Acme stock, and increased input prices lead to lower returns on the Anvil stock. In the conditional variance equation for Anvil, the coefficient on $L_1 \cdot \text{ apex}$ is positive and significant, which indicates that an increase in the return on the Apex stock leads to more variability in the return on the Anvil stock.

Stored results

`mgarch vcc` stores the following in `e()`:

Scalars
- `e(N)` number of observations
- `e(k)` number of parameters
- `e(k_extra)` number of extra estimates added to `_b`
- `e(k_eq)` number of equations in `e(b)`
- `e(k_dv)` number of dependent variables
- `e(df_m)` model degrees of freedom
- `e(ll)` log likelihood
- `e(chi2)` $\chi^2$-value for model test
- `e(p)` $p$-value for model test
- `e(estdf)` 1 if distribution parameter was estimated, 0 otherwise
- `e(usr)` user-provided distribution parameter
- `e(tmin)` minimum time in sample
- `e(tmax)` maximum time in sample
- `e(N_gaps)` number of gaps
- `e(rank)` rank of `e(V)`
- `e(ic)` number of iterations
- `e(rc)` return code
- `e(converged)` 1 if converged, 0 otherwise

Macros
- `e(cmd)` `mgarch`
- `e(model)` `vcc`
- `e(cmdline)` command as typed
- `e(depvar)` names of dependent variables
- `e(covariates)` list of covariates
- `e(dv_eqs)` dependent variables with mean equations
- `e(indeps)` independent variables in each equation
- `e(tvar)` time variable
- `e(hetvars)` variables included in the conditional variance equations
- `e(title)` title in estimation output
- `e(chi2type)` Wald; type of model $\chi^2$ test
- `e(vcex)` vcetype specified in `vce()`
- `e(vcetype)` title used to label Std. Err.
- `e(tmins)` formatted minimum time
- `e(tmaxs)` formatted maximum time
- `e(dist)` distribution for error term: `gaussian` or `t`
- `e(arch)` specified ARCH terms
- `e(garch)` specified GARCH terms
- `e(technique)` maximization technique
- `e(properties)` `b V`
- `e(estat_cmd)` program used to implement `estat`
- `e(predict)` program used to implement `predict`
- `e(marginsok)` predictions allowed by `margins`
- `e(marginsnotok)` predictions disallowed by `margins`
- `e(marginsdefault)` default `predict()` specification for `margins`
- `e(asbalanced)` factor variables `fvset` as `asbalanced`
- `e(asobserved)` factor variables `fvset` as `asobserved`
Matrices
- `e(b)` coefficient vector
- `e(Cns)` constraints matrix
- `e(ilog)` iteration log (up to 20 iterations)
- `e(gradient)` gradient vector
- `e(hessian)` Hessian matrix
- `e(V)` variance–covariance matrix of the estimators
- `e(V_modelbased)` model-based variance
- `e(pinfo)` parameter information, used by `predict`

Functions
- `e(sample)` marks estimation sample

Methods and formulas

`mgarch vcc` estimates the parameters of the varying conditional correlation MGARCH model by maximum likelihood. The log-likelihood function based on the multivariate normal distribution for observation \( t \) is

\[
l_t = -0.5m \log(2\pi) - 0.5 \log {\det(R_t)} - \log \left\{ \det \left( D_t^{1/2} \right) \right\} - 0.5 \tilde{\epsilon}_t R_t^{-1} \tilde{\epsilon}_t'
\]

where \( \tilde{\epsilon}_t = D_t^{-1/2} \epsilon_t \) is an \( m \times 1 \) vector of standardized residuals, \( \epsilon_t = y_t - Cx_t \). The log-likelihood function is \( \sum_{t=1}^{T} l_t \).

If we assume that \( \nu_t \) follow a multivariate \( t \) distribution with degrees of freedom (df) greater than 2, then the log-likelihood function for observation \( t \) is

\[
l_t = \log \Gamma \left( \frac{\text{df} + m}{2} \right) - \log \Gamma \left( \frac{\text{df}}{2} \right) - \frac{m}{2} \log \left\{ (\text{df} - 2)\pi \right\} - 0.5 \log {\det(R_t)} - \log \left\{ \det \left( D_t^{1/2} \right) \right\} - \frac{\text{df} + m}{2} \log \left( 1 + \tilde{\epsilon}_t R_t^{-1} \tilde{\epsilon}_t' \frac{\text{df}}{\text{df} - 2} \right)
\]

The starting values for the parameters in the mean equations and the initial residuals \( \hat{\epsilon}_t \) are obtained by least-squares regression. The starting values for the parameters in the variance equations are obtained by a procedure proposed by Gourieroux and Monfort (1997, sec. 6.2.2). The starting values for the parameters in \( R \) are calculated from the standardized residuals \( \tilde{\epsilon}_t \). Given the starting values for the mean and variance equations, the starting values for the parameters \( \lambda_1 \) and \( \lambda_2 \) are obtained from a grid search performed on the log likelihood.

The initial optimization step is performed in the unconstrained space. Once the maximum is found, we impose the constraints \( \lambda_1 \geq 0, \lambda_2 \geq 0, \) and \( 0 \leq \lambda_1 + \lambda_2 < 1 \), and maximize the log likelihood in the constrained space. This step is reported in the iteration log as the refining step.

GARCH estimators require initial values that can be plugged in for \( \epsilon_{t-i} \epsilon_{t-i}' \) and \( H_{t-j} \) when \( t-i < 1 \) and \( t-j < 1 \). `mgarch vcc` substitutes an estimator of the unconditional covariance of the disturbances

\[
\hat{\Sigma} = T^{-1} \sum_{t=1}^{T} \tilde{\epsilon}_t \tilde{\epsilon}_t'
\]
for $\epsilon_{t-i}\epsilon_{t-i}'$ when $t - i < 1$ and for $H_{t-j}$ when $t - j < 1$, where $\hat{\epsilon}_t$ is the vector of residuals calculated using the estimated parameters.

`mgarch vcc` uses numerical derivatives in maximizing the log-likelihood function.

**References**


**Also see**

[T] `mgarch vcc postestimation` — Postestimation tools for mgarch vcc

[T] `arch` — Autoregressive conditional heteroskedasticity (ARCH) family of estimators

[T] `mgarch` — Multivariate GARCH models

[T] `tsset` — Declare data to be time-series data

[T] `var` — Vector autoregressive models

[U] 20 Estimation and postestimation commands
The following standard postestimation commands are available after `mgarch vcc`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contrast</code></td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>pwcompare</code></td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and conditional variances, covariances, and correlations. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [ type ] { stub* | newvarlist } [ if ] [ in ] [ , statistic options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
<tr>
<td>variance</td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td>correlation</td>
<td>conditional correlations</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>equation( eqnames )</td>
<td>names of equations for which predictions are made</td>
</tr>
<tr>
<td>dynamic( time_constant )</td>
<td>begin dynamic forecast at specified time</td>
</tr>
</tbody>
</table>

Options for predict

```
xb, the default, calculates the linear predictions of the dependent variables.
residuals calculates the residuals.
variance predicts the conditional variances and conditional covariances.
correlation predicts the conditional correlations.
```

equation( eqnames ) specifies the equation for which the predictions are calculated. Use this option to predict a statistic for a particular equation. Equation names, such as `equation(income)`, are used to identify equations.
One equation name may be specified when predicting the dependent variable, the residuals, or the conditional variance. For example, specifying `equation(income)` causes `predict` to predict `income`, and specifying `variance equation(income)` causes `predict` to predict the conditional variance of `income`.

Two equations may be specified when predicting a conditional variance or covariance. For example, specifying `equation(income, consumption) variance` causes `predict` to predict the conditional covariance of `income` and `consumption`.

dynamic(`time_constant`) specifies when `predict` starts producing dynamic forecasts. The specified `time_constant` must be in the scale of the time variable specified in `tsset`, and the `time_constant` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may not be specified with `residuals`.

**margins**

**Description for margins**

`margins` estimates margins of response for linear predictions and conditional variances, covariances, and correlations. All predictions are available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

**Menu for margins**

Statistics > Postestimation

**Syntax for margins**

```plaintext
margins [marginlist] [ , options ]
margins [marginlist], predict(statistic ...) [predict(statistic ...) ...] [options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td><code>xb</code></td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td><code>variance</code></td>
<td>conditional variances and covariances</td>
</tr>
<tr>
<td><code>correlation</code></td>
<td>conditional correlations</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>not allowed with <code>margins</code></td>
</tr>
</tbody>
</table>

`xb` defaults to the first equation.

Statistics not allowed with `margins` are functions of stochastic quantities other than `e(b)`.

For the full syntax, see [R] `margins`. 
## Remarks and examples

We assume that you have already read [TS] `mgarch vcc`. In this entry, we use `predict` after `mgarch vcc` to make in-sample and out-of-sample forecasts.

### Example 1: Dynamic forecasts

In this example, we obtain dynamic forecasts for the Toyota, Nissan, and Honda stock returns modeled in example 2 of [TS] `mgarch vcc`. In the output below, we reestimate the parameters of the model, use `tsappend` (see [TS] `tsappend`) to extend the data, and use `predict` to obtain in-sample one-step-ahead forecasts and dynamic forecasts of the conditional variances of the returns. We graph the forecasts below.

```stata
. use https://www.stata-press.com/data/r16/stocks
(Data from Yahoo! Finance)
. quietly mgarch vcc (toyota nissan = , noconstant)
> (honda = L.nissan, noconstant), arch(1) garch(1)
. tsappend, add(50)
. predict H*, variance dynamic(2016)
```

![Graph of dynamic forecasts](image.png)

Recent in-sample one-step-ahead forecasts are plotted to the left of the vertical line in the above graph, and the dynamic out-of-sample forecasts appear to the right of the vertical line. The graph shows the tail end of the huge increase in return volatility that took place in 2008 and 2009. It also shows that the dynamic forecasts quickly converge.

### Methods and formulas

All one-step predictions are obtained by substituting the parameter estimates into the model. The estimated unconditional variance matrix of the disturbances, \( \hat{\Sigma} \), is the initial value for the ARCH and GARCH terms. The postestimation routines recompute \( \hat{\Sigma} \) using the prediction sample, the parameter estimates stored in \( \text{e}(b) \), and (2) in *Methods and formulas* of [TS] `mgarch vcc`.

For observations in which the residuals are missing, the estimated unconditional variance matrix of the disturbances is used in place of the outer product of the residuals.
Dynamic predictions of the dependent variables use previously predicted values beginning in the period specified by `dynamic()`.

Dynamic variance predictions are implemented by substituting $\hat{\Sigma}$ for the outer product of the residuals beginning in the period specified in `dynamic()`.

Also see

[TS] mgarch vcc — Varying conditional correlation multivariate GARCH models

[U] 20 Estimation and postestimation commands
**mswitch — Markov-switching regression models**

**Description**

`mswitch` fits dynamic regression models that exhibit different dynamics across unobserved states using state-dependent parameters to accommodate structural breaks or other multiple-state phenomena. These models are known as Markov-switching models because the transitions between the unobserved states follow a Markov chain.

Two models are available: Markov-switching dynamic regression (MSDR) models that allow a quick adjustment after the process changes state and Markov-switching autoregression (MSAR) models that allow a more gradual adjustment.

**Quick start**

**MSDR model for the dependent variable y with two state-dependent intercepts using `tsset` data**

```
mswitch dr y
```

Same as above
```
mswitch dr y, states(2)
```

As above, but with three states and switching coefficients on x
```
mswitch dr y, switch(x) states(3)
```

MSDR model with two state-dependent intercepts and variance parameters
```
mswitch dr y, varswitch
```

MSAR model with two state-dependent intercepts and an autoregression (AR) term
```
mswitch ar y, ar(1)
```

As above, but with switching AR coefficients
```
mswitch ar y, ar(1) arswitch
```

**Menu**

Statistics > Time series > Markov-switching model
Syntax

**Markov-switching dynamic regression**

```bash
mswitch dr depvar [nonswitch_varlist] [if] [in] [, options]
```

**Markov-switching AR**

```bash
mswitch ar depvar [nonswitch_varlist], ar(numlist) [msar_options options]
```

`nonswitch_varlist` is a list of variables with state-invariant coefficients.

### options

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td></td>
</tr>
<tr>
<td>states(#)</td>
<td>specify number of states; default is states(2)</td>
</tr>
<tr>
<td>switch([varlist], noconstant)</td>
<td>specify variables with switching coefficients; by default, the constant term is state dependent unless switch(), noconstant) is specified</td>
</tr>
<tr>
<td>constant</td>
<td>allow a state-invariant constant term; may be specified only with switch(), noconstant)</td>
</tr>
<tr>
<td>varswitch</td>
<td>specify state-dependent variance parameters; by default, the variance parameter is constant across all states</td>
</tr>
<tr>
<td>p0(type)</td>
<td>specify initial unconditional probabilities where type is one of transition, fixed, or smoothed; the default is p0(transition)</td>
</tr>
<tr>
<td>constraints(numlist)</td>
<td>apply specified linear constraints</td>
</tr>
<tr>
<td>SE/Robust</td>
<td></td>
</tr>
<tr>
<td>vce(vcetype)</td>
<td>vcetype may be oim or robust</td>
</tr>
<tr>
<td>Reporting</td>
<td></td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>noconsreport</td>
<td>do not display constraints</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling</td>
</tr>
<tr>
<td>EM options</td>
<td></td>
</tr>
<tr>
<td>emiterate(#)</td>
<td>specify the number of expectation-maximization (EM) iterations; default is emiterate(10)</td>
</tr>
<tr>
<td>emlog</td>
<td>show EM iteration log</td>
</tr>
<tr>
<td>emdots</td>
<td>show EM iterations as dots</td>
</tr>
<tr>
<td>Maximization</td>
<td></td>
</tr>
<tr>
<td>maximize_options</td>
<td>control the maximization process</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>
**msar_options**

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ar(numlist)</em></td>
<td>specify the number of AR terms</td>
</tr>
<tr>
<td>arswitch</td>
<td>specify state-dependent AR coefficients</td>
</tr>
</tbody>
</table>

*ar(numlist)* is required.

You must *tsset* your data before using *mswitch*; see [TS] tsset.

*varlist* and *nonswitch_varlist* may contain factor variables; see [U] 11.4.3 Factor variables.

*depvar*, *nonswitch_varlist*, and *varlist* may contain time-series operators; see [U] 11.4.4 Time-series varlists.

*by*, *rolling*, and *statsby* are allowed; see [U] 11.1.10 Prefix commands.

*coeflegend* does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

**Options**

**Model**

- **ar(numlist)** specifies the number of AR terms. This option may be specified only with command *mswitch ar*. *ar()* is required to fit AR models.

- **arswitch** specifies that the AR coefficients vary over the states. *arswitch* may be specified only with option *ar()*.

**Main**

- **states(#)** specifies the number of states. The default is states(2).

- **switch([varlist][, noconstant])** specifies variables whose coefficients vary over the states. By default, the constant term is state dependent and is included in the regression model. You may suppress the constant term by specifying *switch(*, noconstant)*.

- **constant** specifies that a state-invariant constant term be included in the model. This option may be specified only with *switch(*, noconstant)*.

- **varswitch** specifies that the variance parameters are state dependent. The default is constant variance across all states.

- **p0(type)** is rarely used. This option specifies the method for obtaining values for the unconditional transition probabilities. *type* is one of transition, fixed, or smoothed. The default is *p0*(transition), which specifies that the values be computed using the matrix of conditional transition probabilities. Type fixed specifies that each unconditional probability is $1/k$, where $k$ is the number of states. Type smoothed specifies that the unconditional probabilities be estimated as extra parameters of the model.

- **constraints(numlist)** specifies the linear constraints to be applied to the parameter estimates.

**SE/Robust**

- **vce(vcetype)** specifies the type of standard error reported, which includes types that are derived from asymptotic theory (oim) and that are robust to some kinds of misspecification (robust); see [R] vce_option.
Reporting

level(#), nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels,
allbaselevels, noflabel, fwrap(#), fwrapon(style), cformat(%fmt), pformat(%fmt),
sformat(%fmt), and nolstretch; see [R] Estimation options.

EM options

emiterate(#), emlog, and emdots control the EM iterations that take place before estimation
switches to a quasi-Newton method. EM is used to obtain starting values.

emiterate(#) specifies the number of EM iterations; the default is emiterate(10).

emlog specifies that the EM iteration log be shown. The default is to not display the EM iteration
log.

emdots specifies that the EM iterations be shown as dots. The default is to not display the dots.

Maximization

maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace,
gradiant, showstep, hessian, showtolerance, tolerance(#), ltolerance(#),
nrtolerance(#), nonrtolerance, and from(matname); see [R] Maximize for all options
except from(), and see below for information on from().

from(matname) specifies initial values for the maximization process. If from() is specified, the
initial values are used in the EM step to improve the likelihood unless emiterate(0) is also
specified. The coefficients obtained at the end of the EM iterations serve as initial values for
the quasi-Newton method.

matname must be a row vector. The number of columns must equal the number of parameters
in the model, and the values must be in the same order as the parameters in e(b).

The following option is available with mswitch but is not shown in the dialog box:
coeflegend; see [R] Estimation options.

Remarks and examples

mswitch fits Markov-switching models in which the parameters vary over states. The states are
unobserved and follow a Markov process.

mswitch dr fits MSDR models that allow a quick adjustment after a state change and are often
used to model monthly and higher-frequency data. mswitch ar fits MSAR models that allow a more
gradual adjustment after a state change and are often used to model quarterly and lower-frequency
data. Estimation is by maximum likelihood. You must tsset your data before using mswitch; see
[TS] tsset.

Remarks are presented under the following headings:

Introduction
Markov-switching dynamic regression
Markov-switching AR
Video example

If you are new to Markov-switching models, we recommend that you begin with Introduction. A
more technical discussion and examples are presented in the model-specific sections.
Introduction

Markov-switching models are widely applied in the social sciences. For example, in economics, the growth rate of Gross Domestic Product is modeled as a switching process to capture the asymmetrical behavior observed over expansions and recessions (Hamilton 1989). Other economic examples include modeling interest rates (Garcia and Perron 1996) and exchange rates (Engel and Hamilton 1990). In finance, Kim, Nelson, and Startz (1998) model monthly stock returns, while Guidolin (2011b, 2011a) provide many applications of these models to returns, portfolio choice, and asset pricing. In political science, Jones, Kim, and Startz (2010) model Democratic and Republican partisan states in the United States Congress.

These models are also used in health sciences. For example, in psychology, Markov-switching models have been applied to daily data on manic and depressive states for individuals with rapid-cycling bipolar disorder (Hamaker, Grasman, and Kamphuis 2010). In epidemiology, Lu, Zeng, and Chen (2010) and Martinez-Beneito et al. (2008) model the incidence rate of infectious disease in epidemic and nonepidemic states.


Markov-switching models are used for series that are believed to transition over a finite set of unobserved states, allowing the process to evolve differently in each state. The transitions occur according to a Markov process. The time of transition from one state to another and the duration between changes in state is random. For example, these models can be used to understand the process that governs the time at which economic growth transitions between expansion and recession and the duration of each period.

Consider the evolution of a series $y_t$, where $t = 1, 2, \ldots, T$, is characterized by two states, as in the models below

State 1: \[ y_t = \mu_1 + \varepsilon_t \]
State 2: \[ y_t = \mu_2 + \varepsilon_t \]

where $\mu_1$ and $\mu_2$ are the intercept terms in state 1 and state 2, respectively. $\varepsilon_t$ is a white noise error with variance $\sigma^2$. The two states model shifts in the intercept term. If the timing of switches is known, the above model can be expressed as

\[ y_t = s_t \mu_1 + (1 - s_t) \mu_2 + \varepsilon_t \]

where $s_t$ is 1 if the process is in state 1 and 0 otherwise. The above model is a regression with dummy variables and could be estimated with ordinary least squares using, for example, \texttt{regress}.

However, in the case of interest, we never know in which state the process is; that is to say, $s_t$ is not observed. Markov-switching regression models allow the parameters to vary over the unobserved states. In the simplest case, we can express this model as a MSDR model with a state-dependent intercept term

\[ y_t = \mu_{s_t} + \varepsilon_t \]

where $\mu_{s_t}$ is the parameter of interest; $\mu_{s_t} = \mu_1$ when $s_t = 1$, and $\mu_{s_t} = \mu_2$ when $s_t = 2$.

Although one never knows with certainty in which state the process lies, the probabilities of being in each state can be estimated. For a Markov process, the transition probabilities are of greater interest. One-step transition probabilities are given by $p_{s_t,s_{t+1}}$, so for a two-state process, $p_{11}$ denotes the probability of staying in state 1 in the next period given that the process is in state 1 in the current period. Likewise, $p_{22}$ denotes the probability of staying in state 2. Values closer to 1 indicate a more persistent process, or in other words, that it is expected to stay in a given state for a long time.
Estimation of Markov-switching models proceeds by predicting the probabilities of the unobserved state and updating the likelihood at each period, akin to the Kalman filter. However, while the Kalman filter is concerned with making linear updates on continuous latent variables, the filter developed in Hamilton (1989) is a nonlinear algorithm that estimates the probabilities that a discrete, latent variable is in one of several states. Also see Hamilton (1990) for estimation of the model parameters by an EM algorithm, which is also a robust method to find reasonable starting values.

Markov-switching dynamic regression

In this section, we use a series of examples to describe MSDR models and the mswitch dr command.

MSDR models allow a quick adjustment after the process changes state. These models are often used to model monthly and higher-frequency data. When the process is in state \( s \) at time \( t \), a general specification of the MSDR model is written as

\[
y_t = \mu_{s_t} + x_t\alpha + z_t\beta_{s_t} + \varepsilon_s
\]

where \( y_t \) is the dependent variable, \( \mu_{s_t} \) is the state-dependent intercept, \( x_t \) is a vector of exogenous variables with state-invariant coefficients \( \alpha \), \( z_t \) is a vector of exogenous variables with state-dependent coefficients \( \beta_{s_t} \), and \( \varepsilon_s \) is an independent and identically distributed (i.i.d.) normal error with mean 0 and state-dependent variance \( \sigma^2_{s_t} \). \( x_t \) and \( z_t \) may contain lags of \( y_t \). MSDR models allow states to switch according to a Markov process as described in Markov-switching regression models under Methods and formulas.

In the default model fit by mswitch dr, \( s = 2 \) and a constant \( \sigma^2 \) is assumed (\( \sigma^2_1 = \sigma^2_2 = \sigma^2 \)), so three parameters, \( \mu_1, \mu_2 \), and \( \sigma^2 \), are estimated. There is no \( x_t \) or \( z_t \). The number of states may be increased with option states(). To include \( x_t \), you specify a varlist after the command name, and to include \( z_t \), you specify option switch(). The assumption of constant variances over states may be relaxed with option varswitch.

A more complete discussion of the MSDR model is provided in Specification of Markov-switching models under Methods and formulas.

Example 1: MSDR model with switching intercepts

Suppose we wish to model the federal funds rate. One potential model is a constant-only model

\[
r_t = \mu_{s_t} + \varepsilon_t
\]

where \( r_t \) is the federal funds rate, \( s_t \) is the state, and \( \mu_{s_t} \) is the mean in each state. In usmacro.dta, we have data for the series from the third quarter of 1954 to the fourth quarter of 2010 from the Federal Reserve Economic Database, a macroeconomic database provided by the Federal Reserve Bank of Saint Louis. The data are plotted below.
We note that the decades of 1970s and 1980s were characterized by periods of high interest rates while the rest of the sample displays moderate levels.

Thus, a two-state model seems reasonable. $s_t \in \{1, 2\}$ is the state; $\mu_1$ is the mean in the moderate-rate state; and $\mu_2$ is the mean in high-rate state. We can use `mswitch dr` with dependent variable `fedfunds` to estimate the parameters of the model.

```stata
. use https://www.stata-press.com/data/r16/usmacro
    (Federal Reserve Economic Data - St. Louis Fed)
. mswitch dr fedfunds
Performing EM optimization:
Performing gradient-based optimization:
Iteration 0:  log likelihood =  -508.66031
Iteration 1:  log likelihood =  -508.63820
Iteration 2:  log likelihood =  -508.63592
Iteration 3:  log likelihood =  -508.63592
```

Markov-switching dynamic regression

<table>
<thead>
<tr>
<th>Sample: 1954q3 - 2010q4</th>
<th>Number of obs =  226</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of states =  2</td>
<td>AIC =  4.5455</td>
</tr>
<tr>
<td>Unconditional probabilities: transition</td>
<td>HQIC =  4.5760</td>
</tr>
<tr>
<td></td>
<td>SBIC =  4.6211</td>
</tr>
</tbody>
</table>

Log likelihood =  -508.63592

| fedfunds | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|----------|-------|-----------|------|------|----------------------|
| State1 _cons | 3.70877 | .1767083 | 20.99 | 0.000 | 3.362428 - 4.055112 |
| State2 _cons | 9.556793 | .2999889 | 31.86 | 0.000 | 8.968826 - 10.14476 |
| sigma     | 2.107562 | .1008692 | 20.99 | 0.000 | 1.918851 - 2.314831 |
| p11       | .9820939 | .0104002 | .9450805 | .9943119 |
| p21       | .0503587 | .0268434 | .0173432 | .1374344 |

The header reports the sample size, fit statistics, the number of states, and the method used for computing the unconditional state probabilities. The EM algorithm was used to find the starting values.
for the quasi-Newton optimizer, and we see that it took three iterations for the model to converge. Finally, the header reports that the transition method was used to compute the unconditional state probabilities as a function of the transition probabilities; see Methods and formulas.

The estimation table reports results for each state-dependent mean and the constant error variance. Below that, the table displays the elements of the first $k - 1$ rows of the transition matrix, where $k$ is the number of states.

State 1 is the moderate-rate state and has a mean interest rate of 3.71%. State 2 is the high-rate state and has a mean interest rate of 9.56%. $p_{11}$ is the estimated probability of staying in state 1 in the next period given that the process is in state 1 in the current period. The estimate of 0.98 implies that state 1 is highly persistent. Similarly, $p_{21}$ is the probability of transitioning to state 1 from state 2. The probability of staying in state 2 is therefore $1 - 0.05 = 0.95$, which implies that state 2 is also highly persistent.

Note that it is arbitrary which state is called 1 or 2. Changing the initial values for the iterations, for example, can change the state labels for a given model-data combination. The transition probabilities will get swapped in accordance with the change in labels.

Technical note

As mentioned in Introduction, a model with one state is equivalent to linear regression. To estimate a one-state constant-only model for the data in example 1, you type

```
.mswitch dr fedfunds, states(1)
```

This is equivalent to typing

```
.arima fedfunds, technique(nr)
```

or

```
.regress fedfunds
```

The commands produce nearly identical parameter estimates for the coefficients.

Example 2: MSDR model with switching intercepts and coefficients

Continuing example 1, we specify a more complex model that includes the lagged value of the interest rate and allows its coefficient to switch as well. The respecified model is

$$r_t = \mu_{s_t} + \phi_{s_t} r_{t-1} + \varepsilon_t$$
We estimate the switching coefficient by including the `switch()` option.

```
.mswitch dr fedfunds, switch(L.fedfunds)
```

Performing EM optimization:

Performing gradient-based optimization:

```
Iteration 0:  log likelihood = -265.37725
Iteration 1:  log likelihood = -264.74265
Iteration 2:  log likelihood = -264.71073
Iteration 3:  log likelihood = -264.71069
Iteration 4:  log likelihood = -264.71069
```

Markov-switching dynamic regression

Sample: 1954q4 - 2010q4  Number of obs = 225
Number of states = 2  AIC = 2.4152
Unconditional probabilities: transition  HQIC = 2.4581
SBIC = 2.5215

Log likelihood = -264.71069

|       | Coef.  | Std. Err. | z      | P>|z|    | [95% Conf. Interval] |
|-------|--------|-----------|--------|-------|---------------------|
| fedfunds State1 | .7631424 | .0337234 | 22.63 | 0.000 | .6970457 -.8292392  |
|       | L1.    |           |        |       |                     |
| _cons | .724457 | .288657   | 2.51  | 0.012 | .158626 .1290231   |
| fedfunds State2 | 1.061174 | .0185031 | 57.35 | 0.000 | 1.024908 1.097439 |
|       | L1.    |           |        |       |                     |
| _cons | -.0988764 | .1183838 | -0.84 | 0.404 | -.3309043 .1331515 |
| sigma | .6915759 | .0358644 |        |       | .6247373 .7655663 |
| p11   | .6378175 | .1202616 |        |       | .3883032 .830089  |
| p21   | .1306295 | .0495924 |        |       | .0600137 .2612432 |

The output indicates that the coefficients on the lagged dependent variable in the two states are significant. Also, we favor this model over the constant-only model because the Schwarz Bayesian information criterion (SBIC) for this model, 2.52, is lower than the SBIC for the constant-only model, 4.62.

Example 3: Changing the number of states for an MSDR model

Continuing example 2, we now specify a Taylor-rule model with two and three states and select the preferred number of states.

Taylor-rule models specify that the interest rate depends on its own lag, the current value of inflation, and the output gap. In our dataset, `ogap` is the output gap and `inflation` is inflation.
First, we fit a two-state MSDR Taylor-rule model with \texttt{fedfunds} as the interest rate.

\begin{verbatim}
. mswitch dr fedfunds, switch(L.fedfunds ogap inflation)
Performing EM optimization:
Performing gradient-based optimization:
Iteration 0:  log likelihood = -229.43752
Iteration 1:  log likelihood = -229.25718
Iteration 2:  log likelihood = -229.25614
Iteration 3:  log likelihood = -229.25614
Markov-switching dynamic regression
Sample: 1955q3 - 2010q4 Number of obs = 222
Number of states = 2 AIC = 2.1645
Unconditional probabilities: transition HQIC = 2.2325
SBIC = 2.3331
Log likelihood = -229.25614
\end{verbatim}

| fedfunds | Coef.   | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|----------|---------|-----------|-------|-----|---------------------|
| State1 fedfunds L1. | .8314458 | .0333236  | 24.95 | 0.000 | .7661328 .8967587 |
| ogap | .1355425 | .0294113  | 4.61  | 0.000 | .0778975 .1931875 |
| inflation | -.0273928 | .0408057 | -0.67 | 0.502 | -.1073704 .0525849 |
| _cons | .6554954 | .1373889  | 4.77  | 0.000 | .386218 .9247727 |
| State2 fedfunds L1. | .9292574 | .0270852  | 34.31 | 0.000 | .8761713 .9823435 |
| ogap | .0343072 | .0240138  | 1.43  | 0.153 | -.0127589 .0813733 |
| inflation | .2125275 | .0297351  | 7.15  | 0.000 | .1542477 .2708072 |
| _cons | -.0944924 | .1279231 | -0.74 | 0.460 | -.3452171 .1562324 |
| sigma | .5764495 | .0302562  | .520968 | .638908 |
| p11 | .7279288 | .0929915  | .5159594 | .8703909 |
| p21 | .2114578 | .0641179  | .1120643 | .3629704 |

The results indicate that inflation does not significantly affect \texttt{fedfunds} in state 1 but that it does in state 2.
Would a model with three states be better than the above two-state model?

```
.mswitch dr fedfunds, switch(L.fedfunds ogap inflation) states(3)
(output omitted)
```

Markov-switching dynamic regression

Sample: 1955q3 - 2010q4  Number of obs = 222
Number of states = 3  AIC = 1.8819
Unconditional probabilities: transition  HQIC = 1.9995
SBIC = 2.1732

Log likelihood = -189.89493

| fedfunds | Coef. | Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|----------|-------|-----------|------|------|---------------------|
| **State 1** | | | | | |
| fedfunds | 0.8464551 | 0.033543 | 25.38 | 0.000 | 0.781082 - 0.9118283 |
| L1. ogap | 0.1201952 | 0.0232718 | 5.16 | 0.000 | 0.0745834 - 0.1658071 |
| inflation | -0.0425603 | 0.0354281 | -1.20 | 0.230 | -0.1119981 - 0.0268775 |
| _cons | 0.5261292 | 0.1266916 | 4.15 | 0.000 | 0.2778182 - 0.7744403 |

**State 2**

| fedfunds | Coef. | Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|----------|-------|-----------|------|------|---------------------|
| fedfunds | 0.9690088 | 0.0264823 | 36.59 | 0.000 | 0.9171046 - 1.020913 |
| L1. ogap | 0.0464136 | 0.0200197 | 2.32 | 0.020 | 0.0071758 - 0.0856515 |
| inflation | 0.1298906 | 0.0246793 | 5.26 | 0.000 | 0.0815202 - 0.1782611 |
| _cons | -0.0034106 | 0.107303 | -0.03 | 0.975 | -0.2137206 - 0.2068995 |

**State 3**

| fedfunds | Coef. | Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|----------|-------|-----------|------|------|---------------------|
| fedfunds | 0.4178913 | 0.0809412 | 5.16 | 0.000 | 0.2592494 - 0.5765333 |
| L1. ogap | 0.1075357 | 0.1131713 | 0.95 | 0.342 | -0.1142761 - 0.3293474 |
| inflation | 0.9099168 | 0.0732991 | 12.41 | 0.000 | 0.7662531 - 1.053581 |
| _cons | 0.6015991 | 0.8894513 | 0.68 | 0.499 | -1.141693 - 2.344892 |

| sigma | Coef. | Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|-------|-------|-----------|------|------|---------------------|
| 0.438375 | 0.0247954 | 3.923738 | 0.4897694 |
| p11 | 0.7253684 | 0.0807816 | 0.5440011 | 0.8539642 |
| p12 | 0.2564055 | 0.0784303 | 1.334279 | 0.4357355 |
| p21 | 0.1641252 | 0.0548584 | 0.8222928 | 0.3006711 |
| p22 | 0.7994204 | 0.0578707 | 6.626908 | 0.8899317 |
| p31 | 0.6178282 | 0.3481683 | 0.0824324 | 0.9667674 |
| p32 | 0.3821718 | 0.3481683 | 0.0332326 | 0.9175676 |

We favor the three-state model over the two-state model because it has the lower SBIC. The three states, in this case, can be thought of as representing low, moderate, and high-interest rate states.

The results for the three-state model indicate that inflation does not affect the interest rate in state 1, but it does affect the interest rate in states 2 and 3. The results also indicate that when the coefficient on inflation is large and significant in state 3, the output gap coefficient is not significant.
Technical note

In some situations, the quasi-Newton optimization will not converge, which implies that the parameters of the specified model are not identified by the data. These convergence problems most frequently arise when attempting to fit a model with too many states.

Example 4: Switching variances

All examples thus far have assumed a constant variance across states. In some cases, we may wish to relax this assumption. For example, in the snp500 dataset we have weekly data on the absolute returns of the S&P 500 index from the period 2004w17 to 2014w18, which we present below. The graph indicates that there were high-volatility periods in 2008 to 2009 and in late 2011. It would be unreasonable to assume that the variance in this high-volatility state is equal to the variance in the low-volatility state.
Below we fit `areturns`, the absolute returns, with an MSDR model in which the coefficients on the lagged dependent variable and the variances differ by state.

```
. use https://www.stata-press.com/data/r16/snp500
(Federal Reserve Economic Data - St. Louis Fed)
. mswitch dr areturns, switch(L.areturns) varswitch
```

Performing EM optimization:

Performing gradient-based optimization:

```
Iteration 0:   log likelihood = -753.27687
Iteration 1:   log likelihood = -746.54052
Iteration 2:   log likelihood = -745.80829
Iteration 3:   log likelihood = -745.79770
Iteration 4:   log likelihood = -745.79770
```

Markov-switching dynamic regression

Sample: 2004w19 - 2014w18
Number of obs = 520
Number of states = 2
AIC = 2.8992
Unconditional probabilities: transition

| HQIC = 2.9249 |
| SBIC = 2.9647 |

Log likelihood = -745.7977

```
        Coef. Std. Err.     z   P>|z|   [95% Conf. Interval]
------------- -------- -------- -------- -------- -----------------------------
State1         
    areturns     .0790744   .0301862   2.62  0.009    .0199105    .1382384
        L1.        
    _cons      .7641424   .0782852   9.76  0.000    .6107062    .9175785
State2         
    areturns     .527953   .0857841   6.15  0.000    .3598193    .6960867
        L1.        
    _cons      1.972771   .2784204   7.09  0.000    1.427077    2.518465
    sigma1     .5895792   .0517753   .9663544    .7003135
    sigma2     1.605333   .1262679   1.375985    1.872908
    p11        .7530865   .0634387   .6097999    .856167
    p21        .6825357   .0662574   .5414358    .7965346
```

The estimated standard deviations, reported in `sigma1` and `sigma2`, indicate that state 1 corresponds to the low-volatility periods and that state 2 corresponds to the high-volatility periods.

Example 5: An MSDR model of population health

We can apply these same methods to noneconomic data that exhibit similar periods of high and low volatility. For example, in public health and epidemiology, the process that determines the incidence of disease over time may evolve with changes in health practices.

In `mumpspc.dta`, we have monthly data on the number of new mumps cases and the interpolated population in New York City between January 1928 to December 1972. The `mumpspc` variable represents the number of new mumps cases per 10,000 residents. We apply seasonal differencing to the population-adjusted `mumpspc` variable using time-series operators, and we plot the resulting series; see [U] 11.4.4 Time-series varlists.
The data clearly show periods of high and low volatility. We fit a two-state MSDR model to the seasonally differenced dependent variable with state-dependent variances and state-dependent coefficients on the lagged dependent variable. This model does not have a constant term, so we specify suboption noconstant in `switch()` after the variable with a switching coefficient.

```
. use https://www.stata-press.com/data/r16/mumpspc
(Mumps data from Hipel and Mcleod (1994) with interpolated population)
. mswitch dr S12.mumpspc, varswitch switch(LS12.mumpspc, noconstant)
(output omitted)
```

Markov-switching dynamic regression

|           | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-----------|--------|-----------|-------|------|----------------------|
| S12.mumpspc |        |           |       |      |                      |
| State1 mumpspc LS12. | .420275| .0167461  | 25.10 | 0.000 | .3874533 .4530968    |
| State2 mumpspc LS12. | .9847369 | .0258383 | 38.11 | 0.000 | .9340947 1.035379    |
| sigma1     | .0562405 | .0050954 | 0.000 | .067169 |
| sigma2     | .2611362 | .0111191 | 2.381 | 0.01733 |
| p11        | .762733 | .0362619 | .000 | .8264175 |
| p21        | .1473767 | .0257599 | .000 | .2052939 |

The output indicates that there are two distinct states; state 1 is the low-volatility state and state 2 is the high-volatility state. While the lagged seasonally differenced number of mumps cases is a significant predictor of current seasonally differenced cases, the estimates differ between states. Both states are persistent.
Markov-switching AR

In this section, we use a series of examples to describe MSAR models and the `mswitch ar` command.

MSAR models allow a gradual adjustment after the process changes state. These models are often used to model quarterly and lower-frequency data. An MSAR model with two state-dependent AR terms for the dependent variable that is in state $s$ at time $t$ is

$$ y_t = \mu_{s_t} + x_t \alpha + z_t \beta_{s_t} $$

$$ + \phi_{1,s_t}(y_{t-1} - \mu_{s_{t-1}} - x_{t-1} \alpha - z_{t-1} \beta_{s_{t-1}}) $$

$$ + \phi_{2,s_t}(y_{t-2} - \mu_{s_{t-2}} - x_{t-2} \alpha - z_{t-2} \beta_{s_{t-2}}) $$

$$ + \varepsilon_{s_t} $$

where $y_t$ is the dependent variable at time $t$; $\mu_{s_t}$ is the state-dependent intercept; $x_t$ are covariates whose coefficients $\alpha$ are state invariant; $z_t$ are covariates whose coefficients $\beta_{s_t}$ are state-dependent; $\phi_{1,s_t}$ is the first AR term in state $s_t$; $\phi_{2,s_t}$ is the second AR term in state $s_t$; and $\varepsilon_{s_t}$ is the i.i.d. normal error with mean 0 and state-dependent variance. As in MSDR models, $x_t$ and $z_t$ may contain lags of $y_t$.

Note that $\mu_{s_{t-1}}$ is the intercept corresponding to the state that the process was in the previous period and that $\mu_{s_{t-2}}$ is the intercept corresponding to the state that the process was in at $t - 2$. Similarly, $\beta_{s_{t-1}}$ is the coefficient vector on $z_{t-1}$ corresponding to the state that the process was in the previous period, and $\beta_{s_{t-2}}$ is the coefficient vector on $z_{t-2}$ corresponding to the state that the process was in at $t - 2$.

In the default model fit by `mswitch ar`, $s = 2$ and a constant $\sigma^2$ is assumed ($\sigma_1^2 = \sigma_2^2 = \sigma^2$). In the simplest form, a single AR term is specified and the coefficient is common to both states, so four parameters, $\mu_1$, $\mu_2$, $\phi$, and $\sigma^2$, are estimated. There is no $x_t$ or $z_t$.

The number of AR terms may be increased by specifying a numlist in `ar()`. To allow the estimated parameters for the AR terms to vary across states, you specify option `arswitch`. The number of states may be increased with option `states()`. To include $x_t$, you specify a varlist after the command name, and to include $z_t$, you specify option `switch()`. The assumption of constant variances over states may be relaxed with option `varswitch`.

MSAR models allow states to switch according to a Markov process, as described in Methods and formulas under Markov-switching regression models. A more complete discussion of the MSAR model is provided in Specification of Markov-switching regression models under Methods and formulas.
Example 6: MSAR model with switching intercepts

Hamilton (1989) and Hamilton (1994, chap. 22) fit an MSAR to the growth of quarterly U.S. real gross national product using data from 1952q1 to 1984q4. We replicate those results here using rgnp.dta.

```
. use https://www.stata-press.com/data/r16/rgnp
(Data from Hamilton (1989))
. mswitch ar rgnp, ar(1/4)
```

Performing EM optimization:
Performing gradient-based optimization:

```
Iteration 0: log likelihood = -182.54411 (not concave)
Iteration 1: log likelihood = -182.12714 (not concave)
Iteration 2: log likelihood = -181.68653
Iteration 3: log likelihood = -181.42342
Iteration 4: log likelihood = -181.26492
Iteration 5: log likelihood = -181.26339
Iteration 6: log likelihood = -181.26339
```

Markov-switching autoregression

```
Sample: 1952q2 - 1984q4 Number of obs = 131
Number of states = 2 AIC = 2.9048
Unconditional probabilities: transition HQIC = 2.9851
SBIC = 3.1023
Log likelihood = -181.26339
```

| rgnp | Coef. | Std. Err. | z | P>|z| | [95% Conf. Interval] |
|------|-------|-----------|---|-----|---------------------|
| rgnp | ar    |           |   |     |                     |
| L1.  | .0134871 | .1199942  | 0.11 | 0.911 | -.2216971 - .2486713 |
| L2.  | -.0575211 | .137663 | -0.42 | 0.676 | -.3273357 - .2122934 |
| L3.  | -.2469833 | .1069103 | -2.31 | 0.021 | -.4565235 - .0374433 |
| L4.  | -.2129214 | .1105311 | -1.93 | 0.054 | -.4295583 - .0037155 |

State1
| _cons | -.3588127 | .2645396 | -1.36 | 0.175 | -.8773007 - .1596754 |

State2
| _cons | 1.163517 | .0745187 | 15.61 | 0.000 | 1.017463 - 1.309571 |
| sigma | .7690048 | .0667396 | .6487179 | .9115957 |
| p11   | .754671 | .0965189 | .5254555 | .8952432 |
| p21   | .0959153 | .0377362 | .0432569 | .1993222 |

The output indicates that the average growth rate of U.S. real gross national product during expansions is 1.16% and during recessions is −0.36%, with each state being persistent.

Example 7: Switching AR coefficients

Continuing example 6, we now fit an MSAR with state-dependent AR coefficients to the same dataset. We include only two AR terms in each state to better estimate the parameters.
. mswitch ar rgnp, ar(1/2) arswitch
Performing EM optimization:
Performing gradient-based optimization:
Iteration 0:  log likelihood = -179.68471
Iteration 1:  log likelihood = -179.56238
Iteration 2:  log likelihood = -179.32917
Iteration 3:  log likelihood = -179.32356
Iteration 4:  log likelihood = -179.32354
Iteration 5:  log likelihood = -179.32354
Markov-switching autoregression
Sample: 1951q4 - 1984q4  Number of obs = 133
Number of states = 2  AIC = 2.8319
Unconditional probabilities: transition  HQIC = 2.9114
SBIC = 3.0275
Log likelihood = -179.32354

| rgnp | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|------|--------|-----------|-------|------|----------------------|
| State1 | ar     |          |       |      |                      |
|       | L1.     | .3710719  | .1754383 | 2.12 | 0.034  | .0272191 .7149246  |
|       | L2.     | .7002937  | .187409  | 3.74 | 0.000  | .3329787 1.067609 |
|       | _cons   | -.0055216 | .2057086 | -0.03| 0.979  | -.408703 .3976599 |
| State2 | ar     |          |       |      |                      |
|       | L1.     | .4621503  | .1652473 | 2.80 | 0.005  | .1382714 .7860291 |
|       | L2.     | -.3206652 | .1295937 | -2.47| 0.013  | -.5746642 -.0666661 |
|       | _cons   | 1.195482  | .1225987 | 9.75 | 0.000  | .9551925 1.435771 |
| sigma |        | .6677098  | .0719638 | 9.50 | 0.000  | .5405648 .8247604 |
| p11   |        | .3812383  | .1424841 | 2.68 | 0.003  | .1586724 .6680876 |
| p21   |        | .3564492  | .0994742 | 3.50 | 0.000  | .1914324 .5644178 |

The results indicate that state 1 has negative average growth that is different than the positive average growth in state 2. The AR coefficients for state 1 indicate that shocks will die out very slowly, while the AR coefficients for state 2 indicate that shocks will die out moderately quickly. In other words, shocks in a recession last a long time, while shocks in an expansion die out moderately quickly.

Example 8: Markov-switching regression model with constraints

mswitch can fit models subject to constraints. To facilitate the optimization, mswitch estimates a logit transform of the transition probabilities (see Methods and formulas) and a log transformation of the variance parameter. Therefore, all constraints must be specified to the transformed parameter.

In example 6, the estimated transition probability of staying in state 1 was about 0.75. In this example, we constrain that probability to be 0.75 and estimate the remaining parameters. For this case, the transformed value is $q = -\ln(0.75/0.25) = -1.0986123$. We use the constraint command to define this constraint; see [R] constraint.
. constraint 1 [p11]_cons = -1.0986123
. mswitch ar rgnp, ar(1/4) constraints(1)
Performing EM optimization:
Performing gradient-based optimization:
Iteration 0: log likelihood = -182.86708
Iteration 1: log likelihood = -182.05084 (not concave)
Iteration 2: log likelihood = -181.79995
Iteration 3: log likelihood = -181.29355
Iteration 4: log likelihood = -181.26463
Iteration 5: log likelihood = -181.26456
Iteration 6: log likelihood = -181.26456
Markov-switching autoregression
Sample: 1952q2 - 1984q4 Number of obs = 131
Number of states = 2 AIC = 2.8895
Unconditional probabilities: transition HQIC = 2.9609
SBIC = 3.0651
Log likelihood = -181.26456
( 1) [p11]_cons = -1.098612

|         | Coef.  | Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|---------|--------|-----------|-------|-------|----------------------|
| rgnp    |        |           |       |       |                      |
| ar      |        |           |       |       |                      |
| L1.     | .0133924 | .1196067  | 0.11  | 0.911 | -.2210324            | .2478172 |
| L2.     | -.0591073 | .133834   | -0.44 | 0.659 | -.3214172            | .2032026 |
| L3.     | -.2473259 | .1067244  | -2.32 | 0.020 | -.456502             | -.0381499 |
| L4.     | -.2130605 | .1106088  | -1.93 | 0.054 | -.4298498            | .0037288 |
| State1  | _cons  |          |       |       |                      |
| _cons   | -.3648129 | .23039    | -1.58 | 0.113 | -.816369             | .0867432 |
| State2  | _cons  |          |       |       |                      |
| _cons   | 1.163125 | .0738402  | 15.75 | 0.000 | 1.018401             | 1.307849 |
| sigma   |        |           |       |       |                      |
|        | .7682327 | .0644585  | 15.75 | 0.000 | 1.018401             | 1.307849 |
| p11     |        |           |       |       |                      |
|        | .75 (constrained) | | | | | |
| p21     |        |           |       |       |                      |
|        | .0962226 | .037246   | 0.43  | 0.68  | .0439668             | .1977399 |

The point estimates are similar to those reported in example 6 while the standard errors reported here are slightly smaller.

Note that an MSAR model with no AR terms is equivalent to estimating an MSDR model, so typing

. mswitch ar rgnp, ar(0)

is the same as typing

. mswitch dr rgnp

Technical note

Achieving convergence in Markov-switching models can be difficult due to the existence of multiple local minima. Furthermore, a model with switching variance is able to generate a likelihood function that is unbounded when \( \mu = y_i \) and \( \sigma^2 \to 0 \) (Frühwirth-Schnatter 2006, chap. 6). Four methods for
overcoming convergence problems are 1) selecting an alternate optimization algorithm by using the \texttt{technique()} option; 2) using alternative starting values by specifying the \texttt{from()} option; 3) using improved starting values, which can be obtained by increasing the number of EM iterations specified in the \texttt{emiterate()} option; and 4) transforming the variables to be on the same scale.

\textbf{Video example}

\textit{Markov-switching models in Stata}

\textbf{Stored results}

\texttt{mswitch} stores the following in \texttt{e()}:  

\texttt{Scalars}

\begin{itemize}
  \item \texttt{e(N)} number of observations  
  \item \texttt{e(N_gaps)} number of gaps  
  \item \texttt{e(k)} number of parameters  
  \item \texttt{e(k_eq)} number of equations in \texttt{e(b)}  
  \item \texttt{e(k_aux)} number of auxiliary parameters  
  \item \texttt{e(states)} number of states  
  \item \texttt{e(ll)} log likelihood  
  \item \texttt{e(rank)} rank of \texttt{e(V)}  
  \item \texttt{e(aic)} Akaike information criterion  
  \item \texttt{e(hqic)} Hannan–Quinn information criterion  
  \item \texttt{e(sbic)} Schwarz–Bayesian information criterion  
  \item \texttt{e(tmin)} minimum time  
  \item \texttt{e(tmax)} maximum time  
  \item \texttt{e(emiter)} number of EM iterations  
\end{itemize}

\texttt{Macros}

\begin{itemize}
  \item \texttt{e(cmd)} \texttt{mswitch}  
  \item \texttt{e(cmdline)} command as typed  
  \item \texttt{e(eqnames)} names of equations  
  \item \texttt{e(depvar)} name of dependent variable  
  \item \texttt{e(switchvars)} list of switching variables  
  \item \texttt{e(nonswitchvars)} list of nonswitching variables  
  \item \texttt{e(model)} \texttt{dr} or \texttt{ar}  
  \item \texttt{e(title)} title in estimation output  
  \item \texttt{e(tsfmt)} format for the current time variable  
  \item \texttt{e(timevar)} time variable specified in \texttt{tsset}  
  \item \texttt{e(tmins)} formatted minimum time  
  \item \texttt{e(tmaxs)} formatted maximum time  
  \item \texttt{e(vce)} \texttt{vctype} specified in \texttt{vce()}  
  \item \texttt{e(vcetype)} title use to label Std. Err.  
  \item \texttt{e(technique)} maximization technique  
  \item \texttt{e(p0)} unconditional probabilities  
  \item \texttt{e(varswitch)} \texttt{varswitch}, if specified  
  \item \texttt{e(arswitch)} \texttt{arswitch}, if specified  
  \item \texttt{e(ar)} list of AR lags, if \texttt{ar()} is specified  
  \item \texttt{e(properties)} \texttt{b V}  
  \item \texttt{e(estat_cmd)} program used to implement \texttt{estat}  
  \item \texttt{e(predict)} program used to implement \texttt{predict}  
  \item \texttt{e(marginsnotok)} predictions disallowed by \texttt{margins}  
  \item \texttt{e(asbalanced)} factor variables \texttt{fvset} as \texttt{asbalanced}  
  \item \texttt{e(asobserved)} factor variables \texttt{fvset} as \texttt{asobserved}  
\end{itemize}
**Matrices**
- \( e(b) \): coefficient vector
- \( e(Cns) \): constraints matrix
- \( e(V) \): variance–covariance matrix of the estimators
- \( e(V_{modelbased}) \): model-based variance
- \( e(initvals) \): matrix of initial values
- \( e(uncprob) \): matrix of unconditional probabilities

**Functions**
- \( e(sample) \): marks estimation sample

---

**Methods and formulas**

Methods and formulas are presented under the following headings:

- **Markov-switching regression models**
- **Markov chains**
- **Specification of Markov-switching models**
  - Markov-switching dynamic regression
  - Markov-switching AR
- **Likelihood function with latent states**
- **Smoothed probabilities**
- **Unconditional probabilities**

**Markov-switching regression models**

Consider the evolution of \( y_t \), where \( t = 1, 2, \ldots, T \), that is characterized by two states or regimes as in the models below:

**State 1:** \( y_t = \mu_1 + \phi y_{t-1} + \varepsilon_t \)

**State 2:** \( y_t = \mu_2 + \phi y_{t-1} + \varepsilon_t \)

where \( \mu_1 \) and \( \mu_2 \) are the intercept terms in state 1 and state 2, respectively; \( \phi \) is the AR parameter; and \( \varepsilon_t \) is a white noise error with variance \( \sigma^2 \). The two states model abrupt shifts in the intercept term. If the timing of switches is known, the above model can be expressed as

\[
y_t = s_t \mu_1 + (1 - s_t) \mu_2 + \phi y_{t-1} + \varepsilon_t
\]

where \( s_t \) is 1 if the process is in state 1 and 0 otherwise. Estimation in this case can be performed using standard procedures.

In the case of interest, we never know in which state the process is; that is to say, \( s_t \) is not observed. Markov-switching regression models specify that the unobserved \( s_t \) follows a Markov chain. In the simplest case, we can express this model as a state-dependent intercept term for \( k \) states

\[
y_t = \mu_{s_t} + \phi y_{t-1} + \varepsilon_t
\]

where \( \mu_{s_t} = \mu_1 \) when \( s_t = 1 \), \( \mu_{s_t} = \mu_2 \) when \( s_t = 2 \), \ldots, and \( \mu_{s_t} = \mu_k \) when \( s_t = k \). The conditional density of \( y_t \) is assumed to be dependent only on the realization of the current state \( s_t \) and is given by \( f(y_t | s_t = i, y_{t-1}; \theta) \), where \( \theta \) is a vector of parameters. There are \( k \) conditional densities for \( k \) states, and estimation of \( \theta \) is performed by updating the conditional likelihood using a nonlinear filter.
Markov chains

$s_t$ is an irreducible, aperiodic Markov chain starting from its ergodic distribution $\pi = (\pi_1, \ldots, \pi_k)$; see Hamilton (1994, chap. 22). The probability that $s_t$ is equal to $j \in (1, \ldots, k)$ depends only on the most recent realization, $s_{t-1}$, and is given by

$$\Pr(s_t = j | s_{t-1} = i) = p_{ij}$$

All possible transitions from one state to the other can be collected in a $k \times k$ transition matrix

$$P = \begin{bmatrix}
p_{11} & \cdots & p_{k1} \\
p_{12} & \cdots & p_{k2} \\
\vdots & \ddots & \vdots \\
p_{1k} & \cdots & p_{kk}
\end{bmatrix}$$

which governs the evolution of the Markov chain. All elements of $P$ are nonnegative and each column sums to 1. For an excellent introduction on the properties of Markov chains, refer to Hamilton (1994, chap. 22) and Frühwirth-Schnatter (2006, chap. 10). For a deeper treatment, see Karlin and Taylor (1975, chap. 2 and 3).

The fact that $\sum_{j=1}^{k} p_{ij} = 1$ causes some numerical complications. We handle these complications by estimating functions of $p_{ij}$ and by normalizing $p_{ik}$. In particular, we estimate $q_{ij}$ in

$$p_{ij} = \frac{\exp(-q_{ij})}{1 + \exp(-q_{i1}) + \exp(-q_{i2}) + \cdots + \exp(-q_{i,k-1})}$$

for $j \in \{1, 2, \ldots, k - 1\}$. We normalize $p_{ik}$ by imposing

$$p_{ik} = \frac{1}{1 + \exp(-q_{i1}) + \exp(-q_{i2}) + \cdots + \exp(-q_{i,k-1})}$$

The estimates of $p_{ij}$ are displayed, but the estimates of $q_{ij}$ are stored in $e(b)$.

Specification of Markov-switching models

Consider an AR(1) model given by

$$y_t = \mu + \phi y_{t-1} + \varepsilon_t$$

This model can be rewritten in terms of an AR(1) error specification as

$$y_t = \nu + e_t$$

$$e_t = \rho e_{t-1} + \varepsilon_t$$

which can be written as the single equation

$$y_t = \nu + \rho (y_{t-1} - \nu) + \varepsilon_t$$

such that $\phi = \rho$ and $\mu = \nu (1 - \rho)$.
This result, however, does not hold in the case of Markov-switching regression models, as seen below in a simple two-state case where the constant term is state dependent. Consider the following models:

Model I: \[ y_t = \mu_{s_t} + \phi y_{t-1} + \varepsilon_t \]

Model II: \[ y_t = \mu_{s_t} + \phi(y_{t-1} - \mu_{s_{t-1}}) + \varepsilon_t \]

Model I is also referred to as a MSDR model or a Markov-switching intercept model (Krolzig 1997). It may consist of other switching parameters, but for simplicity, we only consider the switching-intercept case. The evolution of \( y_t \) depends on the realization of the switching intercept at time \( t \). The discrete latent state \( s_t \) that governs the value of the intercept at time \( t \) has a transition matrix

\[
P = \begin{bmatrix}
p_{11} & p_{21} \\
p_{12} & p_{22}
\end{bmatrix}
\]

This specification allows for two possible intercepts at any given time \( t \).

By contrast, the evolution of \( y_t \) in model II depends on the value of the switching mean at its current state and its lagged value. Model II is also referred to as MSAR or Markov-switching mean (Krolzig 1997). At any given time \( t \), there are four possible values of the intercept given by

\[
\begin{align*}
\mu_1 - \rho \mu_1 \\
\mu_2 - \rho \mu_1 \\
\mu_1 - \rho \mu_2 \\
\mu_2 - \rho \mu_2
\end{align*}
\]

which implies that models I and II do not yield equivalent representations as compared with the AR(1) model with no switching.

The MA(\( \infty \)) representation shown below better illustrates the different dynamics of \( y_t \) obtained as a result of these specifications.

Model I: \[ y_t = \sum_{i=0}^{\infty} \phi^i \mu_{s_{t-i}} + \sum_{i=0}^{\infty} \phi^i \varepsilon_t \]

Model II: \[ y_t = \mu_{s_t} + \sum_{i=0}^{\infty} \phi^i \varepsilon_t \]

In model I, the effect of a one-time change in state accumulates over time similar to a permanent shift in the error term \( \varepsilon_t \). In model II, the effect of a one-time change in state is the same for all time periods. Also see Hamilton (1993).

Model II allows \( y_t \) to depend on lagged values of the state \( s_{t-1} \), which in turn leads to four conditional densities. We define a new state variable \( s^*_t \) such that \( s^*_t \) is a four-state Markov chain and \( y_t \) depends only on the current state as

\[
\begin{align*}
s^*_t = 1 & \quad \text{if} \quad s_t = 1 \quad \text{and} \quad s_{t-1} = 1 \\
s^*_t = 2 & \quad \text{if} \quad s_t = 2 \quad \text{and} \quad s_{t-1} = 1 \\
s^*_t = 3 & \quad \text{if} \quad s_t = 1 \quad \text{and} \quad s_{t-1} = 2 \\
s^*_t = 4 & \quad \text{if} \quad s_t = 2 \quad \text{and} \quad s_{t-1} = 2
\end{align*}
\]
The corresponding $4 \times 4$ transition matrix is

$$
P = \begin{bmatrix}
p_{11} & 0 & p_{11} & 0 \\
p_{12} & 0 & p_{12} & 0 \\
0 & p_{21} & 0 & p_{21} \\
0 & p_{22} & 0 & p_{22}
\end{bmatrix}
$$

The conditional density of $y_t$ is given by $f(y_t|s^*_t = i, y_{t-1}; \theta)$ for $i = 1, \ldots, 4$. Also see Hamilton (1994, chap. 22). More generally, for MSAR models, $s^*_t$ is a $k^{(p+1)}$-state Markov chain, where $p$ is the number of lagged states. Because MSAR models require larger state vectors, they are often used with low-frequency data and smaller AR lags. However, the state vector in MSDR models does not depend on the AR lags and can thus be used to accommodate high-frequency data and higher AR lags.

**Markov-switching dynamic regression**

A general specification of the MSDR model is written as

$$
y_t = \mu_{s^*_t} + x_t \alpha + z_t \beta_{s^*_t} + \epsilon_{s^*_t}
$$

where $y_t$ is the dependent variable, $\mu_{s^*_t}$ is the state-dependent intercept, $x_t$ is a vector of exogenous variables with state-invariant coefficients $\alpha$, $z_t$ is a vector of exogenous variables with state-dependent coefficients $\beta_{s^*_t}$, and $\epsilon_{s^*_t}$ is an i.i.d. normal error with mean 0 and state-dependent variance $\sigma^2_{s^*_t}$. $x_t$ and $z_t$ may contain lags of $y_t$.

**Markov-switching AR**

A general specification of the MSAR model is written as

$$
y_t = \mu_{s^*_t} + x_t \alpha + z_t \beta_{s^*_t} + \sum_{i=1}^{p} \phi_{i,s^*_t} (y_{t-i} - \mu_{s^*_{t-i}} - x_{t-i} \alpha - z_{t-i} \beta_{s^*_{t-i}}) + \epsilon_{s^*_t}
$$

where $y_t$ is the dependent variable at time $t$, $\mu_{s^*_t}$ is the state-dependent intercept, $x_t$ are covariates whose coefficients $\alpha$ are state-invariant, and $z_t$ are covariates whose coefficients $\beta_{s^*_t}$ are state-dependent. As in MSDR models, $x_t$ and $z_t$ may contain lags of $y_t$.

$\phi_{i,s^*_t}$ is the $i$th AR term in state $s^*_t$. Note that $\mu_{s^*_{t-i}}$ is the intercept corresponding to the state that the process was in at period $t - i$. Similarly, $\beta_{s^*_{t-i}}$ is the coefficient vector on $z_{t-i}$ corresponding to the state that the process was in at period $t - i$.

$\epsilon_{s^*_t}$ is the i.i.d. normal error with mean 0 and state-dependent variance.

This representation clarifies that the demeaned, lagged errors depend on the state previously occupied by the process. This dependence is not present in the MSDR model.
Likelihood function with latent states

The conditional density of $y_t$ is given by $f(y_t|s_t = i, y_{t-1}; \theta)$ for $i = 1, \ldots, k$. The marginal density of $y_t$ is obtained by weighting the conditional densities by their respective probabilities. This is written as follows:

$$f(y_t|\theta) = \sum_{i=1}^{k} f(y_t|s_t = i, y_{t-1}; \theta) \Pr(s_t = i; \theta)$$

Let $\eta_t$ denote a $k \times 1$ vector of conditional densities given by

$$\eta_t = \begin{bmatrix}
    f(y_t|s_t = 1; y_{t-1}; \theta) \\
    f(y_t|s_t = 2; y_{t-1}; \theta) \\
    \vdots \\
    f(y_t|s_t = k; y_{t-1}; \theta)
\end{bmatrix}$$

Constructing the likelihood function requires estimating the probability that $s_t$ takes on a specific value using the data through time $t$ and model parameters $\theta$. Let $\Pr(s_t = i|y_t; \theta)$ denote the conditional probability of observing $s_t = i$ based on data until time $t$. Then

$$\Pr(s_t = i|y_{t-1}; \theta) = \frac{f(y_t|s_t = i, y_{t-1}; \theta) \Pr(s_t = i|y_{t-1}; \theta)}{f(y_t|y_{t-1}; \theta)}$$

where $f(y_t|y_{t-1}; \theta)$ is the likelihood of $y_t$ and $\Pr(s_t = i|y_{t-1}; \theta)$ is the forecasted probability of $s_t = i$ given observation until time $t - 1$. Then

$$\Pr(s_t = 1|y_{t-1}; \theta) = \sum_{j=1}^{k} \Pr(s_t = i|s_{t-1} = j, y_{t-1}; \theta) \Pr(s_{t-1} = j|y_{t-1}; \theta)$$

Let $\xi_{t|t}$ and $\xi_{t|t-1}$ denote $k \times 1$ vectors of conditional probabilities $\Pr(s_t = i|y_{t}; \theta)$ and $\Pr(s_t = i|y_{t-1}; \theta)$. The likelihood is then obtained by iterating on the following equations [Hamilton (1994, chap. 22)]:

$$\xi_{t|t} = \frac{(\xi_{t|t-1} \odot \eta_t)}{1'(\xi_{t|t-1} \odot \eta_t)}$$

$$\xi_{t+1|t} = P \xi_{t|t}$$

where $1$ is a $k \times 1$ vector of 1s. The log-likelihood function is obtained as

$$L(\theta) = \sum_{t=1}^{T} \log f(y_t|y_{t-1}; \theta)$$

where

$$f(y_t|y_{t-1}; \theta) = 1'(\xi_{t|t-1} \odot \eta_t)$$
Smoothed probabilities

Let $\xi_{t|T}$, where $t < T$, denote the $k \times 1$ vector of conditional probabilities $\Pr(s_t = i|y_T; \theta)$, which represents the probability of $s_t = i$ using observations available through time $T$. The smoothed probabilities are calculated using an algorithm developed in Kim (1994).

$$\xi_{t|T} = \xi_{t|t} \odot \{P' (\xi_{t+1|T} ÷ \xi_{t+1|t})\}$$

where $(÷)$ denotes element-by-element division. The smoothed probabilities are obtained by iterating backwards from $t = T - 1, T - 2, \ldots, 1$.

Unconditional probabilities

The log-likelihood function has a recursive structure that starts from the unconditional state probabilities $\xi_{1|0}$. These unconditional state probabilities are unknown. There are three standard ways to obtain them.

By default, or by option $p0(\text{transition})$, the unconditional state probabilities are estimated from the conditional transition probabilities and the Markov structure of the model. Specifically, the vector of unconditional state probabilities is obtained as

$$\pi = (A'A)^{-1}A' e_{k+1}$$

where $A$ is a $(k + 1) \times k$ matrix given by

$$A = \begin{bmatrix} I_k - P \\ 1' \end{bmatrix}$$

$I_k$ denotes a $k \times k$ identity matrix, and $e_k$ denotes the $k$th column of $I_k$.

Sometimes researchers prefer to estimate unconditional state probabilities by adding $k - 1$ additional parameters to the model. This method is seldom used because it requires enough observations to estimate the additional parameters. mswitch uses this method when option $p0(\text{smoothed})$ is specified.

Sometimes researchers prefer to set the unconditional state probabilities to $1/k$. mswitch uses this method when option $p0(\text{fixed})$ is specified.

References


Also see

[TS] **mswitch postestimation** — Postestimation tools for mswitch

[TS] **arima** — ARIMA, ARMAX, and other dynamic regression models

[TS] **sspace** — State-space models

[TS] **threshold** — Threshold regression

[TS] **tsset** — Declare data to be time-series data

[TS] **ucm** — Unobserved-components model

[U] 20 Estimation and postestimation commands
### Postestimation commands

The following postestimation commands are of special interest after `mswitch`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat transition</code></td>
<td>display transition probabilities in a table</td>
</tr>
<tr>
<td><code>estat duration</code></td>
<td>display expected duration of states in a table</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contrast</code></td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>pwcompare</code></td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates new variables containing predictions such as predicted values, probabilities, residuals, and standardized residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [ `type` ] `{ `stub* | newvarlist` } [ `if` ] [ `in` ] [ , statistic options ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>yhat</td>
<td>predicted values; the default</td>
</tr>
<tr>
<td>xb</td>
<td>equation-specific predicted values; default is predicted values for the first equation</td>
</tr>
<tr>
<td>pr</td>
<td>compute probabilities of being in a given state; default is one-step-ahead probabilities</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
<tr>
<td>rstandard</td>
<td>standardized residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

options

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
</tr>
<tr>
<td>smethod(<code>method</code>)</td>
</tr>
<tr>
<td>rmse(`stub*</td>
</tr>
<tr>
<td>dynamic(<code>time_constant</code>)</td>
</tr>
<tr>
<td>equation(<code>eqnnames</code>)</td>
</tr>
</tbody>
</table>

method

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onestep</td>
</tr>
<tr>
<td>filter</td>
</tr>
<tr>
<td>smooth</td>
</tr>
</tbody>
</table>
**Options for predict**

- **yhat**, `xb`, `pr`, `residuals`, and `rstandard` specify the statistic to be predicted.

  - `yhat`, the default, calculates the weighted and state-specific linear predictions of the observed variables.
  - `xb` calculates the equation-specific linear predictions of the observed variables.
  - `pr` calculates the probabilities of being in a given state.
  - `residuals` calculates the residuals in the equations for observable variables.
  - `rstandard` calculates the standardized residuals, which are the residuals normalized to have unit variances.

- **Options**
  
  - `smethod(method)` specifies the method for predicting the unobserved states; `smethod(onestep)`, `smethod(filter)`, and `smethod(smooth)` allow different amounts of information on the dependent variables to be used in predicting the states at each time period. `smethod()` may not be specified with `xb`.

  - `smethod(onestep)`, the default, causes `predict` to estimate the states at each time period using previous information on the dependent variables. The nonlinear filter is performed on previous periods, but only the one-step predictions are made for the current period.

  - `smethod(filter)` causes `predict` to estimate the states at each time period using previous and contemporaneous data by using the nonlinear filter. The filtering is performed on previous periods and the current period.

  - `smethod(smooth)` causes `predict` to estimate the states at each time period using all sample data by using the smoothing algorithm.

  - `rmse(stub* | newvarlist)` puts the root mean squared errors of the predicted statistics into the specified new variables. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error.

  - `dynamic(time_constant)` specifies when `predict` starts producing dynamic forecasts. The specified `time_constant` must be in the scale of the time variable specified in `tsset`, and the `time_constant` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2014q4))` causes dynamic predictions to begin in the fourth quarter of 2014, assuming that the time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may not be specified with `xb`, `pr`, `residuals`, or `rstandard`.

  - `equation(eqnames)` specifies the equations for which the predictions are to be calculated. If you do not specify `equation()` or `stub*`, the results are the same as if you had specified the name of the first equation for the predicted statistic. `equation()` may be specified with `xb` only.

    You specify a list of equation names, such as `equation(income consumption)` or `equation(factor1 factor2)`, to identify the equations.

    `equation()` may not be specified with `stub*`. 
estat

Description for estat

estat transition displays all of the transition probabilities in tabular form.

estat duration computes the expected duration that the process spends in each state and displays the results in a table.

Menu for estat

Statistics > Postestimation

Syntax for estat

Display transition probabilities in a table

\[
\text{estat transition } [, \text{ level}()]\]

Display expected duration of states in a table

\[
\text{estat duration } [, \text{ level}()]\]

Option for estat

level(#) specifies the confidence level, as a percentage, for confidence intervals. The default is level(95) or as set by set level; see [U] 20.8 Specifying the width of confidence intervals.

Remarks and examples

Remarks are presented under the following headings:

One-step predictions
Dynamic predictions
Model fit and state predictions

We assume that you have already read [TS] mswitch. In this entry, we illustrate some of the features of predict after using mswitch to estimate the parameters of a Markov-switching model.

All the predictions after mswitch depend on the unobserved states, which are estimated recursively using a nonlinear filter. Changing the sample can alter the state estimates, which can change all other predictions.

One-step predictions

One-step predictions in a Markov-switching model are the forecasted values of the dependent variable using one-step-ahead predicted probabilities.
Example 1: One-step predictions for a series

In example 3 of [TS] `mswitch`, we estimated the parameters of a Markov-switching dynamic regression for the federal funds rate `fedfunds` as a function of its lag, the output gap `ogap`, and inflation.

```
. use https://www.stata-press.com/data/r16/usmacro
   (Federal Reserve Economic Data - St. Louis Fed)
   . mswitch dr fedfunds, switch(L.fedfunds ogap inflation)
   (output omitted)
```

We obtain the one-step predictions for the dependent variable using the default settings for `predict`. The predictions are stored in the new variable `fedf`.

```
   . predict fedf
   (option `yhat` assumed; predicted values)
```

Next, we graph the actual values, `fedfunds`, and predicted values, `fedf`, using `tsline`. We change the label for `fedf` to “Predicted values”; see [TS] `tsline`.

```
   . tsline fedfunds fedf, legend(label(2 "Predicted values"))
```

The graph shows that one-step-ahead predictions account for large swings in the federal funds rate.

Example 2: State-specific one-step predictions

Continuing example 1, we may also wish to obtain state-specific predictions. This allows us to compare the predictions obtained for different states.

Note that this time, we specify `fedf*` rather than `fedf` so that `predict` generates two state-specific predictions with the prefix `fedf` instead of a single weighted prediction. Also note that the predicted values obtained in example 1 are the weighted average of the state-specific predictions, the weights being the one-step-ahead probabilities.
. predict fedf*  
   (option yhat assumed; predicted values)
. tsline fedf1 fedf2, legend(label(1 "State 1 predictions")  
   > label(2 "State 2 predictions"))

The graph shows that, as expected, the predicted values of fedfunds are higher in state 2, the high-interest rate state, than in state 1, the moderate-interest rate state.

Dynamic predictions

Dynamic predictions are out-of-sample forecasted values of the dependent variable using one-step-ahead probabilities.

Example 3: Dynamic predictions for Markov-switching autoregression

In example 6 of [TS] mswitch, we estimated the parameters of a Markov-switching autoregression for the U.S. real gross national product as a function of its own lags.

. use https://www.stata-press.com/data/r16/rgnp, clear  
   (Data from Hamilton (1989))
. mswitch ar rgnp, ar(1/4)  
   (output omitted)

To obtain dynamic predictions, we use predict with the dynamic() option. The dynamic() option requires that all exogenous variables be present for the whole predicted sample. In this example, we have not specified any exogenous variables, so we do not check for that. However, we do need to have time values available for the predictions. So before submitting our predict command, we use tsappend to extend the dataset by eight periods.

Within dynamic(), we specify that dynamic predictions will begin in the first quarter of 1985, and we use the convenience function tq() to convert 1985q1 into a numeric date that Stata understands; see [FN] Date and time functions.
We again use `tsline` to plot the in- and out-of-sample predictions. We restrict the range to quarters 1982q1 to 1986q4 using function `tin()`.

```
.tsline rgnp_f if tin(1982q1,1986q4), ytitle("Out-of-sample predictions")
> tline(1985q1)
```

The vertical line shows where our out-of-sample prediction begins.

### Model fit and state predictions

#### Example 4: Assessing model fit

In this example, we examine the model fit by comparing the fitted values of U.S. real gross national product and the residuals with the actual data. The fitted values are obtained using smoothed probabilities that consider all sample information.

```
.predict yhat, smethod(smooth)
(option _yhat_ assumed; predicted values)
.predict res, residuals smethod(smooth)
.tsline yhat res rgnp, legend(label(1 "Fitted values") label(2 "Residuals"))
```
We see in the graph above that we do not obtain a good fit; the residuals account for much of the variation in the dependent variable.

Example 5: Filtered probabilities

Continuing example 4, recall that the states in the model correspond to recession periods and expansion periods for the U.S. economy. State 1 was the recession state. Here we compare the predicted probability of being in state 1 with the National Bureau of Economic Research recession periods stored in the indicator variable `recession`.

To obtain the filtered probabilities, typically used to predict state probabilities, we specify options `pr` and `smethod(filter)` with `predict`.

```
.predict fprob, pr smethod(filter)
.tsline fprob recession
```
The predictions of recession and expansion states fit well with the NBER dates. Thus, it appears that while our model does not have good fit, it does a good job of predicting the probability of being in a given state.

We could also have specified \texttt{smethod(smooth)} to obtain better estimates of the state probability using all sample information.

\section*{Example 6: Expected duration}

Rather than predicting which state the series is in at a point in time, we may wish to know the average time it spends in a given state. We can compute the expected duration of the process being in a given state and show the result in a table using \texttt{estat duration}.

Continuing \texttt{example 5}, we can calculate the average length of recession periods and expansion periods for the U.S. economy.

\begin{verbatim}
. estat duration
Number of obs = 131

<table>
<thead>
<tr>
<th>Expected Duration</th>
<th>Estimate</th>
<th>Std. Err.</th>
<th>[95% Conf. Interval]</th>
</tr>
</thead>
<tbody>
<tr>
<td>State1</td>
<td>4.076159</td>
<td>1.603668</td>
<td>2.107284 9.545915</td>
</tr>
<tr>
<td>State2</td>
<td>10.42587</td>
<td>4.101872</td>
<td>5.017004 23.11771</td>
</tr>
</tbody>
</table>
\end{verbatim}

The table indicates that state 1, the recession state, will typically persist for about 4 quarters and state 2, the expansion state, will persist for about 10 quarters.

\section*{Stored results}

\texttt{estat transition} stores the following in \texttt{r(}):

- Scalars
  - \texttt{r(level)}: confidence level of confidence intervals

- Macros
  - \texttt{r(label#)}: label for transition probability

- Matrices
  - \texttt{r(prob)}: vector of transition probabilities
  - \texttt{r(se)}: vector of standard errors of transition probabilities
  - \texttt{r(ci#)}: vector of confidence interval (lower and upper) for transition probability

\texttt{estat duration} stores the following in \texttt{r(}):

- Scalars
  - \texttt{r(d#)}: expected duration for state #
  - \texttt{r(se#)}: standard error of expected duration for state #
  - \texttt{r(level)}: confidence level of confidence intervals

- Macros
  - \texttt{r(label#)}: label for state #

- Matrices
  - \texttt{r(ci#)}: vector of confidence interval (lower and upper) for expected duration for state #
Methods and formulas

Forecasting a Markov-switching model requires estimating the probability of the process being at any given state at each time period. The forecasts are then computed by weighting the state-specific forecasts by the estimated probabilities. Refer to Hamilton (1993) and Davidson (2004) for more details on forecasting Markov-switching regression models.

By default and with the smethod(filter) option, predict estimates the probability of being at a state in each period by applying a nonlinear filter on all previous periods and the current period. (See Methods and formulas of [TS] mswitch for the filter equations.)

With the smethod(smooth) option, predict estimates the probabilities by applying a smoothing algorithm (Kim 1994) using all the sample information. With the smethod(onestep) option, predict estimates the probabilities using information from all previous periods to make one-step-ahead predictions.

The dependent variable is predicted by averaging the state-specific forecasts by the estimated probabilities. The residuals are computed as the difference between the predicted and the realized dependent variable. The standardized residuals are the residuals normalized to have unit variances.

Using an if or in qualifier to alter the prediction sample can change the estimate of the unobserved states in the period prior to beginning the dynamic predictions and hence alter the dynamic predictions. The initial values for the probabilities are obtained by calculating the ergodic probabilities from the transition matrix.

References


Also see

[TS] mswitch — Markov-switching regression models

[U] 20 Estimation and postestimation commands
newey — Regression with Newey–West standard errors

Description

newey produces Newey–West standard errors for coefficients estimated by OLS regression. The error structure is assumed to be heteroskedastic and possibly autocorrelated up to some lag.

Quick start

OLS regression of y on x1 and x2 with Newey–West standard errors robust to heteroskedasticity and first-order autocorrelation using tsset data

```
newey y x1 x2, lag(1)
```

With heteroskedasticity-robust standard errors

```
newey y x1 x2, lag(0)
```

Menu

Statistics > Time series > Regression with Newey-West std. errors

Syntax

```
newey depvar [ indepvars ] [ if ] [ in ] [ weight ], lag(#) [ options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*lag(#)</td>
<td>set maximum lag order of autocorrelation</td>
</tr>
<tr>
<td>noconstant</td>
<td>suppress constant term</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

*lag(#) is required.

You must tsset your data before using newey; see [TS] tsset.

indepvars may contain factor variables; see [U] 11.4.3 Factor variables.

depvar and indepvars may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.

aweights are allowed; see [U] 11.1.6 weight.

coeflegend does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

470
Options

lag(#) specifies the maximum lag to be considered in the autocorrelation structure. If you specify lag(0), the output is the same as `regress, vce(robust)`. lag() is required.

noconstant; see [R] Estimation options.

level(#); see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nolabel, fvwrap(#), fvwrapon(style), cformat(%fmt), pformat(%fmt), sformat(%fmt), and nolstretch; see [R] Estimation options.

The following option is available with `newey` but is not shown in the dialog box:

coeflegend; see [R] Estimation options.

Remarks and examples

The Huber/White/sandwich robust variance estimator (see White [1980]) produces consistent standard errors for OLS regression coefficient estimates in the presence of heteroskedasticity. The Newey–West (1987) variance estimator is an extension that produces consistent estimates when there is autocorrelation in addition to possible heteroskedasticity.

The Newey–West variance estimator handles autocorrelation up to and including a lag of $m$, where $m$ is specified by stipulating the lag() option. Thus, it assumes that any autocorrelation at lags greater than $m$ can be ignored.

If lag(0) is specified, the variance estimates produced by `newey` are simply the Huber/White/sandwich robust variances estimates calculated by `regress, vce(robust)`; see [R] `regress`.

Example 1

newey, lag(0) is equivalent to `regress, vce(robust)`:

```
. use https://www.stata-press.com/data/r16/auto
    (1978 Automobile Data)
. regress price weight displ, vce(robust)
Linear regression
Number of obs = 74
F(2, 71) = 14.44
Prob > F = 0.0000
R-squared = 0.2909
Root MSE = 2518.4

    price      Coef.  Std. Err.      t    P>|t|     [95% Conf. Interval]
    weight    1.823366   .7808755     2.34   0.022     .2663445    3.380387
   displacement    2.087054   7.436967     0.28   0.780   -12.74184    16.91595
    _cons    247.9070   1129.6028     0.22   0.827   -2004.455    2500.269
```

. generate t = _n
. tsset t
time variable: t, 1 to 74
delta: 1 unit

. newey price weight displ, lag(0)
Regression with Newey-West standard errors Number of obs = 74
maximum lag: 0 F( 2, 71) = 14.44
Prob > F = 0.0000

<table>
<thead>
<tr>
<th></th>
<th>Newey-West</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coef.</td>
</tr>
<tr>
<td>price</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>1.823366</td>
</tr>
<tr>
<td>displ</td>
<td>2.087054</td>
</tr>
<tr>
<td>_cons</td>
<td>247.907</td>
</tr>
</tbody>
</table>

Because newey requires the dataset to be tsset, we generated a dummy time variable t, which in this example played no role in the estimation.

Example 2

Say that we have time-series measurements on variables usr and idle and now wish to fit an OLS model but obtain Newey–West standard errors allowing for a lag of up to 3:

. use https://www.stata-press.com/data/r16/idle2, clear
. tsset time
time variable: time, 1 to 30
delta: 1 unit

. newey usr idle, lag(3)
Regression with Newey-West standard errors Number of obs = 30
maximum lag: 3 F( 1, 28) = 10.90
Prob > F = 0.0026

<table>
<thead>
<tr>
<th></th>
<th>Newey-West</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coef.</td>
</tr>
<tr>
<td>usr</td>
<td></td>
</tr>
<tr>
<td>idle</td>
<td>-.2281501</td>
</tr>
<tr>
<td>_cons</td>
<td>23.13483</td>
</tr>
</tbody>
</table>
newey — Regression with Newey–West standard errors

Stored results

_Stored results_ newey stores the following in e():

Scalars
- `e(N)` number of observations
- `e(df_m)` model degrees of freedom
- `e(df_r)` residual degrees of freedom
- `e(F)` \( F \) statistic
- `e(lag)` maximum lag
- `e(rank)` rank of `e(V)`

Macros
- `e(cmd)` newey
- `e(cmdline)` command as typed
- `e(depvar)` name of dependent variable
- `e(wtype)` weight type
- `e(wexp)` weight expression
- `e(title)` title in estimation output
- `e(vcetype)` title used to label Std. Err.
- `e(properties)` `b V`
- `e(estat_cmd)` program used to implement estat
- `e(predict)` program used to implement predict
- `e(asbalanced)` factor variables fvset as asbalanced
- `e(asobserved)` factor variables fvset as asobserved

Matrices
- `e(b)` coefficient vector
- `e(V)` variance–covariance matrix of the estimators

Functions
- `e(sample)` marks estimation sample

Methods and formulas

_Methods and formulas_ newey calculates the estimates

\[
\hat{\beta}_{OLS} = (X'X)^{-1}X'y
\]
\[
\text{Var}(\hat{\beta}_{OLS}) = (X'X)^{-1}X'\hat{\Omega}X(X'X)^{-1}
\]

That is, the coefficient estimates are simply those of OLS linear regression.

For `lag(0)` (no autocorrelation), the variance estimates are calculated using the White formulation:

\[
X'\hat{\Omega}X = X'\hat{\Omega}_0X = \frac{n}{n-k} \sum_i \hat{c}_i^2 x_i'x_i
\]

Here \( \hat{c}_i = y_i - x_i\hat{\beta}_{OLS} \), where \( x_i \) is the \( i \)th row of the \( X \) matrix, \( n \) is the number of observations, and \( k \) is the number of predictors in the model, including the constant if there is one. The above formula is the same as that used by `regress`, `vce(robust)` with the regression-like formula (the default) for the multiplier \( q_c \); see _Methods and formulas_ of \([R]\) `regress`.

For `lag(m)`, \( m > 0 \), the variance estimates are calculated using the Newey–West (1987) formulation

\[
X'\hat{\Omega}X = X'\hat{\Omega}_0X + \frac{n}{n-k} \sum_{l=1}^{m} \left( 1 - \frac{l}{m+1} \right) \sum_{t=l+1}^{n} \hat{c}_t \hat{c}_{t-l}(x_t'x_{t-l} + x_{t-l}'x_t)
\]

where \( x_t \) is the row of the \( X \) matrix observed at time \( t \).
Whitney K. Newey (1954– ) earned degrees in economics at Brigham Young University and MIT. After a period at Princeton, he returned to MIT as a professor in 1990. His interests in theoretical and applied econometrics include bootstrapping, nonparametric estimation of models, semiparametric models, and choosing the number of instrumental variables.

Kenneth D. West (1953– ) earned a bachelor’s degree in economics and mathematics at Wesleyan University and then a PhD in economics at MIT. After a period at Princeton, he joined the University of Wisconsin in 1988. His interests include empirical macroeconomics and time-series econometrics.

References


Also see

[TS] `newey postestimation` — Postestimation tools for newey

[TS] `arima` — ARIMA, ARMAX, and other dynamic regression models

[TS] `forecast` — Econometric model forecasting

[TS] `tsset` — Declare data to be time-series data

[R] `regress` — Linear regression

[U] 20 Estimation and postestimation commands
### Postestimation commands

The following postestimation commands are available after `newey`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contrast</code></td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>linktest</code></td>
<td>link test for model specification</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>pwcompare</code></td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

    predict [ type ] newvar [ if ] [ in ] [ , statistic ]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>standard error of the linear prediction</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for the estimation sample.

Options for predict

xb, the default, calculates the linear prediction.

stdp calculates the standard error of the linear prediction.

residuals calculates the residuals.
margins

Description for margins

margins estimates margins of response for linear predictions.

Menu for margins

Statistics > Postestimation

Syntax for margins

margins [marginlist] [ , options ]
margins [marginlist], predict(statistic ...) [options ]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

Statistics not allowed with margins are functions of stochastic quantities other than e(b).
For the full syntax, see [R] margins.

Remarks and examples

Example 1

We use the test command after newey to illustrate the importance of accounting for the presence of serial correlation in the error term. The dataset contains daily stock returns of three car manufacturers from January 2, 2003, to December 31, 2010, in the variables toyota, nissan, and honda.

We fit a model for the Nissan stock returns on the Honda and Toyota stock returns, and we use estat bgodfrey to test for serial correlation of order one:

```
. use https://www.stata-press.com/data/r16/stocks
(Data from Yahoo! Finance)
. regress nissan honda toyota
(output omitted)
. estat bgodfrey
Breusch-Godfrey LM test for autocorrelation

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.415</td>
<td>1</td>
<td>0.0113</td>
</tr>
</tbody>
</table>

H0: no serial correlation
```
The result implies that the error term is serially correlated; therefore, we should rather fit the model with *newey*. But let’s use the outcome from *regress* to conduct a test for the statistical significance of a particular linear combination of the two coefficients in the regression:

```
.test 1.15*honda+toyota = 1
      ( 1) 1.15*honda + toyota = 1
      F( 1, 2012) =  5.52
      Prob > F =  0.0189
```

We reject the null hypothesis that the linear combination is valid. Let’s see if the conclusion remains the same when we fit the model with *newey*, obtaining the Newey–West standard errors for the OLS coefficient estimates.

```
.newey nissan honda toyota,lag(1)
(output omitted)
.test 1.15*honda+toyota = 1
      ( 1) 1.15*honda + toyota = 1
      F( 1, 2012) =  2.57
      Prob > F =  0.1088
```

The conclusion would be the opposite, which illustrates the importance of using the proper estimator for the standard errors.

Example 2

We want to produce forecasts based on dynamic regressions for each of the three stocks. We will treat the stock returns for *toyota* as a leading indicator for the two other stocks. We also check for autocorrelation with the Breusch–Godfrey test.

```
.use https://www.stata-press.com/data/r16/stocks
(Data from Yahoo! Finance)
.regress toyota l(1/2).toyota
(output omitted)
.estat bgodfrey
Breusch-Godfrey LM test for autocorrelation

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.373</td>
<td>1</td>
<td>0.0365</td>
</tr>
</tbody>
</table>

H0: no serial correlation

.regress nissan l(1/2).nissan l.toyota
(output omitted)
.estat bgodfrey
Breusch-Godfrey LM test for autocorrelation

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.099</td>
<td>1</td>
<td>0.7536</td>
</tr>
</tbody>
</table>

H0: no serial correlation
. regress honda 1(1/2).honda l.toyota
(output omitted)
. estat bgodfrey
Breusch-Godfrey LM test for autocorrelation

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.923</td>
<td>1</td>
<td>0.3367</td>
</tr>
</tbody>
</table>

H0: no serial correlation

The first result indicates that we should consider using newey to fit the model for toyota. The point forecasts would not be actually affected because newey produces the same OLS coefficient estimates reported by regress. However, if we were interested in obtaining measures of uncertainty surrounding the point forecasts, we should then use the results from newey for that first equation.

Let's illustrate the use of forecast with newey for the first equation and regress for the two other equations. We first declare the forecast model:

. forecast create stocksmodel
Forecast model stocksmodel started.

Then we refit the equations and add them to the forecast model:

. quietly newey toyota 1(1/2).toyota, lag(1)
. estimates store eq_toyota
. forecast estimates eq_toyota
   Added estimation results from newey.
   Forecast model stocksmodel now contains 1 endogenous variable.
. quietly regress nissan 1(1/2).nissan l.toyota
. estimates store eq_nissan
. forecast estimates eq_nissan
   Added estimation results from regress.
   Forecast model stocksmodel now contains 2 endogenous variables.
. quietly regress honda 1(1/2).honda l.toyota
. estimates store eq_honda
. forecast estimates eq_honda
   Added estimation results from regress.
   Forecast model stocksmodel now contains 3 endogenous variables.

We use tsappend to add the number of periods for the forecast, and then we obtain the predicted values with forecast solve:

. tsappend, add(7)
. forecast solve, prefix(stk_)
Computing dynamic forecasts for model stocksmodel.

Starting period:  2016
Ending period:   2022
Forecast prefix: stk_
2016: ............
2017: ............
2018: ............
2019: ............
2020: ............
2021: ............
2022: ............
Forecast 3 variables spanning 7 periods.
The graph below shows several interesting results. First, the stock returns of the competitor (toyota) does not seem to be a leading indicator for the stock returns of the two other companies (otherwise, the patterns for the movements in nissan and honda would be following the recent past movements in toyota). You can actually fit the models above for nissan and honda to confirm that the coefficient estimate for the first lag of toyota is not significant in any of the two equations. Second, immediately after the second forecasted period, there is basically no variation in the predictions, which indicates the very short-run predicting influence of past history on the forecasts of the three stock returns.

![Current and forecasted stock returns](image)

Also see

[TS] newey — Regression with Newey–West standard errors
[U] 20 Estimation and postestimation commands
**Description**

`pergram` plots the log-standardized periodogram for a dense time series.

**Quick start**

Plot periodogram of \( y \) using `tsset` data

```
pergram y
```

As above, but generate variable `newvar` to hold raw periodogram values

```
pergram y, generate(newvar)
```

As above, but suppress display of periodogram graph

```
pergram y, generate(newvar) nograph
```

**Menu**

Statistics > Time series > Graphs > Periodogram
## Syntax

```plaintext
pergram varname [if] [in] [, options]
```

### Options

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>generate(newvar)</td>
<td>generate newvar to contain the raw periodogram values</td>
</tr>
<tr>
<td>Plot</td>
<td></td>
</tr>
<tr>
<td>cline_options</td>
<td>affect rendition of the plotted points connected by lines</td>
</tr>
<tr>
<td>marker_options</td>
<td>change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td>marker_label_options</td>
<td>add marker labels; change look or position</td>
</tr>
<tr>
<td>Add plots</td>
<td></td>
</tr>
<tr>
<td>addplot(plot)</td>
<td>add other plots to the generated graph</td>
</tr>
<tr>
<td>Y axis, X axis, Titles, Legend, Overall</td>
<td></td>
</tr>
<tr>
<td>twoway_options</td>
<td>any options other than by() documented in [G-3] twoway_options</td>
</tr>
<tr>
<td>nograph</td>
<td>suppress the graph</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `pergram`; see [TS] tsset. Also, the time series must be dense (nonmissing with no gaps in the time variable) in the specified sample. `varname` may contain time-series operators; see [U] 11.4.4 Time-series varlists. `nograph` does not appear in the dialog box.

## Options

**Main**

- `generate(newvar)` specifies a new variable to contain the raw periodogram values. The generated graph log-transforms and scales the values by the sample variance and then truncates them to the `[-6, 6]` interval before graphing them.

**Plot**

- `cline_options` affect the rendition of the plotted points connected by lines; see [G-3] cline_options.
- `marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] marker_options.
- `marker_label_options` specify if and how the markers are to be labeled; see [G-3] marker_label_options.

**Add plots**

- `addplot(plot)` adds specified plots to the generated graph; see [G-3] addplot_option.

**Y axis, X axis, Titles, Legend, Overall**

- `twoway_options` are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option).
The following option is available with `pergram` but is not shown in the dialog box: `nograph` prevents `pergram` from constructing a graph.

Remarks and examples

A good discussion of the periodogram is provided in Chatfield (2004), Hamilton (1994), and Newton (1988). Chatfield is also a good introductory reference for time-series analysis. Another classic reference is Box, Jenkins, and Reinsel (2008). `pergram` produces a scatterplot in which the points of the scatterplot are connected. The points themselves represent the log-standardized periodogram, and the connections between points represent the (continuous) log-standardized sample spectral density.

In the following examples, we present the periodograms with an interpretation of the main features of the plots.

Example 1

We have time-series data consisting of 144 observations on the monthly number of international airline passengers (in thousands) between 1949 and 1960 (Box, Jenkins, and Reinsel 2008, Series G). We can graph the raw series and the log periodogram for these data by typing

```
use https://www.stata-press.com/data/r16/air2
(scatter air time, m(o) c(l)
```

![Airline Passengers (1949−1960)
1950 1955 1960
Time (in months)](image)
The periodogram highlights the annual cycle together with the harmonics. Notice the peak at a frequency of about 0.08 cycles per month (cpm). The period is the reciprocal of frequency, and the reciprocal of 0.08 cpm is approximately 12 months per cycle. The similarity in shape of each group of 12 observations reveals the annual cycle. The magnitude of the cycle is increasing, resulting in the peaks in the periodogram at the harmonics of the principal annual cycle.

Example 2

This example uses 215 observations on the annual number of sunspots from 1749 to 1963 (Box and Jenkins 1976, Series E). The graph of the raw series and the log periodogram for these data are given as

```
use https://www.stata-press.com/data/r16/sunspot
(scatter spot time, m(o) c(l)
```

The periodogram highlights the annual cycle together with the harmonics. Notice the peak at a frequency of about 0.08 cycles per month (cpm). The period is the reciprocal of frequency, and the reciprocal of 0.08 cpm is approximately 12 months per cycle. The similarity in shape of each group of 12 observations reveals the annual cycle. The magnitude of the cycle is increasing, resulting in the peaks in the periodogram at the harmonics of the principal annual cycle.

Example 2

This example uses 215 observations on the annual number of sunspots from 1749 to 1963 (Box and Jenkins 1976, Series E). The graph of the raw series and the log periodogram for these data are given as

```
use https://www.stata-press.com/data/r16/sunspot
(scatter spot time, m(o) c(l)
```
The periodogram peaks at a frequency of slightly less than 0.10 cycles per year, indicating a 10- to 12-year cycle in sunspot activity.

Example 3

Here we examine the number of trapped Canadian lynx from 1821 through 1934 (Newton 1988, 587). The raw series and the log periodogram are given as

```
. use https://www.stata-press.com/data/r16/lynx2
(TIMESLAB: Canadian lynx)
. scatter lynx time, m(o) c(l)
```
The periodogram indicates that there is a cycle with a duration of about 10 years for these data but that it is otherwise random.

Example 4

To more clearly highlight what the periodogram depicts, we present the result of analyzing a time series of the sum of four sinusoids (of different periods). The periodogram should be able to decompose the time series into four different sinusoids whose periods may be determined from the plot.
The periodogram clearly shows the four contributions to the original time series. From the plot, we can see that the periods of the summands were 3, 6, 12, and 36, although you can confirm this by using

```
. generate double omega = (_n-1)/144
. generate double period = 1/omega
(1 missing value generated)
. list period omega if ordinate> 1e-5 & omega <=.5
```

<table>
<thead>
<tr>
<th>period</th>
<th>omega</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.</td>
<td>36</td>
</tr>
<tr>
<td>13.</td>
<td>12</td>
</tr>
<tr>
<td>25.</td>
<td>6</td>
</tr>
<tr>
<td>49.</td>
<td>3</td>
</tr>
</tbody>
</table>

Methods and formulas

We use the notation of Newton (1988) in the following discussion.

A time series of interest is decomposed into a unique set of sinusoids of various frequencies and amplitudes.

A plot of the sinusoidal amplitudes (ordinates) versus the frequencies for the sinusoidal decomposition of a time series gives us the spectral density of the time series. If we calculate the sinusoidal amplitudes for a discrete set of “natural” frequencies \((1/n, 2/n, \ldots, q/n)\), we obtain the periodogram.

Let \(x(1), \ldots, x(n)\) be a time series, and let \(\omega_k = (k - 1)/n\) denote the natural frequencies for \(k = 1, \ldots, (n/2) + 1\). Define

\[
C_k^2 = \frac{1}{n^2} \left| \sum_{t=1}^{n} x(t)e^{2\pi i(t-1)\omega_k} \right|^2
\]

A plot of \(nC_k^2\) versus \(\omega_k\) is then called the periodogram.
The sample spectral density is defined for a continuous frequency \( \omega \) as

\[
\hat{f}(\omega) = \begin{cases} 
\frac{1}{n} \left| \sum_{t=1}^{n} x(t) e^{2\pi i (t-1) \omega} \right|^2 & \text{if } \omega \in [0, 0.5] \\
\hat{f}(1 - \omega) & \text{if } \omega \in [0.5, 1] 
\end{cases}
\]

The periodogram (and sample spectral density) is symmetric about \( \omega = 0.5 \). Further standardize the periodogram such that

\[
\frac{1}{n} \sum_{k=2}^{n} nC_k^2 \hat{\sigma}^2 = 1
\]

where \( \hat{\sigma}^2 \) is the sample variance of the time series so that the average value of the ordinate is one.

Once the amplitudes are standardized, we may then take the natural log of the values and produce the log periodogram. In doing so, we truncate the graph at \( \pm 6 \). We drop the word “log” and simply refer to the “log periodogram” as the “periodogram” in text.

References


Also see

[TS] **corrgram** — Tabulate and graph autocorrelations
[TS] **cums** — Graph cumulative spectral distribution
[TS] **tsset** — Declare data to be time-series data
[TS] **wntestb** — Bartlett’s periodogram-based test for white noise
pperron — Phillips–Perron unit-root test

Description

pperron performs the Phillips–Perron (1988) test that a variable has a unit root. The null hypothesis is that the variable contains a unit root, and the alternative is that the variable was generated by a stationary process. pperron uses Newey–West (1987) standard errors to account for serial correlation, whereas the augmented Dickey–Fuller test implemented in dfuller (see [TS] dfuller) uses additional lags of the first-differenced variable.

Quick start

Phillips–Perron unit-root test for \( y \) using \texttt{tsset} data

\begin{verbatim}
pperron y
\end{verbatim}

As above, and include a trend in the specification

\begin{verbatim}
pperron y, trend
\end{verbatim}

As above, but use 10 lags when calculating Newey–West standard errors

\begin{verbatim}
pperron y, trend lags(10)
\end{verbatim}

As above, but without a trend or constant in the specification

\begin{verbatim}
pperron y, lags(10) noconstant
\end{verbatim}

Menu

Statistics > Time series > Tests > Phillips-Perron unit-root test
Syntax

```
pperron  varname  [if]  [in]  [,  options]
```

options       Description

Main

- `noconstant`  suppress constant term
- `trend`       include trend term in regression
- `regress`     display regression table
- `lags(#)`     use # Newey–West lags

You must `tsset` your data before using `pperron`; see `[TS] tsset`.

`varname` may contain time-series operators; see `[U] 11.4.4 Time-series varlists`.

Options

- `noconstant` suppresses the constant term (intercept) in the model.
- `trend` specifies that a trend term be included in the associated regression. This option may not be specified if `noconstant` is specified.
- `regress` specifies that the associated regression table appear in the output. By default, the regression table is not produced.
- `lags(#)` specifies the number of Newey–West lags to use in calculating the standard error. The default is to use \( \text{int} \left( \frac{4(T/100)^2/9}{T} \right) \) lags.

Remarks and examples

As noted in `[TS] dfuller`, the Dickey–Fuller test involves fitting the regression model

\[
y_t = \alpha + \rho y_{t-1} + \delta t + u_t
\]

by ordinary least squares (OLS), but serial correlation will present a problem. To account for this, the augmented Dickey–Fuller test’s regression includes lags of the first differences of \( y_t \).

The Phillips–Perron test involves fitting (1), and the results are used to calculate the test statistics. Phillips and Perron (1988) proposed two alternative statistics, which `pperron` presents. Phillips and Perron’s test statistics can be viewed as Dickey–Fuller statistics that have been made robust to serial correlation by using the Newey–West (1987) heteroskedasticity- and autocorrelation-consistent covariance matrix estimator.
Hamilton (1994, chap. 17) and [TS] dfuller discuss four different cases into which unit-root tests can be classified. The Phillips–Perron test applies to cases one, two, and four but not to case three. Cases one and two assume that the variable has a unit root without drift under the null hypothesis, the only difference being whether the constant term $\alpha$ is included in regression (1). Case four assumes that the variable has a random walk, with or without drift, under the null hypothesis. Case three, which assumes that the variable has a random walk with drift under the null hypothesis, is just a special case of case four, so the fact that the Phillips–Perron test does not apply is not restrictive.

The table below summarizes the relevant cases:

<table>
<thead>
<tr>
<th>Case</th>
<th>Process under null hypothesis</th>
<th>Regression restrictions</th>
<th>dfuller option</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Random walk without drift</td>
<td>$\alpha = 0, \delta = 0$</td>
<td>noconstant</td>
</tr>
<tr>
<td>2</td>
<td>Random walk without drift</td>
<td>$\delta = 0$</td>
<td>(default)</td>
</tr>
<tr>
<td>4</td>
<td>Random walk with or without drift</td>
<td>(none)</td>
<td>trend</td>
</tr>
</tbody>
</table>

The critical values for the Phillips–Perron test are the same as those for the augmented Dickey–Fuller test. See Hamilton (1994, chap. 17) for more information.

Example 1

Here we use the international airline passengers dataset (Box, Jenkins, and Reinsel 2008, Series G). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. Because the data exhibit a clear upward trend over time, we will use the trend option.

```
. use https://www.stata-press.com/data/r16/air2
  (TIMESLAB: Airline passengers)
  . pperron air, lags(4) trend regress
```

Phillips–Perron test for unit root

<table>
<thead>
<tr>
<th>Test</th>
<th>1% Critical Value</th>
<th>5% Critical Value</th>
<th>10% Critical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z(rho)</td>
<td>-46.405</td>
<td>-27.687</td>
<td>-20.872</td>
</tr>
<tr>
<td>Z(t)</td>
<td>-5.049</td>
<td>-4.026</td>
<td>-3.444</td>
</tr>
</tbody>
</table>

MacKinnon approximate p-value for $Z(t) = 0.0002$

| air | Coef. | Std. Err. | t | P>|t| | [95% Conf. Interval] |
|-----|-------|-----------|---|-----|---------------------|
| air | .7318116 | .0578092 | 12.66 | 0.000 | .6175196 | .8461035 |
| L1. | .7107559 | .1670563 | 4.25 | 0.000 | .3804767 | 1.041035 |
| _trend | 25.95168 | 7.325951 | 3.54 | 0.001 | 11.46788 | 40.43547 |

Just as in the example in [TS] dfuller, we reject the null hypothesis of a unit root at all common significance levels. The interpolated critical values for $Z_t$ differ slightly from those shown in the example in [TS] dfuller because the sample sizes are different: with the augmented Dickey–Fuller regression we lose observations because of the inclusion of lagged difference terms as regressors.
pperron stores the following in \( r() \):  

Scalars  
\[ r(N) \] number of observations  
\[ r(lags) \] number of lagged differences used  
\[ r(pval) \] MacKinnon approximate \( p \)-value (not included if noconstant specified)  
\[ r(Zt) \] Phillips–Perron \( \tau \) test statistic  
\[ r(Zrho) \] Phillips–Perron \( \rho \) test statistic

Methods and formulas

In the OLS estimation of an AR(1) process with Gaussian errors,  
\[ y_i = \rho y_{i-1} + \epsilon_i \]

where \( \epsilon_i \) are independent and identically distributed as \( N(0, \sigma^2) \) and \( y_0 = 0 \), the OLS estimate (based on an \( n \)-observation time series) of the autocorrelation parameter \( \rho \) is given by  
\[ \hat{\rho}_n = \frac{\sum_{i=1}^{n} y_{i-1}y_i}{\sum_{i=1}^{n} y_i^2} \]

If \( |\rho| < 1 \), then \( \sqrt{n}(\hat{\rho}_n - \rho) \rightarrow N(0, 1 - \rho^2) \). If this result were valid for when \( \rho = 1 \), then the resulting distribution would have a variance of zero. When \( \rho = 1 \), the OLS estimate \( \hat{\rho} \) still converges to one, though we need to find a nondegenerate distribution so that we can test \( H_0: \rho = 1 \). See Hamilton (1994, chap. 17).

The Phillips–Perron test involves fitting the regression  
\[ y_i = \alpha + \rho y_{i-1} + \epsilon_i \]

where we may exclude the constant or include a trend term. There are two statistics, \( Z_{\rho} \) and \( Z_{\tau} \), calculated as

\[
Z_{\rho} = n(\hat{\rho}_n - 1) - \frac{1}{2} \frac{n^2 \sigma^2}{\lambda_n^2} (\hat{\lambda}_n^2 - \gamma_{0,n})
\]

\[
Z_{\tau} = \sqrt{\frac{\gamma_{0,n} \hat{\rho}_n - 1}{\lambda_n^2} \frac{1}{\sigma} - \frac{1}{2} \left( \hat{\lambda}_n^2 - \gamma_{0,n} \right) \frac{1}{\lambda_n} \frac{n\sigma}{s_n^2}}
\]

\[
\hat{\gamma}_{j,n} = \frac{1}{n} \sum_{i=j+1}^{n} \hat{u}_i \hat{u}_{i-j}
\]

\[
\hat{\lambda}_n^2 = \gamma_{0,n} + 2 \sum_{j=1}^{q} \left( 1 - \frac{j}{q + 1} \right) \hat{\gamma}_{j,n}
\]

\[
s_n^2 = \frac{1}{n-k} \sum_{i=1}^{n} \hat{u}_i^2
\]

\[
\gamma_{0,n} = \frac{1}{n \gamma_{0,n}} \sum_{i=1}^{n} \hat{u}_i^2
\]
where \( u_i \) is the OLS residual, \( k \) is the number of covariates in the regression, \( q \) is the number of Newey–West lags to use in calculating \( \hat{\lambda}^2_n \), and \( \hat{\sigma} \) is the OLS standard error of \( \hat{\rho} \).

The critical values, which have the same distribution as the Dickey–Fuller statistic (see Dickey and Fuller 1979) included in the output, are linearly interpolated from the table of values that appear in Fuller (1996), and the MacKinnon approximate \( p \)-values use the regression surface published in MacKinnon (1994).

Peter Charles Bonest Phillips (1948– ) was born in Weymouth, England, and earned degrees in economics at the University of Auckland in New Zealand, and the London School of Economics. After periods at the Universities of Essex and Birmingham, Phillips moved to Yale in 1979. He also holds appointments at the University of Auckland and the University of York. His main research interests are in econometric theory, financial econometrics, time-series and panel-data econometrics, and applied macroeconomics.

Pierre Perron (1959– ) was born in Québec, Canada, and earned degrees at McGill, Queen’s, and Yale in economics. After posts at Princeton and the Université de Montréal, he joined Boston University in 1997. His research interests include time-series analysis, econometrics, and applied macroeconomics.

References


Also see

[TS] dfgls — DF-GLS unit-root test
[TS] dfuller — Augmented Dickey–Fuller unit-root test
[TS] tsset — Declare data to be time-series data
[XT] xtunitroot — Panel-data unit-root tests
**Description**

`prais` uses the generalized least-squares method to estimate the parameters in a linear regression model in which the errors are serially correlated. Specifically, the errors are assumed to follow a first-order autoregressive process.

**Quick start**

Prais–Winsten regression of $y$ on $x$ estimating the autocorrelation parameter by a single-lag OLS regression of residuals using `tsset` data

```
prais y x
```

As above, but estimate the autocorrelation parameter using a single-lead OLS of residuals

```
prais y x, rhotype(freg)
```

As above, but estimate the autocorrelation parameter using autocorrelation of residuals

```
prais y x, rhotype(tscorr)
```

Cochrane–Orcutt regression of $y$ on $x$ with first-order serial correlation

```
prais y x, corc
```
Syntax

```
prais depvar [ indepvars ] [ if ] [ in ] [ , options ]
```

**options**

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>base ρ on single-lag OLS of residuals; the default</td>
<td><code>rho</code>type(regress)</td>
</tr>
<tr>
<td>base ρ on single-lead OLS of residuals</td>
<td><code>rho</code>type(freg)</td>
</tr>
<tr>
<td>base ρ on autocorrelation of residuals</td>
<td><code>rho</code>type(tscorr)</td>
</tr>
<tr>
<td>base ρ on autocorrelation based on Durbin–Watson</td>
<td><code>rho</code>type(dw)</td>
</tr>
<tr>
<td>base ρ on adjusted autocorrelation</td>
<td><code>rho</code>type(theil)</td>
</tr>
<tr>
<td>base ρ on adjusted Durbin–Watson</td>
<td><code>rho</code>type(nagar)</td>
</tr>
<tr>
<td>use Cochrane–Orcutt transformation</td>
<td><code>corc</code></td>
</tr>
<tr>
<td>search for ρ that minimizes SSE</td>
<td><code>sseseach</code></td>
</tr>
<tr>
<td>stop after the first iteration</td>
<td><code>twostep</code></td>
</tr>
<tr>
<td>suppress constant term</td>
<td><code>noconstant</code></td>
</tr>
<tr>
<td>has user-defined constant</td>
<td><code>hascons</code></td>
</tr>
<tr>
<td>conserve memory during estimation</td>
<td><code>savespace</code></td>
</tr>
<tr>
<td><code>vcetype</code> may be ols, robust, cluster <code>clustvar</code>, hc2, or hc3</td>
<td><code>vce(vcetype)</code></td>
</tr>
<tr>
<td>set confidence level; default is level(95)</td>
<td><code>level(#)</code></td>
</tr>
<tr>
<td>do not report the Durbin–Watson statistic</td>
<td><code>nodw</code></td>
</tr>
<tr>
<td>control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling</td>
<td><code>display_options</code></td>
</tr>
<tr>
<td>control the optimization process; seldom used</td>
<td><code>optimize_options</code></td>
</tr>
<tr>
<td>display legend instead of statistics</td>
<td><code>coeflegend</code></td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `prais`; see [TS] `tsset`.

`indepvars` may contain factor variables; see [U] 11.4.3 Factor variables.

`depvar` and `indepvars` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, fp, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.

`coeflegend` does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
Options

`rhostype(rhomethod)` selects a specific computation for the autocorrelation parameter $\rho$, where `rhomethod` can be

- `regress`: $\rho_{\text{reg}} = \beta$ from the residual regression $\epsilon_t = \beta \epsilon_{t-1}$
- `freg`: $\rho_{\text{freg}} = \beta$ from the residual regression $\epsilon_t = \beta \epsilon_{t+1}$
- `tscorr`: $\rho_{\text{tscorr}} = \epsilon' \epsilon_{t-1}/\epsilon' \epsilon$, where $\epsilon$ is the vector of residuals
- `dw`: $\rho_{\text{dw}} = 1 - \text{dw}/2$, where dw is the Durbin–Watson $d$ statistic
- `theil`: $\rho_{\text{theil}} = \rho_{\text{tscorr}} (N - k)/N$
- `nagar`: $\rho_{\text{nagar}} = (\rho_{\text{dw}} \ast N^2 + k^2)/(N^2 - k^2)$

The `prais` estimator can use any consistent estimate of $\rho$ to transform the equation, and each of these estimates meets that requirement. The default is `regress`, which produces the minimum sum-of-squares solution (`ssesearch` option) for the Cochrane–Orcutt transformation—none of these computations will produce the minimum sum-of-squares solution for the full Prais–Winsten transformation. See Judge et al. (1985) for a discussion of each estimate of $\rho$.

`corc` specifies that the Cochrane–Orcutt transformation be used to estimate the equation. With this option, the Prais–Winsten transformation of the first observation is not performed, and the first observation is dropped when estimating the transformed equation; see `Methods and formulas` below.

`ssesearch` specifies that a search be performed for the value of $\rho$ that minimizes the sum-of-squared errors of the transformed equation (Cochrane–Orcutt or Prais–Winsten transformation). The search method is a combination of quadratic and modified bisection searches using golden sections.

`twostep` specifies that `prais` stops on the first iteration after the equation is transformed by $\rho$—the two-step efficient estimator. Although iterating these estimators to convergence is customary, they are efficient at each step.

`noconstant`; see [R] Estimation options.

`hascons` indicates that a user-defined constant, or a set of variables that in linear combination forms a constant, has been included in the regression. For some computational concerns, see the discussion in [R] regress.

`savespace` specifies that `prais` attempt to save as much space as possible by retaining only those variables required for estimation. The original data are restored after estimation. This option is rarely used and should be used only if there is insufficient space to fit a model without the option.

`vce(vcetype)` specifies the estimator for the variance–covariance matrix of the estimator; see [R] vce_option.

`vce(ols)`, the default, uses the standard variance estimator for ordinary least-squares regression.

`vce(robust)` specifies to use the Huber/White/sandwich estimator.

`vce(cluster clustvar)` specifies to use the intragroup correlation estimator.

`vce(hc2)` and `vce(hc3)` specify an alternative bias correction for the `vce(robust)` variance calculation; for more information, see [R] regress. You may specify only one of `vce(hc2)`, `vce(hc3)`, or `vce(robust)`. 

---

496  prais — Prais–Winsten and Cochrane–Orcutt regression
All estimates from *prais* are conditional on the estimated value of $\rho$. Robust variance estimates here are robust only to heteroskedasticity and are not generally robust to misspecification of the functional form or omitted variables. The estimation of the functional form is intertwined with the estimation of $\rho$, and all estimates are conditional on $\rho$. Thus estimates cannot be robust to misspecification of functional form. For these reasons, it is probably best to interpret `vce(robust)` in the spirit of White’s (1980) original paper on estimation of heteroskedastic-consistent covariance matrices.

**Reporting**

*level(#)***; see [R] Estimation options.


`display_options`: `noci`, `nopvalues`, `noomitted`, `vsquish`, `nomemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(style)`, `cformat(%,fnt)`, `pformat(%,fnt)`, `sformat(%,fnt)`, and `nolstretch`; see [R] Estimation options.

**Optimization**

`optimize_options`: `iterate(#)`, `[no]log, tolerance(#)`. `iterate()` specifies the maximum number of iterations. `log/nolog` specifies whether to show the iteration log (see `set iterlog` in [R] set iter). `tolerance()` specifies the tolerance for the coefficient vector; `tolerance(1e-6)` is the default. These options are seldom used.

The following option is available with *prais* but is not shown in the dialog box: `coeflegend`; see [R] Estimation options.

**Remarks and examples**

*prias* fits a linear regression of *depvar* on *indepvars* that is corrected for first-order serially correlated residuals by using the Prais–Winsten (1954) transformed regression estimator, the Cochrane–Orcutt (1949) transformed regression estimator, or a version of the search method suggested by Hildreth and Lu (1960). Davidson and MacKinnon (1993) provide theoretical details on the three methods (see pages 333–335 for the latter two and pages 343–351 for Prais–Winsten). See Becketti (2013) for more examples showing how to use *prais*.

The most common autocorrelated error process is the first-order autoregressive process. Under this assumption, the linear regression model can be written as

$$y_t = x_t \beta + u_t$$

where the errors satisfy

$$u_t = \rho u_{t-1} + e_t$$

and the $e_t$ are independent and identically distributed as $N(0, \sigma^2)$. The covariance matrix $\Psi$ of the error term $u$ can then be written as

$$\Psi = \frac{1}{1 - \rho^2} \begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{T-1} \\ \rho & 1 & \rho & \cdots & \rho^{T-2} \\ \rho^2 & \rho & 1 & \cdots & \rho^{T-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{T-1} & \rho^{T-2} & \rho^{T-3} & \cdots & 1 \end{bmatrix}$$
The Prais–Winsten estimator is a generalized least-squares (GLS) estimator. The Prais–Winsten method (as described in Judge et al. 1985) is derived from the AR(1) model for the error term described above. Whereas the Cochrane–Orcutt method uses a lag definition and loses the first observation in the iterative method, the Prais–Winsten method preserves that first observation. In small samples, this can be a significant advantage.

Technical note

To fit a model with autocorrelated errors, you must specify your data as time series and have (or create) a variable denoting the time at which an observation was collected. The data for the regression should be equally spaced in time.

Example 1

Say that we wish to fit a time-series model of `usr` on `idle` but are concerned that the residuals may be serially correlated. We will declare the variable `t` to represent time by typing

```
   . use https://www.stata-press.com/data/r16/idle
   . tsset t
      time variable:  t, 1 to 30
      delta:  1 unit
```

We can obtain Cochrane–Orcutt estimates by specifying the `corc` option:

```
   . prais usr idle, corc
   Iteration 0:  rho = 0.0000
   Iteration 1:  rho = 0.3518
   (output omitted)
   Iteration 13:  rho = 0.5708

   Cochrane-Orcutt AR(1) regression -- iterated estimates
   Number of obs = 29  
   F(1, 27) = 6.49
   Prob > F = 0.0168
   R-squared = 0.1938
   Adj R-squared = 0.1640
   Root MSE = 2.4862

   usr       Coef.    Std. Err.     t     P>|t|     [95% Conf. Interval]
   -------------    -----------    -----    ------    ---------------    --------
   idle        -.1254511    .0492356    -2.55    0.017     -.2264742    -.024428
   _cons       14.54641    4.272299     3.40    0.002     5.78038    23.31245
   rho          .5707918

   Durbin-Watson statistic (original)  1.295766
   Durbin-Watson statistic (transformed)  1.466222
```

The fitted model is

\[
usr_t = -0.1255 \text{idle}_t + 14.55 + u_t \quad \text{and} \quad u_t = 0.5708 u_{t-1} + e_t
\]
We can also fit the model with the Prais–Winsten method,

```
.prais usr idle
Iteration 0: rho = 0.0000
Iteration 1: rho = 0.3518
(output omitted)
Iteration 14: rho = 0.5535
```

Prais-Winsten AR(1) regression -- iterated estimates

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>Number of obs = 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>43.0076941</td>
<td>1</td>
<td>43.0076941</td>
<td>F(1, 28) = 7.12</td>
</tr>
<tr>
<td>Residual</td>
<td>169.165739</td>
<td>28</td>
<td>6.04163354</td>
<td>Prob &gt; F = 0.0125</td>
</tr>
<tr>
<td>Total</td>
<td>212.173433</td>
<td>29</td>
<td>7.31632528</td>
<td>R-squared = 0.2027</td>
</tr>
</tbody>
</table>

| usr         | Coef.  | Std. Err. | t    | P>|t|  | [95% Conf. Interval] |
|------------|--------|-----------|------|------|----------------------|
| idle       | -.1356522 | .0472195  | -2.87 | 0.008 | -.2323769 -.0389275  |
| _cons      | 15.20415 | 4.160391  | 3.65  | 0.001 | 6.681978  23.72633  |

| rho         | .5535476 |

Durbin-Watson statistic (original) 1.295766
Durbin-Watson statistic (transformed) 1.476004

where the Prais–Winsten fitted model is

\[ \text{usr}_t = -0.1357 \text{idle}_t + 15.20 + u_t \quad \text{and} \quad u_t = 0.5535 u_{t-1} + e_t \]

As the results indicate, for these data there is little difference between the Cochrane–Orcutt and Prais–Winsten estimators, whereas the OLS estimate of the slope parameter is substantially different.

---

**Example 2**

We have data on quarterly sales, in millions of dollars, for 5 years, and we would like to use this information to model sales for company X. First, we fit a linear model by OLS and obtain the Durbin–Watson statistic by using `estat dwatson`; see [R] `regress postestimation time series`.

```
.use https://www.stata-press.com/data/r16/qsales
.regress csales isales
```

```
Source | SS          | df  | MS        | Number of obs = 20 |
-------|-------------|-----|-----------|--------------------|
Model  | 110.256901  | 1   | 110.256901| F(1, 18) = 14888.15 |
Residual | .133302302  | 18  | .007405683| Prob > F = 0.0000 |
Total  | 110.390204  | 19  | 5.81001072| R-squared = 0.9988 |

| csales      | Coef.   | Std. Err. | t     | P>|t|  | [95% Conf. Interval] |
|-------------|---------|-----------|-------|------|----------------------|
| isales      | .1762828 | .0014447  | 122.02| 0.000 | .1732475 .1793181   |
| _cons       | -1.454753| .2141461  | -6.79 | 0.000 | -1.904657 -1.004849 |

| isales      | Coef.   | Std. Err. | t     | P>|t|  | [95% Conf. Interval] |
|-------------|---------|-----------|-------|------|----------------------|
| _cons       | -1.454753| .2141461  | -6.79 | 0.000 | -1.904657 -1.004849 |
```

```
.estat dwatson
Durbin-Watson d-statistic( 2, 20) = .7347276
```

---

Although Stata does not perform the Prais–Winsten procedure by default, it is easy to write a program that does.
Because the Durbin–Watson statistic is far from 2 (the expected value under the null hypothesis of no serial correlation) and well below the 5% lower limit of 1.2, we conclude that the disturbances are serially correlated. (Upper and lower bounds for the $d$ statistic can be found in most econometrics texts; for example, Harvey [1990]. The bounds have been derived for only a limited combination of regressors and observations.) To reinforce this conclusion, we use two other tests to test for serial correlation in the error distribution.

```
. estat bgodfrey, lags(1)
Breusch-Godfrey LM test for autocorrelation

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.998</td>
<td>1</td>
<td>0.0047</td>
</tr>
</tbody>
</table>

H0: no serial correlation
```

```
. estat durbinalt
Durbin's alternative test for autocorrelation

<table>
<thead>
<tr>
<th>lags(p)</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.329</td>
<td>1</td>
<td>0.0008</td>
</tr>
</tbody>
</table>

H0: no serial correlation
```

`estat bgodfrey` reports the Breusch–Godfrey Lagrange multiplier test statistic, and `estat durbinalt` reports the Durbin’s alternative test statistic. Both tests give a small $p$-value and thus reject the null hypothesis of no serial correlation. These two tests are asymptotically equivalent when testing for AR(1) process. See [R] `regress postestimation time series` if you are not familiar with these two tests.

We correct for autocorrelation with the `ssesearch` option of `prais` to search for the value of $\rho$ that minimizes the sum-of-squared residuals of the Cochrane–Orcutt transformed equation. Normally, the default Prais–Winsten transformations is used with such a small dataset, but the less-efficient Cochrane–Orcutt transformation allows us to demonstrate an aspect of the estimator’s convergence.

```
. prais csales isales, corc ssesearch
Iteration 1: rho = 0.8944 , criterion = -.07298558
Iteration 2: rho = 0.8944 , criterion = -.07298558
(output omitted)
Iteration 15: rho = 0.9588 , criterion = -.07167037
Cochrane-Orcutt AR(1) regression -- SSE search estimates

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>Number of obs = 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>2.33199178</td>
<td>1 2.33199178</td>
<td>Prob &gt; F = 0.0000</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>.071670369</td>
<td>17 .004215904</td>
<td>R-squared = 0.9702</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2.40366215</td>
<td>18 .133536786</td>
<td>Root MSE = .06493</td>
<td></td>
</tr>
</tbody>
</table>

| csales     | Coef.    | Std. Err. | t      | P>|t|  | [95% Conf. Interval] |
|------------|----------|-----------|--------|------|----------------------|
| isales     | .1605233 | .0068253  | 23.52  | 0.000 | .1461233 .1749234    |
| _cons      | 1.738946 | 1.432674  | 1.21   | 0.241 | -1.283732 4.761624   |
| rho        | .9588209 |           |        |      |                      |

Durbin-Watson statistic (original) 0.734728
Durbin-Watson statistic (transformed) 1.724419
We noted in *Options* that, with the default computation of $\rho$, the Cochrane–Orcutt method produces an estimate of $\rho$ that minimizes the sum-of-squared residuals—the same criterion as the `ssesearch` option. Given that the two methods produce the same results, why would the search method ever be preferred? It turns out that the back-and-forth iterations used by Cochrane–Orcutt may have difficulty converging if the value of $\rho$ is large. Using the same data, the Cochrane–Orcutt iterative procedure requires more than 350 iterations to converge, and a higher tolerance must be specified to prevent premature convergence:

```
.prais csales isales, corc tol(1e-9) iterate(500)
```

\[
\begin{array}{l}
\text{Iteration 0: rho = 0.0000} \\
\text{Iteration 1: rho = 0.6312} \\
\text{Iteration 2: rho = 0.6866} \\
\text{(output omitted)} \\
\text{Iteration 377: rho = 0.9588} \\
\text{Iteration 378: rho = 0.9588} \\
\text{Iteration 379: rho = 0.9588} \\
\end{array}
\]

Once convergence is achieved, the two methods produce identical results.


**Stored results**

`prais` stores the following in `e()`:

**Scalars**
- `e(N)` number of observations
- `e(N_gaps)` number of gaps
- `e(mss)` model sum of squares
- `e(df_m)` model degrees of freedom
- `e(rss)` residual sum of squares
- `e(df_r)` residual degrees of freedom
- `e(r2)` \(R^2\)
- `e(r2_a)` adjusted \(R^2\)
- `e(F)` \(F\) statistic
- `e(rmse)` root mean squared error
- `e(ll)` log likelihood
- `e(N_clust)` number of clusters
- `e(rho)` autocorrelation parameter \(\rho\)
- `e(dw)` Durbin–Watson \(d\) statistic for transformed regression
- `e(dw_0)` Durbin–Watson \(d\) statistic of untransformed regression
- `e(rank)` rank of `e(V)`
- `e(tol)` target tolerance
- `e(max_ic)` maximum number of iterations
- `e(ic)` number of iterations

**Macros**
- `e(cmd)` `prais`
- `e(cmdline)` command as typed
- `e(depvar)` name of dependent variable
- `e(title)` title in estimation output
- `e(clustvar)` name of cluster variable
- `e(cons)` noconstant or not reported
- `e(method)` twostep, iterated, or SSE search
- `e(tranmeth)` corc or prais
- `e(rhotype)` method specified in `rhotype()` option
- `e(vce)` `vce()` specified in `vce()`
- `e(vcetype)` title used to label Std. Err.
- `e(properties)` \(b\) \(V\)
- `e(predict)` program used to implement `predict`
- `e(marginsok)` predictions allowed by `margins`
- `e(asbalanced)` factor variables `fvset` as `asbalanced`
- `e(asobserved)` factor variables `fvset` as `asobserved`

**Matrices**
- `e(b)` coefficient vector
- `e(V)` variance–covariance matrix of the estimators
- `e(V_modelbased)` model-based variance

**Functions**
- `e(sample)` estimation sample

---

**Methods and formulas**

Consider the command `prais y x z`. The 0th iteration is obtained by estimating \(a\), \(b\), and \(c\) from the standard linear regression:

\[
y_t = ax_t + bz_t + c + u_t
\]

An estimate of the correlation in the residuals is then obtained. By default, `prais` uses the auxiliary regression:

\[
u_t = \rho u_{t-1} + e_t
\]

This can be changed to any computation noted in the `rhotype()` option.
Next we apply a Cochrane–Orcutt transformation (1) for observations $t = 2, \ldots, n$

$$y_t - \rho y_{t-1} = a(x_t - \rho x_{t-1}) + b(z_t - \rho z_{t-1}) + c(1 - \rho) + v_t$$  \hspace{1cm} (1)

and the transformation (1') for $t = 1$

$$\sqrt{1 - \rho^2} y_1 = a(\sqrt{1 - \rho^2} x_1) + b(\sqrt{1 - \rho^2} z_1) + c\sqrt{1 - \rho^2} + \sqrt{1 - \rho^2} v_1$$  \hspace{1cm} (1')

Thus the differences between the Cochrane–Orcutt and the Prais–Winsten methods are that the latter uses (1') in addition to (1), whereas the former uses only (1), necessarily decreasing the sample size by one.

Equations (1) and (1') are used to transform the data and obtain new estimates of $a$, $b$, and $c$.

When the *twostep* option is specified, the estimation process stops at this point and reports these estimates. Under the default behavior of iterating to convergence, this process is repeated until the change in the estimate of $\rho$ is within a specified tolerance.

The new estimates are used to produce fitted values

$$\hat{y}_t = \hat{a}x_t + \hat{b}z_t + \hat{c}$$

and then $\rho$ is reestimated using, by default, the regression defined by

$$y_t - \hat{y}_t = \rho(y_{t-1} - \hat{y}_{t-1}) + u_t$$  \hspace{1cm} (2)

We then reestimate (1) by using the new estimate of $\rho$ and continue to iterate between (1) and (2) until the estimate of $\rho$ converges.

Convergence is declared after *iterate()* iterations or when the absolute difference in the estimated correlation between two iterations is less than *tol()*; see [R] Maximize. Sargan (1964) has shown that this process will always converge.

Under the *sesearch* option, a combined quadratic and bisection search using golden sections searches for the value of $\rho$ that minimizes the sum-of-squared residuals from the transformed equation. The transformation may be either the Cochrane–Orcutt (1 only) or the Prais–Winsten (1 and 1').

All reported statistics are based on the $\rho$-transformed variables, and $\rho$ is assumed to be estimated without error. See Judge et al. (1985) for details.

The Durbin–Watson $d$ statistic reported by prais and estat dwatson is

$$d = \frac{\sum_{j=1}^{n-1} (u_{j+1} - u_j)^2}{\sum_{j=1}^{n} u_j^2}$$

where $u_j$ represents the residual of the $j$th observation.

This command supports the Huber/White/sandwich estimator of the variance and its clustered version using *vce(robust)* and *vce(cluster clustvar)*, respectively. See [P] _robust, particularly Introduction and Methods and formulas.
All estimates from `prais` are conditional on the estimated value of $\rho$. Robust variance estimates here are robust only to heteroskedasticity and are not generally robust to misspecification of the functional form or omitted variables. The estimation of the functional form is intertwined with the estimation of $\rho$, and all estimates are conditional on $\rho$. Thus estimates cannot be robust to misspecification of functional form. For these reasons, it is probably best to interpret `vce(robust)` in the spirit of White’s original paper on estimation of heteroskedastic-consistent covariance matrices.

**Acknowledgment**

We thank Richard Dickens of the Department of Economics, University of Sussex, UK, for testing and assistance with an early version of this command.

Sigbert Jon Prais (1928–2014) was born in Frankfurt and moved to Britain in 1934 as a refugee. After earning degrees at the universities of Birmingham and Cambridge and serving in various posts in research and industry, he settled at the National Institute of Economic and Social Research. Prais’s interests extended widely across economics, including studies of the influence of education on economic progress.

Christopher Blake Winsten (1923–2005) was born in Welwyn Garden City, England; the son of the writer Stephen Winsten and the painter and sculptress Clare Blake. He was educated at the University of Cambridge and worked with the Cowles Commission at the University of Chicago and at the universities of Oxford, London (Imperial College) and Essex, making many contributions to economics and statistics, including the Prais–Winsten transformation and joint authorship of a celebrated monograph on transportation economics.

Donald Cochrane (1917–1983) was an Australian economist and econometrician. He was born in Melbourne and earned degrees at Melbourne and Cambridge. After wartime service in the Royal Australian Air Force, he held chairs at Melbourne and Monash, being active also in work for various international organizations and national committees.

Guy Henderson Orcutt (1917–2006) was born in Michigan and earned degrees in physics and economics at the University of Michigan. He worked at Harvard, the University of Wisconsin, and Yale. He contributed to econometrics and economics in several fields, most distinctively in developing microanalytical models of economic behavior.

**References**

Becketti, S. 2013. *Introduction to Time Series Using Stata*. College Station, TX: Stata Press.


Also see

- [TS] prais postestimation — Postestimation tools for prais
- [TS] arima — ARIMA, ARMAX, and other dynamic regression models
- [TS] mswitch — Markov-switching regression models
- [TS] tsset — Declare data to be time-series data
- [R] regress — Linear regression
- [R] regress postestimation time series — Postestimation tools for regress with time series
- [U] 20 Estimation and postestimation commands
# Postestimation commands

The following standard postestimation commands are available after `prais`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contrast</code></td>
<td>contrasts and ANOVA-style joint tests of estimates</td>
</tr>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>linktest</code></td>
<td>link test for model specification</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>pwcompare</code></td>
<td>pairwise comparisons of estimates</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] newvar [if] [in] [, statistic]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>standard error of the linear prediction</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Options for predict

`xb`, the default, calculates the fitted values—the prediction of $x_j b$ for the specified equation. This is the linear predictor from the fitted regression model; it does not apply the estimate of $\rho$ to prior residuals.

`stdp` calculates the standard error of the prediction for the specified equation, that is, the standard error of the predicted expected value or mean for the observation’s covariate pattern. The standard error of the prediction is also referred to as the standard error of the fitted value. As computed for `prais`, this is strictly the standard error from the variance in the estimates of the parameters of the linear model and assumes that $\rho$ is estimated without error.

`residuals` calculates the residuals from the linear prediction.
margins

Description for margins

margins estimates margins of response for linear predictions.

Menu for margins

Statistics  >  Postestimation

Syntax for margins

margins [marginlist] [ , options ]
margins [marginlist] , predict(statistic ...) [options ]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

Statistics not allowed with margins are functions of stochastic quantities other than \( e(b) \).

For the full syntax, see [R] margins.

Also see

[TS] prais — Prais–Winsten and Cochrane–Orcutt regression

[U] 20 Estimation and postestimation commands
Title

psdensity — Parametric spectral density estimation after arima, arfima, and ucm

Description

psdensity estimates the spectral density of a stationary process using the parameters of a previously estimated parametric model.

psdensity works after arfima, arima, and ucm; see [TS] arfima, [TS] arima, and [TS] ucm.

Quick start

Obtain spectral density values spden and corresponding frequencies sfreq

```
psdensity spden sfreq
```

Obtain power spectrum values pspec and corresponding frequencies pfreq

```
psdensity pspec pfreq, pspectrum
```

As above, but limit the frequency range to between 0 and 1

```
psdensity pspec pfreq, pspectrum range(0 1)
```

After ucm, obtain spectral density values for the second stochastic cycle cpden and corresponding frequencies cfreq

```
psdensity cpden cfreq, cycle(2)
```

Menu

Statistics > Time series > Postestimation > Parametric spectral density
Syntax

\texttt{psdensity [type] newvar.sd newvar.f [if] [in] [, options]}

where \texttt{newvar.sd} is the name of the new variable that will contain the estimated spectral density and \texttt{newvar.f} is the name of the new variable that will contain the frequencies at which the spectral density estimate is computed.

\begin{center}
\begin{tabular}{ll}
\textbf{options} & \textbf{Description} \\
\hline
\texttt{pspectrum} & estimate the power spectrum rather than the spectral density \\
\texttt{range(a b)} & limit the frequency range to \([a, b]\) \\
\texttt{cycle(\#)} & estimate the spectral density from the specified stochastic cycle; only allowed after \texttt{ucm} \\
\texttt{smemory} & estimate the spectral density of the short-memory component of the ARFIMA process; only allowed after \texttt{arfima} \\
\end{tabular}
\end{center}

Options

\texttt{pspectrum} causes \texttt{psdensity} to estimate the power spectrum rather than the spectral density. The power spectrum is equal to the spectral density times the variance of the process.

\texttt{range(a b)} limits the frequency range. By default, the spectral density is computed over \([0, \pi]\). Specifying \texttt{range(a b)} causes the spectral density to be computed over \([a, b]\). We require that \(0 \leq a < b < \pi\).

\texttt{cycle(\#)} causes \texttt{psdensity} to estimate the spectral density from the specified stochastic cycle after \texttt{ucm}. By default, the spectral density from the first stochastic cycle is estimated. \texttt{cycle(\#)} must specify an integer that corresponds to a cycle in the model fit by \texttt{ucm}.

\texttt{smemory} causes \texttt{psdensity} to ignore the ARFIMA fractional integration parameter. The spectral density computed is for the short-memory ARMA component of the model.

Remarks and examples

Remarks are presented under the following headings:

- \textit{The frequency-domain approach to time series}
- \textit{Some ARMA examples}

The frequency-domain approach to time series

A stationary process can be decomposed into random components that occur at the frequencies \(\omega \in [0, \pi]\). The spectral density of a stationary process describes the relative importance of these random components. \texttt{psdensity} uses the estimated parameters of a parametric model to estimate the spectral density of a stationary process.

We need some concepts from the frequency-domain approach to time-series analysis to interpret estimated spectral densities. Here we provide a simple, intuitive explanation. More technical presentations can be found in Priestley (1981), Harvey (1989, 1993), Hamilton (1994), Fuller (1996), and Wei (2006).
In the time domain, the dependent variable evolves over time because of random shocks. The autocovariances \( \gamma_j, j \in \{0, 1, \ldots, \infty\} \), of a covariance-stationary process \( y_t \) specify its variance and dependence structure, and the autocorrelations \( \rho_j, j \in \{1, 2, \ldots, \infty\} \), provide a scale-free measure of its dependence structure. The autocorrelation at lag \( j \) specifies whether realizations at time \( t \) and realizations at time \( t - j \) are positively related, unrelated, or negatively related.

In the frequency domain, the dependent variable is generated by an infinite number of random components that occur at the frequencies \( \omega \in [0, \pi] \). The spectral density specifies the relative importance of these random components. The area under the spectral density in the interval \((\omega, \omega + d\omega)\) is the fraction of the variance of the process that can be attributed to the random components that occur at the frequencies in the interval \((\omega, \omega + d\omega)\).

The spectral density and the autocorrelations provide the same information about the dependence structure, albeit in different domains. The spectral density can be written as a weighted average of the autocorrelations of \( y_t \), and it can be inverted to retrieve the autocorrelations as a function of the spectral density.

Like autocorrelations, the spectral density is normalized by \( \gamma_0 \), the variance of \( y_t \). Multiplying the spectral density by \( \gamma_0 \) yields the power spectrum of \( y_t \), which changes with the units of \( y_t \).

A peak in the spectral density around frequency \( \omega \) implies that the random components around \( \omega \) make an important contribution to the variance of \( y_t \).

A random variable primarily generated by low-frequency components will tend to have more runs above or below its mean than an independent and identically distributed (i.i.d.) random variable, and its plot may look smoother than the plot of the i.i.d. variable. A random variable primarily generated by high-frequency components will tend to have fewer runs above or below its mean than an i.i.d. random variable, and its plot may look more jagged than the plot of the i.i.d. variable.

**Technical note**

A more formal specification of the spectral density allows us to be more specific about how the spectral density specifies the relative importance of the random components.

If \( y_t \) is a covariance-stationary process with absolutely summable autocovariances, its spectrum is given by

\[
g_y(\omega) = \frac{1}{2\pi} \gamma_0 + \frac{1}{\pi} \sum_{k=1}^{\infty} \gamma_k \cos(\omega k) \tag{1}
\]

where \( g_y(\omega) \) is the spectrum of \( y_t \) at frequency \( \omega \) and \( \gamma_k \) is the \( k \)th autocovariance of \( y_t \). Taking the inverse Fourier transform of each side of (1) yields

\[
\gamma_k = \int_{-\pi}^{\pi} g_y(\omega) e^{i\omega k} d\omega \tag{2}
\]

where \( i \) is the imaginary number \( i = \sqrt{-1} \).

Evaluating (2) at \( k = 0 \) yields

\[
\gamma_0 = \int_{-\pi}^{\pi} g_y(\omega) d\omega
\]
which means that the variance of $y_t$ can be decomposed in terms of the spectrum $g_y(\omega)$. In particular, $g_y(\omega)d\omega$ is the contribution to the variance of $y_t$ attributable to the random components in the interval $(\omega, \omega + d\omega)$.

The spectrum depends on the units in which $y_t$ is measured, because it depends on the $\gamma_0$. Dividing both sides of (1) by $\gamma_0$ gives us the scale-free spectral density of $y_t$:

$$f_y(\omega) = \frac{1}{2\pi} + \frac{1}{\pi} \sum_{k=1}^{\infty} \rho_k \cos(\omega k)$$

By construction,

$$\int_{-\pi}^{\pi} f_y(\omega) d\omega = 1$$

so $f_y(\omega)d\omega$ is the fraction of the variance of $y_t$ attributable to the random components in the interval $(\omega, \omega + d\omega)$.

### Some ARMA examples

In this section, we estimate and interpret the spectral densities implied by the estimated ARMA parameters. The examples illustrate some of the essential relationships between covariance-stationary processes, the parameters of ARMA models, and the spectral densities implied by the ARMA-model parameters.

See [TS] ucm for a discussion of unobserved-components models and the stochastic-cycle model derived by Harvey (1989) for stationary processes. The stochastic-cycle model has a different parameterization of the spectral density, and it tends to produce spectral densities that look more like probability densities than ARMA models. See Remarks and examples in [TS] ucm for an introduction to these models, some examples, and some comparisons between the stochastic-cycle model and ARMA models.
Example 1

Let’s consider the changes in the number of manufacturing employees in the United States, which we plot below.

```
. use https://www.stata-press.com/data/r16/manemp2
   (FRED data: Number of manufacturing employees in U.S.)
. tsline D.manemp, yline(-0.206)
```

![Graph showing changes in number of manufacturing employees](image)

We added a horizontal line at the sample mean of \(-0.0206\) to highlight that there appear to be more runs above or below the mean than we would expect in data generated by an i.i.d. process.

As a first pass at modeling this dependence, we use `arima` to estimate the parameters of a first-order autoregressive (AR(1)) model. Formally, the AR(1) model is given by

\[
y_t = \alpha y_{t-1} + \epsilon_t
\]

where \(y_t\) is the dependent variable, \(\alpha\) is the autoregressive coefficient, and \(\epsilon_t\) is an i.i.d. error term. See [TS] `arima` for an introduction to ARMA modeling and the `arima` command.
. arima D.manemp, ar(1) noconstant

(Chaning optimization to BHHH)

Iteration 0: log likelihood = -870.64844
Iteration 1: log likelihood = -870.64794
Iteration 2: log likelihood = -870.64789
Iteration 3: log likelihood = -870.64787
Iteration 4: log likelihood = -870.64786

(Switching optimization to BFGS)
Iteration 5: log likelihood = -870.64786
Iteration 6: log likelihood = -870.64786

ARIMA regression
Sample: 1950m2 - 2011m2               Number of obs = 733
Wald chi2(1) = 730.51
Log likelihood = -870.6479        Prob > chi2 = 0.0000

|             | Coef.  | Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|-------------|--------|-----------|-------|-------|----------------------|
| D.manemp    |        |           |       |       |                      |
| ARMA        |        |           |       |       |                      |
| ar          | .5179561 | .0191638  | 27.03 | 0.000 | .4803959 .5555164   |
| L1.         |        |           |       |       |                      |
| /sigma      | .7934554 | .0080636  | 98.40 | 0.000 | .777651 .8092598    |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

The statistically significant estimate of 0.518 for the autoregressive coefficient indicates that there is an important amount of positive autocorrelation in this series.

The spectral density of a covariance-stationary process is symmetric around 0. Following convention, psdensity estimates the spectral density over the interval [0, π) at the points given in Methods and formulas.

Now we use psdensity to estimate the spectral density of the process implied by the estimated ARMA parameters. We specify the names of two new variables in the call to psdensity. The first new variable will contain the estimated spectral density. The second new variable will contain the frequencies at which the spectral density is estimated.
The above graph is typical of a spectral density of an AR(1) process with a positive coefficient. The curve is highest at frequency 0, and it tapers off toward zero or a positive asymptote. The estimated spectral density is telling us that the low-frequency random components are the most important random components of an AR(1) process with a positive autoregressive coefficient.

The closer the $\alpha$ is to 1, the more important are the low-frequency components relative to the high-frequency components. To illustrate this point, we plot the spectral densities implied by AR(1) models with $\alpha = 0.1$ and $\alpha = 0.9$. 

```
  . psdensity psden1 omega
  . line psden1 omega
```
As $\alpha$ gets closer to 1, the plot of the spectral density gets closer to being a spike at frequency 0, implying that only the lowest-frequency components are important.

> Example 2

Now let’s consider a dataset for which the estimated coefficient from an AR(1) model is negative. Below we plot the changes in initial claims for unemployment insurance in the United States.

```
. use https://www.stata-press.com/data/r16/icsa1, clear
. tsline D.icsa, yline(0.08)
```

The plot looks a little more jagged than we would expect from an i.i.d. process, but it is hard to tell. Below we estimate the AR(1) coefficient.

```
. arima D.icsa, ar(1) noconstant
(setting optimization to BHHH)
Iteration 0: log likelihood = -9934.0659
Iteration 1: log likelihood = -9934.0657
Iteration 2: log likelihood = -9934.0657
ARIMA regression
Sample: 14jan1967 - 19feb2011 Number of obs = 2302
Wald chi2(1) = 666.06
Log likelihood = -9934.066 Prob > chi2 = 0.0000

             D.icsa               OPG
Coef. Std. Err.     z     P>|z|   [95% Conf. Interval]
---------  -------------  --------  -------  --------  -------------  -------------
ARMA       
       ar    -.2756024  .0106789  -25.81  0.000   -.2965326   -.2546722
       L1.     .1810988  .1176556   153.92 0.000    17.87928    18.34048
/sigma
```

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

The estimated coefficient is negative and statistically significant.
The spectral density implied by the estimated parameters is

```
psdensity psden2 omega2
line psden2 omega2
```

The above graph is typical of a spectral density of an AR(1) process with a negative coefficient. The curve is lowest at frequency 0, and it monotonically increases to its highest point, which occurs when the frequency is $\pi$.

When the coefficient of an AR(1) model is negative, the high-frequency random components are the most important random components of the process. The closer the $\alpha$ is to $-1$, the more important are the high-frequency components relative to the low-frequency components. To illustrate this point, we plot the spectral densities implied by AR(1) models with $\alpha = -0.1$, and $\alpha = -0.9$.
As $\alpha$ gets closer to $-1$, the plot of the spectral density shifts toward becoming a spike at frequency $\pi$, implying that only the highest-frequency components are important.

For examples of \texttt{psdensity} after \texttt{arfima} and \texttt{ucm}, see \texttt{[TS] arfima} and \texttt{[TS] ucm}.

### Methods and formulas

Methods and formulas are presented under the following headings:

- Introduction
- Spectral density after arima or arfima
- Spectral density after ucm

**Introduction**

The spectral density $f(\omega)$ is estimated at the values $\omega \in \{\omega_1, \omega_2, \ldots, \omega_N\}$ using one of the formulas given below. Given a sample of size $N$, after accounting for any if or in restrictions, the $N$ values of $\omega$ are given by $\omega_i = \pi(i-1)/(N-1)$ for $i \in \{1, 2, \ldots, N\}$.

In the rare case in which the dataset in memory has insufficient observations for the desired resolution of the estimated spectral density, you may use \texttt{tsappend} or \texttt{set obs} (see \texttt{[TS] tsappend} or \texttt{[D] obs}) to increase the number of observations in the current dataset.

You may use an if restriction or an in restriction to restrict the observations to handle panel data or to compute the estimates for a subset of the observations.

**Spectral density after arima or arfima**

Let $\phi_k$ and $\theta_k$ denote the $p$ autoregressive and $q$ moving-average parameters of an ARMA model, respectively. Box, Jenkins, and Reinsel (2008) show that the spectral density implied by the ARMA parameters is

$$f_{\text{ARMA}}(\omega; \phi, \theta, \sigma^2_\epsilon, \gamma_0) = \frac{\sigma^2_\epsilon}{2\pi \gamma_0} \left| 1 + \theta_1 e^{-i\omega} + \theta_2 e^{-i2\omega} + \ldots + \theta_q e^{-iq\omega} \right|^2 \left| 1 - \phi_1 e^{-i\omega} - \phi_2 e^{-i2\omega} - \ldots - \phi_p e^{-ip\omega} \right|^2$$

where $\omega \in [0, \pi]$ and $\sigma^2_\epsilon$ is the variance of the idiosyncratic error and $\gamma_0$ is the variance of the dependent variable. We estimate $\gamma_0$ using the \texttt{arima} parameter estimates.

The spectral density for the ARFIMA model is

$$f_{\text{ARFIMA}}(\omega; \phi, \theta, d, \sigma^2_\epsilon, \gamma_0) = |1 - e^{i\omega}|^{-2d} f_{\text{ARMA}}(\omega; \phi, \theta, \sigma^2_\epsilon)$$

where $d, -1/2 < d < 1/2$, is the fractional integration parameter. The spectral density goes to infinity as the frequency approaches 0 for $0 < d < 1/2$, and it is zero at frequency 0 for $-1/2 < d < 0$.

The \texttt{smemory} option causes \texttt{psdensity} to perform the estimation with $d = 0$, which is equivalent to estimating the spectral density of the fractionally differenced series.

The power spectrum omits scaling by $\gamma_0$. 

Spectral density after ucm

The spectral density of an order-$k$ stochastic cycle with frequency $\lambda$ and damping $\rho$ is (Trimbur 2006)

\[
f(\omega; \rho, \lambda, \sigma^2_\kappa) = \left\{ \frac{(1 - \rho^2)^2k-1}{\sigma^2_\kappa \sum_{i=0}^{k-1} \binom{k-1}{i}^2 \rho^{2i}} \right\} \times \frac{\sum_{j=0}^{k} \sum_{i=0}^{k} (-1)^{j+i} \binom{k}{j} \binom{k}{i} \rho^{j+i} \cos \lambda(j - i) \cos \omega(j - i)}{2\pi \{1 + 4\rho^2 \cos^2 \lambda + \rho^4 - 4\rho(1 + \rho^2) \cos \lambda \cos \omega + 2\rho^2 \cos 2\omega \}^k}
\]

where $\sigma^2_\kappa$ is the variance of the cycle error term.

The variance of the cycle is

\[
\sigma^2_\omega = \frac{\sigma^2_\kappa \sum_{i=0}^{k-1} \binom{k-1}{i}^2 \rho^{2i}}{(1 - \rho^2)^2k-1}
\]

and the power spectrum omits scaling by $\sigma^2_\omega$.

References


Also see

[TS] arfima — Autoregressive fractionally integrated moving-average models

[TS] arima — ARIMA, ARMAX, and other dynamic regression models

[TS] ucm — Unobserved-components model
**Description**

*rolling* executes a command on each of a series of windows of observations and stores the results. *rolling* can perform what are commonly called rolling regressions, recursive regressions, and reverse recursive regressions. However, *rolling* is not limited to just linear regression analysis: any command that stores results in `e()` or `r()` can be used with *rolling*.

**Quick start**

Fit an AR(1) model for *y* with a 20-period rolling window using `tsset` data

```
rolling, window(20): arima y, ar(1)
```

Recursive rolling window estimation with a fixed starting period

```
rolling, window(20) recursive: arima y, ar(1)
```

As above, but specify that estimation start in 1990 and end in 2011

```
rolling, window(20) recursive start(1990) end(2011): arima y, ar(1)
```

Reverse recursive rolling window estimation with the last period fixed

```
rolling, window(20) rrecursive start(1990) end(2011): arima y, ar(1)
```

Save results from a 20-period rolling window estimation to new dataset *mydata.dta*

```
rolling, window(20) saving(mydata): arima y, ar(1)
```

Note: Any command that accepts the *rolling* prefix may be substituted for *arima* above.

**Menu**

Statistics > Time series > Rolling-window and recursive estimation
### Syntax

```
rolling [exp_list] [if] [in] window(#) [options]: command
```

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>window(#)</code></td>
<td>number of consecutive data points in each sample</td>
</tr>
<tr>
<td><code>recursive</code></td>
<td>use recursive samples</td>
</tr>
<tr>
<td><code>recursive</code></td>
<td>use reverse recursive samples</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>replace data in memory with results</td>
</tr>
<tr>
<td><code>saving(filename, ...)</code></td>
<td>save results to <code>filename</code>; save statistics in double precision;</td>
</tr>
<tr>
<td></td>
<td>save results to <code>filename</code> every # replications</td>
</tr>
<tr>
<td><code>stepsizes(#)</code></td>
<td>number of periods to advance window</td>
</tr>
<tr>
<td><code>start(time_constant)</code></td>
<td>period at which rolling is to start</td>
</tr>
<tr>
<td><code>end(time_constant)</code></td>
<td>period at which rolling is to end</td>
</tr>
<tr>
<td><code>keep(varname[, start])</code></td>
<td>save <code>varname</code> with results; optionally, use value at left edge of window</td>
</tr>
<tr>
<td><code>nodots</code></td>
<td>suppress replication dots</td>
</tr>
<tr>
<td><code>dots(#)</code></td>
<td>display dots every # replications</td>
</tr>
<tr>
<td><code>noisily</code></td>
<td>display any output from <code>command</code></td>
</tr>
<tr>
<td><code>trace</code></td>
<td>trace <code>command</code>’s execution</td>
</tr>
<tr>
<td><code>reject(exp)</code></td>
<td>identify invalid results</td>
</tr>
</tbody>
</table>

* `window(#)` is required.

You must `tsset` your data before using `rolling`; see [TS] `tsset`.

`command` is any command that follows standard Stata syntax and allows the `if` qualifier. The by prefix cannot be part of `command`.

`aweights` are allowed in `command` if `command` accepts `aweights`; see [U] 11.1.6 `weight`.

`exp_list` contains \( (name: elist) \)

- `elist`
- `eexp`

`elist` contains \( newvar = (exp) \)

- `(exp)`

`eexp` is \( specname\)

- `[eqno]specname`

`specname` is

- `_b`
- `_b[]`
- `_se`
- `_se[]`

`eqno` is

- `##`
- `name`

`exp` is a standard Stata expression; see [U] 13 Functions and expressions.

Distinguish between [], which are to be typed, and [], which indicate optional arguments.
**Options**

`window(#)` defines the window size used each time `command` is executed. The window size refers to calendar periods, not the number of observations. If there are missing data (for example, because of weekends), the actual number of observations used by `command` may be less than `window(#).` `window(#)` is required.

`recursive` specifies that a recursive analysis be done. The starting period is held fixed, the ending period advances, and the window size grows.

`rrecursive` specifies that a reverse recursive analysis be done. Here the ending period is held fixed, the starting period advances, and the window size shrinks.

`clear` specifies that Stata replace the data in memory with the collected statistics even though the current data in memory have not been saved to disk.

`saving(filename[, suboptions])` creates a Stata data file (.dta file) consisting of (for each statistic in `exp_list`) a variable containing the window replicates.

`double` specifies that the results for each replication be saved as doubles, meaning 8-byte reals. By default, they are saved as floats, meaning 4-byte reals.

`every(#)` specifies that results be written to disk every #th replication. `every()` should be specified in conjunction with `saving()` only when `command` takes a long time for each replication. This will allow recovery of partial results should your computer crash. See [P] postfile.

`start(time_constant)` specifies the date on which `rolling` is to start. `start()` may be specified as an integer or as a date literal.

`end(time_constant)` specifies the date on which `rolling` is to end. `end()` may be specified as an integer or as a date literal.

`keep(varname[, start])` specifies a variable to be posted along with the results. The value posted is the value that corresponds to the right edge of the window. Specifying the `start()` option requests that the value corresponding to the left edge of the window be posted instead. This option is often used to record calendar dates.

`nodots` and `dots(#)` specify whether to display replication dots. By default, one dot character is displayed for each window. A red ‘x’ is displayed if `command` returns an error or if any value in `exp_list` is missing. You can also control whether dots are printed using `set dots`; see [R] set.

`nodots` suppresses display of the replication dot for each window on which `command` is executed.

`dots(#)` displays dots every # replications. `dots(0)` is a synonym for `nodots`.

`noisily` causes the output of `command` to be displayed for each window on which `command` is executed. This option implies the `nodots` option.

`trace` causes a trace of the execution of `command` to be displayed. This option implies the `noisily` and `nodots` options.
reject\((exp)\) identifies an expression that indicates when results should be rejected. When \(exp\) is true, the saved statistics are set to missing values.

Remarks and examples

`rolling` is a moving sampler that collects statistics from `command` after executing `command` on subsets of the data in memory. Typing

```
. rolling `exp', window(50) clear: `command'
```

executes `command` on sample windows of span 50. That is, `rolling` will first execute `command` by using periods 1–50 of the dataset, and then using periods 2–51, 3–52, and so on.

`command` defines the statistical command to be executed. Most Stata commands and user-written programs can be used with `rolling`, as long as they follow standard Stata syntax and allow the `if` qualifier; see `[U] 11 Language syntax`. The by prefix cannot be part of `command`.

`exp_list` specifies the statistics to be collected from the execution of `command`. If no expressions are given, `exp_list` assumes a default of `b` if `command` stores results in `e()` and of all the scalars if `command` stores results in `r()` and not in `e()`. Otherwise, not specifying an expression in `exp_list` is an error.

Suppose that you have data collected at 100 consecutive points in time, numbered 1–100, and you wish to perform a rolling regression with a window size of 20 periods. Typing

```
. rolling _b, window(20) clear: regress depvar indepvar
```

causes Stata to `regress depvar on indepvar` using periods 1–20, store the regression coefficients (_b), run the regression using periods 2–21, and so on, finishing with a regression using periods 81–100 (the last 20 periods).

The `stepsize()` option specifies how far ahead the window is moved each time. For example, if you specify `step(2)`, then `command` is executed on periods 1–20, and then 3–22, 5–24, etc. By default, `rolling` replaces the dataset in memory with the computed statistics unless the `saving()` option is specified, in which case the computed statistics are saved in the filename specified. If the dataset in memory has been changed since it was last saved and you do not specify `saving()`, you must use `clear`.

`rolling` can also perform recursive and reverse recursive analyses. In a recursive analysis, the starting date is held fixed, and the window size grows as the ending date is advanced. In a reverse recursive analysis, the ending date is held fixed, and the window size shrinks as the starting date is advanced.

> Example 1

We have data on the daily returns to IBM stock (`ibm`), the S&P 500 (`spx`), and short-term interest rates (`irx`), and we want to create a series containing the beta of IBM by using the previous 200 trading days at each date. We will also record the standard errors, so that we can obtain 95% confidence intervals for the betas. See, for example, Stock and Watson (2019, 112) for more information on estimating betas. We type
. use https://www.stata-press.com/data/r16/ibm
(Source: Yahoo! Finance)
. tsset t
  time variable:  t, 1 to 494
delta:  1 unit
. generate ibmadj = ibm - irx
(1 missing value generated)
. generate spxadj = spx - irx
(1 missing value generated)
. rolling _b _se, window(200) saving(betas, replace) keep(date):
   > regress ibmadj spxadj
(running regress on estimation sample)
Rolling replications (295)
.................................................. 50
.................................................. 100
.................................................. 150
.................................................. 200
.................................................. 250
.............................................
file betas.dta saved

Our dataset has both a time variable \( t \) that runs consecutively and a date variable \( \text{date} \) that measures the calendar date and therefore has gaps at weekends and holidays. Had we used the \( \text{date} \) variable as our time variable, \rolling would have used windows consisting of 200 calendar days instead of 200 trading days, and each window would not have exactly 200 observations. We used the \( \text{keep(date)} \) option so that we could refer to the \( \text{date} \) variable when working with the results dataset.

We can list a portion of the dataset created by \rolling to see what it contains:

. use betas, clear
(rolling: regress)
. sort date
. list in 1/3, abbreviate(10) table

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>date</th>
<th>_b_spxadj</th>
<th>_b_cons</th>
<th>_se_spxadj</th>
<th>_se_cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1</td>
<td>200</td>
<td>1.043422</td>
<td>-.0181504</td>
<td>.0658531</td>
<td>.0748295</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>201</td>
<td>1.039024</td>
<td>-.0126876</td>
<td>.0656893</td>
<td>.074609</td>
</tr>
<tr>
<td>3.</td>
<td>3</td>
<td>202</td>
<td>1.038371</td>
<td>-.0235616</td>
<td>.0654591</td>
<td>.0743851</td>
</tr>
</tbody>
</table>

The variables \( \text{start} \) and \( \text{end} \) indicate the first and last observations used each time that \rolling called \( \text{regress} \), and the \( \text{date} \) variable contains the calendar date corresponding the period represented by \( \text{end} \). The remaining variables are the estimated coefficients and standard errors from the regression. In our example, \( _\text{b_spxadj} \) contains the estimated betas, and \( _\text{b_cons} \) contains the estimated alphas. The variables \( _\text{se_spxadj} \) and \( _\text{se_cons} \) have the corresponding standard errors.
Finally, we compute the confidence intervals for the betas and examine how they have changed over time:

```stata
. generate lower = _b_spxadj - 1.96*_se_spxadj
. generate upper = _b_spxadj + 1.96*_se_spxadj
. twoway (line _b_spxadj date) (rline lower upper date) if date>=td(1oct2003), ytitle("Beta")
```

As 2004 progressed, IBM’s stock returns were less influenced by returns in the broader market. Beginning in June of 2004, IBM’s beta became significantly different from unity at the 95% confidence level, as indicated by the fact that the confidence interval does not contain one from then onward.

In addition to rolling-window analyses, `rolling` can also perform recursive ones. Suppose again that you have data collected at 100 consecutive points in time, and now you type

```stata
. rolling _b, window(20) recursive clear: regress depvar indepvar
```

Stata will first `regress depvar` on `indepvar` by using observations 1–20, store the coefficients, run the regression using observations 1–21, observations 1–22, and so on, finishing with a regression using all 100 observations. Unlike a rolling regression, in which case the number of observations is held constant and the starting and ending points are shifted, a recursive regression holds the starting point fixed and increases the number of observations. Recursive analyses are often used in forecasting situations. As time goes by, more information becomes available that can be used in making forecasts. See Kmenta (1997, 423–424).

> **Example 2**

Using the same dataset, we type

```stata
. use https://www.stata-press.com/data/r16/ibm, clear
(Source: Yahoo! Finance)
. tsset t
time variable: t, 1 to 494
delta: 1 unit
. generate ibmadj = ibm - irx
(1 missing value generated)
```
. generate spxadj = spx - irx
   (1 missing value generated)
. rolling _b _se, recursive window(200) clear: regress ibmadj spxadj
   (output omitted)
. list in 1/3, abbrev(10)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>_b_spxadj</th>
<th>_b_cons</th>
<th>_se_spxadj</th>
<th>_se_cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.043422</td>
<td>-.0181504</td>
<td>.0658531</td>
<td>.0748295</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.039024</td>
<td>-.0126876</td>
<td>.0656893</td>
<td>.074609</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1.037687</td>
<td>-.016475</td>
<td>.0655896</td>
<td>.0743481</td>
</tr>
</tbody>
</table>

Here the starting period remains fixed and the window grows larger.

In a reverse recursive analysis, the ending date is held fixed, and the window size becomes smaller as the starting date is advanced. For example, with a dataset that has observations numbered 1–100,
typing

. rolling _b, window(20) reverse recursive clear: regress depvar indepvar

creates a dataset in which the first observation has the results based on periods 1–100, the second observation has the results based on 2–100, the third having 3–100, and so on, up to the last observation having results based on periods 81–100 (the last 20 observations).

Example 3

Using the data on stock returns, we want to build a model in which we predict today’s IBM stock return on the basis of yesterday’s returns on IBM and the S&P 500. That is, letting $i_t$ and $s_t$ denote the returns to IBM and the S&P 500 on date $t$, we want to fit the regression model

$$i_t = \beta_0 + \beta_1 i_{t-1} + \beta_2 s_{t-1} + \epsilon_t$$

where $\epsilon_t$ is a regression error term, and then compute

$$\hat{i}_{t+1} = \hat{\beta}_0 + \hat{\beta}_1 i_t + \hat{\beta}_2 s_t$$

We will use recursive regression because we suspect that the more data we have to fit the regression model, the better the model will predict returns. We will use at least 20 periods in fitting the regression.

. use https://www.stata-press.com/data/r16/ibm, clear
   (Source: Yahoo! Finance)
. tsset t
   time variable:  t, 1 to 494
   delta:  1 unit

One alternative would be to use rolling with the recursive option to fit the regressions, collect the coefficients, and then compute the predicted values afterward. However, we will instead write a short program that computes the forecasts automatically and then use rolling, recursive on that program. The program must accept an if expression so that rolling can indicate to the program which observations are to be used. Our program is
program myforecast, rclass
    syntax [if]
    regress ibm L.ibm L.spx `if'
    // Find last time period of estimation sample and
    // make forecast for period just after that
    summ t if e(sample)
    local last = `r(max)'
    local fcast = _b[_cons] + _b[L.ibm]*ibm[`last'] + ///
        _b[L.spx]*spx[`last']
    return scalar forecast = `fcast'
    // Next period’s actual return
    // Will return missing value for final period
    return scalar actual = ibm[`last'+1]
end

Now we call rolling:

. rolling actual=r(actual) forecast=r(forecast), recursive window(20): myforecast
   (output omitted)
. corr actual forecast
   (obs=474)

      actual       forecast
      -------------
    actual |    1.0000
    forecast |  -0.0957    1.0000

Our model does not work too well—the correlation between actual returns and our forecasts is negative.

Stored results

rolling sets no r- or e-class macros. The results from the command used with rolling, depending on the last window of data used, are available after rolling has finished.

Acknowledgment

We thank Christopher F. Baum of the Department of Economics at Boston College and author of the Stata Press books *An Introduction to Modern Econometrics Using Stata* and *An Introduction to Stata Programming* for an earlier rolling regression command.

References


Also see

[D] statsby — Collect statistics for a command across a by list
[R] Stored results — Stored results
**sspace** estimates the parameters of linear state-space models by maximum likelihood. Linear state-space models are very flexible and many linear time-series models can be written as linear state-space models.

**sspace** uses two forms of the Kalman filter to recursively obtain conditional means and variances of both the unobserved states and the measured dependent variables that are used to compute the likelihood.

The covariance-form syntax and the error-form syntax of **sspace** reflect the two different forms in which researchers specify state-space models. Choose the syntax that is easier for you; the two forms are isomorphic.

**Quick start**

AR(1) model for y with unobserved state u modeled as lag of itself in the state equation, and requiring the coefficient of u constrained to 1 in the observation equation

```
constraint 1 [y]u = 1
sspace (u L.u, state noconstant) (y u, noerror), constraints(1)
```

Dynamic-factor model of the first difference of y1, y2, and y3 as linear functions of an unobserved factor that follows a first-order autoregressive process

```
constraint 1 [y1]u = 1
sspace (u L.u, state noconstant) (d.y1 u) (d.y2 u) (d.y3 u), nolog
```
## Syntax

**Covariance-form syntax**

```markdown
sspace state_ceq [ state_ceq ... state_ceq ] obs_ceq [ obs_ceq ... obs_ceq ]
    [ if ] [ in ] [ , options ]
```

where each `state_ceq` is of the form

```markdown
(statevar [ lagged_statevars ] [ indepvars ], state [ noerror noconstant ])
```

and each `obs_ceq` is of the form

```markdown
(obs [ statevars ] [ indepvars ] [, noerror noconstant ])
```

**Error-form syntax**

```markdown
sspace state_efeq [ state_efeq ... state_efeq ] obs_efeq [ obs_efeq ... obs_efeq ]
    [ if ] [ in ] [ , options ]
```

where each `state_efeq` is of the form

```markdown
(statevar [ lagged_statevars ] [ indepvars ] [ state_errors ], state [ noconstant ])
```

and each `obs_efeq` is of the form

```markdown
(obs [ statevars ] [ indepvars ] [ obs_errors ] [, noconstant ])
```

- `statevar` is the name of an unobserved state, not a variable. If there happens to be a variable of the same name, the variable is ignored and plays no role in the estimation.
- `lagged_statevars` is a list of lagged `statevar`s. Only first lags are allowed.
- `state_errors` is a list of state-equation errors that enter a state equation. Each state error has the form `e.statevar`, where `statevar` is the name of a state in the model.
- `obs_errors` is a list of observation-equation errors that enter an equation for an observed variable. Each error has the form `e.depvar`, where `depvar` is an observed dependent variable in the model.

<table>
<thead>
<tr>
<th>equation-level options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>specifies that the equation is a state equation</td>
</tr>
<tr>
<td>noerror</td>
<td>specifies that there is no error term in the equation</td>
</tr>
<tr>
<td>noconstant</td>
<td>suppresses the constant term from the equation</td>
</tr>
</tbody>
</table>
options  Description

Model

covstate(covform)  specifies the covariance structure for the errors in the state variables

covobserved(covform)  specifies the covariance structure for the errors in the observed
dependent variables

constraints(constraints)  apply specified linear constraints

SE/Robust

vce(vcetype)  vcetype may be oim or robust

Reporting

level(#)  set confidence level; default is level(95)

noconsreport  do not display constraints

display_options  control columns and column formats, row spacing, display of
omitted variables and base and empty cells, and
factor-variable labeling

Maximization

maximize_options  control the maximization process; seldom used

Advanced

method(method)  specify the method for calculating the log likelihood; seldom used

coeflegend  display legend instead of statistics

covform  Description

identity  identity matrix; the default for error-form syntax

diagidential scalar matrix

diagonal  diagonal matrix; the default for covariance-form syntax

unstructured  symmetric, positive-definite matrix; not allowed with error-form
syntax

method  Description

hybrid  use the stationary Kalman filter and the De Jong diffuse Kalman
filter; the default

dejong  use the stationary De Jong Kalman filter and the De Jong diffuse
Kalman filter

kdiffuse  use the stationary Kalman filter and the nonstationary large-\(\kappa\) diffuse Kalman filter; seldom used

You must tsset your data before using sspace; see [TS] tsset.

indepvars may contain factor variables; see [U] 11.4.3 Factor variables.

indepvars and depvar may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.

coefflegend does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
Options

**Equation-level options**

- **state** specifies that the equation is a state equation.
- **noerror** specifies that there is no error term in the equation. **noerror** may not be specified in the error-form syntax.
- **noconstant** suppresses the constant term from the equation.

**Options**

- **covstate(covform)** specifies the covariance structure for the state errors.
  - **covstate(identity)** specifies a covariance matrix equal to an identity matrix, and it is the default for the error-form syntax.
  - **covstate(dscalar)** specifies a covariance matrix equal to $\sigma_{state}^2$ times an identity matrix.
  - **covstate(diagonal)** specifies a diagonal covariance matrix, and it is the default for the covariance-form syntax.
  - **covstate(unstructured)** specifies a symmetric, positive-definite covariance matrix with parameters for all variances and covariances. **covstate(unstructured)** may not be specified with the error-form syntax.

- **covobserved(covform)** specifies the covariance structure for the observation errors.
  - **covobserved(identity)** specifies a covariance matrix equal to an identity matrix, and it is the default for the error-form syntax.
  - **covobserved(dscalar)** specifies a covariance matrix equal to $\sigma_{observed}^2$ times an identity matrix.
  - **covobserved(diagonal)** specifies a diagonal covariance matrix, and it is the default for the covariance-form syntax.
  - **covobserved(unstructured)** specifies a symmetric, positive-definite covariance matrix with parameters for all variances and covariances. **covobserved(unstructured)** may not be specified with the error-form syntax.

- **constraints(constraints)**; see [R] Estimation options.

**SE/Robust**

- **vce(vcetype)** specifies the estimator for the variance–covariance matrix of the estimator.
  - **vce(oim)**, the default, causes sspace to use the observed information matrix estimator.
  - **vce(robust)** causes sspace to use the Huber/White/sandwich estimator.

**Reporting**

- **level(#)**, **nocnsreport**; see [R] Estimation options.

- **display_options**: **noci**, **nopvalues**, **noomitted**, **vsquish**, **noemptycells**, **baselevels**, **allbaselevels**, **nofvlabel**, **fvwrap(#)**, **fvwрап(style)**, **cformat(%.fmt)**, **pformat(%.fmt)**, and **sformat(%.fmt)**; see [R] Estimation options.
Maximization options: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), and from(matname)`; see `[R] Maximize` for all options except `from()`, and see below for information on `from()`. These options are seldom used.

`from(matname)` specifies initial values for the maximization process. `from(b0)` causes `sspace` to begin the maximization algorithm with the values in `b0`. `b0` must be a row vector; the number of columns must equal the number of parameters in the model; and the values in `b0` must be in the same order as the parameters in `e(b)`.

**Advanced**

`method(method)` specifies how to compute the log likelihood. This option is seldom used.

- `method(hybrid)`, the default, uses the Kalman filter with model-based initial values for the states when the model is stationary and uses the De Jong (1988, 1991) diffuse Kalman filter when the model is nonstationary.
- `method(dejong)` uses the Kalman filter with the De Jong (1988) method for estimating the initial values for the states when the model is stationary and uses the De Jong (1988, 1991) diffuse Kalman filter when the model is nonstationary.
- `method(kdiffuse)` is a seldom used method that uses the Kalman filter with model-based initial values for the states when the model is stationary and uses the large-κ diffuse Kalman filter when the model is nonstationary.

The following option is available with `sspace` but is not shown in the dialog box:

`coeflegend`; see `[R] Estimation options`.

### Remarks and examples

Remarks are presented under the following headings:

- An introduction to state-space models
- Some stationary state-space models
- Some nonstationary state-space models

### An introduction to state-space models

Many linear time-series models can be written as linear state-space models, including vector autoregressive moving-average (VARMA) models, dynamic-factor (DF) models, and structural time-series (STS) models. The solutions to some stochastic dynamic-programming problems can also be written in the form of linear state-space models. We can estimate the parameters of a linear state-space model by maximum likelihood (ML). The Kalman filter or a diffuse Kalman filter is used to write the likelihood function in prediction-error form, assuming normally distributed errors. The quasi–maximum likelihood (QML) estimator, which drops the normality assumption, is consistent and asymptotically normal when the model is stationary. Chang, Miller, and Park (2009) establish consistency and asymptotic normality of the QML estimator for a class of nonstationary state-space models. The QML estimator differs from the ML estimator only in the `VCE`; specify the `vce(robust)` option to obtain the QML estimator.
Hamilton (1994a, 1994b), Harvey (1989), and Brockwell and Davis (1991) provide good introductions to state-space models. Anderson and Moore’s (1979) text is a classic reference; they produced many results used subsequently. Caines (1988) and Hannan and Deistler (1988) provide excellent, more advanced, treatments.

`sspace` estimates linear state-space models with time-invariant coefficient matrices, which cover the models listed above and many others. `sspace` can estimate parameters from state-space models of the form

\[
\begin{align*}
  z_t &= A z_{t-1} + B x_t + C \epsilon_t \\
  y_t &= D z_t + F w_t + G \nu_t
\end{align*}
\]

where

- \( z_t \) is an \( m \times 1 \) vector of unobserved state variables;
- \( x_t \) is a \( k_x \times 1 \) vector of exogenous variables;
- \( \epsilon_t \) is a \( q \times 1 \) vector of state-error terms, \( q \leq m \);
- \( y_t \) is an \( n \times 1 \) vector of observed endogenous variables;
- \( w_t \) is a \( k_w \times 1 \) vector of exogenous variables;
- \( \nu_t \) is an \( r \times 1 \) vector of observation-error terms, \( r \leq n \);
- \( A, B, C, D, F, \) and \( G \) are parameter matrices.

The equations for \( z_t \) are known as the state equations, and the equations for \( y_t \) are known as the observation equations.

The error terms are assumed to be zero mean, normally distributed, serially uncorrelated, and uncorrelated with each other;

\[
\begin{align*}
  \epsilon_t &\sim N(0, Q) \\
  \nu_t &\sim N(0, R) \\
  E[\epsilon_t \epsilon_s'] &= 0 \text{ for all } s \neq t \\
  E[\epsilon_t \nu_s'] &= 0 \text{ for all } s \text{ and } t
\end{align*}
\]

The state-space form is used to derive the log likelihood of the observed endogenous variables conditional on their own past and any exogenous variables. When the model is stationary, a method for recursively predicting the current values of the states and the endogenous variables, known as the Kalman filter, is used to obtain the prediction error form of the log-likelihood function. When the model is nonstationary, a diffuse Kalman filter is used. How the Kalman filter and the diffuse Kalman filter initialize their recursive computations depends on the `method()` option; see `Methods and formulas`.

The linear state-space models with time-invariant coefficient matrices defined above can be specified in the covariance-form syntax and the error-form syntax. The covariance-form syntax requires that \( C \) and \( G \) be selection matrices, but places no restrictions on \( Q \) or \( R \). In contrast, the error-form syntax places no restrictions \( C \) or \( G \), but requires that \( Q \) and \( R \) be either diagonal, diagonal-scalar, or identity matrices. Some models are more easily specified in the covariance-form syntax, while others are more easily specified in the error-form syntax. Choose the syntax that is easiest for your application.
Some stationary state-space models

Example 1: An AR(1) model

Following Hamilton (1994b, 373–374), we can write the first-order autoregressive (AR(1)) model

\[ y_t - \mu = \alpha(y_{t-1} - \mu) + \epsilon_t \]

as a state-space model with the observation equation

\[ y_t = \mu + u_t \]

and the state equation

\[ u_t = \alpha u_{t-1} + \epsilon_t \]

where the unobserved state is \( u_t = y_t - \mu \).

Here we fit this model to data on the capacity utilization rate. The variable \( \ln\text{caputil} \) contains data on the natural log of the capacity utilization rate for the manufacturing sector of the U.S. economy. We treat the series as first-difference stationary and fit its first difference to an AR(1) process. Here we estimate the parameters of the above state-space form of the AR(1) model:

```
use https://www.stata-press.com/data/r16/manufac
(St. Louis Fed (FRED) manufacturing data)
constraint 1 [D.lncaputil]u = 1
.sspace (u L.u, state noconstant) (D.lncaputil u, noerror), constraints(1)
searching for initial values ............
(setting technique to bhhh)
Iteration 0: log likelihood = 1515.8693
Iteration 1: log likelihood = 1516.4187
(output omitted)
Refining estimates:
Iteration 0: log likelihood = 1516.44
Iteration 1: log likelihood = 1516.44

State-space model
Sample: 1972m2 - 2008m12   Number of obs = 443
Wald chi2(1) = 61.73
Log likelihood = 1516.44   Prob > chi2 = 0.0000
( 1) [D.lncaputil]u = 1


<table>
<thead>
<tr>
<th></th>
<th>OIM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coef.</td>
</tr>
<tr>
<td>ln\text{caputil}</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td>0.3524</td>
</tr>
<tr>
<td>D.lncaputil</td>
<td></td>
</tr>
<tr>
<td>_cons</td>
<td>1   (constrained)</td>
</tr>
<tr>
<td></td>
<td>-.0003</td>
</tr>
<tr>
<td>/state</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0000</td>
</tr>
</tbody>
</table>
```

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
The iteration log has three parts: the dots from the search for initial values, the log from finding the maximum, and the log from a refining step. Here is a description of the logic behind each part:

1. The quality of the initial values affects the speed and robustness of the optimization algorithm. `sspace` takes a few iterations in a nonlinear least-squares (NLS) algorithm to find good initial values and reports a dot for each (NLS) iteration.

2. This iteration log is the standard method by which Stata reports the search for the maximum likelihood estimates of the parameters in a nonlinear model.

3. Some of the parameters are transformed in the maximization process that `sspace` reports in part 2. After a maximum candidate is found in part 2, `sspace` looks for a maximum in the unconstrained space, checks that the Hessian of the log-likelihood function is of full rank, and reports these iterations as the refining step.

The header in the output describes the estimation sample, reports the log-likelihood function at the maximum, and gives the results of a Wald test against the null hypothesis that the coefficients on all the independent variables, state variables, and lagged state variables are zero. In this example, the null hypothesis that the coefficient on \( L1.u \) is zero is rejected at all conventional levels.

The estimation table reports results for the state equations, the observation equations, and the variance–covariance parameters. The estimated autoregressive coefficient of 0.3524 indicates that there is persistence in the first differences of the log of the manufacturing rate. The estimated mean of the differenced series is \(-0.0004\), which is smaller in magnitude than its standard error, indicating that there is no deterministic linear trend in the series.

Typing

```
. arima D.lncaputil, ar(1) technique(nr)
(output omitted)
```

produces nearly identical parameter estimates and standard errors for the mean and the autoregressive parameter. Because `sspace` estimates the variance of the state error while `arima` estimates the standard deviation, calculations are required to obtain the same results. The different parameterization of the variance parameter can cause small numerical differences.

---

### Technical note

In some situations, the second part of the iteration log terminates but the refining step never converges. Only when the refining step converges does the maximization algorithm find interpretable estimates. If the refining step iterates without convergence, the parameters of the specified model are not identified by the data. (See Rothenberg [1971], Drukker and Wiggins [2004], and Davidson and MacKinnon [1993, sec. 5.2] for discussions of identification.)

---

### Example 2: An ARMA(1,1) model

Following Harvey (1993, 95–96), we can write a zero-mean, first-order, autoregressive moving-average (ARMA(1,1)) model

\[
y_t = \alpha y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t
\]  

(1)
as a state-space model with state equations

\[
\begin{pmatrix}
y_t \\
\theta \epsilon_t
\end{pmatrix} = \begin{pmatrix} \alpha & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_{t-1} \\
\theta \epsilon_{t-1}
\end{pmatrix} + \begin{pmatrix} 1 \\
\theta \end{pmatrix} \epsilon_t
\]

(2)

and observation equation

\[
y_t = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} y_t \\
\theta \epsilon_t
\end{pmatrix}
\]

(3)

The unobserved states in this model are \( u_{1t} = y_t \) and \( u_{2t} = \theta \epsilon_t \). We set the process mean to zero because economic theory and the previous example suggest that we should do so. Below we estimate the parameters in the state-space model by using the error-form syntax:

. constraint 2 [u1]L.u2 = 1
. constraint 3 [u1]e.u1 = 1
. constraint 4 [D.lncaputil]u1 = 1
. sspace (u1 L.u1 L.u2 e.u1, state noconstant) (u2 e.u1, state noconstant) > (D.lncaputil u1, noconstant), constraints(2/4) covstate(diagonal)
searching for initial values

(setting technique to bhhh)
Iteration 0: log likelihood = 1478.5361
Iteration 1: log likelihood = 1490.5202
(output omitted)
Refining estimates:
Iteration 0: log likelihood = 1531.255
Iteration 1: log likelihood = 1531.255

State-space model
Sample: 1972m2 - 2008m12

| OIM             | Coef.       | Std. Err.     | z    | P>|z|    | [95% Conf. Interval] |
|-----------------|-------------|---------------|------|---------|---------------------|
| lnncaputil      |             |               |      |         |                     |
| u1              |             |               |      |         |                     |
| u1 L1.          | .8056815    | .0522661      | 15.41| 0.000   | .7032418 .9081212   |
| u2 L1.          | 1 (constrained) |               |      |         |                     |
| e.u1            | 1 (constrained) |               |      |         |                     |
| e.u1            | -.5188453   | .0701985      | -7.39| 0.000   | -.6564317 -.3812588 |
| D.lncaputil u1  | 1 (constrained) |               |      |         |                     |
| /state var(u1)  | .00000582   | 3.91e-06      | 14.88| 0.000   | .00000505 .00000659 |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
The command in the above output specifies two state equations, one observation equation, and two options. The first state equation defines $u_{1t}$ and the second defines $u_{2t}$ according to (2) above. The observation equation defines the process for $\Delta \text{lncaputil}$ according to the one specified in (3) above. Several coefficients in (2) and (3) are set to 1, and constraints 2–4 place these restrictions on the model.

The estimated coefficient on $L.u1$ in equation $u1$, 0.806, is the estimate of $\alpha$ in (2), which is the autoregressive coefficient in the ARMA model in (1). The estimated coefficient on $e.u1$ in equation $u2$, −0.519, is the estimate of $\theta$, which is the moving-average term in the ARMA model in (1).

This example highlights a difference between the error-form syntax and the covariance-form syntax. The error-form syntax used in this example includes only explicitly included errors. In contrast, the covariance-form syntax includes an error term in each equation, unless the noerror option is specified.

The default for covstate() also differs between the error-form syntax and the covariance-form syntax. Because the coefficients on the errors in the error-form syntax are frequently used to estimate the standard deviation of the errors, covstate(identity) is the default for the error-form syntax. In contrast, unit variances are less common in the covariance-form syntax, for which covstate(diagonal) is the default. In this example, we specified covstate(diagonal) to estimate a nonunitary variance for the state.

```
Typing

. arima D.lncaputil, noconstant ar(1) ma(1) technique(nr)
(output omitted)
```

produces nearly identical results. As in the AR(1) example above, arima estimates the standard deviation of the error term, while sspace estimates the variance. Although they are theoretically equivalent, the different parameterizations give rise to small numerical differences in the other parameters.

Example 3: A VAR(1) model

The variable lnhours contains data on the log of manufacturing hours, which we treat as first-difference stationary. We have a theory in which the process driving the changes in the log utilization rate affects the changes in the log of hours, but changes in the log hours do not affect changes in the log utilization rate. In line with this theory, we estimate the parameters of a lower triangular, first-order vector autoregressive (VAR(1)) process

$$
\begin{pmatrix}
\Delta \text{lncaputil}_t \\
\Delta \text{lnhours}_t
\end{pmatrix} =
\begin{pmatrix}
\alpha_1 & 0 \\
\alpha_2 & \alpha_3
\end{pmatrix}
\begin{pmatrix}
\Delta \text{lncaputil}_{t-1} \\
\Delta \text{lnhours}_{t-1}
\end{pmatrix} +
\begin{pmatrix}
\epsilon_{1t} \\
\epsilon_{2t}
\end{pmatrix}
$$

(4)

where $\Delta y_t = y_t - y_{t-1}$, $\epsilon_t = (\epsilon_{1t}, \epsilon_{2t})'$ and Var($\epsilon$) = $\Sigma$. We can write this VAR(1) process as a state-space model with state equations

$$
\begin{pmatrix}
u_{1t} \\
u_{2t}
\end{pmatrix} =
\begin{pmatrix}\alpha_1 & 0 \\
\alpha_2 & \alpha_3 \end{pmatrix}
\begin{pmatrix}u_{1(t-1)} \\
u_{2(t-1)}
\end{pmatrix} +
\begin{pmatrix}\epsilon_{1t} \\
\epsilon_{2t}
\end{pmatrix}
$$

(5)

with Var($\epsilon$) = $\Sigma$ and observation equations

$$
\begin{pmatrix}
\Delta \text{lncaputil} \\
\Delta \text{lnhours}
\end{pmatrix} =
\begin{pmatrix}
u_{1t} \\
u_{2t}
\end{pmatrix}
$$
Below we estimate the parameters of the state-space model:

- constraint 5 \([D.lncaputil]u1 = 1\)
- constraint 6 \([D.lnhours]u2 = 1\)
- \texttt{sspace} (u1 L.u1, state noconstant) (u2 L.u1 L.u2, state noconstant) \(> (D.lncaputil u1, noconstant noerror) (D.lnhours u2, noconstant noerror), \)
- constraints(5/6) covstate(unstructured)
- searching for initial values ............
- (setting technique to bhhh)
- Iteration 0: log likelihood = 2789.6095
- Iteration 1: log likelihood = 2957.8299
- (output omitted)
- Refining estimates:
- Iteration 0: log likelihood = 3211.7532
- Iteration 1: log likelihood = 3211.7532

State-space model

Sample: 1972m2 - 2008m12 Number of obs = 443
Wald chi2(3) = 166.87
Log likelihood = 3211.7532 Prob > chi2 = 0.0000

( 1) \([D.lncaputil]u1 = 1\)
( 2) \([D.lnhours]u2 = 1\)

| OIM | Coef. | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-----|-------|-----------|-------|------|----------------------|
| u1  | u1    | .353257   | .0448456 | 7.88 | 0.000 | .2653612 .4411528   |
|     | L1.   |           |         |      |         |                      |
| u2  | u1    | .1286218  | .0394742 | 3.26 | 0.001 | .0512537 .2059899   |
|     | L1.   |           |         |      |         |                      |
|     | u2    | -.3707083 | .0434255 | -8.54| 0.000 | -.4558208 -.2855959 |
|     | L1.   |           |         |      |         |                      |

D.lncaputil

| Coef. | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-------|-----------|-------|------|----------------------|
| u1    | 1         | (constrained) |

D.lnhours

| Coef. | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-------|-----------|-------|------|----------------------|
| u2    | 1         | (constrained) |

/state

<table>
<thead>
<tr>
<th>var(u1)</th>
<th>0.0000623</th>
<th>4.19e-06</th>
<th>14.88</th>
<th>0.000</th>
<th>0.000541 .0000705</th>
</tr>
</thead>
<tbody>
<tr>
<td>cov(u1,u2)</td>
<td>0.000026</td>
<td>2.67e-06</td>
<td>9.75</td>
<td>0.000</td>
<td>0.000208 .0000312</td>
</tr>
<tr>
<td>var(u2)</td>
<td>0.000386</td>
<td>2.61e-06</td>
<td>14.76</td>
<td>0.000</td>
<td>0.000335 .0000437</td>
</tr>
</tbody>
</table>

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

Specifying \texttt{covstate(unstructured)} caused \texttt{sspace} to estimate the off-diagonal element of \(\Sigma\). The output indicates that this parameter, \texttt{/state:cov(u1,u2)}, is small but statistically significant.

The estimated coefficient on \(L.u1\) in equation \(u1, 0.353\), is the estimate of \(\alpha_1\) in (5). The estimated coefficient on \(L.u1\) in equation \(u2, 0.129\), is the estimate of \(\alpha_2\) in (5). The estimated coefficient on \(L.u1\) in equation \(u2, -0.371\), is the estimate of \(\alpha_3\) in (5).

For the \texttt{VAR(1)} model in (4), the estimated autoregressive coefficient for \(D.lncaputil\) is similar to the corresponding estimate in the univariate results in \texttt{example 1}. The estimated effect of \(LD.lncaputil\) on \(D.lnhours\) is 0.129, the estimated autoregressive coefficient of \(D.lnhours\) is \(-0.371\), and both are statistically significant.
These estimates can be compared with those produced by typing

```
. constraint 101 [D.lncaputil]LD.lnhours = 0
. var D.lncaputil D.lnhours, lags(1) noconstant constraints(101)
(output omitted)
. matrix list e(Sigma)
(output omitted)
```

The `var` estimates are not the same as the `sspace` estimates because the generalized least-squares estimator implemented in `var` is only asymptotically equivalent to the ML estimator implemented in `sspace`, but the point estimates are similar. The comparison is useful for pedagogical purposes because the `var` estimator is relatively simple.

Some problems require constraining a covariance term to zero. If we wanted to constrain \([/state]\text{cov}(u1,u2)\) to zero, we could type

```
. constraint 7 [;/state]\text{cov}(u1,u2) = 0
. sspace (u1 L.u1, state noconstant)
> (u2 L.u1 L.u2, state noconstant)
> (D.lncaputil u1, noconstant noerror)
> (D.lnhours u2, noconstant noerror),
> constraints(5/7) covstate(unstructured)
(output omitted)
```

Example 4: A VARMA(1,1) model

We now extend the previous example by modeling \(D.lncaputil\) and \(D.lnhours\) as a first-order vector autoregressive moving-average (VARMA(1,1)) process. Building on the previous examples, we allow the lag of \(D.lncaputil\) to affect \(D.lnhours\) but we do not allow the lag of \(D.lnhours\) to affect the lag of \(D.lncaputil\). Previous univariate analysis revealed that \(D.lnhours\) is better modeled as an autoregressive process than as an ARMA(1,1) process. As a result, we estimate the parameters of

\[
\begin{pmatrix}
\Delta lncaputil_t \\
\Delta lnhours_t
\end{pmatrix} = \begin{pmatrix}
\alpha_1 & 0 \\
\alpha_2 & \alpha_3
\end{pmatrix}
\begin{pmatrix}
\Delta lncaputil_{t-1} \\
\Delta lnhours_{t-1}
\end{pmatrix} + \begin{pmatrix}
\theta_1 & 0 \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
\epsilon_{1(t-1)} \\
\epsilon_{2(t-1)}
\end{pmatrix} + \begin{pmatrix}
\epsilon_{1t} \\
\epsilon_{2t}
\end{pmatrix}
\]

We can write this VARMA(1,1) process as a state-space model with state equations

\[
\begin{pmatrix}
s_{1t} \\
s_{2t} \\
s_{3t}
\end{pmatrix} = \begin{pmatrix}
\alpha_1 & 1 & 0 \\
0 & 0 & 0 \\
\alpha_2 & 0 & \alpha_3
\end{pmatrix}
\begin{pmatrix}
s_{1(t-1)} \\
s_{2(t-1)} \\
s_{3(t-1)}
\end{pmatrix} + \begin{pmatrix}
1 & 0 \\
\theta_1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
\epsilon_{1t} \\
\epsilon_{2t}
\end{pmatrix}
\]

where the states are

\[
\begin{pmatrix}
s_{1t} \\
s_{2t} \\
s_{3t}
\end{pmatrix} = \begin{pmatrix}
\Delta lncaputil_t \\
\theta_1 \epsilon_{1t} \\
\Delta lnhours_t
\end{pmatrix}
\]

and we simplify the problem by assuming that

\[
\text{Var}\begin{pmatrix}
\epsilon_{1t} \\
\epsilon_{2t}
\end{pmatrix} = \begin{pmatrix}
\sigma_1^2 & 0 \\
0 & \sigma_2^2
\end{pmatrix}
\]
Below we estimate the parameters of this model by using \texttt{sspace}:

```plaintext
constraint 7 [u1]L.u2 = 1
constraint 8 [u1]e.u1 = 1
constraint 9 [u3]e.u3 = 1
constraint 10 [D.lncaputil]u1 = 1
constraint 11 [D.lnhours]u3 = 1

\texttt{sspace (u1 L.u1 L.u2 e.u1, state noconstant) (u2 e.u1, state noconstant)}
> (u3 L.u1 L.u3 e.u3, state noconstant)
> (D.lncaputil u1, noconstant) (D.lnhours u3, noconstant),
> constraints(7/11) technique(nr) covstate(diagonal)
searching for initial values ...........
(output omitted)
```

Refining estimates:

```
Iteration 0:  log likelihood = 3156.0564
Iteration 1:  log likelihood = 3156.0564
```

State-space model

```
Sample: 1972m2 - 2008m12  Number of obs = 443
Log likelihood = 3156.0564  Prob > chi2 = 0.0000
Wald chi2(4) = 427.55
( 1) [u1]L.u2 = 1
( 2) [u1]e.u1 = 1
( 3) [u3]e.u3 = 1
( 4) [D.lncaputil]u1 = 1
( 5) [D.lnhours]u3 = 1
```

| OIM                        | Coef.    | Std. Err. | z      | P>|z|  | [95% Conf. Interval] |
|----------------------------|----------|-----------|--------|------|----------------------|
| u1                         |          |           |        |      |                      |
| u1                         |          |           |        |      |                      |
| L1.                        | .8058031 | .0522493  | 15.42  | 0.000| .7033964 .9082098   |
| e.u1                       | -.518907 | .0701848  | -7.39  | 0.000| -.6564667 -.3813474 |
| u2                         |          |           |        |      |                      |
| L1.                        | .1734868 | .0405156  | 4.28   | 0.000| .0940776 .252896    |
| e.u3                       | -.4809376| .0498574  | -9.65  | 0.000| -.5786563 -.3832188 |
| D.lncaputil u1             | 1 (constrained) |          |        |      |                      |
| D.lnhours u3               | 1 (constrained) |          |        |      |                      |
| /state                     |          |           |        |      |                      |
| var(u1)                    | .00000582| 3.91e-06  | 14.88  | 0.000| .00000505 .0000659  |
| var(u3)                    | .0000382 | 2.56e-06  | 14.88  | 0.000| .0000331 .0000432  |
```

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
The estimates of the parameters in the model for D.lncaputil are similar to those in the univariate model fit in example 2. The estimates of the parameters in the model for D.lnhours indicate that the lag of D.lncaputil has a positive effect on D.lnhours.

Technical note

The technique(nr) option facilitates convergence in example 4. Fitting state-space models is notoriously difficult. Convergence problems are common. Four methods for overcoming convergence problems are 1) selecting an alternate optimization algorithm by using the technique() option, 2) using alternative starting values by specifying the from() option, 3) using starting values obtained by estimating the parameters of a restricted version of the model of interest, or 4) putting the variables on the same scale.

Example 5: A dynamic-factor model

Stock and Watson (1989, 1991) wrote a simple macroeconomic model as a dynamic-factor model, estimated the parameters by ML, and extracted an economic indicator. In this example, we estimate the parameters of a dynamic-factor model. In [TS] sspace postestimation, we extend this example and extract an economic indicator for the differenced series.

We have data on an industrial-production index, ipman; an aggregate weekly hours index, hours; and aggregate unemployment, unemp. income is real disposable income divided by 100. We rescaled real disposable income to avoid convergence problems.

We postulate a latent factor that follows an AR(2) process. Each measured variable is then related to the current value of that latent variable by a parameter. The state-space form of our model is

\[
\begin{pmatrix}
  f_t \\
  f_{t-1}
\end{pmatrix}
= \begin{pmatrix}
  \theta_1 & \theta_2 \\
  1 & 0
\end{pmatrix}
\begin{pmatrix}
  f_{t-1} \\
  f_{t-2}
\end{pmatrix}
+ \begin{pmatrix}
  \nu_t
\end{pmatrix}
\]

\[
\begin{pmatrix}
  \Delta \text{ipman}_t \\
  \Delta \text{income}_t \\
  \Delta \text{hours}_t \\
  \Delta \text{unemp}_t
\end{pmatrix}
= \begin{pmatrix}
  \gamma_1 \\
  \gamma_2 \\
  \gamma_3 \\
  \gamma_4
\end{pmatrix}
\begin{pmatrix}
  f_t
\end{pmatrix}
+ \begin{pmatrix}
  \epsilon_{1t} \\
  \epsilon_{2t} \\
  \epsilon_{3t} \\
  \epsilon_{4t}
\end{pmatrix}
\]

where

\[
\text{Var} \begin{pmatrix}
  \epsilon_{1t} \\
  \epsilon_{2t} \\
  \epsilon_{3t} \\
  \epsilon_{4t}
\end{pmatrix}
= \begin{pmatrix}
  \sigma_1^2 & 0 & 0 & 0 \\
  0 & \sigma_2^2 & 0 & 0 \\
  0 & 0 & \sigma_3^2 & 0 \\
  0 & 0 & 0 & \sigma_4^2
\end{pmatrix}
\]
The parameter estimates are

```
use https://www.stata-press.com/data/r16/dfex
(St. Louis Fed (FRED) macro data)
constraint 12 [lf]L.f = 1
.sspace (f L.f L.1f, state noconstant)
> (lf L.f, state noconstant noerror)
> (D.ipman f, noconstant)
> (D.income f, noconstant)
> (D.hours f, noconstant)
> (D.unemp f, noconstant),
> covstate(identity) constraints(12)
searching for initial values ..................
(setting technique to bhhh)
Iteration 0: log likelihood = -674.18497
Iteration 1: log likelihood = -667.23913
(output omitted)
Refining estimates:
Iteration 0: log likelihood = -662.09507
Iteration 1: log likelihood = -662.09507
State-space model
Sample: 1972m2 - 2008m11 Number of obs = 442
Wald chi2(6) = 751.95
Log likelihood = -662.09507 Prob > chi2 = 0.0000
( 1) [lf]L.f = 1
```

| OIM          | Coef. | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|--------------|-------|-----------|-------|------|----------------------|
| f            |       |           |       |      |                      |
| f            |       |           |       |      |                      |
| L1.          | .2651932 | .0568663 | 4.66  | 0.000 | .1537372 .3766491    |
| L1.          | .4820398 | .0624635 | 7.72  | 0.000 | .3596136 .604466     |
| lf           |       |           |       |      |                      |
| L1.          | .4820398 | .0624635 | 7.72  | 0.000 | .3596136 .604466     |
| D.ipman      |       |           |       |      |                      |
| f            | .3502249 | .0287389 | 12.19 | 0.000 | .2938976 .4065522    |
| D.income     |       |           |       |      |                      |
| f            | .0746338 | .0217319 | 3.43  | 0.001 | .0320401 .1172276    |
| D.hours      |       |           |       |      |                      |
| f            | .2177469 | .0186769 | 11.66 | 0.000 | .1811407 .254353     |
| D.unemp      |       |           |       |      |                      |
| f            | -.0676016 | .0071022 | -9.52 | 0.000 | -.0815217 -.0536816  |
| /observable  |       |           |       |      |                      |
| var(D.ipman) | .1383158 | .0167086 | 8.28  | 0.000 | .1055675 .1710641    |
| var(D.income)| .2773808 | .0188302 | 14.73 | 0.000 | .2404743 .3142873    |
| var(D.hours)| .0911446 | .0080847 | 11.27 | 0.000 | .0752988 .1069903    |
| var(D.unemp)| .0237232 | .0017932 | 13.23 | 0.000 | .0202086 .0272378    |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
The output indicates that the unobserved factor is quite persistent and that it is a significant predictor for each of the observed variables.

These models are frequently used to forecast the dependent variables and to estimate the unobserved factors. We present some illustrative examples in [TS] sspace postestimation. The dfactor command estimates the parameters of dynamic-factor models; see [TS] dfactor.

Some nonstationary state-space models

Example 6: A local-level model

Harvey (1989) advocates the use of STS models. These models parameterize the trends and seasonal components of a set of time series. The simplest STS model is the local-level model, which is given by

\[ y_t = \mu_t + \epsilon_t \]

where

\[ \mu_t = \mu_{t-1} + \nu_t \]

The model is called a local-level model because the level of the series is modeled as a random walk plus an idiosyncratic noise term. (The model is also known as the random-walk-plus-noise model.) The local-level model is nonstationary because of the random-walk component. When the variance of the idiosyncratic-disturbance \( \epsilon_t \) is zero and the variance of the level-disturbance \( \nu_t \) is not zero, the local-level model reduces to a random walk. When the variance of the level-disturbance \( \nu_t \) is zero and the variance of the idiosyncratic-disturbance \( \epsilon_t \) is not zero,

\[ \mu_t = \mu_{t-1} = \mu \]

and the local-level model reduces to

\[ y_t = \mu + \epsilon_t \]

which is a simple regression with a time-invariant mean. The parameter \( \mu \) is not estimated in the state-space formulation below.

In this example, we fit weekly levels of the Standard and Poor’s 500 Index to a local-level model. Because this model is already in state-space form, we fit close by typing
use https://www.stata-press.com/data/r16/sp500w
.
constraint 13 [z]L.z = 1
.
constraint 14 [close]z = 1
.
sspace (z L.z, state noconstant) (close z, noconstant), constraints(13 14)
searching for initial values ...........
(setting technique to bhhh)
Iteration 0: log likelihood = -12582.89
Iteration 1: log likelihood = -12577.146
(output omitted)

State-space model
Sample: 1 - 3093 Number of obs = 3,093
Log likelihood = -12576.99
( 1) [z]L.z = 1
( 2) [close]z = 1

```
+-------------------------------+-----------------------------+
|                      Coef.  Std. Err. z    P>|z|    [95% Conf. Interval] |
|-------------------------------+-----------------------------|
| close                        OIM                      |
| z                             1 (constrained)  |
| L1.                           |
| close                        OIM                      |
| z                             1 (constrained)  |
| /state                       var(z)                   |
| 170.3456 7.584909 22.46 0.000 155.4794 185.2117 |
| /observable                   var(close)              |
| 15.24858 3.392457 4.49 0.000 8.599486 21.89767 |
+-------------------------------+-----------------------------+

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The results indicate that both components have nonzero variances. The output footer informs us that the model is nonstationary at the estimated parameter values.

Technical note

In the previous example, we estimated the parameters of a nonstationary state-space model. The model is nonstationary because one of the eigenvalues of the A matrix has unit modulus. That all the coefficients in the A matrix are fixed is also important. See Lütkepohl (2005, 636–637) for why the ML estimator for the parameters of a nonstationary state-model that is nonstationary because of eigenvalues with unit moduli from a fixed A matrix is still consistent and asymptotically normal.

Example 7: A local linear-trend model

In another basic STS model, known as the local linear-trend model, both the level and the slope of a linear time trend are random walks. Here are the state equations and the observation equation for a local linear-trend model for the level of industrial production contained in variable ipman:

\[
\begin{pmatrix}
\mu_t \\
\beta_t
\end{pmatrix} = \begin{pmatrix}
1 & 1 \\
0 & 1
\end{pmatrix} \begin{pmatrix}
\mu_{t-1} \\
\beta_{t-1}
\end{pmatrix} + \begin{pmatrix}
\nu_{1t} \\
\nu_{2t}
\end{pmatrix}
\]

\[ipman_t = \mu_t + \epsilon_t\]
The estimated parameters are

```
  . use https://www.stata-press.com/data/r16/dfex
     (St. Louis Fed (FRED) macro data)
  . constraint 15 [f1]L.f1 = 1
  . constraint 16 [f1]L.f2 = 1
  . constraint 17 [f2]L.f2 = 1
  . constraint 18 [ipman]f1 = 1
  . sspace (f1 L.f1 L.f2, state noconstant) (f2 L.f2, state noconstant)
     > (ipman f1, noconstant), constraints(15/18)
  searching for initial values ...........
     (setting technique to bhhh)
  Iteration 0:  log likelihood = -362.93861
  Iteration 1:  log likelihood = -362.12048
```

(output omitted)

State-space model

Sample: 1972m1 - 2008m11  Number of obs = 443
Log likelihood = -359.1266

( 1) [f1]L.f1 = 1
( 2) [f1]L.f2 = 1
( 3) [f2]L.f2 = 1
( 4) [ipman]f1 = 1

| OIM                  | Coef. | Std. Err. |     z  |    P>|z|   | [95% Conf. Interval] |
|----------------------|-------|-----------|--------|--------|----------------------|
|                      |       |           |        |        |                      |
| ipman                |       |           |        |        |                      |
| f1                   |       |           |        |        |                      |
| f1                   |       |           |        |        |                      |
| L1.                  |       |           |        |        |                      |
| f2                   |       |           |        |        |                      |
| L1.                  |       |           |        |        |                      |
| f2                   |       |           |        |        |                      |
| L1.                  |       |           |        |        |                      |
| ipman                |       |           |        |        |                      |
| f1                   |       |           |        |        |                      |

/state

| var(f1) | .1473071 | .0407156 | 3.62   | 0.000 | .067506   | .2271082 |
| var(f2) | .0178752 | .0065743 | 2.72   | 0.003 | .0049898  | .0307606 |

/observable

| var(ipman) | .0354429 | .0148186 | 2.39   | 0.008 | .0063989  | .0644868 |

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

There is little evidence that either of the variance parameters are zero. The fit obtained indicates that we could now proceed with specification testing and checks to see how well this model forecasts these data.
Stored results

`sspace` stores the following in `e()`:

Scalars
- `e(N)` number of observations
- `e(k)` number of parameters
- `e(k_eq)` number of equations in `e(b)`
- `e(k_dv)` number of dependent variables
- `e(k_obser)` number of observation equations
- `e(k_state)` number of state equations
- `e(k_obser_err)` number of observation-error terms
- `e(k_state_err)` number of state-error terms
- `e(df_m)` model degrees of freedom
- `e(ll)` log likelihood
- `e(chi2)` $\chi^2$ statistic
- `e(p)` $p$-value for model test
- `e(tmin)` minimum time in sample
- `e(tmax)` maximum time in sample
- `e(stationary)` 1 if the estimated parameters indicate a stationary model, 0 otherwise
- `e(rank)` rank of VCE
- `e(ic)` number of iterations
- `e(rc)` return code
- `e(converged)` 1 if converged, 0 otherwise

Macros
- `e(cmd)` `sspace`
- `e(cmdline)` command as typed
- `e(depvar)` unoperated names of dependent variables in observation equations
- `e(obser_deps)` names of dependent variables in observation equations
- `e(state_deps)` names of dependent variables in state equations
- `e(covariates)` list of covariates
- `e(tvar)` variable denoting time within groups
- `e(eqnames)` names of equations
- `e(title)` title in estimation output
- `e(tmins)` formatted minimum time
- `e(tmaxs)` formatted maximum time
- `e(R_structure)` structure of observed-variable-error covariance matrix
- `e(Q_structure)` structure of state-error covariance matrix
- `e(chi2type)` Wald; type of model $\chi^2$ test
- `e(vce)` `vcetype` specified in `vce()`
- `e(vcetype)` title used to label Std. Err.
- `e(opt)` type of optimization
- `e(method)` likelihood method
- `e(initial_values)` type of initial values
- `e(technique)` maximization technique
- `e(tech_steps)` iterations taken in maximization technique
- `e(datasignature)` the checksum
- `e(datasignaturevars)` variables used in calculation of checksum
- `e(properties)` `b V`
- `e(estat_cmd)` program used to implement `estat`
- `e(predict)` program used to implement `predict`
- `e(marginsnotok)` predictions disallowed by `margins`
- `e(asbalanced)` factor variables `fvset` as `asbalanced`
- `e(asobserved)` factor variables `fvset` as `asobserved`
Matrices

- \( e(b) \): parameter vector
- \( e(Cns) \): constraints matrix
- \( e(ilog) \): iteration log (up to 20 iterations)
- \( e(gradient) \): gradient vector
- \( e(gamma) \): mapping from parameter vector to state-space matrices
- \( e(A) \): estimated \( A \) matrix
- \( e(B) \): estimated \( B \) matrix
- \( e(C) \): estimated \( C \) matrix
- \( e(D) \): estimated \( D \) matrix
- \( e(F) \): estimated \( F \) matrix
- \( e(G) \): estimated \( G \) matrix
- \( e(chol_R) \): Cholesky factor of estimated \( R \) matrix
- \( e(chol_Q) \): Cholesky factor of estimated \( Q \) matrix
- \( e(chol_{Sz0}) \): Cholesky factor of initial state covariance matrix
- \( e(z0) \): initial state vector augmented with a matrix identifying nonstationary components
- \( e(d) \): additional term in diffuse initial state vector, if nonstationary model
- \( e(T) \): inner part of quadratic form for initial state covariance in a partially nonstationary model
- \( e(M) \): outer part of quadratic form for initial state covariance in a partially nonstationary model
- \( e(V) \): variance–covariance matrix of the estimators
- \( e(V_{modelbased}) \): model-based variance

Functions

- \( e(sample) \): marks estimation sample

### Methods and formulas

Recall that our notation for linear state-space models with time-invariant coefficient matrices is

\[
\begin{align*}
\mathbf{z}_t &= \mathbf{A}\mathbf{z}_{t-1} + \mathbf{B}\mathbf{x}_t + \mathbf{C}\mathbf{\epsilon}_t \\
\mathbf{y}_t &= \mathbf{D}\mathbf{z}_t + \mathbf{F}\mathbf{w}_t + \mathbf{G}\mathbf{\nu}_t
\end{align*}
\]

where

- \( \mathbf{z}_t \) is an \( m \times 1 \) vector of unobserved state variables;
- \( \mathbf{x}_t \) is a \( k_x \times 1 \) vector of exogenous variables;
- \( \mathbf{\epsilon}_t \) is a \( q \times 1 \) vector of state-error terms, \( (q \leq m) \);
- \( \mathbf{y}_t \) is an \( n \times 1 \) vector of observed endogenous variables;
- \( \mathbf{w}_t \) is a \( k_w \times 1 \) vector of exogenous variables;
- \( \mathbf{\nu}_t \) is an \( r \times 1 \) vector of observation-error terms, \( (r \leq n) \); and

\( \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{F}, \) and \( \mathbf{G} \) are parameter matrices.

The equations for \( \mathbf{z}_t \) are known as the state equations, and the equations for \( \mathbf{y}_t \) are known as the observation equations.

The error terms are assumed to be zero mean, normally distributed, serially uncorrelated, and uncorrelated with each other;

\[
\begin{align*}
\mathbf{\epsilon}_t &\sim N(0, \mathbf{Q}) \\
\mathbf{\nu}_t &\sim N(0, \mathbf{R}) \\
E[\mathbf{\epsilon}_t \mathbf{\epsilon}_s'] &= 0 \text{ for all } s \neq t \\
E[\mathbf{\epsilon}_t \mathbf{\nu}_s'] &= 0 \text{ for all } s \text{ and } t
\end{align*}
\]
sspace estimates the parameters of linear state-space models by maximum likelihood. The Kalman filter is a method for recursively obtaining linear, least-squares forecasts of $y_t$ conditional on past information. These forecasts are used to construct the log likelihood, assuming normality and stationarity. When the model is nonstationary, a diffuse Kalman filter is used.

Hamilton (1994a; 1994b, 389) shows that the QML estimator, obtained when the normality assumption is dropped, is consistent and asymptotically normal, although the variance–covariance matrix of the estimator (VCE) must be estimated by the Huber/White/sandwich estimator. Hamilton’s discussion applies to stationary models, and specifying `vce(robust)` produces a consistent estimator of the VCE when the errors are not normal.

Methods for computing the log likelihood differ in how they calculate initial values for the Kalman filter when the model is stationary, how they compute a diffuse Kalman filter when the model is nonstationary, and whether terms for initial states are included. sspace offers the method(hybrid), method(dejong), and method(kdiffuse) options for computing the log likelihood. All three methods handle both stationary and nonstationary models.

method(hybrid), the default, uses the initial values for the states implied by stationarity to initialize the Kalman filter when the model is stationary. Hamilton (1994b, 378) discusses this method of computing initial values for the states and derives a log-likelihood function that does not include terms for the initial states. When the model is nonstationary, method(hybrid) uses the De Jong (1988, 1991) diffuse Kalman filter and log-likelihood function, which includes terms for the initial states.


method(kdiffuse) implements the seldom-used large-$\kappa$ diffuse approximation to the diffuse Kalman filter when the model is nonstationary and uses initial values for the states implied by stationarity when the model is stationary. The log likelihood does not include terms for the initial states in either case. We recommend that you do not use method(kdiffuse) except to replicate older results computed using this method.

De Jong (1988, 1991) and De Jong and Chu-Chun-Lin (1994) derive the log likelihood and a diffuse Kalman filter for handling nonstationary data. De Jong (1988) replaces the stationarity assumption with a time-immemorial assumption, which he uses to derive the log-likelihood function, an initial state vector, and a covariance of the initial state vector when the model is nonstationary. By default, and when method(hybrid) or method(dejong) is specified, sspace uses the diffuse Kalman filter given in definition 5 of De Jong and Chu-Chun-Lin (1994). This method uses theorem 3 of De Jong and Chu-Chun-Lin (1994) to compute the covariance of the initial states. When using this method, sspace saves the matrices from their theorem 3 in $e()$, although the names are changed. $e(Z)$ is their $U_1$, $e(T)$ is their $U_2$, $e(A)$ is their $T$, and $e(M)$ is their $M$.


Practical estimation and inference require that the maximum likelihood estimator be consistent and normally distributed in large samples. These statistical properties of the maximum likelihood estimator are well established when the model is stationary; see Caines (1988, chap. 5 and 7), Hamilton (1994b, 388–389), and Hannan and Deistler (1988, chap. 4). When the model is nonstationary, additional assumptions must hold for the maximum likelihood estimator to be consistent and asymptotically normal; see Harvey (1989, sec. 3.4), Lütkepohl (2005, 636–637), and Schneider (1988). Chang, Miller, and Park (2009) show that the ML and the QML estimators are consistent and asymptotically normal for a class of nonstationary state-space models.
We now give an intuitive version of the Kalman filter. sspace uses theoretically equivalent, but numerically more stable, methods. For each time $t$, the Kalman filter produces the conditional expected state vector $z_{t|t-1}$ and the conditional covariance matrix $\Omega_{t|t-1}$; both are conditional on information up to and including time $t$. Using the model and previous period results, for each $t$ we begin with

$$
\begin{align*}
  z_{t|t-1} &= Az_{t-1|t-1} + Bx_t \\
  \Omega_{t|t-1} &= A\Omega_{t-1|t-1}A' + CQC' \\
  y_{t|t-1} &= Dz_{t|t-1} + Fw_t
\end{align*}
$$

(6)

The residuals and the mean squared error (MSE) matrix of the forecast error are

$$
\begin{align*}
  \tilde{\nu}_{t|t} &= y_t - y_{t|t-1} \\
  \Sigma_{t|t} &= D\Omega_{t|t-1}D' + GRG'
\end{align*}
$$

(7)

In the last steps, we update the conditional expected state vector and the conditional covariance with the time $t$ information:

$$
\begin{align*}
  z_{t|t} &= z_{t|t-1} + \Omega_{t|t-1}D\Sigma_{t|t}^{-1}\tilde{\nu}_{t|t} \\
  \Omega_{t|t} &= \Omega_{t|t-1} - \Omega_{t|t-1}D\Sigma_{t|t}^{-1}D'\Omega_{t|t-1}
\end{align*}
$$

(8)

Equations (6)–(8) are the Kalman filter. The equations denoted by (6) are the one-step predictions. The one-step predictions do not use contemporaneous values of $y_t$; only past values of $y_t$, past values of the exogenous $x_t$, and contemporaneous values of $x_t$ are used. Equations (7) and (8) form the update step of the Kalman filter; they incorporate the contemporaneous dependent variable information into the predicted states.

The Kalman filter requires initial values for the states and a covariance matrix for the initial states to start off the recursive process. Hamilton (1994b) discusses how to compute initial values for the Kalman filter assuming stationarity. This method is used by default when the model is stationary. De Jong (1988) discusses how to estimate initial values by maximum likelihood; this method is used when method(dejong) is specified.

Letting $\delta$ be the vector of parameters in the model, Lütkepohl (2005) and Harvey (1989) show that the log-likelihood function for the parameters of a stationary model is given by

$$
\ln L(\delta) = -0.5 \left\{ nT \ln(2\pi) + \sum_{t=1}^{T} \ln(|\Sigma_{t|t-1}|) + \sum_{t=1}^{T} e_t'\Sigma_{t|t-1}^{-1}e_t \right\}
$$

where $e_t = (y_t - y_{t|t-1})$ depends on $\delta$ and $\Sigma$ also depends on $\delta$.

The variance–covariance matrix of the estimator (VCE) is estimated by the observed information matrix (OIM) estimator by default. Specifying vce(robust) causes sspace to use the Huber/White/sandwich estimator. Both estimators of the VCE are standard and documented in Hamilton (1994b).

Hamilton (1994b), Hannan and Deistler (1988), and Caines (1988) show that the ML estimator is consistent and asymptotically normal when the model is stationary. Schneider (1988) establishes consistency and asymptotic normality when the model is nonstationary because $A$ has some eigenvalues with modulus 1 and there are no unknown parameters in $A$. 
Not all state-space models are identified, as discussed in Hamilton (1994b) and Lütkepohl (2005). \texttt{sspace} checks for local identification at the optimum. \texttt{sspace} will not declare convergence unless the Hessian is full rank. This check for local identifiability is due to Rothenberg (1971).

Specifying \texttt{method(dejong)} causes \texttt{sspace} to maximize the log-likelihood function given in section 2 (vii) of De Jong (1988). This log-likelihood function includes the initial states as parameters to be estimated. We use some of the methods in Casals, Sotoca, and Jerez (1999) for computing the De Jong (1988) log-likelihood function.

References


Also see

[TS] sspace postestimation — Postestimation tools for sspace
[TS] arima — ARIMA, ARMAX, and other dynamic regression models
[TS] dfactor — Dynamic-factor models
[TS] tsset — Declare data to be time-series data
[TS] ucm — Unobserved-components model
[TS] var — Vector autoregressive models

[U] 20 Estimation and postestimation commands

Postestimation commands

The following standard postestimation commands are available after `sspace`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as expected values. The root mean squared error is available for all predictions. All predictions are also available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] {stub* | newvarlist} [if] [in] [ , statistic options]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>observable variables</td>
</tr>
<tr>
<td>states</td>
<td>latent state variables</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
<tr>
<td>rstandard</td>
<td>standardized residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td></td>
</tr>
<tr>
<td>equation(eqnames)</td>
<td>name(s) of equation(s) for which predictions are to be made</td>
</tr>
<tr>
<td>rmse(stub*</td>
<td>newvarlist)</td>
</tr>
<tr>
<td>dynamic(time_constant)</td>
<td>begin dynamic forecast at specified time</td>
</tr>
</tbody>
</table>

| Advanced       |                           |
| smethod(method) | method for predicting unobserved states |

<table>
<thead>
<tr>
<th>method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onestep</td>
<td>predict using past information</td>
</tr>
<tr>
<td>smooth</td>
<td>predict using all sample information</td>
</tr>
<tr>
<td>filter</td>
<td>predict using past and contemporaneous information</td>
</tr>
</tbody>
</table>
Options for predict

**Main**

- **xb**, **states**, **residuals**, and **rstandard** specify the statistic to be predicted.
  - **xb**, the default, calculates the linear predictions of the observed variables.
  - **states** calculates the linear predictions of the latent state variables.
  - **residuals** calculates the residuals in the equations for observable variables. **residuals** may not be specified with **dynamic()**.
  - **rstandard** calculates the standardized residuals, which are the residuals normalized to be uncorrelated and to have unit variances. **rstandard** may not be specified with **smethod(filter)**, **smethod(smooth)**, or **dynamic()**.

**Options**

- **equation(eqnames)** specifies the equation(s) for which the predictions are to be calculated. If you do not specify **equation()** or **stub***, the results are the same as if you had specified the name of the first equation for the predicted statistic.
  - You specify a list of equation names, such as **equation(income consumption)** or **equation(factor1 factor2)**, to identify the equations. Specify names of state equations when predicting **states** and names of observable equations in all other cases.
  - **equation()** may not be specified with **stub***.

- **rmse(stub* newvarlist)** puts the root mean squared errors of the predicted statistics into the specified new variables. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error.

- **dynamic(time_constant)** specifies when **predict** starts producing dynamic forecasts. The specified **time_constant** must be in the scale of the time variable specified in **tsset**, and the **time_constant** must be inside a sample for which observations on the dependent variables are available. For example, **dynamic(tq(2008q4))** causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. **dynamic()** may not be specified with **rstandard**, **residuals**, or **smethod(smooth)**.

**Advanced**

- **smethod(method)** specifies the method for predicting the unobserved states; **smethod(onestep)**, **smethod(filter)**, and **smethod(smooth)** cause different amounts of information on the dependent variables to be used in predicting the states at each time period.
  - **smethod(onestep)**, the default, causes **predict** to estimate the states at each time period using previous information on the dependent variables. The Kalman filter is performed on previous periods, but only the one-step predictions are made for the current period.
  - **smethod(smooth)** causes **predict** to estimate the states at each time period using all the sample data by the Kalman smoother. **smethod(smooth)** may not be specified with **rstandard**.
  - **smethod(filter)** causes **predict** to estimate the states at each time period using previous and contemporaneous data by the Kalman filter. The Kalman filter is performed on previous periods and the current period. **smethod(filter)** may be specified only with **states**.
Remarks and examples

We assume that you have already read [TS] sspace. In this entry, we illustrate some of the features of `predict` after using `sspace` to estimate the parameters of a state-space model.

All the predictions after `sspace` depend on the unobserved states, which are estimated recursively. Changing the sample can alter the state estimates, which can change all other predictions.

Example 1: One-step predictions

In example 5 of [TS] sspace, we estimated the parameters of the dynamic-factor model

\[
\begin{align*}
\begin{bmatrix}
    f_t \\
    f_{t-1}
\end{bmatrix}
&= 
\begin{bmatrix}
    \theta_1 & \theta_2 \\
    1 & 0
\end{bmatrix}
\begin{bmatrix}
    f_{t-1} \\
    f_{t-2}
\end{bmatrix}
+ 
\begin{bmatrix}
    \nu_t
\end{bmatrix}

\begin{bmatrix}
    \Delta ipman_t \\
    \Delta income_t \\
    \Delta hours_t \\
    \Delta unemp_t
\end{bmatrix}
&= 
\begin{bmatrix}
    \gamma_1 \\
    \gamma_2 \\
    \gamma_3 \\
    \gamma_4
\end{bmatrix}
\begin{bmatrix}
    f_t
\end{bmatrix}
+ 
\begin{bmatrix}
    \epsilon_{1t} \\
    \epsilon_{2t} \\
    \epsilon_{3t} \\
    \epsilon_{4t}
\end{bmatrix}
\end{align*}
\]

where

\[
\text{Var}\begin{pmatrix}
\epsilon_{1t} \\
\epsilon_{2t} \\
\epsilon_{3t} \\
\epsilon_{4t}
\end{pmatrix} = 
\begin{pmatrix}
\sigma_1^2 & 0 & 0 & 0 \\
0 & \sigma_2^2 & 0 & 0 \\
0 & 0 & \sigma_3^2 & 0 \\
0 & 0 & 0 & \sigma_4^2
\end{pmatrix}
\]

by typing

```
. use https://www.stata-press.com/data/r16/dfex (St. Louis Fed (FRED) macro data)
. constraint 1 [lf]L.f = 1
. sspace (f L.f L.lf, state noconstant)
> (lf L.f, state noconstant noerror)
> (D.ipman f, noconstant)
> (D.income f, noconstant)
> (D.hours f, noconstant)
> (D.unemp f, noconstant),
> covstate(identity) constraints(1)
```

(output omitted)
Below we obtain the one-step predictions for each of the four dependent variables in the model, and then we graph the actual and predicted \textit{ipman}:

\begin{verbatim}
. predict dep*
  (option xb assumed; fitted values)
. tsline D.ipman dep1, lcolor(gs10) xtitle("") legend(rows(2))
\end{verbatim}

The graph shows that the one-step predictions account for only a small part of the swings in the realized \textit{ipman}.

\textbf{Example 2: Out-of-sample, dynamic predictions}

We use the estimates from the previous example to make out-of-sample predictions. After using \texttt{tsappend} to extend the dataset by six periods, we use \texttt{predict} with the \texttt{dynamic()} option and graph the result.

\begin{verbatim}
. tsappend, add(6)
. predict Dipman_f, dynamic(tm(2008m12)) equation(D.ipman)
. tsline D.ipman Dipman_f if month>=tm(2008m1), xtitle("") legend(rows(2))
\end{verbatim}
The model predicts that the changes in industrial production will remain negative for the forecast horizon, although they increase toward zero.

Example 3: Estimating an unobserved factor

In this example, we want to estimate the unobserved factor instead of predicting a dependent variable. Specifying `smethod(smooth)` causes `predict` to use all sample information in estimating the states by the Kalman smoother.

Below we estimate the unobserved factor by using the estimation sample, and we graph `ipman` and the estimated factor:

```
. predict fac if e(sample), states smethod(smooth) equation(f)
. tsline D.ipman fac, xtitle("") legend(rows(2))
```
Example 4: Calculating residuals

The residuals and the standardized residuals are frequently used to review the specification of the model.

Below we calculate the standardized residuals for each of the series and display them in a combined graph:

\[
\begin{align*}
&\text{. predict sres1-sres4 if e(sample), rstandard} \\
&\text{. tsline sres1, xtitle("") name(sres1)} \\
&\text{. tsline sres2, xtitle("") name(sres2)} \\
&\text{. tsline sres3, xtitle("") name(sres3)} \\
&\text{. tsline sres4, xtitle("") name(sres4)} \\
&\text{. graph combine sres1 sres2 sres3 sres4, name(combined)}
\end{align*}
\]

Methods and formulas

Estimating the unobserved states is key to predicting the dependent variables.

By default and with the smethod(onestep) option, predict estimates the states in each period by applying the Kalman filter to all previous periods and only making the one-step predictions to the current period. (See Methods and formulas of \texttt{[TS] sspace} for the Kalman filter equations.)

With the smethod(filter) option, predict estimates the states in each period by applying the Kalman filter on all previous periods and the current period. The computational difference between smethod(onestep) and smethod(filter) is that smethod(filter) performs the update step on the current period while smethod(onestep) does not. The statistical difference between
ssmethod(onestep) and ssmethod(filter) is that ssmethod(filter) uses contemporaneous information on the dependent variables while ssmethod(onestep) does not.

As noted in [TS] sspace, sspace has both a stationary and a diffuse Kalman filter. predict uses the same Kalman filter used for estimation.

With the ssmethod(smooth) option, predict estimates the states in each period using all the sample information by applying the Kalman smoother. predict uses the Harvey (1989, sec. 3.6.2) fixed-interval smoother with model-based initial values to estimate the states when the estimated parameters imply a stationary model. De Jong (1989) provides a computationally efficient method. Hamilton (1994) discusses the model-based initial values for stationary state-space models. When the model is nonstationary, the De Jong (1989) diffuse Kalman smoother is used to predict the states. The smoothed estimates of the states are subsequently used to predict the dependent variables.

The dependent variables are predicted by plugging in the estimated states. The residuals are calculated as the differences between the predicted and the realized dependent variables. The root mean squared errors are the square roots of the diagonal elements of the mean squared error matrices that are computed by the Kalman filter. The standardized residuals are the residuals normalized by the Cholesky factor of their mean squared error produced by the Kalman filter.

predict uses the Harvey (1989, sec. 3.5) methods to compute the dynamic forecasts and the root mean squared errors. Let \( \tau \) be the period at which the dynamic forecasts begin; \( \tau \) must either be in the specified sample or be in the period immediately following the specified sample.

The dynamic forecasts depend on the predicted states in the period \( \tau - 1 \), which predict obtains by running the Kalman filter or the diffuse Kalman filter on the previous sample observations. The states in the periods prior to starting the dynamic predictions may be estimated using ssmethod(onestep) or ssmethod(smooth).

Using an if or in qualifier to alter the prediction sample can change the estimate of the unobserved states in the period prior to beginning the dynamic predictions and hence alter the dynamic predictions. The initial states are estimated using e(b) and the prediction sample.

References


Also see

[TS] sspace — State-space models
[TS] dfactor — Dynamic-factor models
[TS] dfactor postestimation — Postestimation tools for dfactor
[U] 20 Estimation and postestimation commands
threshold — Threshold regression

**Description**

threshold extends linear regression to allow coefficients to differ across regions. Those regions are identified by a threshold variable being above or below a threshold value. The model may have multiple thresholds, and you can either specify a known number of thresholds or let threshold find that number for you through the Bayesian information criterion (BIC), Akaike information criterion (AIC), or Hannan–Quinn information criterion (HQIC).

**Quick start**

Threshold regression model for the dependent variable \( y \) with region-dependent intercepts for two regions of \( x \)

\[
\text{threshold } y, \text{ threshvar}(x)
\]

Add the first lag of \( x \) as a region-invariant variable

\[
\text{threshold } y \ 1.x, \text{ threshvar}(x)
\]

Add the first lag of \( y \) as a region-dependent variable

\[
\text{threshold } y \ 1.x, \text{ threshvar}(x) \ \text{regionvars}(1.y)
\]

Threshold regression model of \( y \) with region-dependent intercepts for three regions determined by two threshold values of \( x \)

\[
\text{threshold } y, \text{ threshvar}(x) \ nthresholds(2)
\]

Use BIC to select the number of thresholds from a maximum of 5 thresholds

\[
\text{threshold } y, \text{ threshvar}(x) \ \text{optthresh}(5)
\]

**Menu**

Statistics > Time series > Threshold regression model
Syntax

threshold depvar [ indepvars ] [ if ] [ in ], threshvar(varname) [ options ]

indepvars is a list of variables with region-invariant coefficients.

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>*threshvar(varname)</td>
<td>threshold variable</td>
</tr>
<tr>
<td>regionvars(varlist)</td>
<td>include region-varying coefficients for specified covariates</td>
</tr>
<tr>
<td>consoinvariant</td>
<td>replace region-varying constant with a region-invariant constant</td>
</tr>
<tr>
<td>noconstant</td>
<td>suppress region-varying constant terms</td>
</tr>
<tr>
<td>trim(#)</td>
<td>trimming percentage; default is trim(10)</td>
</tr>
<tr>
<td>nthresholds(#)</td>
<td>number of thresholds; default is nthresholds(1); not allowed with optthresh()</td>
</tr>
<tr>
<td>optthresh(#[, ictype ])</td>
<td>select optimal number of thresholds less than or equal to #; not allowed with nthresholds()</td>
</tr>
<tr>
<td>SE/Robust</td>
<td></td>
</tr>
<tr>
<td>vce(vcetype)</td>
<td>vcetype may be oim or robust</td>
</tr>
<tr>
<td>Reporting</td>
<td></td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>nocnsreport</td>
<td>do not display constraints</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling</td>
</tr>
<tr>
<td>nodots</td>
<td>suppress replication dots</td>
</tr>
<tr>
<td>dots(#)</td>
<td>display dots every # replications</td>
</tr>
<tr>
<td>Advanced</td>
<td></td>
</tr>
<tr>
<td>ssrs(stub*</td>
<td>newvarlist)</td>
</tr>
<tr>
<td>constraints(numlist)</td>
<td>apply specified linear constraints; not allowed with optthresh()</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

*threshvar() is required.

You must tsset your data before using threshold; see [TS] tsset.

indepvars and varlist may contain factor variables; see [U] 11.4.3 Factor variables.

depvar, indepvars, varlist, and varname may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.

coeflegend does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

<table>
<thead>
<tr>
<th>ictype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bic</td>
<td>Bayesian information criterion (BIC); the default</td>
</tr>
<tr>
<td>aic</td>
<td>Akaike information criterion (AIC)</td>
</tr>
<tr>
<td>hqic</td>
<td>Hannan–Quinn information criterion (HQIC)</td>
</tr>
</tbody>
</table>
Options

\textbf{Model}

\texttt{threshvar(varname)} specifies the variable from which values are to be selected as thresholds. \texttt{threshvar()} is required.

\texttt{regionvars(varlist)} specifies additional variables whose coefficients vary over the regions defined by the estimated thresholds. By default, only the constant term varies over regions.

c\texttt{oninvariant} specifies that the constant term should be region invariant instead of region varying.

\texttt{noconstant} suppresses the region-varying constant terms (intercepts) in the model.

\texttt{trim(#)} specifies that \texttt{threshold} treat the value at the \#th percentile of the threshold variable as the first possible threshold and the value at the \((100 - \#)\)th percentile as the last possible threshold. \# must be an integer between 1 and 49. The default is \texttt{trim(10)}.

\texttt{nthresholds(#)} specifies the number of thresholds. Specifying the number of thresholds is equivalent to specifying the number of regions because the number of regions is equal to \# + 1 thresholds. The default is \texttt{nthresholds(1)}, equivalent to 2 regions.

\texttt{optthresh(# [, ictype])} specifies that \texttt{threshold} choose the optimal number of thresholds, up to a possible \#. By default, the optimal number of thresholds is based on the BIC, but you may specify the information criterion (\texttt{ictype}) to be used. \texttt{ictype} may be \texttt{bic} (the default), \texttt{aic}, or \texttt{hqic}.

\textbf{SE/Robust}

\texttt{vce(vcetype)} specifies the type of standard error reported, which includes types that are derived from asymptotic theory (\texttt{oim}) and that are robust to some kinds of misspecification (\texttt{robust}); see [\texttt{R}] \texttt{vce \_option}.

\textbf{Reporting}

\texttt{level(#), nocnsreport}; see [\texttt{R}] Estimation options.

\textit{display \_options}: \texttt{noci}, \texttt{nopvalues}, \texttt{noomitted}, \texttt{vsquish}, \texttt{noemptycells}, \texttt{baselevels}, \texttt{allbaselevels}, \texttt{nofvlabel}, \texttt{fwrap(#)}, \texttt{fwrapon(style)}, \texttt{cformat(\%fmt)}, \texttt{pformat(\%fmt)}, \texttt{sformat(\%fmt)}, and \texttt{nolstretch}; see [\texttt{R}] Estimation options.

\texttt{nodots} and \texttt{dots(#)} specify whether to display replication dots. By default, one dot character is displayed for each successful replication. A red 'x' is displayed if \texttt{command} returns an error. You can also control whether dots are displayed using \texttt{set dots}; see [\texttt{R}] \texttt{set}.

\texttt{nodots} suppresses display of the replication dots.

\texttt{dots(#)} displays dots every \# replications. \texttt{dots(0)} is a synonym for \texttt{nodots}.

\textbf{Advanced}

\texttt{ssrs(stub* | newvarlist)} creates a variable containing the sum of squared residuals (SSRs) that was computed for each tentative threshold value during the search for the \(k\)th threshold. For observations where the value of the threshold variable specified in \texttt{threshvar()} is not a tentative threshold, the corresponding value of the variable created by \texttt{ssrs()} for that observation will be missing.

If you specify \texttt{stub*}, Stata will create \(k\) new variables with the names \texttt{stub1}, \ldots, \texttt{stubk}, which will contain the SSRs for the 1st, \ldots, \(k\)th thresholds, where \(k\) is the \# specified in \texttt{nthresholds()} or the optimal number of thresholds if \texttt{optthresh()} is specified.
If you specify a list of new variable names, you may request SSRs for up to the # specified in nthresholds(). If you specify optthresh(#) and the optimal number of thresholds is less than #, any additional variables will contain only missing values.

constraints(numlist) specifies the constraints by number after they have been defined by using the constraint command; see [R] constraint. constraints() may not be specified with optthresh()

The following option is available with threshold but is not shown in the dialog box:

coefflegend; see [R] Estimation options.

Remarks and examples

threshold extends linear regression to allow coefficients to differ across regions. Those regions are identified by a threshold variable being above or below a threshold value. The model may have multiple thresholds, and you can either specify a known number of thresholds or let threshold find that number for you by minimizing an information criterion.

These models are good alternatives to linear models for capturing abrupt breaks or asymmetries observed in most macroeconomic time series over the course of a business cycle. Common threshold regression models include the threshold autoregression model and self-exciting threshold model. In the threshold autoregression model, proposed by Tong (1983), the dependent variable is a function of its own lags; see Tong (1990) for details. In the self-exciting threshold model, the lagged dependent variable is used as the threshold variable. For a survey of threshold regression models in economics, refer to Hansen (2011).

Formally, consider a threshold regression with two regions defined by a threshold γ. This is written as

\[ y_t = x_t\beta + z_t\delta_1 + \epsilon_t \quad \text{if} \quad -\infty < w_t \leq \gamma \]
\[ y_t = x_t\beta + z_t\delta_2 + \epsilon_t \quad \text{if} \quad \gamma < w_t < \infty \]

where \( y_t \) is the dependent variable, \( x_t \) is a \( 1 \times k \) vector of covariates possibly containing lagged values of \( y_t \), \( \beta \) is a \( k \times 1 \) vector of region-invariant parameters, \( \epsilon_t \) is an IID error with mean 0 and variance \( \sigma^2 \), \( z_t \) is a vector of exogenous variables with region-specific coefficient vectors \( \delta_1 \) and \( \delta_2 \), and \( w_t \) is a threshold variable that may also be one of the variables in \( x_t \) or \( z_t \).

The parameters of interest are \( \beta \), \( \delta_1 \), and \( \delta_2 \). Region 1 is defined as the subset of observations in which the value of \( w_t \) is less than the threshold \( \gamma \). Similarly, Region 2 is defined as the subset of observations in which the value of \( w_t \) is greater than \( \gamma \). Inference on the nuisance parameter \( \gamma \) is complicated because of its nonstandard asymptotic distribution; see Hansen (1997, 2000).

threshold uses conditional least squares to estimate the parameters of the threshold regression model. The threshold value is estimated by minimizing the SSR obtained for all tentative thresholds; see Methods and Formulas for details.

Example 1: Threshold regression model

We are interested in the effect of inflation and the output gap on interest rates in a typical business cycle. Our dataset, usmacro.dta, contains quarterly data from 1954q3 to 2010q4 on the U.S. federal funds interest rate (fedfunds), the current inflation rate (inflation), and the output gap (ogap). These data were obtained from the Federal Reserve Economic Database, a macroeconomic database provided by the Federal Reserve Bank of Saint Louis; see [D] import fred.
In our model, we assume that the Federal Reserve sets the federal funds interest rate based on its most recent lag ($l.\text{fedfunds}$), the current inflation rate, and the output gap. We use the first lag of the federal funds interest rate as the threshold variable, and we assume one threshold, or two regions, so the model may be written as

\[
\text{fedfunds}_t = \delta_{10} + \delta_{11} l.\text{fedfunds} + \delta_{12} \text{inflation} + \delta_{13} \text{ogap} + \epsilon_t \text{ if } -\infty < l.\text{fedfunds} \leq \gamma
\]

\[
\text{fedfunds}_t = \delta_{20} + \delta_{21} l.\text{fedfunds} + \delta_{22} \text{inflation} + \delta_{23} \text{ogap} + \epsilon_t \text{ if } \gamma < l.\text{fedfunds} < \infty
\]

The output consists of two tables. The first table reports the estimated threshold and the corresponding SSR. The column labeled Order ranks the order in which the threshold was estimated. Because there is only a single threshold in this example, the order of 1 corresponds to the threshold value that contributes most in minimizing the SSR. The order is more relevant in the case of multiple thresholds.

The estimated threshold of 9.35% splits the sample into two regions. Region1 corresponds to the portion of the sample in which the federal funds interest rate from last quarter is less than or equal to 9.35%. Region2 corresponds to the portion of the sample in which the federal funds interest rate from last quarter is greater than 9.35%.
Coefficient estimates appear in the second table. In Region1, or the low federal funds interest rate region, the coefficient of 0.93 on the lag of fedfunds indicates that fedfunds is highly persistent. The coefficient on inflation is not significantly different from zero, which implies that the Federal Reserve does not attach any weight to the inflation rate in the low federal funds interest rate region and cares more about the output gap. In Region2, or the high federal funds interest rate region, the coefficient on the lag of fedfunds is only 0.70, which indicates that fedfunds is not as persistent as in Region1. In Region 2, the coefficient on ogap is not significantly different from zero, but the coefficient on inflation is, so we may infer that the Federal Reserve cares more about inflation than it does about the output gap.

Example 2: Selecting the threshold variable

In example 1, we use l.fedfunds as the threshold variable. The Federal Reserve may also consider the output gap to be an important factor that determines the interest rate. In this example, we fit models using the first and second lags of output gap as threshold variables. We store the estimates of each model for comparison using estimates store.

First, we store the estimates of example 1 as Model1.

. estimates store Model1

Next, we fit two models, one with l.ogap as the threshold variable and the other with l2.ogap as the threshold variable. We store the estimates as Model2 and Model3, respectively.

. threshold fedfunds, regionvars(l.fedfunds inflation ogap) threshvar(l.ogap)  
(output omitted)
. estimates store Model2

. threshold fedfunds, regionvars(l.fedfunds inflation ogap) threshvar(l2.ogap)  
(output omitted)
. estimates store Model3
We compare the SSR and information criteria of all fitted models. Combining all estimates, we get the following table:

\[
\begin{array}{l}
\text{Variable} & \text{Model1} & \text{Model2} & \text{Model3} \\
\text{Region1} \\
\text{fedfunds L1.} & 0.92689581 & 0.90860624 & 0.8533835 \\
\text{inflation} & 0.0602282 & 0.19755936 & 0.28187753 \\
\text{ogap} & 0.0990296 & 0.29553563 & 0.1449944 \\
\text{_cons} & 0.19662232 & 1.4172835 & 0.54280799 \\
\text{Region2} \\
\text{fedfunds L1.} & 0.69741126 & 0.90512493 & 0.90879685 \\
\text{inflation} & 0.16764486 & 0.0896271 & 0.08361366 \\
\text{ogap} & 0.05587384 & 0.15549667 & 0.15233276 \\
\text{_cons} & 2.1626095 & 1.4172835 & 0.54280799 \\
\text{Statistics} \\
\text{ssr} & 155.42663 & 145.96457 & 142.0608 \\
\text{aic} & -63.143795 & -77.087586 & -83.105746 \\
\text{bic} & -35.922376 & -49.866167 & -55.884327 \\
\text{hqic} & -52.153481 & -66.097272 & -72.115432 \\
\end{array}
\]

From the table above, we see that Model3 provides the best fit. This is the model that uses the second lag of output gap as the threshold variable.

Example 3: Selecting the number of thresholds

Instead of assuming a known number of thresholds, we can use model selection to choose the number of thresholds that minimizes a certain information criterion. In example 2, using 12.ogap as the threshold variable provided the best fit. We fit a model with an unknown number of thresholds using 12.ogap as the threshold variable. We can do this by specifying the maximum number of thresholds in the optthresh() option. In this example, we specify 5 as the maximum number of thresholds.
. threshold fedfunds, regionvars(l.fedfunds inflation ogap) threshvar(l2.ogap) > optthresh(5) nodots

Searching for threshold: 1
(Running 177 regressions)
Searching for threshold: 2
(Running 146 regressions)
Searching for threshold: 3
(Running 105 regressions)
Searching for threshold: 4
(Running 52 regressions)
Searching for threshold: 5
(Running 40 regressions)

Threshold regression

Full sample: 1955q3 - 2010q4
Number of obs = 222
Number of thresholds = 2
Max thresholds = 5
Threshold variable: L2.ogap
BIC = -60.0780

<table>
<thead>
<tr>
<th>Order</th>
<th>Threshold</th>
<th>SSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-3.1787</td>
<td>142.0608</td>
</tr>
<tr>
<td>2</td>
<td>-0.5351</td>
<td>126.4718</td>
</tr>
</tbody>
</table>

| fedfunds | Coef. | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|----------|-------|-----------|-------|-------|----------------------|
| Region1  | fedfunds | L1.    | .8533835 | .0435617 | 19.59 | 0.000 | .7680042 | .9387628 |
|          | inflation |       | .2818775 | .0679414 | 4.15 | 0.000 | .1487148 | .4150403 |
|          | ogap     |       | .1444994 | .072028  | 2.01 | 0.045 | .0033272 | .2856717 |
|          | _cons    |       | .542808  | .4297171 | 1.26 | 0.207 | -.299422 | 1.385038 |
| Region2  | fedfunds | L1.    | .9406721 | .0338085 | 27.82 | 0.000 | .8744087 | 1.006935 |
|          | inflation |       | -.0191805 | .0462729 | -0.41 | 0.679 | -.1098737 | .0715128 |
|          | ogap     |       | .2387934 | .0565521 | 4.22 | 0.000 | .1279534 | .3496334 |
|          | _cons    |       | .638354  | .1591717 | 4.01 | 0.000 | .3263832 | .9503249 |
| Region3  | fedfunds | L1.    | .8892742 | .0593484 | 14.98 | 0.000 | .7729535 | 1.005595 |
|          | inflation |       | .1851127 | .0532112 | 3.48 | 0.001 | .0808206 | .2894047 |
|          | ogap     |       | .1984744 | .039236  | 5.06 | 0.000 | .1215733 | .2753754 |
|          | _cons    |       | -.3086232 | .2215645 | -1.39 | 0.164 | -.7428817 | .1256352 |

We estimate two thresholds using the default BIC (bic). The first estimated threshold is 12.ogap = -3.18. A negative value of 12.ogap implies low economic growth two quarters ago. The second estimated threshold is -0.54 and also represents a negative output gap, although with a smaller magnitude. The two thresholds split the sample into three regions.

In the first region, Region1, the second lag of output gap is less than or equal to -3.18, indicating a recession period. In this case, the coefficients on inflation and ogap are both significantly different from zero, which implies that the Federal Reserve considers the current inflation rate and the output gap as important predictors of federal funds interest rate.
In the second region, Region2, the second lag of output gap is between $-3.18$ and $-0.54$. In this case, only the coefficient on output gap is significantly different from zero, which implies that the Federal Reserve only considers the output gap as a predictor of federal funds interest rate.

In the third region, Region3, the second lag of output gap is greater than $-0.54$. In this case, the coefficients on current inflation rate and output gap are both significantly different from zero, which implies that the Federal Reserve considers the current inflation rate and the output gap as important predictors of federal funds interest rate.

**Stored results**

`threshold` stores the following in `e()`:

Scalars
- `e(n)`: number of observations
- `e(k)`: number of parameters
- `e(nthresholds)`: number of thresholds
- `e(optthresh)`: number of maximum thresholds; if specified
- `e(ssr)`: sum of squared residuals of the model
- `e(rank)`: rank of `e(V)`
- `e(aic)`: Akaike information criterion
- `e(bic)`: Bayesian information criterion
- `e(hqic)`: Hannan–Quinn information criterion
- `e(tmin)`: minimum time
- `e(tmax)`: maximum time

Macros
- `e(cmd)`: `threshold`
- `e(cmdline)`: command as typed
- `e(eqnames)`: names of equations
- `e(depvar)`: name of dependent variable
- `e(threshvar)`: name of the threshold variable
- `e(criteria)`: information criteria if `optthresh(#, ic)type)` is specified
- `e(title)`: title in estimation output
- `e(tsfmt)`: format for the current time variable
- `e(tmins)`: formatted minimum time
- `e(tmaxs)`: formatted maximum time
- `e(vce)`: `vcetype` specified in `vce()`
- `e(vcetype)`: title used to label Std. Err.
- `e(properties)`: `b V`
- `e(predict)`: program used to implement `predict`
- `e(marginsnotok)`: predictions disallowed by `margins`
- `e(asbalanced)`: factor variables `fvset` as `asbalanced`
- `e(asobserved)`: factor variables `fvset` as `asobserved`

Matrices
- `e(b)`: coefficient vector
- `e(Cns)`: constraints matrix
- `e(V)`: variance–covariance matrix of the estimators
- `e(ssrmat)`: sum of squared residuals for each estimated threshold
- `e(thresholds)`: matrix of estimated thresholds
- `e(nobs)`: number of observations in each region

Functions
- `e(sample)`: marks estimation sample
Methods and formulas

Methods and formulas are presented under the following headings:

Introduction
Model with more than two regions

Introduction

Consider a threshold regression with two regions defined by a threshold $\gamma$. This is written as

$$
y_t = x_t \beta + z_t \delta_1 + \epsilon_t \quad \text{if} \quad -\infty < w_t \leq \gamma
$$

$$
y_t = x_t \beta + z_t \delta_2 + \epsilon_t \quad \text{if} \quad \gamma < w_t < \infty
$$

where $y_t$ is the dependent variable, $x_t$ is a $1 \times k$ vector of covariates possibly containing lagged values of $y_t$, $\beta$ is a $k \times 1$ vector of region-invariant parameters, $z_t$ is a vector of exogenous variables with region-specific coefficient vectors $\delta_1$ and $\delta_2$, $w_t$ is a threshold variable that may also be one of the variables in $x_t$ or $z_t$, and $\epsilon_t$ is an IID error with mean 0 and variance $\sigma^2$.

The estimated threshold ($\hat{\gamma}$) is one of the values in the threshold variable $w_t$. To estimate the threshold, we minimize the least squares of the following regression with $T$ observations and two regions,

$$
y_t = x_t \beta + z_t \delta_1 I(-\infty < w_t \leq \gamma) + z_t \delta_2 I(\gamma < w_t < \infty) + \epsilon_t
$$

for a sequence of $T_1$ values in $w_t$, where $T_1 < T$. The default trimming percentage is set to 10%, which implies that $T_1$ corresponds to the number of observations between the 10th and the 90th percentile of $w_t$. The estimator for the threshold is

$$
\hat{\gamma} = \arg \min_{\gamma \in \Gamma} S_{T_1}(\gamma)
$$

where $\Gamma = (-\infty, \infty)$,

$$
S_{T_1}(\gamma) = \sum_{t=1}^{T} \{y_t - x_t \beta - z_t \delta_1 I(-\infty < w_t \leq \gamma) - z_t \delta_2 I(\gamma < w_t < \infty)\}^2
$$

is a $T_1 \times 1$ vector of SSR, and $\gamma$ is a $T_1 \times 1$ vector of tentative thresholds.

Model with more than two regions

In general, a threshold regression model with $m$ thresholds has $m+1$ regions. Let $j = 1, \ldots, m+1$ index the regions. We can write the model as

$$
y_t = x_t \beta + z_t \delta_1 I_1(\gamma_1, w_t) + \cdots + z_t \delta_{m+1} I_{m+1}(\gamma_{m+1}, w_t) + \epsilon_t
$$

$$
y_t = x_t \beta + \sum_{j=1}^{m+1} z_t \delta_j I_j(\gamma_j, w_t) + \epsilon_t
$$

where $\gamma_1 < \gamma_2 < \cdots < \gamma_m$ are ordered thresholds with $\gamma_0 = -\infty$ and $\gamma_{m+1} = \infty$. $I_j(\gamma_j, w_t) = I(\gamma_{j-1} < w_t \leq \gamma_j)$ is an indicator for the $j$th region. Conditional on all estimated thresholds ($\hat{\gamma}_1, \ldots, \hat{\gamma}_m$), the threshold regression model is linear, and the remaining parameters are estimated using least squares.
The thresholds are estimated sequentially as described below. Let $\gamma_1^*, \ldots, \gamma_m^*$ represent the $m$ thresholds in the order of estimation. Gonzalo and Pitarakis (2002) show that the thresholds estimated sequentially are $T$ consistent. The first threshold $(\gamma_1^*)$ is estimated assuming a model with two regions as described in the previous section. Conditional on the first threshold, the second threshold is estimated as the value that yields the minimum sum of squared errors over all observations in $w_t$ excluding the first threshold. The estimator of the second threshold $\gamma_2^*$ is obtained by minimizing the least squares of a regression with three regions conditional on the first estimated threshold $\hat{\gamma}_1^*$. The estimator is given by
\[
\hat{\gamma}_2^* = \arg \min_{\gamma_2^* \in \Gamma_2} S_{T_2}(\gamma_2^* | \hat{\gamma}_1^*)
\]
where $\Gamma_2 = (\gamma_0, \hat{\gamma}_1^*) \cup (\hat{\gamma}_1^*, \gamma_3)$ and $T_2 < T_1$.

In general, the $l$th threshold minimizes the SSR conditional on the $l - 1$ estimated thresholds and is given by
\[
\hat{\gamma}_l^* = \arg \min_{\gamma_l^* \in \Gamma_l} S_{T_l}(\gamma_l^* | \hat{\gamma}_1^*, \ldots, \hat{\gamma}_{l-1}^*)
\]
where $\Gamma_l = (\gamma_0, \gamma_{m+1})$ excluding $\hat{\gamma}_1^*, \ldots, \hat{\gamma}_{l-1}^*$.

When the number of thresholds is not known a priori, threshold selects the optimal number of thresholds based on AIC, BIC, or HQIC, which is defined based on SSR from the fitted model as
\[
\begin{align*}
\text{AIC} &= T \ln(\text{SSR}/T) + 2k \\
\text{BIC} &= T \ln(\text{SSR}/T) + k \ln(T) \\
\text{HQIC} &= T \ln(\text{SSR}/T) + 2k \ln\{ \ln(T) \}
\end{align*}
\]
where $k$ is the number of parameters in the model. See Gonzalo and Pitarakis (2002) for Monte Carlo studies of selecting the number of thresholds based on information criteria.

References


Also see

- [TS] threshold postestimation — Postestimation tools for threshold
- [TS] mswitch — Markov-switching regression models
- [TS] tsset — Declare data to be time-series data
- [R] regress — Linear regression
- [U] 20 Estimation and postestimation commands
Postestimation commands

The following standard postestimation commands are available after `threshold`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions, standard errors, and residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

\[ \text{predict} \ [ \text{type} \] \text{newvar} \ [ \text{if} \] \ [ \text{in} \] \ [ , \text{statistic} \ \text{dynamic}(\text{time}\_\text{constant})] \]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>standard error of the linear prediction</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for the estimation sample.

Options for predict

\underline{Main}

xb, the default, calculates the linear prediction.

stdp calculates the standard error of the linear prediction.

residuals calculates the residuals in the equations for observable variables.

\underline{Options}

dynamic(\text{time}\_\text{constant}) specifies that predict begin producing dynamic forecasts at \text{time}\_\text{constant}, which must be in the sample for which observations on the dependent variable exist and given in the scale of the time variable specified in tsset. For example, dynamic(tq(2014q4)) causes dynamic predictions to begin in the fourth quarter of 2014, assuming that the time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. dynamic() may not be specified with stdp or residuals.

Remarks and examples

We assume that you have already read [TS] threshold. In this entry, we illustrate some of the features of predict after using threshold to estimate the parameters of a threshold regression model.
Example 1: One-step ahead predictions

In example 3 of [TS] threshold, we selected the threshold autoregression model where the federal funds interest rate (fedfunds) is a function of its own first lag, inflation (inflation), and output gap (ogap). We estimated two thresholds using the Bayesian information criterion and 12.ogap as the threshold variable. We refit that model here, but we directly specify two thresholds by using the nthresholds(2) option. We obtain the one-step ahead predictions for the dependent variable using the default settings for predict. The predictions are stored in the new variable fedf.

```
. use https://www.stata-press.com/data/r16/usmacro
   (Federal Reserve Economic Data - St. Louis Fed)
. threshold fedfunds, regionvars(l.fedfunds inflation ogap) threshvar(l2.ogap)
   > nthresholds(2)
     (output omitted)
. predict fedf
   (option xb assumed; predicted values)
```

Next, we graph the actual values (fedfunds) and predicted values (fedf) using tsline. We change the label for fedf to “Predicted values”; see [TS] tsline.

```
. tsline fedfunds fedf, legend(label(2 "Predicted values"))
```

The one-step ahead predicted values follow the actual federal funds interest rate closely.

Example 2: Dynamic predictions

Continuing the example above, we may instead want to obtain dynamic predictions and compare them with the actual values. Within dynamic(), we specify that dynamic predictions will begin in the first quarter of 2003. Our data were tsset as quarterly data, so we use the function tq() to convert 2003q1 into a numeric date that Stata understands; see [FN] Date and time functions.

```
. predict fedfdyn, dynamic(tq(2003q1))
   (option xb assumed; predicted values)
```

We again use tsline to plot the actual data and dynamic predictions. We use the tline() option to add a vertical line that shows the beginning date of our dynamic predictions and restrict the range to quarters 2002q1 to the end of the sample using tin().
The dynamic forecast does well tracking the increase in the federal funds interest rate beginning 2004q3 and the subsequent decline around 2007q1.

Also see

[TS] `threshold` — Threshold regression

[U] 20 Estimation and postestimation commands
tsappend — Add observations to a time-series dataset

Description

`tsappend` appends observations to a time-series dataset or to a panel dataset. `tsappend` uses and updates the information set by `tsset` or `xtset`. Any gaps in the dataset are removed.

Quick start

Add 10 time periods to `tsset` data

```
tsappend, add(10)
```

Incorporate additional months to data up to the third month of 1999

```
tsappend, last(1999m3) tsfmt(tm)
```

Add 2 time periods to the panel identified by `pvar = 333` after `xtset pvar tvar`

```
tsappend, add(2) panel(333)
```

Menu

Statistics > Time series > Setup and utilities > Add observations to time-series dataset
tsappend — Add observations to a time-series dataset

Syntax

```
  tsappend, { add(#) | last(date | clock) tsfmt(string) } [ options ]
```

Options

- `add(#)` specifies the number of observations to add.
- `last(date | clock)` and `tsfmt(string)` must be specified together and are an alternative to `add()`.
- `tsfmt(string)` specifies the Stata time-series function to use in converting the date specified in `last()` to an integer. The function names are tc (clock), tC (Clock), td (daily), tw (weekly), tm (monthly), tq (quarterly), and th (half-yearly).
- `panel(panel_id)` specifies that observations be added only to panels with the ID specified in `panel()`.

Remarks and examples

Remarks are presented under the following headings:

- **Introduction**
- Using tsappend with time-series data
- Using tsappend with panel data

Introduction

`tsappend` adds observations to a time-series dataset or to a panel dataset. You must `tsset` or `xtset` your data before using `tsappend`. `tsappend` simultaneously removes any gaps from the dataset.

There are two ways to use `tsappend`: you can specify the `add(#)` option to request that # observations be added, or you can specify the `last(date | clock)` option to request that observations be appended until the date specified is reached. If you specify `last()`, you must also specify `tsfmt()`. `tsfmt()` specifies the Stata time-series date function that converts the date held in `last()` to an integer.

`tsappend` works with time series of panel data. With panel data, `tsappend` adds the requested observations to all the panels, unless the `panel()` option is also specified.
Using `tsappend` with time-series data

`tsappend` can be useful for appending observations when dynamically predicting a time series. Consider an example in which `tsappend` adds the extra observations before dynamically predicting from an AR(1) regression:

```
. use https://www.stata-press.com/data/r16/tsappend1
. regress y l.y
```

```
Source |        SS       df       MS  Number of obs =  479
-------+------------------------------  F(1, 477) = 119.29
Model  |  115.349555     1  115.349555  Prob > F    = 0.0000
Residual |  461.241577    477  .966963473  R-squared    = 0.2001
         |  576.591132    478  1.2062576  Adj R-squared    = 0.1984
```

```
|     | Coef. | Std. Err. | t    | P>|t|  | [95% Conf. Interval] |
|-----|-------|-----------|------|------|----------------------|
| y   |       |           |      |      |                      |
| L1. |  .4493507 |  .0411417 | 10.92 | 0.000 | .3685093       | .5301921 |
| _cons |  11.11877 |  .8314581 | 13.37 | 0.000 | 9.484993       | 12.75254 |
```

```
. matrix b = e(b)
. matrix colnames b = L.xb one
. tsset
time variable:  t2, 1960m2 to 2000m1
   delta:  1 month
. tsappend, add(12)
. tsset
time variable:  t2, 1960m2 to 2001m1
   delta:  1 month
. predict xb if t2<=tm(2000m2)
   (option xb assumed; fitted values)
(12 missing values generated)
. generate one=1
. matrix score xb=b if t2>=tm(2000m2), replace
```

The calls to `tsset` before and after `tsappend` were made without a time variable; thus both commands display how the data are currently `tsset`. The results from the first `tsset` command show that we have monthly data and that our time variable, `t2`, starts at 1960m2 and ends at 2000m1.

`tsappend` with the `add(12)` option used these results to add 12 months to the dataset. The results of the second `tsset` command show that this new year of data has been added, as shown by the end year now being 2001m1. We could have skipped these calls to `tsset`, but they are shown here to illustrate how `tsappend` uses and updates time-series settings of the dataset.

We then used `predict` and `matrix score` to obtain the dynamic predictions, which allows us to produce the following graph:
In the call to `tsappend`, instead of saying that we wanted to add 12 observations, we could have specified that we wanted to fill in observations through the first month of 2001:

```
. use https://www.stata-press.com/data/r16/tsappend1, clear
. tsset
time variable: t2, 1960m2 to 2000m1
delta:  1 month
. tsappend, last(2001m1) tsfmt(tm)
. tsset
time variable: t2, 1960m2 to 2001m1
delta:  1 month
```

We specified the `tm()` function in the `tsfmt()` option. [FN] Date and time functions contains a list of time-series functions for translating date literals to integers. Because we have monthly data, and since [FN] Date and time functions tells us that we want to use the `tm()` function, we specified the `tsfmt(tm)` option. The following table shows the most common types of time-series data, their formats, the appropriate translation functions, and the corresponding options for `tsappend`:

<table>
<thead>
<tr>
<th>Description</th>
<th>Format</th>
<th>Function</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>%tc</td>
<td>tc()</td>
<td>tsfmt(tc)</td>
</tr>
<tr>
<td>time</td>
<td>%tC</td>
<td>tC()</td>
<td>tsfmt(tC)</td>
</tr>
<tr>
<td>daily</td>
<td>%td</td>
<td>td()</td>
<td>tsfmt(td)</td>
</tr>
<tr>
<td>weekly</td>
<td>%tw</td>
<td>tw()</td>
<td>tsfmt(tw)</td>
</tr>
<tr>
<td>monthly</td>
<td>%tm</td>
<td>tm()</td>
<td>tsfmt(tm)</td>
</tr>
<tr>
<td>quarterly</td>
<td>%tq</td>
<td>tq()</td>
<td>tsfmt(tq)</td>
</tr>
<tr>
<td>half-yearly</td>
<td>%th</td>
<td>th()</td>
<td>tsfmt(th)</td>
</tr>
<tr>
<td>yearly</td>
<td>%ty</td>
<td>ty()</td>
<td>tsfmt(ty)</td>
</tr>
</tbody>
</table>
Using tsappend with panel data

`tsappend`'s actions on panel data are similar to its action on time-series data, except that `tsappend` performs those actions on each time series within the panels. To work within panels, a panel variable must have been specified with `tsset` or `xtset`. It does not matter which command you use; the two are equivalent.

If the end dates vary over panels, `last()` and `add()` will produce different results. `add(#)` always adds # observations to each panel. If the data end at different periods before `tsappend`, `add()` is used, the data will still end at different periods after `tsappend`, `add()`. In contrast, `tsappend`, `last() tsfmt()` will cause all the panels to end on the specified last date. If the beginning dates differ across panels, using `tsappend`, `last() tsfmt()` to provide a uniform ending date will not create balanced panels because the number of observations per panel will still differ.

Consider the panel data summarized in the output below:

```
. use https://www.stata-press.com/data/r16/tsappend3, clear
. xtdescribe
id: 1, 2, ..., 3
   n = 3
  t2: 1998m1, 1998m2, ..., 2000m1
      Delta(t2) = 1 month
      Span(t2) = 25 periods
      (id*t2 uniquely identifies each observation)
Distribution of T_i:  min  5%  25%  50%  75%  95%  max
                      13   13   13    20   24   24   24
Freq. Percent Cum. Pattern
       1  33.33   33.33
       1  33.33  66.67
       1  33.33 100.00
 13   33.33   33.33
13   33.33 100.00
```

The output from `xtdescribe` and `summarize` on these data tells us that one panel starts later than the other, that another panel ends before the other two, and that the remaining panel has a gap in the time variable but otherwise spans the entire time frame.
Now consider the data after a call to `tsappend, add(6)`:

```
.tsappend, add(6)
.xtdescribe
  id: 1, 2, ..., 3 n = 3
t2: 1998m1, 1998m2, ..., 2000m7 T = 31
  Delta(t2) = 1 month
  Span(t2) = 31 periods
  (id*t2 uniquely identifies each observation)
Distribution of T_i: min 5% 25% 50% 75% 95% max
  19 19 19 26 31 31 31
  Freq. Percent Cum. Pattern
    1 33.33 33.33 ................11111111111111111111
    1 33.33 66.67 1111111111111111111111111111
    1 33.33 100.00 1111111111111111111111111111111

  3 100.00 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
.xtdescribe
  .by id: summarize t2
d-> id = 1
  Variable | Obs  Mean   Std. Dev.   Min   Max
  t2       | 19  477.0  5.627314  468  486
```

```
d-> id = 2
  Variable | Obs  Mean   Std. Dev.   Min   Max
  t2       | 26  468.5  7.648529  456  481
```

```
d-> id = 3
  Variable | Obs  Mean   Std. Dev.   Min   Max
  t2       | 31  471.0  9.092121  456  486
```

This output from `xtdescribe` and `summarize` after the call to `tsappend` shows that the call to `tsappend, add(6)` added 6 observations to each panel and filled in the gap in the time variable in the second panel. `tsappend, add()` did not cause a uniform end date over the panels.

The following output illustrates the contrast between `tsappend, add()` and `tsappend, last() tsfmt()` with panel data that end at different dates. The output from `xtdescribe` and `summarize` shows that the call to `tsappend, last() tsfmt()` filled in the gap in `t2` and caused all the panels to end at the specified end date. The output also shows that the panels remain unbalanced because one panel has a later entry date than the other two.
. use https://www.stata-press.com/data/r16/tsappend2, clear  
. tsappend, last(2000m7) tsfmt(tm)  
. xtdescribe  

    id:  1, 2, ..., 3    n = 3  
    t2:  1998m1, 1998m2, ..., 2000m7    T = 31  
     Delta(t2) = 1 month  
Span(t2) = 31 periods  
(id*t2 uniquely identifies each observation)  

<table>
<thead>
<tr>
<th>Distribution of T_{i}:</th>
<th>min</th>
<th>5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>95%</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>66.67</td>
<td>66.67</td>
<td>11111111111111111111111111111111</td>
</tr>
<tr>
<td>1</td>
<td>33.33</td>
<td>100.00</td>
<td>............11111111111111111</td>
</tr>
<tr>
<td>3</td>
<td>100.00</td>
<td></td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
</tbody>
</table>

. by id: summarize t2  

--> id = 1  

<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>19</td>
<td>477</td>
<td>5.627314</td>
<td>468</td>
<td>486</td>
</tr>
</tbody>
</table>

--> id = 2  

<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>31</td>
<td>471</td>
<td>9.092121</td>
<td>456</td>
<td>486</td>
</tr>
</tbody>
</table>

--> id = 3  

<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>31</td>
<td>471</td>
<td>9.092121</td>
<td>456</td>
<td>486</td>
</tr>
</tbody>
</table>

**Stored results**  

`tsappend` stores the following in `r()`:  

Scalars  
`r(add)` number of observations added

**Also see**  

[TS] `tsset` — Declare data to be time-series data  
[XT] `xtset` — Declare data to be panel data
Title

**tsfill** — Fill in gaps in time variable

<table>
<thead>
<tr>
<th>Description</th>
<th>Quick start</th>
<th>Menu</th>
<th>Syntax</th>
</tr>
</thead>
</table>

**Description**

tsfill is used to fill in gaps in time-series data and gaps in panel data with new observations, which contain missing values. tsfill is not needed to obtain correct lags, leads, and differences when gaps exist in a series because Stata’s time-series operators handle gaps automatically.

**Quick start**

Add new observations with missing values for missing time periods in a time-series dataset that has been `tsset`

```
tsfill
```

Add new observations with missing values to eliminate gaps in a panel dataset that has been `xtset`

```
tsfill
```

As above, but making the panel strongly balanced

```
tsfill, full
```

**Menu**

Statistics > Time series > Setup and utilities > Fill in gaps in time variable
Syntax

tsfill [ , full ]

You must tsset or xtset your data before using tsfill; see [TS] tsset and [XT] xtset.

Option

full is for use with panel data only. With panel data, tsfill by default fills in observations for each panel according to the minimum and maximum values of timevar for the panel. Thus if the first panel spanned the times 5–20 and the second panel the times 1–15, after tsfill they would still span the same periods; observations would be created to fill in any missing times from 5–20 in the first panel and from 1–15 in the second.

If full is specified, observations are created so that both panels span the time 1–20, the overall minimum and maximum of timevar across panels.

Remarks and examples

Remarks are presented under the following headings:

Introduction
Using tsfill with time-series data
Using tsfill with panel data
Video example

Introduction

tsfill is used after tsset or xtset to fill gaps in time-series data and gaps in panel data with new observations. Each new observation contains the appropriate values of the time variable, timevar, and, when specified, the panel variable, panelvar, and missing values for all other variables in the dataset. For instance, perhaps observations for timevar = 1, 3, 5, 6, ..., 22 exist. tsfill would create observations for timevar = 2 and timevar = 4 containing all missing values.

tsfill is intended as an intermediate step in a data management process. For example, you may wish to use tsfill with time-series data if you plan to interpolate missing values or with panel data if you intend to impute missing values.

You do not need to use tsfill to correctly create variables with lags, leads, and differencing, because Stata’s time-series operators handle gaps in the series for you; see [U] 11.4.4 Time-series varlists. These operators consider timevar, not the observation number. For example, suppose we have data on GNP in the years 1989–1991 and 1993–1995. Referring to L.gnp to obtain lagged gnp values would correctly produce a missing value of lagged gnp for timevar = 1989 and timevar = 1993 even if missing values were not explicitly created using tsfill.
Using tsfill with time-series data

You have monthly data, with gaps:

. use https://www.stata-press.com/data/r16/tsfillxmpl

.time variable: mdate, 1995m7 to 1996m3, but with gaps
.delta: 1 month

.list mdate income

<table>
<thead>
<tr>
<th>mdate</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>1995m11</td>
<td>1236</td>
</tr>
<tr>
<td>1995m12</td>
<td>1297</td>
</tr>
<tr>
<td>1996m1</td>
<td>1265</td>
</tr>
<tr>
<td>1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>

You can fill in the gaps by interpolation easily with tsfill and ipolate. tsfill creates the missing observations:

.tsfill

.list mdate income

<table>
<thead>
<tr>
<th>mdate</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>1995m9</td>
<td>.</td>
</tr>
<tr>
<td>1995m10</td>
<td>.</td>
</tr>
<tr>
<td>1995m11</td>
<td>1236</td>
</tr>
<tr>
<td>1995m12</td>
<td>1297</td>
</tr>
<tr>
<td>1996m1</td>
<td>1265</td>
</tr>
<tr>
<td>1996m2</td>
<td>.</td>
</tr>
<tr>
<td>1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>

We can now use ipolate (see [D] ipolate) to fill them in:

.ipolate income mdate, gen(ipinc)

.list mdate income ipinc

<table>
<thead>
<tr>
<th>mdate</th>
<th>income</th>
<th>ipinc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995m7</td>
<td>1153</td>
<td>1153</td>
</tr>
<tr>
<td>1995m8</td>
<td>1181</td>
<td>1181</td>
</tr>
<tr>
<td>1995m9</td>
<td>.</td>
<td>1199.3333</td>
</tr>
<tr>
<td>1995m10</td>
<td>.</td>
<td>1217.6667</td>
</tr>
<tr>
<td>1995m11</td>
<td>1236</td>
<td>1236</td>
</tr>
<tr>
<td>1995m12</td>
<td>1297</td>
<td>1297</td>
</tr>
<tr>
<td>1996m1</td>
<td>1265</td>
<td>1265</td>
</tr>
<tr>
<td>1996m2</td>
<td>.</td>
<td>1273.5</td>
</tr>
<tr>
<td>1996m3</td>
<td>1282</td>
<td>1282</td>
</tr>
</tbody>
</table>
Using tsfill with panel data

You have the following panel dataset:

```
. use https://www.stata-press.com/data/r16/tsfillxmpl2, clear
. tsset
   panel variable: edlevel (unbalanced)
   time variable: year, 1988 to 1992, but with a gap
      delta: 1 unit
. list edlevel year income
   edlevel year income
     1. 1 1988 14500
     2. 1 1989 14750
     3. 1 1990 14950
     4. 1 1991 15100
     5. 2 1989 22100
     6. 2 1990 22200
     7. 2 1992 22800
```

Just as with nonpanel time-series datasets, you can use `tsfill` to fill in the gaps within each panel:

```
. tsfill
. list edlevel year income
   edlevel year income
     1. 1 1988 14500
     2. 1 1989 14750
     3. 1 1990 14950
     4. 1 1991 15100
     5. 2 1989 22100
     6. 2 1990 22200
     7. 2 1991 . ← new
     8. 2 1992 22800
```

You could instead use `tsfill` to produce fully balanced panels with the `full` option:

```
. tsfill, full
. list edlevel year income, sep(0)
   edlevel year income
     1. 1 1988 14500
     2. 1 1989 14750
     3. 1 1990 14950
     4. 1 1991 15100
     5. 1 1992 . ← new
     6. 2 1988 . ← new
     7. 2 1989 22100
     8. 2 1990 22200
     9. 2 1991 . ← new
    10. 2 1992 22800
```
Video example

Formatting and managing dates

Also see

[TS] tsappend — Add observations to a time-series dataset
[TS] tsset — Declare data to be time-series data
[XT] xtset — Declare data to be panel data
tsfilter — Filter a time series for cyclical components

Description

*tsfilter* separates a time series into trend and cyclical components. The trend component may contain a deterministic or a stochastic trend. The stationary cyclical component is driven by stochastic cycles at the specified periods.

Syntax

Filter one variable

```
*tsfilter* filter [type] newvar = varname [if] [in] [ , options]
```

Filter multiple variables, unique names

```
*tsfilter* filter [type] newvarlist = varlist [if] [in] [ , options]
```

Filter multiple variables, common name stub

```
*tsfilter* filter [type] stub* = varlist [if] [in] [ , options]
```

<table>
<thead>
<tr>
<th>filter</th>
<th>Name</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>bk</td>
<td>Baxter–King</td>
<td>[TS] tsfilter bk</td>
</tr>
<tr>
<td>bw</td>
<td>Butterworth</td>
<td>[TS] tsfilter bw</td>
</tr>
<tr>
<td>cf</td>
<td>Christiano–Fitzgerald</td>
<td>[TS] tsfilter cf</td>
</tr>
<tr>
<td>hp</td>
<td>Hodrick–Prescott</td>
<td>[TS] tsfilter hp</td>
</tr>
</tbody>
</table>

You must *tsset* or *xtset* your data before using *tsfilter*; see [TS] tsset and [XT] xtset. *varname* and *varlist* may contain time-series operators; see [U] 11.4.4 Time-series varlists. *options* differ across the filters and are documented in each *filter*'s manual entry.

Remarks and examples

The time-series filters implemented in *tsfilter* separate a time-series $y_t$ into trend and cyclical components:

$$y_t = \tau_t + c_t$$

where $\tau_t$ is the trend component and $c_t$ is the cyclical component. $\tau_t$ may be nonstationary; it may contain a deterministic or a stochastic trend, as discussed below.

The primary objective of the methods implemented in *tsfilter* is to estimate $c_t$, a stationary cyclical component that is driven by stochastic cycles within a specified range of periods. The trend component $\tau_t$ is calculated by the difference $\tau_t = y_t - c_t$. 
Although the filters implemented in \texttt{tsfilter} have been widely applied by macroeconomists, they are general time-series methods and may be of interest to other researchers.

Remarks are presented under the following headings:

\begin{itemize}
  \item An example dataset
  \item A baseline method: Symmetric moving-average (SMA) filters
  \item An overview of filtering in the frequency domain
  \item SMA revisited: The Baxter–King filter
  \item Filtering a random walk: The Christiano–Fitzgerald filter
  \item A one-parameter high-pass filter: The Hodrick–Prescott filter
  \item A two-parameter high-pass filter: The Butterworth filter
\end{itemize}

### An example dataset

Time series are frequently filtered to remove unwanted characteristics, such as trends and seasonal components, or to estimate components driven by stochastic cycles from a specific range of periods. Although the filters implemented in \texttt{tsfilter} can be used for both purposes, their primary purpose is the latter, and we restrict our discussion to that use.

We explain the methods implemented in \texttt{tsfilter} by estimating the business-cycle component of a macroeconomic variable, because they are frequently used for this purpose. We estimate the business-cycle component of the natural log of an index of the industrial production of the United States, which is plotted below.

#### Example 1: A trending time series

```stata
. use https://www.stata-press.com/data/r16/ipq
    (Federal Reserve Economic Data, St. Louis Fed)
. tsline ip_ln
```

The above graph shows that \texttt{ip\_ln} contains a trend component. Time series may contain deterministic trends or stochastic trends. A polynomial function of time is the most common deterministic time trend. An integrated process is the most common stochastic trend. An integrated process is a random variable that must be differenced one or more times to be stationary; see Hamilton (1994) for a discussion. The different filters implemented in \texttt{tsfilter} allow for different orders of deterministic time trends or integrated processes.
We now illustrate the four methods implemented in tsfilter, each of which will remove the trend and estimate the business-cycle component. Burns and Mitchell (1946) defined oscillations in business data with recurring periods between 1.5 and 8 years to be business-cycle fluctuations; we use their commonly accepted definition.

**A baseline method: Symmetric moving-average (SMA) filters**

Symmetric moving-average (SMA) filters form a baseline method for estimating a cyclical component because of their properties and simplicity. An SMA filter of a time series $y_t$, $t \in \{1, \ldots, T\}$, is the data transform defined by

$$y_t^* = \sum_{j=-q}^{q} \alpha_j y_{t-j}$$

for each $t \in \{q+1, \ldots, T-q\}$, where $\alpha_{-j} = \alpha_j$ for $j \in \{-q, \ldots, q\}$. Although the original series has $T$ observations, the filtered series has only $T - 2q$, where $q$ is known as the order of the SMA filter.

SMA filters with weights that sum to zero remove deterministic and stochastic trends of order 2 or less, as shown by Fuller (1996) and Baxter and King (1999).

**Example 2: A trend-removing SMA filter**

This trend-removal property of SMA filters with coefficients that sum to zero may surprise some readers. For illustration purposes, we filter ip\_ln by the filter

\[-0.2ip\_ln_{t-2} - 0.2ip\_ln_{t-1} + 0.8ip\_ln_t - 0.2ip\_ln_{t+1} - 0.2ip\_ln_{t+2}\]

and plot the filtered series. We do not even need tsfilter to implement this second-order SMA filter; we can use generate.

```
. generate ip_sma = -.2*L2.ip\_ln-.2*L.ip\_ln+.8*ip\_ln-.2*F.ip\_ln-.2*F2.ip\_ln
(4 missing values generated)
. tsline ip_sma
```

![Graph showing the filtered series](image)
The filter has removed the trend.

There is no good reason why we chose that particular SMA filter. Baxter and King (1999) derived a class of SMA filters with coefficients that sum to zero and get as close as possible to keeping only the specified cyclical component.

### An overview of filtering in the frequency domain

We need some concepts from the frequency-domain approach to time-series analysis to motivate how Baxter and King (1999) defined “as close as possible”. These concepts also motivate the other filters in tsfilter. The intuitive explanation presented here glosses over many technical details discussed by Priestley (1981), Hamilton (1994), Fuller (1996), and Wei (2006).

As with much time-series analysis, the basic results are for covariance-stationary processes with additional results handling some nonstationary cases. We present some useful results for covariance-stationary processes and discuss how to handle nonstationary series below.

The autocovariances \( \gamma_j, j \in \{0, 1, \ldots, \infty\} \), of a covariance-stationary process \( y_t \) specify its variance and dependence structure. In the frequency-domain approach to time-series analysis, \( y_t \) and the autocovariances are specified in terms of independent stochastic cycles that occur at frequencies \( \omega \in [-\pi, \pi] \). The spectral density function \( f_y(\omega) \) specifies the contribution of stochastic cycles at each frequency \( \omega \) relative to the variance of \( y_t \), which is denoted by \( \sigma_y^2 \). The variance and the autocovariances can be expressed as an integral of the spectral density function. Formally,

\[
\gamma_j = \int_{-\pi}^{\pi} e^{i\omega j} f_y(\omega) d\omega
\]  

(1)

where \( i \) is the imaginary number \( i = \sqrt{-1} \).

Equation (1) can be manipulated to show what fraction of the variance of \( y_t \) is attributable to stochastic cycles in a specified range of frequencies. Hamilton (1994, 156) discusses this point in more detail.

Equation (1) implies that if \( f_y(\omega) = 0 \) for \( \omega \in [\omega_1, \omega_2] \), then stochastic cycles at these frequencies contribute zero to the variance and autocovariances of \( y_t \).

The goal of time-series filters is to transform the original series into a new series \( y_t^* \) for which the spectral density function of the filtered series \( f_{y^*}(\omega) \) is zero for unwanted frequencies and equal to \( f_y(\omega) \) for desired frequencies.

A linear filter of \( y_t \) can be written as

\[
y_t^* = \sum_{j=-\infty}^{\infty} \alpha_j y_{t-j} = \alpha(L) y_t
\]

where we let \( y_t \) be an infinitely long series as required by some of the results below. To see the impact of the filter on the components of \( y_t \) at each frequency \( \omega \), we need an expression for \( f_{y^*}(\omega) \) in terms of \( f_y(\omega) \) and the filter weights \( \alpha_j \). Wei (2006, 282) shows that for each \( \omega \),

\[
f_{y^*}(\omega) = |\alpha(e^{i\omega})|^2 f_y(\omega)
\]  

(2)
where \(|\alpha(e^{i\omega})|\) is known as the gain of the filter. Equation (2) makes explicit that the squared gain function \(|\alpha(e^{i\omega})|^2\) converts the spectral density of the original series, \(f_y(\omega)\), into the spectral density of the filtered series, \(f_y^*(\omega)\). In particular, (2) says that for each frequency \(\omega\), the spectral density of the filtered series is the product of the square of the gain of the filter and the spectral density of the original series.

As we will see in the examples below, the gain function provides a crucial interpretation of what a filter is doing. We want a filter for which \(f_y^*(\omega) = 0\) for unwanted frequencies and for which \(f_y^*(\omega) = f_y(\omega)\) for desired frequencies. So we seek a filter for which the gain is 0 for unwanted frequencies and for which the gain is 1 for desired frequencies.

In practice, we cannot find such an ideal filter exactly, because the constraints an ideal filter places on filter coefficients cannot be satisfied for time series with only a finite number of observations. The expansive literature on filters is a result of the trade-offs involved in designing implementable filters that approximate the ideal filter.

Ideally, filters pass or block the stochastic cycles at specified frequencies by having a gain of 1 or 0. Band-pass filters, such as the Baxter–King (BK) and the Christiano–Fitzgerald (CF) filters, pass through stochastic cycles in the specified range of frequencies and block all the other stochastic cycles. High-pass filters, such as the Hodrick–Prescott (HP) and Butterworth filters, only allow the stochastic cycles at or above a specified frequency to pass through and block the lower-frequency stochastic cycles. For band-pass filters, let \([\omega_0, \omega_1]\) be the set of desired frequencies with all other frequencies being undesired. For high-pass filters, let \(\omega_0\) be the cutoff frequency with only those frequencies \(\omega \geq \omega_0\) being desired.

\section*{SMA revisited: The Baxter–King filter}

We now return to the class of SMA filters with coefficients that sum to zero and get as close as possible to keeping only the specified cyclical component as derived by Baxter and King (1999).

For an infinitely long series, there is an ideal band-pass filter for which the gain function is 1 for \(\omega \in [\omega_0, \omega_1]\) and 0 for all other frequencies. It just so happens that this ideal band-pass filter is an SMA filter with coefficients that sum to zero. Baxter and King (1999) derive the coefficients of this ideal band-pass filter and then define the BK filter to be the SMA filter with \(2q + 1\) terms that are as close as possible to those of the ideal filter. There is a trade-off in choosing \(q\): larger values of \(q\) cause the gain of the BK filter to be closer to the gain of the ideal filter, but larger values also increase the number of missing observations in the filtered series.

Although the mathematics of the frequency-domain approach to time-series analysis is in terms of stochastic cycles at frequencies \(\omega \in [-\pi, \pi]\), applied work is generally in terms of periods \(p\), where \(p = 2\pi/\omega\). So the options for the \texttt{tsfilter} subcommands are in terms of periods.

\section*{Example 3: A BK estimate of the business-cycle component}

Below we use \texttt{tsfilter bk}, which implements the BK filter, to estimate the business-cycle component composed of stochastic cycles between 6 and 32 periods, and then we graph the estimated component.
The above graph tells us what the estimated business-cycle component looks like, but it presents no evidence as to how well we have estimated the component. A periodogram is better for this purpose. A periodogram is an estimator of a transform of the spectral density function; see [TS] pergram for details. Below we plot the periodogram for the BK estimate of the business-cycle component. \texttt{pergram} displays the results in natural frequencies, which are the standard frequencies divided by $2\pi$. We use the \texttt{xline()} option to draw vertical lines at the lower natural-frequency cutoff ($1/32 = 0.03125$) and the upper natural-frequency cutoff ($1/6 \approx 0.16667$).

```
. pergram ip_bk, xline(0.03125 0.16667)
```

If the filter completely removed the stochastic cycles corresponding to the unwanted frequencies, the periodogram would be a flat line at the minimum value of $-6$ outside the range identified by the vertical lines. That the periodogram takes on values greater than $-6$ outside the specified range indicates the inability of the BK filter to pass through only stochastic cycles at frequencies inside the specified band.
We can also evaluate the BK filter by plotting its gain function against the gain function of an ideal filter. In the output below, we reestimate the business-cycle component to store the gain of the BK filter for the specified parameters. (The coefficients and the gain of the BK filter are completely determined by the specified minimum period, the maximum period, and the order of the SMA filter.) We label the variable bkgain for the graph below.

```
. drop ip_bk
. tsfilter bk ip_bk = ip_ln, minperiod(6) maxperiod(32) gain(bkgain abk)
. label variable bkgain "BK filter"
```

Below we generate ideal, the gain function of the ideal band-pass filter at the frequencies f. Then we plot the gain of the ideal filter and the gain of the BK filter.

```
. generate f = _pi*(_n-1)/_N
. generate ideal = cond(f<_pi/16, 0, cond(f<_pi/3, 1,0))
. label variable ideal "Ideal filter"
. twoway line ideal f || line bkgain abk
```

The graph reveals that the gain of the BK filter deviates markedly from the square-wave gain of the ideal filter. Increasing the symmetric moving average via the `smaorder()` option will cause the gain of the BK filter to more closely approximate the gain of the ideal filter at the cost of lost observations in the filtered series.

Filtering a random walk: The Christiano–Fitzgerald filter

Although Baxter and King (1999) minimized the error between the coefficients in their filter and the ideal band-pass filter, Christiano and Fitzgerald (2003) minimized the mean squared error between the estimated component and the true component, assuming that the raw series is a random-walk process. Christiano and Fitzgerald (2003) give three important reasons for using their filter:

1. The true dependence structure of the data affects which filter is optimal.
2. Many economic time series are well approximated by random-walk processes.
3. Their filter does a good job passing through stochastic cycles of desired frequencies and blocking stochastic cycles from unwanted frequencies on a range of processes that are close to being a random-walk process.

The CF filter obtains its optimality properties at the cost of an additional parameter that must be estimated and a loss of robustness. The CF filter is optimal for a random-walk process. If the true process is a random walk with drift, then the drift term must be estimated and removed; see [TS] tsfilter cf for details. The CF filter is not symmetric, so it will not remove second-order deterministic or second-order integrated processes. tsfilter cf also implements another filter that Christiano and Fitzgerald (2003) derived that is an SMA filter with coefficients that sum to zero. This filter is designed to be as close as possible to the random-walk optimal filter under the constraint that it be an SMA filter with constraints that sum to zero; see [TS] tsfilter cf for details.

Technical note

A random-walk process is a first-order integrated process; it must be differenced once to produce a stationary process. Formally, a random-walk process is given by \( y_t = y_{t-1} + \epsilon_t \), where \( \epsilon_t \) is a zero-mean stationary random variable. A random-walk-plus-drift process is given by \( \tilde{y}_t = \mu + \tilde{y}_{t-1} + \epsilon_t \), where \( \epsilon_t \) is a zero-mean stationary random variable.

Example 4: A CF estimate of the business-cycle component

In this example, we use the CF filter to estimate the business-cycle component, and we plot the periodogram of the CF estimates. We specify the drift option because \( \text{ip\_ln} \) is well approximated by a random-walk-plus-drift process.

```
.tsfilter cf ip\_cf = ip\_ln, minperiod(6) maxperiod(32) drift
.pergram ip\_cf, xline(0.03125 0.16667)
```

The periodogram of the CF estimates of the business-cycle component indicates that the CF filter did a better job than the BK filter of passing through only the desired stochastic cycles. Given that \( \text{ip\_ln} \) is well approximated by a random-walk-plus-drift process, the relative performance of the CF filter is not surprising.
As with the BK filter, plotting the gain of the CF filter and the gain of the ideal filter gives an impression of how well the filter isolates the specified components. In the output below, we reestimate the business-cycle component, using the `gain()` option to store the gain of the CF filter, and we plot the gain functions.

```
.drop ip_cf
.tsfilter cf ip_cf = ip_ln, minperiod(6) maxperiod(32) drift gain(cfgain acf)
.label variable cfgain "CF filter"
.twoway line ideal f || line cfgain acf
```

Comparing this graph with the graph of the BK gain function reveals that the CF filter is closer to the gain of the ideal filter than is the BK filter. The graph also reveals that the gain of the CF filter oscillates above and below 1 for desired frequencies.

The choice between the BK or the CF filter is one between robustness or efficiency. The BK filter handles a broader class of stochastic processes, but the CF filter produces a better estimate of $c_t$ if $y_t$ is close to a random-walk process or a random-walk-plus-drift process.

A one-parameter high-pass filter: The Hodrick–Prescott filter

Hodrick and Prescott (1997) motivated the Hodrick–Prescott (HP) filter as a trend-removal technique that could be applied to data that came from a wide class of data-generating processes. In their view, the technique specified a trend in the data, and the data were filtered by removing the trend. The smoothness of the trend depends on a parameter $\lambda$. The trend becomes smoother as $\lambda \to \infty$. Hodrick and Prescott (1997) recommended setting $\lambda$ to 1,600 for quarterly data.

King and Rebelo (1993) showed that removing a trend estimated by the HP filter is equivalent to a high-pass filter. They derived the gain function of this high-pass filter and showed that the filter would make integrated processes of order 4 or less stationary, making the HP filter comparable with the band-pass filters discussed above.
Example 5: An HP estimate of the business-cycle component

We begin by applying the HP high-pass filter to $ip_{\ln}$ and plotting the periodogram of the estimated business-cycle component. We specify the `gain()` option because will use the gain of the filter in the next example.

```plaintext
tsfilter hp ip_hp = ip_{\ln}, gain(hpg1600 ahp1600)
lable variable hpg1600 "HP(1600) filter"
pergram ip_hp, xline(0.03125)
```

Because the HP filter is a high-pass filter, the high-frequency stochastic cycles corresponding to those periods below 6 remain in the estimated component. Of more concern is the presence of the low-frequency stochastic cycles that the filter should remove. We address this issue in the example below.

Example 6: Choosing the parameters for the HP filter

Hodrick and Prescott (1997) argued that the smoothing parameter $\lambda$ should be 1,600 on the basis of a heuristic argument that specified values for the variance of the cyclical component and the variance of the second difference of the trend component, both recorded at quarterly frequencies. In this example, we choose the smoothing parameter to be 677.13, which sets the gain of the filter to 0.5 at the frequency corresponding to 32 periods, as explained in the technical note below. We then plot the periodogram of the filtered series.
. tsfilter hp ip_hp2 = ip_ln, smooth(677.13) gain(hpg677 ahp677)
. label variable hpg677 "HP(677) filter"
. pergram ip_hp, xline(0.03125)

Although the periodogram looks better than the periodogram with the default smoothing, the HP filter still did not zero out the low-frequency stochastic cycles as well as the CF filter did. We take another look at this issue by plotting the gain functions for these filters along with the gain function from the ideal band-pass filter.

. twoway line ideal f || line hpg677 ahp677

Comparing the gain graphs reveals that the gain of the CF filter is closest to the gain of the ideal filter. Both the BK and the HP filters allow some low-frequency stochastic cycles to pass through. The plot also illustrates that the HP filter is a high-pass filter because its gain is 1 for those stochastic cycles at frequencies above 6 periods, whereas the other gain functions go to zero.
Technical note

Conventionally, economists have used $\lambda = 1600$, which Hodrick and Prescott (1997) recommended for quarterly data. Ravn and Uhlig (2002) derived values for $\lambda$ at monthly and annual frequencies that are rescalings of the conventional $\lambda = 1600$ for quarterly data. These heuristic values are the default values; see [TS] tsfilter hp for details. In the filter literature, filter parameters are set as functions of the cutoff frequency; see Pollock (2000, 324), for instance. This method finds the filter parameter that sets the gain of the filter equal to $1/2$ at the cutoff frequency. Applying this method to selecting $\lambda$ at the cutoff frequency of 32 periods requires solving

$$\frac{1}{2} = \frac{4\lambda \{1 - \cos(2\pi/32)\}^2}{1 + 4\lambda \{1 - \cos(2\pi/32)\}^2}$$

for $\lambda$, which yields $\lambda \approx 677.13$, which was used in the previous example.

The gain function of the HP filter is a function of the parameter $\lambda$, and $\lambda$ sets both the location of the cutoff frequency and the slope of the gain function. The graph below illustrates this dependence by plotting the gain function of the HP filter for $\lambda$ set to 10, 677.13, and 1,600 along with the gain function for the ideal band-pass filter with cutoff periods of 32 periods and 6 periods.

A two-parameter high-pass filter: The Butterworth filter

Engineers have used Butterworth filters for a long time because they are “maximally flat”. The gain functions of these filters are as close as possible to being a flat line at 0 for the unwanted periods and a flat line at 1 for the desired periods; see Butterworth (1930) and Bianchi and Sorrentino (2007, 17–20).

Pollock (2000) showed that Butterworth filters can be derived from some axioms that specify properties we would like a filter to have. Although the Butterworth and BK filters share the properties of symmetry and phase neutrality, the coefficients of Butterworth filters do not need to sum to zero. (Phase-neutral filters do not shift the signal forward or backward in time; see Pollock [1999].) Although the BK filter relies on the detrending properties of SMA filters with coefficients that sum to zero, Pollock (2000) shows that Butterworth filters have detrending properties that depend on the filters’ parameters.
tsfilter bw implements the high-pass Butterworth filter using the computational method that Pollock (2000) derived. This filter has two parameters: the cutoff period and the order of the filter denoted by \( m \). The cutoff period sets the location where the gain function starts to filter out the high-period (low-frequency) stochastic cycles, and \( m \) sets the slope of the gain function for a given cutoff period. For a given cutoff period, the slope of the gain function at the cutoff period increases with \( m \). For a given \( m \), the slope of the gain function at the cutoff period increases with the cutoff period.

We cannot obtain a vertical slope at the cutoff frequency, which is the ideal, because the computation becomes unstable; see Pollock (2000). The \( m \) for which the computation becomes unstable depends on the cutoff period.

Pollock (2000) and Gómez (1999) argue that the additional flexibility produced by the additional parameter makes the high-pass Butterworth filter a better filter than the HP filter for estimating the cyclical components.

Pollock (2000) shows that the high-pass Butterworth filter can estimate the desired components of the \( d \)th difference of a \( d \)th-order integrated process as long as \( m \geq d \).

Example 7: A Butterworth filter that removes low-frequency components

Below we use tsfilter bw to estimate the components driven by stochastic cycles greater than 32 periods using Butterworth filters of order 2 and order 6. We also compute, label, and plot the gain functions for each filter.

```
.tsfilter bw ip_bw1 = ip_ln, gain(bwgain1 abw1) maxperiod(32) order(2)
.label variable bwgain1 "BW 2"
.tsfilter bw ip_bw6 = ip_ln, gain(bwgain6 abw6) maxperiod(32) order(6)
.label variable bwgain6 "BW 6"
.twoway line ideal f || line bwgain1 abw1 || line bwgain6 abw6
```

The graph illustrates that the slope of the gain function increases with the order of the filter.

The graph below provides another perspective by plotting the gain function from the ideal band-pass filter on a graph with plots of the gain functions from the Butterworth filter of order 6, the CF filter, and the HP(677) filter.
Although the slope of the gain function from the CF filter is closer to being vertical at the cutoff frequency, the gain function of the Butterworth filter does not oscillate above and below 1 after it first reaches the value of 1. The flatness of the Butterworth filter below and above the cutoff frequency is not an accident; it is one of the filter’s properties.

Example 8: A Butterworth filter that removes high-frequency components

In the previous example, we used the Butterworth filter of order 6 to remove low-frequency stochastic cycles, and we saved the results in `ip_bw6`. The Butterworth filter did not address the high-frequency stochastic cycles below 6 periods because it is a high-pass filter. We remove those high-frequency stochastic cycles in this example by keeping the trend produced by refiltering the previously filtered series.

This example uses a common trick: keeping the trend produced by a high-pass filter turns that high-pass filter into a low-pass filter. Because we want to remove the high-frequency stochastic cycles still in the previously filtered series `ip_bw6`, we need a low-pass filter. So we keep the trend produced by refiltering the previously filtered series.

In the output below, we apply a Butterworth filter of order 20 to the previously filtered series `ip_bw6`. We explain why we used order 20 in the next example. We specify the `trend()` option to keep the low-frequency components from these filters. Then we compute and graph the periodogram for the trend variable.
. tsfilter bw ip_bw20 = ip_bw6, gain(bwg20 fbw20) maxperiod(6) order(20)
> trend(ip_bwb)
. label variable bwg20 "BW upper filter 20"
. pergram ip_bwb, xline(0.03125 0.16667)

The periodogram reveals that the two-pass process has passed the original series \( \text{ip}_\text{ln} \) through a band-pass filter. It also reveals that the two-pass process did a reasonable job of filtering out the stochastic cycles corresponding to the unwanted frequencies.

Example 9: Choosing the order of a Butterworth filter

In the previous example, when the cutoff period was 6, we set the order of the Butterworth filter to 20. In contrast, in example 7, when the cutoff period was 32, we set the order of the Butterworth filter to 6. We had to increase filter order because the slope of the gain function of the Butterworth filter is increasing with the cutoff period. We needed a larger filter order to get an acceptable slope at the lower cutoff period.

We illustrate this point in the output below. We apply Butterworth filters of orders 1 and 6 to the previously filtered series \( \text{ip}_\text{bw6} \), we compute the gain functions, we label the gain variables, and then we plot the gain functions from the ideal filter and the Butterworth filters.
. tsfilter bw ip_bwu1 = ip_bw6, gain(bwg1 fbw1) maxperiod(6) order(2)
. label variable bwg1 "BW upper filter 2"
. tsfilter bw ip_bwu6 = ip_bw6, gain(bwg6 fbw6) maxperiod(6) order(6)
. label variable bwg6 "BW upper filter 6"
. twoway line ideal f || line bwg1 fbw1 || line bwg6 fbw6 || line bwg20 fbw20

Because the cutoff period is 6, the gain functions for $m = 2$ and $m = 6$ are much flatter than the
gain functions for $m = 2$ and $m = 6$ in example 7 when the cutoff period was 32. The gain function
for $m = 20$ is reasonably close to vertical, so we used it in example 8. We mentioned above that
for any given cutoff period, the computation eventually becomes unstable for larger values of $m$. For
instance, when the cutoff period is 32, $m = 20$ is not numerically feasible.

> Example 10: Comparing the Butterworth and CF estimates

As a conclusion, we plot the business-cycle components estimated by the CF filter and by the
two passes of Butterworth filters. The shaded areas identify recessions. The two estimates are close
but the differences could be important. Which estimate is better depends on whether the oscillations
around 1 in the graph of the CF gain function (the second graph of example 7) cause more problems
than the nonvertical slopes at the cutoff periods that occur in the BW6 gain function of that same
graph and the BW upper filter 20 gain function graphed above.
There is a long tradition in economics of using models to estimate components. Instead of comparing filters by their gain functions, some authors compare filters by finding underlying models for which the filter parameters are the model parameters. For instance, Harvey and Jaeger (1993), Gómez (1999, 2001), Pollock (2000, 2006), and Harvey and Trimbur (2003) derive models that correspond to the HP or the Butterworth filter. Some of these references also compare components estimated by filters with components estimated by making predictions from estimated models. In effect, these references point out that arima, dfactor, sspace, and ucm (see [TS] arima, [TS] dfactor, [TS] sspace, and [TS] ucm) implement alternative methods to component estimation.

Methods and formulas

All filters work with both time-series data and panel data when there are many observations on each panel. When used with panel data, the calculations are performed separately within each panel.

For these filters, the default minimum and maximum periods of oscillation correspond to the boundaries used by economists (Burns and Mitchell 1946) for business cycles. Burns and Mitchell defined business cycles as oscillations in business data with recurring periods between 1.5 and 8 years. Their definition continues to be cited by economists investigating correlations between business cycles.

If $y_t$ is a time series, then the cyclical component is

$$c_t = B(L)y_t = \sum_{j=-\infty}^{\infty} b_j y_{t-j}$$

where $b_j$ are the coefficients of the impulse–response sequence of some ideal filter. The impulse–response sequence is the inverse Fourier transform of either a square wave or step function depending upon whether the filter is a band-pass or high-pass filter, respectively.
In finite sequences, it is necessary to approximate this calculation with a finite impulse–response sequence \( \hat{b}_j \):

\[
\hat{c}_t = \hat{B}_t(L)y_t = \sum_{j=-n_1}^{n_2} \hat{b}_j y_{t-j}
\]

The infinite-order impulse–response sequence for the filters implemented in \texttt{tsfilter} are symmetric and time-invariant.

In the frequency domain, the relationships between the true cyclical component and its finite estimates respectively are

\[
c(\omega) = B(\omega) y(\omega)
\]

and

\[
\hat{c}(\omega) = \hat{B}(\omega) y(\omega)
\]

where \( B(\omega) \) and \( \hat{B}(\omega) \) are the frequency transfer functions of the filters \( B \) and \( \hat{B} \).

The frequency transfer function for \( B(\omega) \) can be expressed in polar form as

\[
B(\omega) = |B(\omega)| \exp\{i\theta(\omega)\}
\]

where \(|B(\omega)|\) is the filter’s gain function and \( \theta(\omega) \) is the filter’s phase function. The gain function determines whether the amplitude of the stochastic cycle is increased or decreased at a particular frequency. The phase function determines how a cycle at a particular frequency is shifted forward or backward in time.

In this form, it can be shown that the spectrum of the cyclical component, \( f_c(\omega) \), is related to the spectrum of \( y_t \) series by the squared gain:

\[
f_c(\omega) = |B(\omega)|^2 f_y(\omega)
\]

Each of the four filters in \texttt{tsfilter} has an option for returning an estimate of the gain function together with its associated scaled frequency \( a = \omega/\pi \), where \( 0 \leq \omega \leq \pi \). These are consistent estimates of \(|B(\omega)|\), the gain from the ideal linear filter.

The band-pass filters implemented in \texttt{tsfilter}, the \( BK \) and \( CF \) filters, use a square wave as the ideal transfer function:

\[
B(\omega) = \begin{cases} 
1 & \text{if } |\omega| \in [\omega_l, \omega_h] \\
0 & \text{if } |\omega| \notin [\omega_l, \omega_h]
\end{cases}
\]

The high-pass filters, the Hodrick–Prescott and Butterworth filters, use a step function as the ideal transfer function:

\[
B(\omega) = \begin{cases} 
1 & \text{if } |\omega| \geq \omega_h \\
0 & \text{if } |\omega| < \omega_h
\end{cases}
\]

**Acknowledgments**

We thank Christopher F. Baum of the Department of Economics at Boston College and author of the Stata Press books \textit{An Introduction to Modern Econometrics Using Stata} and \textit{An Introduction to Stata Programming} for his previous implementations of these filters: Baxter–King (\texttt{bking}), Christiano–Fitzgerald (\texttt{cfitzrw}), Hodrick–Prescott (\texttt{hprescott}), and Butterworth (\texttt{butterworth}).
We also thank D. S. G. Pollock of the Department of Economics at the University of Leicester, UK, for his helpful responses to our questions about Butterworth filters and the methods that he has developed.

References


Also see

[TS] tsset — Declare data to be time-series data

[TS] tssmooth — Smooth and forecast univariate time-series data

[XT] xtset — Declare data to be panel data
Description

`tsfilter bk` uses the Baxter and King (1999) band-pass filter to separate a time series into trend and cyclical components. The trend component may contain a deterministic or a stochastic trend. The stationary cyclical component is driven by stochastic cycles at the specified periods.

See [TS] `tsfilter` for an introduction to the methods implemented in `tsfilter bk`.

Quick start

Use the Baxter–King filter for `y`, and obtain cyclical component `ct` using `tsset` data

```
tsfilter bk ct=y
```

As above, but filter out stochastic cycles with periods smaller than 5 and those with periods larger than 20

```
tsfilter bk ct=y, minperiod(5) maxperiod(20)
```

As above, and save the trend component in the variable `trendvar`

```
tsfilter bk ct=y, minperiod(5) maxperiod(20) trend(trendvar)
```

Save gain in `gain` and angular frequency in `angle`

```
tsfilter bk ct=y, gain(gain angle)
```

Use the Baxter–King filter for variables `y1`, `y2`, and `y3` to obtain cyclical components with prefix `cycl`

```
tsfilter bk cycl*'=y1 y2 y3
```

Note: The above commands can also be used to apply the filter separately to each panel of a panel dataset when a `panelvar` has been specified by using `tsset` or `xtset`.

Menu

Statistics > Time series > Filters for cyclical components > Baxter-King
Syntax

Filter one variable

```plaintext
tsfilter bk [type] newvar = varname [if] [in] [ , options ]
```

Filter multiple variables, unique names

```plaintext
tsfilter bk [type] newvarlist = varlist [if] [in] [ , options ]
```

Filter multiple variables, common name stub

```plaintext
tsfilter bk [type] stub* = varlist [if] [in] [ , options ]
```

**options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
</tr>
<tr>
<td>minperiod(#)</td>
</tr>
<tr>
<td>maxperiod(#)</td>
</tr>
<tr>
<td>smaorder(#)</td>
</tr>
<tr>
<td>stationary</td>
</tr>
<tr>
<td><strong>Trend</strong></td>
</tr>
<tr>
<td>trend(newvar</td>
</tr>
<tr>
<td><strong>Gain</strong></td>
</tr>
<tr>
<td>gain(gainvar anglevar)</td>
</tr>
</tbody>
</table>

You must `tsset` or `xtset` your data before using `tsfilter`; see [TS] `tsset` and [XT] `xtset`. `varname` and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

**Main**

- `minperiod(#)` filters out stochastic cycles at periods smaller than #, where # must be at least 2 and less than `maxperiod()`. By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, or half-yearly, then # is set to the number of periods equivalent to 1.5 years; yearly data use `minperiod(2)`; otherwise, the default value is `minperiod(6)`.

- `maxperiod(#)` filters out stochastic cycles at periods larger than #, where # must be greater than `minperiod()`. By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, half-yearly, or yearly, then # is set to the number of periods equivalent to 8 years; otherwise, the default value is `maxperiod(32)`.

- `smaorder(#)` sets the order of the symmetric moving average, denoted by q. The order is an integer that specifies the number of observations in each direction used in calculating the symmetric moving average estimate of the cyclical component. This number must be an integer greater than zero and less than \((T − 1)/2\). The estimate for the cyclical component for the \(t\)th observation, \(y_t\), is based upon the \(2q + 1\) values \(y_{t−q}, y_{t−q+1}, \ldots, y_t, y_{t+1}, \ldots, y_{t+q}\). By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, half-yearly, or yearly, then # is set to the number of periods equivalent to 3 years; otherwise, the default value is `smaorder(12)`.
stationary modifies the filter calculations to those appropriate for a stationary series. By default, the series is assumed nonstationary.

Trend

trend(newvar | newvarlist | stub*) saves the trend component(s) in the new variable(s) specified by newvar, newvarlist, or stub*.

Gain

gain(gainvar anglevar) saves the gain in gainvar and its associated angular frequency in anglevar. Gains are calculated at the $N$ angular frequencies that uniformly partition the interval $(0, \pi]$, where $N$ is the sample size.

Remarks and examples

We assume that you have already read [TS] tsfilter, which provides an introduction to filtering and the methods implemented in tsfilter bk, more examples using tsfilter bk, and a comparison of the four filters implemented by tsfilter. In particular, an understanding of gain functions as presented in [TS] tsfilter is required to understand these remarks.

tsfilter bk uses the Baxter–King (BK) band-pass filter to separate a time-series $y_t$ into trend and cyclical components:

$$y_t = \tau_t + c_t$$

where $\tau_t$ is the trend component and $c_t$ is the cyclical component. $\tau_t$ may be nonstationary; it may contain a deterministic or a stochastic trend, as discussed below.

The primary objective is to estimate $c_t$, a stationary cyclical component that is driven by stochastic cycles within a specified range of periods. The trend component $\tau_t$ is calculated by the difference $\tau_t = y_t - c_t$.

Although the BK band-pass filter implemented in tsfilter bk has been widely applied by macroeconomists, it is a general time-series method and may be of interest to other researchers.

Symmetric moving-average (SMA) filters with coefficients that sum to zero remove stochastic and deterministic trends of first and second order; see Fuller (1996), Baxter and King (1995), and Baxter and King (1999).

For an infinitely long series, there is an ideal band-pass filter for which the gain function is 1 for $\omega \in [\omega_0, \omega_1]$ and 0 for all other frequencies; see [TS] tsfilter for an introduction to gain functions. It just so happens that this ideal band-pass filter is an SMA filter with coefficients that sum to zero. Baxter and King (1999) derive the coefficients of this ideal band-pass filter and then define the BK filter to be the SMA filter with $2q + 1$ terms that are as close as possible to those of the ideal filter. There is a trade-off in choosing $q$: larger values of $q$ cause the gain of the BK filter to be closer to the gain of the ideal filter, but they also increase the number of missing observations in the filtered series.

The smaorder() option specifies $q$. The default value of smaorder() is the number of periods equivalent to 3 years, following the Baxter and King (1999) recommendation.

Although the mathematics of the frequency-domain approach to time-series analysis is in terms of stochastic cycles at frequencies $\omega \in [-\pi, \pi]$, applied work is generally in terms of periods $p$, where $p = 2\pi/\omega$. So tsfilter bk has the minperiod() and maxperiod() options to specify the desired range of stochastic cycles.
Among economists, the BK filter is commonly used for investigating business cycles. Burns and Mitchell (1946) defined business cycles as stochastic cycles in business data corresponding to periods between 1.5 and 8 years. The default values for minperiod() and maxperiod() are the Burns–Mitchell values of 1.5 and 8 years, scaled to the frequency of the dataset. The calculations of the default values assume that the time variable is formatted as daily, weekly, monthly, quarterly, half-yearly, or yearly; see [D] format.

For each variable, the band-pass BK filter estimate of $c_t$ is put in the corresponding new variable, and when the trend() option is specified, the estimate of $\tau_t$ is put in the corresponding new variable.

tsfilt bk automatically detects panel data from the information provided when the dataset was tsset or xtset. All calculations are done separately on each panel. Missing values at the beginning and end of the sample are excluded from the sample. The sample may not contain gaps.

Baxter and King (1999) derived their method for nonstationary time series, but they noted that a small modification makes it applicable to stationary time series. Imposing the condition that the filter coefficients sum to zero is what makes their method applicable to nonstationary time series; dropping this condition yields a filter for stationary time series. Specifying the stationary option causes tsfilter bk to use coefficients calculated without the constraint that they sum to zero.

Example 1: Estimating a business-cycle component

In this and the subsequent examples, we use tsfilter bk to estimate the business-cycle component of the natural log of real gross domestic product (GDP) of the United States. Our sample of quarterly data goes from 1952q1 to 2010q4. Below we read in and plot the data.

```
. use https://www.stata-press.com/data/r16/gdp2
(Federal Reserve Economic Data, St. Louis Fed)
. tsline gdp_ln
```

The series is nonstationary and is thus a candidate for the BK filter.
Below we use `tsfilter bk` to filter `gdp_ln`, and we use `pergram` (see `[TS] pergram`) to compute and to plot the periodogram of the estimated cyclical component.

```
. tsfilter bk gdp_bk = gdp_ln
. pergram gdp_bk, xline(.03125 .16667)
```

Because our sample is of quarterly data, `tsfilter bk` used the default values of `minperiod(6)`, `maxperiod(32)`, and `smaorder(12)`. The minimum and maximum periods are the Burns and Mitchell (1946) business-cycle periods for quarterly data. The default of `smaorder(12)` was recommend by Baxter and King (1999) for quarterly data.

In the periodogram, we added vertical lines at the natural frequencies corresponding to the conventional Burns and Mitchell (1946) values for business-cycle components. `pergram` displays the results in natural frequencies, which are the standard frequencies divided by $2\pi$. We use the `xline()` option to draw vertical lines at the lower natural-frequency cutoff ($1/32 = 0.03125$) and the upper natural-frequency cutoff ($1/6 \approx 0.16667$).

If the filter completely removed the stochastic cycles at the unwanted frequencies, the periodogram would be a flat line at the minimum value of $-6$ outside the range identified by the vertical lines.

The periodogram reveals that the default value of `smaorder(12)` did not do a good job of filtering out the high-periodicity stochastic cycles, because there are too many points above $-6.00$ to the left of the left-hand vertical line. It also reveals that the filter did not remove enough low-periodicity stochastic cycles, because there are too many points above $-6.00$ to the right of the right-hand vertical line.

We address these problems in the next example.

---

**Example 2: Changing the order of the filter**

In this example, we change the symmetric moving average of the filter via the `smaorder()` option so that it will remove more of the unwanted stochastic cycles. As mentioned, larger values of $q$ cause the gain of the BK filter to be closer to the gain of the ideal filter, but larger values also increase the number of missing observations in the filtered series.
In the output below, we estimate the business-cycle component and compute the gain functions when the SMA-order of the filter is 12 and when it is 20. We also generate `ideal`, the gain function of the ideal band-pass filter at the frequencies \( f \). Then we plot the gain functions from all three filters.

```stata
.tsfilter bk gdp_bk12 = gdp_ln, gain(g12 a12)
.label variable g12 "BK SMA-order 12"
.tsfilter bk gdp_bk20 = gdp_ln, gain(g20 a20) smaorder(20)
.label variable g20 "BK SMA-order 20"
generate f = _pi*(_n-1)/_N
generate ideal = cond(f<_pi/16, 0, cond(f<_pi/3, 1,0))
.label variable ideal "Ideal filter"
twoway line ideal f || line g12 a12 || line g20 a20
```

As discussed in [TS] `tsfilter`, the gain function of the ideal filter is a square wave with a value of 0 at the frequencies corresponding to unwanted frequencies and a value of 1 at the desired frequencies. The vertical lines in the gain function of the ideal filter occur at the frequencies \( \pi/16 \), corresponding to 32 periods, and at \( \pi/3 \), corresponding to 6 periods. (Given that \( p = 2\pi/\omega \), where \( p \) is the period corresponding to the frequency \( \omega \), the frequency is given by \( 2\pi/p \).)

The differences between the gain function of the filter with SMA-order 12 and the gain function of the ideal band-pass filter is the root of the issues mentioned at the end of example 1. The filter with SMA-order 20 is closer to the gain function of the ideal band-pass filter at the cost of 16 more missing values in the filtered series.
Below we compute and graph the periodogram of the series filtered with SMA-order 20.

\[ \text{pergram gdp\_bk20, xline(.03125 .16667)} \]

The above periodogram indicates that the filter of SMA-order 20 removed more of the stochastic cycles at the unwanted periodicities than did the filter of SMA-order 12. Whether removing the stochastic cycles at the unwanted periodicities is worth losing more observations in the filtered series is a judgment call.
Below we plot the estimated business-cycle component with recessions identified by the shaded areas.

Stored results

tsfiltter bk stores the following in r():

Scalars
- r(smaorder) order of the symmetric moving average
- r(minperiod) minimum period of stochastic cycles
- r(maxperiod) maximum period of stochastic cycles

Macros
- r(varlist) original time-series variables
- r(filterlist) variables containing estimates of the cyclical components
- r(trendlist) variables containing estimates of the trend components, if trend() was specified
- r(method) Baxter–King
- r(stationary) yes or no, indicating whether the calculations assumed the series was or was not stationary
- r(unit) units of time variable set using tsset or xtset

Matrices
- r(filter) \((q+1)\times1\) matrix of filter weights, where \(q\) is the order of the symmetric moving average

Methods and formulas

Baxter and King (1999) showed that there is an infinite-order SMA filter with coefficients that sum to zero that can extract the specified components from a nonstationary time series. The components are specified in terms of the minimum and maximum periods of the stochastic cycles that drive these components in the frequency domain. This ideal filter is not feasible, because the constraints imposed on the filter can only be satisfied using an infinite number of coefficients, so Baxter and King (1999) derived a finite approximation to this ideal filter.

The infinite-order, ideal band-pass filter obtains the cyclical component with the calculation

\[
c_t = \sum_{j=-\infty}^{\infty} b_j y_{t-j}
\]
Letting \( p_l \) and \( p_h \) be the minimum and maximum periods of the stochastic cycles of interest, the weights \( b_j \) in this calculation are given by

\[
b_j = \begin{cases} 
\pi^{-1}(\omega_h - \omega_l) & \text{if } j = 0 \\
(j\pi)^{-1}\{\sin(j\omega_h) - \sin(j\omega_l)\} & \text{if } j \neq 0 
\end{cases}
\]

where \( \omega_l = 2\pi/p_l \) and \( \omega_h = 2\pi/p_h \) are the lower and higher cutoff frequencies, respectively.

For the default case of nonstationary time series with finite length, the ideal band-pass filter cannot be used without modification. Baxter and King (1999) derived modified weights for a finite order SMA filter with coefficients that sum to zero.

As a result, Baxter and King (1999) estimate \( c_t \) by

\[
c_t = \sum_{j=-q}^{+q} \hat{b}_j y_{t-j}
\]

The coefficients \( \hat{b}_j \) in this calculation are equal to \( \hat{b}_j = b_j - \bar{b}_q \), where \( \bar{b}_j = \hat{b}_j \) and \( \bar{b}_q \) is the mean of the ideal coefficients truncated at \( \pm q \):

\[
\bar{b}_q = (2q + 1)^{-1} \sum_{j=-q}^{+q} b_j
\]

Note that \( \sum_{j=-q}^{+q} \hat{b}_j = 0 \) and that the first and last \( q \) values of the cyclical component cannot be estimated using this filter.

If the stationary option is set, the BK filter sets the coefficients to the ideal coefficients, that is, \( \hat{b}_j = b_j \). For these weights, \( \hat{b}_j = \hat{b}_{-j} \), and although \( \sum_{j=-\infty}^{\infty} \hat{b}_j = 0 \), for small \( q \), \( \sum_{-q}^{q} \hat{b}_j \neq 0 \).

References


Also see

[TS] \texttt{tsfilter} — Filter a time series for cyclical components

[TS] \texttt{tsset} — Declare data to be time-series data

[TS] \texttt{tssmooth} — Smooth and forecast univariate time-series data

[D] \texttt{format} — Set variables’ output format

[XT] \texttt{xtset} — Declare data to be panel data
**tsfilter bw — Butterworth time-series filter**

### Description

*tsfilter bw* uses the Butterworth high-pass filter to separate a time series into trend and cyclical components. The trend component may contain a deterministic or a stochastic trend. The stationary cyclical component is driven by stochastic cycles at the specified periods.

See 

### Quick start

Use the Butterworth filter for *y*, and obtain cyclical component *ct* using `tsset` data

```
tsfilter bw ct=y
```

As above, but filter periods larger than 20

```
tsfilter bw ct=y, maxperiod(20)
```

As above, and save the trend component in the variable *trendvar*

```
tsfilter bw ct=y, maxperiod(20) trend(trendvar)
```

As above, and save gain and angular frequency with the names *gain* and *angle*

```
tsfilter bw ct=y, maxperiod(20) trend(trendvar) gain(gain angle)
```

Set the order of the filter to 4, and save the trend component in the variable *trendvar*

```
tsfilter bw ct=y, order(4) trend(trendvar)
```

Use Butterworth filter for variables *y1*, *y2*, and *y3* to obtain cyclical components with prefix *cycl*

```
tsfilter bw cycl* = y1 y2 y3
```

Note: The above commands can also be used to apply the filter separately to each panel of a panel dataset when a *panelvar* has been specified by using `tsset` or `xtset`.

### Menu

Statistics > Time series > Filters for cyclical components > Butterworth
Syntax

Filter one variable

```
   tsfilter bw [type] newvar = varname [if] [in] [, options]
```

Filter multiple variables, unique names

```
   tsfilter bw [type] newvarlist = varlist [if] [in] [, options]
```

Filter multiple variables, common name stub

```
   tsfilter bw [type] stub* = varlist [if] [in] [, options]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td>filter out stochastic cycles at periods larger than #</td>
</tr>
<tr>
<td>maxperiod(#)</td>
<td>set the order of the filter; default is order(2)</td>
</tr>
<tr>
<td>order(#)</td>
<td>save the trend component(s) in new variable(s)</td>
</tr>
<tr>
<td>Trend</td>
<td>save the gain and angular frequency</td>
</tr>
<tr>
<td>trend(newvar</td>
<td>newvarlist</td>
</tr>
</tbody>
</table>

Gains are calculated at the $N$ angular frequencies that uniformly partition the interval $(0, \pi]$, where $N$ is the sample size.

You must `tsset` or `xtset` your data before using `tsfilter`; see [TS] `tsset` and [XT] `xtset`.

`varname` and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

**Main**

`maxperiod(#)` filters out stochastic cycles at periods larger than #, where # must be greater than 2.

By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, half-yearly, or yearly, then # is set to the number of periods equivalent to 8 years; otherwise, the default value is `maxperiod(32)`.

`order(#)` sets the order of the Butterworth filter, which must be an integer. The default is `order(2)`.

**Trend**

`trend(newvar | newvarlist | stub*)` saves the trend component(s) in the new variable(s) specified by `newvar`, `newvarlist`, or `stub*`.

**Gain**

`gain(gainvar anglevar)` saves the gain in `gainvar` and its associated angular frequency in `anglevar`.
Remarks and examples

We assume that you have already read [TS] tsfilter, which provides an introduction to filtering and the methods implemented in tsfilter bw, more examples using tsfilter bw, and a comparison of the four filters implemented by tsfilter. In particular, an understanding of gain functions as presented in [TS] tsfilter is required to understand these remarks.

tsfilter bw uses the Butterworth high-pass filter to separate a time-series $y_t$ into trend and cyclical components:

$$y_t = \tau_t + c_t$$

where $\tau_t$ is the trend component and $c_t$ is the cyclical component. $\tau_t$ may be nonstationary; it may contain a deterministic or a stochastic trend, as discussed below.

The primary objective is to estimate $c_t$, a stationary cyclical component that is driven by stochastic cycles within a specified range of periods. The trend component $\tau_t$ is calculated by the difference $\tau_t = y_t - c_t$.

Although the Butterworth high-pass filter implemented in tsfilter bw has been widely applied by macroeconomists and engineers, it is a general time-series method and may be of interest to other researchers.

Engineers have used Butterworth filters for a long time because they are “maximally flat”. The gain functions of these filters are as close as possible to being a flat line at 0 for the unwanted periods and a flat line at 1 for the desired periods; see Butterworth (1930) and Bianchi and Sorrentino (2007, 17–20). (See [TS] tsfilter for an introduction to gain functions.)

The high-pass Butterworth filter is a two-parameter filter. The maxperiod() option specifies the maximum period; the stochastic cycles of all higher periodicities are filtered out. The maxperiod() option sets the location of the cutoff period in the gain function. The order() option specifies the order of the filter, which determines the slope of the gain function at the cutoff frequency.

For a given cutoff period, the slope of the gain function at the cutoff period increases with filter order. For a given filter order, the slope of the gain function at the cutoff period increases with the cutoff period.

We cannot obtain a vertical slope at the cutoff frequency, which is the ideal, because the computation becomes unstable; see Pollock (2000). The filter order for which the computation becomes unstable depends on the cutoff period.

Among economists, the high-pass Butterworth filter is commonly used for investigating business cycles. Burns and Mitchell (1946) defined business cycles as stochastic cycles in business data corresponding to periods between 1.5 and 8 years. For this reason, the default value for maxperiod() is the number of periods in 8 years, if the time variable is formatted as daily, weekly, monthly, quarterly, half-yearly, or yearly; see [D] format. The default value for maxperiod() is 32 for all other time formats.

For each variable, the high-pass Butterworth filter estimate of $c_t$ is put in the corresponding new variable, and when the trend() option is specified, the estimate of $\tau_t$ is put in the corresponding new variable.
	sfilter bw automatically detects panel data from the information provided when the dataset was tsset or xtset. All calculations are done separately on each panel. Missing values at the beginning and end of the sample are excluded from the sample. The sample may not contain gaps.
Example 1: Estimating a business-cycle component

In this and the subsequent examples, we use `tsfilter bw` to estimate the business-cycle component of the natural log of the real gross domestic product (GDP) of the United States. Our sample of quarterly data goes from 1952q1 to 2010q4. Below we read in and plot the data.

```
use https://www.stata-press.com/data/r16/gdp2
(Federal Reserve Economic Data, St. Louis Fed)
.tsline gdp_ln
```

The series is nonstationary. Pollock (2000) shows that the high-pass Butterworth filter can estimate the components driven by the stochastic cycles at the specified frequencies when the original series is nonstationary.

Below we use `tsfilter bw` to filter `gdp_ln` and use `pergram` (see [TS] `pergram`) to compute and to plot the periodogram of the estimated cyclical component.

```
.tsfilter bw gdp_bw = gdp_ln
.pergram gdp_bw, xline(.03125 .16667)
```
tsfilter bw used the default value of `maxperiod(32)` because our sample is of quarterly data. In
the periodogram, we added vertical lines at the natural frequencies corresponding to the conventional
Burns and Mitchell (1946) values for business-cycle components. `pergram` displays the results in
natural frequencies, which are the standard frequencies divided by $2\pi$. We use option `xline()` to draw
vertical lines at the lower natural-frequency cutoff ($1/32 = 0.03125$) and the upper natural-frequency
cutoff ($1/6 \approx 0.16667$).

If the filter completely removed the stochastic cycles at the unwanted frequencies, the periodogram
would be a flat line at the minimum value of $-6$ outside the range identified by the vertical lines.

The periodogram reveals two issues. First, it indicates that the default value of `order(2)` did not
do a good job of filtering out the high-periodicity stochastic cycles, because there are too many points
above $-6.00$ to the left of the left-hand vertical line. Second, it reveals the high-pass nature of the
filter, because none of the low-period (high-frequency) stochastic cycles have been filtered out.

We cope with these two issues in the remaining examples.

### Example 2: Changing the order of the filter

In this example, we change the order of the filter so that it will remove more of the unwanted
low-frequency stochastic cycles. As previously mentioned, increasing the order of the filter increases
the slope of the gain function at the cutoff period.

For orders 2 and 8, we compute the filtered series, compute the gain functions, and label the gain
variables. We also generate `ideal`, the gain function of the ideal band-pass filter at the frequencies
$f$. Then we plot the gain function of the ideal band-pass filter and the gain functions of the high-pass
Butterworth filters of orders 2 and 8.

```
. tsfilter bw gdp_bw2 = gdp_ln, gain(g1 a1)
. label variable g1 "BW order 2"
. tsfilter bw gdp_bw8 = gdp_ln, gain(g8 a8) order(8)
. label variable g8 "BW order 8"
. generate f = _pi*(_n-1)/_N
. generate ideal = cond(f<_pi/16, 0, cond(f<_pi/3, 1,0))
. label variable ideal "Ideal filter"
. twoway line ideal f || line g1 a1 || line g8 a8
```
As discussed in [TS] tssfiter, the gain function of the ideal filter is a square wave with a value of 0 at the frequencies corresponding to unwanted frequencies and a value of 1 at the desired frequencies. The vertical lines in the gain function of the ideal filter occur at \( \pi/16 \), corresponding to 32 periods, and at \( \pi/3 \), corresponding to 6 periods. (Given that \( p = 2\pi/\omega \), where \( p \) is the period corresponding to frequency \( \omega \), the frequency is given by \( 2\pi/p \).

The distance between the gain function of the filter with order 2 and the gain function of the ideal band-pass filter at \( \pi/16 \) is the root of the first issue mentioned at the end of example 1. The filter with order 8 is much closer to the gain function of the ideal band-pass filter at \( \pi/16 \) than is the filter with order 2. That both gain functions are 1 to the right of the vertical line at \( \pi/3 \) reveals the high-pass nature of the filter.

Example 3: Removing the high-frequency component

In this example, we use a common trick to resolve the second issue mentioned at the end of example 1. Keeping the trend produced by a high-pass filter turns that high-pass filter into a low-pass filter. Because we want to remove the high-frequency stochastic cycles still in the previously filtered series \( gdp_{bw8} \), we need to run \( gdp_{bw8} \) through a low-pass filter. So we keep the trend produced by refiltering the previously filtered series.

To determine an order for the filter, we run the filter with \( \text{order}(8) \), then with \( \text{order}(15) \), and then we plot the gain functions along with the gain function of the ideal filter.

```
. tssfiter bw gdp_bw8 = gdp_bw8, gain(gc8 ac8) order(8)
> maxperiod(6) trend(gdp_bwc8)
. label variable gc8 "BW order 8"
. tssfiter bw gdp_bw15 = gdp_bw8, gain(gc15 ac15) order(15)
> maxperiod(6) trend(gdp_bwc15)
. label variable gc15 "BW order 15"
. twoway line ideal f || line gc8 ac8 || line gc15 ac15
```

We specified much higher orders for the filter in this example because the cutoff period is 6 instead of 32. (As previously mentioned, holding the order of the filter constant, the slope of the gain function at the cutoff period decreases when the period decreases.) The above graph indicates that the filter with \( \text{order}(15) \) is reasonably close to the gain function of the ideal filter.
Now we compute and plot the periodogram of the estimated business-cycle component.

```
. pergram gdp_bwc15, xline(.03125 .16667)
```

The graph indicates that the above applications of the Butterworth filter did a reasonable job of filtering out the high-periodicity stochastic cycles but that the low-periodicity stochastic cycles have not been completely removed.

Below we plot the estimated business-cycle component with recessions identified by the shaded areas.
tsfilter bw stores the following in r():

Scalars
- `r(order)` order of the filter
- `r(maxperiod)` maximum period of stochastic cycles

Macros
- `r(varlist)` original time-series variables
- `r(filterlist)` variables containing estimates of the cyclical components
- `r(trendlist)` variables containing estimates of the trend components, if `trend()` was specified
- `r(method)` Butterworth
- `r(unit)` units of time variable set using `tsset` or `xtset`

Methods and formulas

`tsfilter bw` uses the computational methods described in Pollock (2000) to implement the filter. Pollock (2000) shows that the gain of the Butterworth high-pass filter is given by

$$\psi(\omega) = \left[1 + \left\{\frac{\tan(\omega_c/2)}{\tan(\omega/2)}\right\}^{2m}\right]^{-1}$$

where $m$ is the order of the filter, $\omega_c = 2\pi/p_h$ is the cutoff frequency, and $p_h$ is the maximum period.

Here is an outline of the computational procedure that Pollock (2000) derived.

Pollock (2000) showed that the Butterworth filter corresponds to a particular model. Actually, his model is more general than the Butterworth filter, but `tsfilter bw` restricts the computations to the case in which the model corresponds to the Butterworth filter.

The model represents the series to be filtered, $y_t$, in terms of zero mean, covariance stationary, and independent and identically distributed shocks $\nu_t$ and $\varepsilon_t$:

$$y_t = \frac{(1 + L)^m}{(1 - L)^m} \nu_t + \varepsilon_t$$

From this model, Pollock (2000) shows that the optimal estimate for the cyclical component is given by

$$c = \lambda Q(\Omega_L + \lambda \Omega_H)^{-1} Q'y$$

where $\text{Var}\{Q'(y - c)\} = \sigma^2\Omega_L$ and $\text{Var}\{Q'c\} = \sigma^2\Omega_H$. Here $\Omega_L$ and $\Omega_H$ are symmetric Toeplitz matrices with $2m + 1$ nonzero diagonal bands and generating functions $(1 + z)^m(1 + z^{-1})^m$ and $(1 - z)^m(1 - z^{-1})^m$, respectively.

The parameter $\lambda$ in this expression is a function of $p_h$ (the maximum period of stochastic cycles filtered out) and the order of the filter:

$$\lambda = \{\tan(\pi/p_h)\}^{-2m}$$
The matrix $Q'$ in this expression is a function of the coefficients in the polynomial $(1 - L)^d = 1 + \delta_1 L + \cdots + \delta_d L^d$:

$$Q' = \begin{pmatrix}
\delta_d & \ldots & \delta_1 & 1 & \ldots & 0 & 0 & \ldots & 0 & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \ldots & \delta_d & \delta_{d-1} & \ldots & 1 & 0 & \ldots & 0 & 0 \\
0 & \ldots & 0 & \delta_d & \ldots & \delta_1 & 1 & \ldots & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & 0 & \ldots & \delta_d & \delta_{d-1} & \ldots & 1 & 0 \\
0 & \ldots & 0 & 0 & \ldots & 0 & \delta_d & \ldots & \delta_1 & 1 \\
\end{pmatrix}^{(T-d)\times T}$$

It can be shown that $\Omega_H = Q'Q$ and $\Omega_L = |\Omega_H|$, which simplifies the calculation of the cyclical component to

$$c = \lambda Q\{|Q'Q| + \lambda(Q'Q)^{-1}\}^{-1}Q'y$$

References


Also see

[TS] tsfilter — Filter a time series for cyclical components

[TS] tsset — Declare data to be time-series data

[TS] tssmooth — Smooth and forecast univariate time-series data

[D] format — Set variables’ output format

[XT] xtset — Declare data to be panel data
tsfilter cf — Christiano–Fitzgerald time-series filter

Description

\texttt{tsfilter cf} uses the Christiano and Fitzgerald (2003) band-pass filter to separate a time series into trend and cyclical components. The trend component may contain a deterministic or a stochastic trend. The stationary cyclical component is driven by stochastic cycles at the specified periods.

See \texttt{[TS] tsfilter} for an introduction to the methods implemented in \texttt{tsfilter cf}.

Quick start

Use the Christiano–Fitzgerald filter for \texttt{y} to obtain cyclical component \texttt{ct} using \texttt{tsset} data

\begin{verbatim}
  tsfilter cf ct=y
\end{verbatim}

As above, but filter for periods smaller than 5 and larger than 20

\begin{verbatim}
  tsfilter cf ct=y, minperiod(5) maxperiod(20)
\end{verbatim}

As above, and save the trend component in the variable \texttt{trendvar}

\begin{verbatim}
  tsfilter cf ct=y, minperiod(5) maxperiod(20) trend(trendvar)
\end{verbatim}

As above, and save gain and angular frequency with the names \texttt{gain} and \texttt{angle}

\begin{verbatim}
  tsfilter cf ct=y, minperiod(5) maxperiod(20) trend(trendvar) ///
  gain(gain angle)
\end{verbatim}

Use the Christiano–Fitzgerald filter for variables \texttt{y1}, \texttt{y2}, and \texttt{y3} to obtain cyclical components with prefix \texttt{cycl}

\begin{verbatim}
  tsfilter cf cycl*=y1 y2 y3
\end{verbatim}

Note: The above commands can also be used to apply the filter separately to each panel of a panel dataset when a \texttt{panelvar} has been specified by using \texttt{tsset} or \texttt{xtset}.

Menu

Statistics > Time series > Filters for cyclical components > Christiano-Fitzgerald
### Syntax

**Filter one variable**

```
    tsfilter cf [type] newvar = varname [if] [in] [, options]
```

**Filter multiple variables, unique names**

```
    tsfilter cf [type] newvarlist = varlist [if] [in] [, options]
```

**Filter multiple variables, common name stub**

```
    tsfilter cf [type] stub* = varlist [if] [in] [, options]
```

#### options

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
</tr>
<tr>
<td><code>minperiod(#)</code></td>
</tr>
<tr>
<td><code>maxperiod(#)</code></td>
</tr>
<tr>
<td><code>smaorder(#)</code></td>
</tr>
<tr>
<td><code>stationary</code></td>
</tr>
<tr>
<td><code>drift</code></td>
</tr>
<tr>
<td><strong>Trend</strong></td>
</tr>
<tr>
<td>`trend(newvar</td>
</tr>
<tr>
<td><strong>Gain</strong></td>
</tr>
<tr>
<td><code>gain(gainvar anglevar)</code></td>
</tr>
</tbody>
</table>

You must `tsset` or `xtset` your data before using `tsfilter`; see [TS] `tsset` and [XT] `xtset`. `varname` and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

### Options

#### Main

- `minperiod(#)` filters out stochastic cycles at periods smaller than #, where # must be at least 2 and less than `maxperiod()`. By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, or half-yearly, then # is set to the number of periods equivalent to 1.5 years; yearly data use `minperiod(2)`; otherwise, the default value is `minperiod(6)`.

- `maxperiod(#)` filters out stochastic cycles at periods larger than #, where # must be greater than `minperiod()`. By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, half-yearly, or yearly, then # is set to the number of periods equivalent to 8 years; otherwise, the default value is `maxperiod(32)`.

- `smaorder(#)` sets the order of the symmetric moving average, denoted by $q$. By default, `smaorder()` is not set, which invokes the asymmetric calculations for the Christiano–Fitzgerald filter. The order is an integer that specifies the number of observations in each direction used in calculating the symmetric moving average estimate of the cyclical component. This number must be an integer greater than zero and less than $\frac{T - 1}{2}$. The estimate of the cyclical component for the $t$th observation, $y_t$, is based upon the $2q + 1$ values $y_{t-q}, y_{t-q+1}, \ldots, y_t, y_{t+1}, \ldots, y_{t+q}$.
stationary modifies the filter calculations to those appropriate for a stationary series. By default, the series is assumed nonstationary.
drift removes drift using the approach described in Christiano and Fitzgerald (2003). By default, drift is not removed.

Trend

trend(newvar | newvarlist | stub*) saves the trend component(s) in the new variable(s) specified by newvar, newvarlist, or stub*.

Gain

gain(gainvar anglevar) saves the gain in gainvar and its associated angular frequency in anglevar. Gains are calculated at the $N$ angular frequencies that uniformly partition the interval $(0, \pi]$, where $N$ is the sample size.

Remarks and examples

We assume that you have already read [TS] tsfilter, which provides an introduction to filtering and the methods implemented in tsfilter cf, more examples using tsfilter cf, and a comparison of the four filters implemented by tsfilter. In particular, an understanding of gain functions as presented in [TS] tsfilter is required to understand these remarks.

tsfilter cf uses the Christiano–Fitzgerald (CF) band-pass filter to separate a time-series $y_t$ into trend and cyclical components

$$ y_t = \tau_t + c_t $$

where $\tau_t$ is the trend component and $c_t$ is the cyclical component. $\tau_t$ may be nonstationary; it may contain a deterministic or a stochastic trend, as discussed below.

The primary objective is to estimate $c_t$, a stationary cyclical component that is driven by stochastic cycles at a specified range of periods. The trend component $\tau_t$ is calculated by the difference $\tau_t = y_t - c_t$.

Although the CF band-pass filter implemented in tsfilter cf has been widely applied by macroeconomists, it is a general time-series method and may be of interest to other researchers.

As discussed by Christiano and Fitzgerald (2003) and in [TS] tsfilter, if one had an infinitely long series, one could apply an ideal band-pass filter that perfectly separates out cyclical components driven by stochastic cycles at the specified periodicities. In finite samples, it is not possible to exactly satisfy the conditions that a filter must fulfill to perfectly separate out the specified stochastic cycles; the expansive filter literature reflects the trade-offs involved in choosing a finite-length filter to separate out the specified stochastic cycles.

Christiano and Fitzgerald (2003) derive a finite-length CF band-pass filter that minimizes the mean squared error between the filtered series and the series filtered by an ideal band-pass filter that perfectly separates out components driven by stochastic cycles at the specified periodicities. Christiano and Fitzgerald (2003) place two important restrictions on the mean squared error problem that their filter solves. First, the CF filter is restricted to be a linear filter. Second, $y_t$ is assumed to be a random-walk process; in other words, $y_t = y_{t-1} + \epsilon_t$, where $\epsilon_t$ is independent and identically distributed with mean zero and finite variance. The CF filter is the best linear predictor of the series filtered by the ideal band-pass filter when $y_t$ is a random walk.
Christiano and Fitzgerald (2003) make four points in support of the random-walk assumption. First, the mean squared error problem solved by their filter requires that the process for $y_t$ be specified. Second, they provide a method for removing drift so that their filter handles cases in which $y_t$ is a random walk with drift. Third, many economic time series are well approximated by a random-walk-plus-drift process. (We add that many time series encountered in applied statistics are well approximated by a random-walk-plus-drift process.) Fourth, they provide simulation evidence that their filter performs well when the process generating $y_t$ is not a random-walk-plus-drift process but is close to being a random-walk-plus-drift process.

Comparing the CF filter with the Baxter–King (BK) filter provides some intuition and explains the `smaorder()` option in `tsfilter cf`. As discussed in [TS] `tsfilter` and Baxter and King (1999), symmetric moving-average (SMA) filters with coefficients that sum to zero can extract the components driven by stochastic cycles at specified periodicities when the series to be filtered has a deterministic or stochastic trend of order 1 or 2.

The coefficients of the finite-length BK filter are as close as possible to the coefficients of an ideal SMA band-pass filter under the constraints that the BK coefficients are symmetric and sum to zero. The coefficients of the CF filter are not symmetric nor do they sum to zero, but the CF filter was designed to filter out the specified periodicities when $y_t$ has a first-order stochastic trend.

To be robust to second-order trends, Christiano and Fitzgerald (2003) derive a constrained version of the CF filter. The coefficients of the constrained filter are constrained to be symmetric and to sum to zero. Subject to these constraints, the coefficients of the constrained CF filter minimize the mean squared error between the filtered series and the series filtered by an ideal band-pass filter that perfectly separates out the components. Christiano and Fitzgerald (2003) note that the higher-order detrending properties of this constrained filter come at the cost of lost efficiency. If the constraints are binding, the constrained filter cannot predict the series filtered by the ideal filter as well as the unconstrained filter can.

Specifying the `smaorder()` option causes `tsfilter cf` to compute the SMA-constrained CF filter.

The choice between the BK and the CF filters is one between robustness and efficiency. The BK filter handles a broader class of stochastic processes than does the CF filter, but the CF filter produces a better estimate of $c_t$ if $y_t$ is close to a random-walk process or a random-walk-plus-drift process.

Among economists, the CF filter is commonly used for investigating business cycles. Burns and Mitchell (1946) defined business cycles as stochastic cycles in business data corresponding to periods between 1.5 and 8 years. The default values for `minperiod()` and `maxperiod()` are the Burns–Mitchell values of 1.5 and 8 years scaled to the frequency of the dataset. The calculations of the default values assume that the time variable is formatted as daily, weekly, monthly, quarterly, half-yearly, or yearly; see [D] `format`.

When $y_t$ is assumed to be a random-walk-plus-drift process instead of a random-walk process, specify the `drift` option, which removes the linear drift in the series before applying the filter. Drift is removed by transforming the original series to a new series by using the calculation

$$z_t = y_t - \frac{(t - 1)(y_T - y_1)}{T - 1}$$

The cyclical component $c_t$ is calculated from drift-adjusted series $z_t$. The trend component $\tau_t$ is calculated by $\tau_t = y_t - c_t$.

By default, the CF filter assumes the series is nonstationary. If the series is stationary, the `stationary` option is used to change the calculations to those appropriate for a stationary series.

For each variable, the CF filter estimate of $c_t$ is put in the corresponding new variable, and when the `trend()` option is specified, the estimate of $\tau_t$ is put in the corresponding new variable.
tsfilter cf automatically detects panel data from the information provided when the dataset was `tsset` or `xtset`. All calculations are done separately on each panel. Missing values at the beginning and end of the sample are excluded from the sample. The sample may not contain gaps.

Example 1: Estimating a business-cycle component

In this and the subsequent examples, we use `tsfilter cf` to estimate the business-cycle component of the natural log of real gross domestic product (GDP) of the United States. Our sample of quarterly data goes from 1952q1 to 2010q4. Below we read in and plot the data.

```
. use https://www.stata-press.com/data/r16/gdp2
(Federal Reserve Economic Data, St. Louis Fed)
. tsline gdp_ln
```

![Plot of natural log of real GDP over time](image)

The series looks like it might be generated by a random-walk-plus-drift process and is thus a candidate for the CF filter.
Below we use `tsfilter cf` to filter `gdp_ln`, and we use `pergram` (see [TS] `pergram`) to compute and to plot the periodogram of the estimated cyclical component.

```
. tsfilter cf gdp_cf = gdp_ln
. pergram gdp_cf, xline(.03125 .16667)
```

Because our sample is of quarterly data, `tsfilter cf` used the default values of `minperiod(6)` and `maxperiod(32)`. The minimum and maximum periods are the Burns and Mitchell (1946) business-cycle periods for quarterly data.

In the periodogram, we added vertical lines at the natural frequencies corresponding to the conventional Burns and Mitchell (1946) values for business-cycle components. `pergram` displays the results in natural frequencies, which are the standard frequencies divided by $2\pi$. We use the `xline()` option to draw vertical lines at the lower natural-frequency cutoff ($1/32 = 0.03125$) and the upper natural-frequency cutoff ($1/6 \approx 0.16667$).

If the filter completely removed the stochastic cycles at the unwanted frequencies, the periodogram would be a flat line at the minimum value of $-6$ outside the range identified by the vertical lines.

The periodogram reveals that the CF did a reasonable job of filtering out the unwanted stochastic cycles.
Below we plot the estimated business-cycle component with recessions identified by the shaded areas.

![Graph showing estimated business-cycle component with recessions identified by shaded areas.](image)

### Stored results

**tsfilter cf** stores the following in **r()**:

- **Scalars**
  - `r(smaorder)`: order of the symmetric moving average, if specified
  - `r(minperiod)`: minimum period of stochastic cycles
  - `r(maxperiod)`: maximum period of stochastic cycles

- **Macros**
  - `r(varlist)`: original time-series variables
  - `r(filterlist)`: variables containing estimates of the cyclical components
  - `r(trendlist)`: variables containing estimates of the trend components, if `trend()` was specified
  - `r(method)`: Christiano–Fitzgerald
  - `r(symmetric)`: yes or no, indicating whether the symmetric version of the filter was or was not used
  - `r(drift)`: yes or no, indicating whether drift was or was not removed before filtering
  - `r(stationary)`: yes or no, indicating whether the calculations assumed the series was or was not stationary
  - `r(unit)`: units of time variable set using `tsset` or `xtset`

- **Matrices**
  - `r(filter)`: $(q+1) \times 1$ matrix of weights $(\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_q)'$, where $q$ is the order of the symmetric moving average, and the weights are the Christiano–Fitzgerald coefficients; only returned when `smaorder()` is used to set $q$

### Methods and formulas

For an infinitely long series, there is an ideal band-pass filter that extracts the cyclical component by using the calculation

$$c_t = \sum_{j=-\infty}^{\infty} b_j y_{t-j}$$

If $p_l$ and $p_h$ are the minimum and maximum periods of the stochastic cycles of interest, the weights $b_j$ in the ideal band-pass filter are given by
\[ b_j = \begin{cases} \pi^{-1}(\omega_h - \omega_l) & \text{if } j = 0 \\ (j\pi)^{-1}\{\sin(j\omega_h) - \sin(j\omega_l)\} & \text{if } j \neq 0 \end{cases} \]

where \( \omega_l = 2\pi/p_l \) and \( \omega_h = 2\pi/p_h \) are the lower and higher cutoff frequencies, respectively.

Because our time series has finite length, the ideal band-pass filter cannot be computed exactly. Christiano and Fitzgerald (2003) derive the finite-length CF band-pass filter that minimizes the mean squared error between the filtered series and the series filtered by an ideal band-pass filter that perfectly separates out the components. This filter is not symmetric nor do the coefficients sum to zero. The formula for calculating the value of cyclical component \( c_t \) for \( t = 2, 3, \ldots, T - 1 \) using the asymmetric version of the CF filter can be expressed as

\[
c_t = b_0 y_t + \sum_{j=1}^{T-t-1} b_j y_{t+j} + \tilde{b}_{T-t} y_T + \sum_{j=1}^{t-2} b_j y_{t-j} + \tilde{b}_{t-1} y_1
\]

where \( b_0, b_1, \ldots \) are the weights used by the ideal band-pass filter. \( \tilde{b}_{T-t} \) and \( \tilde{b}_{t-1} \) are linear functions of the ideal weights used in this calculation. The CF filter uses two different calculations for \( \tilde{b}_{T-t} \) depending upon whether the series is assumed to be stationary or nonstationary.

For the default nonstationary case with \( 1 < t < T \), Christiano and Fitzgerald (2003) set \( \tilde{b}_{T-t} \) and \( \tilde{b}_{t-1} \) to

\[
\tilde{b}_{T-t} = -\frac{1}{2} b_0 - \sum_{j=1}^{T-t-1} b_j \quad \text{and} \quad \tilde{b}_{t-1} = -\frac{1}{2} b_0 - \sum_{j=1}^{t-2} b_j
\]

which forces the weights to sum to zero.

For the nonstationary case, when \( t = 1 \) or \( t = T \), the two endpoints (\( c_1 \) and \( c_T \)) use only one modified weight, \( \tilde{b}_{T-1} \):

\[
c_1 = \frac{1}{2} b_0 y_1 + \sum_{j=1}^{T-2} b_j y_{j+1} + \tilde{b}_{T-1} y_T \quad \text{and} \quad c_T = \frac{1}{2} b_0 y_T + \sum_{j=1}^{T-2} b_j y_{T-j} + \tilde{b}_{T-1} y_1
\]

When the stationary option is used to invoke the stationary calculations, all weights are set to the ideal filter weight, that is, \( \tilde{b}_j = b_j \).

If the \( \text{smaorder()} \) option is set, the symmetric version of the CF filter is used. This option specifies the length of the symmetric moving average denoted by \( q \). The symmetric calculations for \( c_t \) are similar to those used by the BK filter:

\[
c_t = \hat{b}_q \{L^{-q}(y_t) + L^q(y_t)\} + \sum_{j=-q+1}^{q-1} b_j L^j(y_t)
\]

where, for the default nonstationary calculations, \( \hat{b}_q = -(1/2)b_0 - \sum_{j=1}^{q-1} b_j \). If the \( \text{smaorder()} \) and stationary options are set, then \( \hat{b}_q \) is set equal to the ideal weight \( b_q \).
References


Also see

[TS] `tsfilter` — Filter a time series for cyclical components

[TS] `tsset` — Declare data to be time-series data

[TS] `tssmooth` — Smooth and forecast univariate time-series data

[D] `format` — Set variables’ output format

[XT] `xtset` — Declare data to be panel data
Description

tsfiler hp uses the Hodrick–Prescott high-pass filter to separate a time series into trend and cyclical components. The trend component may contain a deterministic or a stochastic trend. The smoothing parameter determines the periods of the stochastic cycles that drive the stationary cyclical component.

See [TS] tsfilter for an introduction to the methods implemented in tsfilter hp.

Quick start

Use the Hodrick–Prescott filter for y to obtain cyclical component ct using tsset data

    tsfilter hp ct=y

As above, and save the trend component in the variable trendvar

    tsfilter hp ct=y, trend(trendvar)

As above, and save gain and angular frequency with the names gain and angle

    tsfilter hp ct=y, trend(trendvar) gain(gain angle)

As above, but set the Hodrick–Prescott smoothing parameter to be 1700

    tsfilter hp ct=y, trend(trendvar) gain(gain angle) smooth(1700)

Use Hodrick–Prescott filter for variables y1, y2, and y3 to obtain cyclical components with prefix cycl

    tsfilter hp cycl*=y1 y2 y3

Note: The above commands can also be used to apply the filter separately to each panel of a panel dataset when a panelvar has been specified by using tsset or xtset.

Menu

Statistics > Time series > Filters for cyclical components > Hodrick-Prescott
Syntax

Filter one variable

```
  tsfilter hp [type] newvar = varname [if] [in] [, options]
```

Filter multiple variables, unique names

```
  tsfilter hp [type] newvarlist = varlist [if] [in] [, options]
```

Filter multiple variables, common name stub

```
  tsfilter hp [type] stub* = varlist [if] [in] [, options]
```

**options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth(#)</td>
</tr>
<tr>
<td>Trend</td>
</tr>
<tr>
<td>trend(newvar</td>
</tr>
<tr>
<td>Gain</td>
</tr>
<tr>
<td>gain(gainvar anglevar)</td>
</tr>
</tbody>
</table>

You must `tsset` or `xtset` your data before using `tsfilter`; see [TS] `tsset` and [XT] `xtset`. `varname` and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

**Options**

**Main**

- `smooth(#)` sets the smoothing parameter for the Hodrick–Prescott filter. By default, if the units of the time variable are set to daily, weekly, monthly, quarterly, half-yearly, or yearly, then the Ravn–Uhlig rule is used to set the smoothing parameter; otherwise, the default value is `smooth(1600)`. The Ravn–Uhlig rule sets # to 1600\(p^4\), where \(p\) is the number of periods per quarter. The smoothing parameter must be greater than 0.

**Trend**

- `trend(newvar | newvarlist | stub*)` saves the trend component(s) in the new variable(s) specified by `newvar`, `newvarlist`, or `stub*`.

**Gain**

- `gain(gainvar anglevar)` saves the gain in `gainvar` and its associated angular frequency in `anglevar`. Gains are calculated at the \(N\) angular frequencies that uniformly partition the interval \((0, \pi]\), where \(N\) is the sample size.
Remarks and examples

We assume that you have already read [TS] tsfilter, which provides an introduction to filtering and the methods implemented in tsfilter hp, more examples using tsfilter hp, and a comparison of the four filters implemented by tsfilter. In particular, an understanding of gain functions as presented in [TS] tsfilter is required to understand these remarks.

tsfilter hp uses the Hodrick–Prescott (HP) high-pass filter to separate a time-series \( y_t \) into trend and cyclical components

\[
y_t = \tau_t + c_t
\]

where \( \tau_t \) is the trend component and \( c_t \) is the cyclical component. \( \tau_t \) may be nonstationary; it may contain a deterministic or a stochastic trend, as discussed below.

The primary objective is to estimate \( c_t \), a stationary cyclical component that is driven by stochastic cycles at a range of periods. The trend component \( \tau_t \) is calculated by the difference \( \tau_t = y_t - c_t \).

Although the HP high-pass filter implemented in tsfilter hp has been widely applied by macroeconomists, it is a general time-series method and may be of interest to other researchers.

Hodrick and Prescott (1997) motivated the HP filter as a trend-removal technique that could be applied to data that came from a wide class of data-generating processes. In their view, the technique specified a trend in the data and the data was filtered by removing the trend. The smoothness of the trend depends on a parameter \( \lambda \). The trend becomes smoother as \( \lambda \to \infty \), and Hodrick and Prescott (1997) recommended setting \( \lambda \) to 1,600 for quarterly data.

King and Rebelo (1993) showed that removing a trend estimated by the HP filter is equivalent to a high-pass filter. They derived the gain function of this high-pass filter and showed that the filter would make integrated processes of order 4 or less stationary, making the HP filter comparable to the other filters implemented in tsfilter.

Example 1: Estimating a business-cycle component

In this and the subsequent examples, we use tsfilter hp to estimate the business-cycle component of the natural log of real gross domestic product (GDP) of the United States. Our sample of quarterly data goes from 1952q1 to 2010q4. Below we read in and plot the data.

```
. use https://www.stata-press.com/data/r16/gdp2
   (Federal Reserve Economic Data, St. Louis Fed)
. tsline gdp_ln
```

![Graph showing the natural log of real GDP from 1950q1 to 2010q1]
The series is nonstationary and is thus a candidate for the HP filter.

Below we use `tsfilter hp` to filter `gdp_ln`, and we use `pergram` (see [TS] `pergram`) to compute and to plot the periodogram of the estimated cyclical component.

```
.tsfilter hp gdp_hp = gdp_ln
.pergram gdp_hp, xline(.03125 .16667)
```

Because our sample is of quarterly data, `tsfilter hp` used the default value for the smoothing parameter of 1,600.

In the periodogram, we added vertical lines at the natural frequencies corresponding to the conventional Burns and Mitchell (1946) values for business-cycle components of 32 periods and 6 periods. `pergram` displays the results in natural frequencies, which are the standard frequencies divided by $2\pi$. We use the `xline()` option to draw vertical lines at the lower natural-frequency cutoff ($1/32 = 0.03125$) and the upper natural-frequency cutoff ($1/6 \approx 0.16667$).

If the filter completely removed the stochastic cycles at the unwanted frequencies, the periodogram would be a flat line at the minimum value of $-6$ outside the range identified by the vertical lines.

The periodogram reveals a high-periodicity issue and a low-periodicity issue. The points above $-6.00$ to the left of the left-hand vertical line in the periodogram reveal that the filter did not do a good job of filtering out the high-periodicity stochastic cycles with the default value smoothing parameter of 1,600. That there is no tendency of the points to the right of the right-hand vertical line to be smoothed toward $-6.00$ reveals that the HP filter did not remove any of the low-periodicity stochastic cycles. This result is not surprising, because the HP filter is a high-pass filter.

In the next example, we address the high-periodicity issue. See [TS] `tsfilter` and [TS] `tsfilter bw` for how to turn a high-pass filter into a band-pass filter.

**Example 2: Choosing the filter parameters**

In the filter literature, filter parameters are set as functions of the cutoff frequency; see Pollock (2000, 324), for instance. This method finds the filter parameter that sets the gain of the filter equal to $1/2$ at the cutoff frequency. In a technical note in [TS] `tsfilter`, we showed that applying this method to selecting $\lambda$ at the cutoff frequency of 32 periods suggests setting $\lambda \approx 677.13$. In the output below, we
estimate the business-cycle component using this value for the smoothing parameter, and we compute and plot the periodogram of the estimated business-cycle component.

```
.tsfilter hp gdp_hp677 = gdp_ln, smooth(677.13)
.pergram gdp_hp677, xline(.03125 .16667)
```

A comparison of the two periodograms reveals that setting the smoothing parameter to 677.13 removes more of the high-periodicity stochastic cycles than does the default 1,600. In [TS] tsfilter, we found that the HP filter was not as good at removing the high-periodicity stochastic cycles as was the Christiano–Fitzgerald filter implemented in tsfilter cf or as was the Butterworth filter implemented in tsfilter bw.

Below we plot the estimated business-cycle component with recessions identified by the shaded areas.

```
tsfilter hp — Hodrick–Prescott time-series filter 637
```

`tsfilter hp` automatically detects panel data from the information provided when the dataset was `tsset` or `xtset`. All calculations are done separately on each panel. Missing values at the beginning and end of the sample are excluded from the sample. The sample may not contain gaps.
tsfilter hp stores the following in r():

Scalars
r(smooth) smoothing parameter \( \lambda \)

Macros
r(varlist) original time-series variables
r(filterlist) variables containing estimates of the cyclical components
r(trendlist) variables containing estimates of the trend components, if trend() was specified
r(method) Hodrick–Prescott
r(unit) units of time variable set using tsset or xtset

Methods and formulas

Formally, the filter is defined as the solution to the following optimization problem for \( \tau_t \)

\[
\min_{\tau_t} \left[ \sum_{t=1}^{T} (y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T-1} \left\{ (\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1}) \right\}^2 \right]
\]

where the smoothing parameter \( \lambda \) is set fixed to a value.

If \( \lambda = 0 \), the solution degenerates to \( \tau_t = y_t \), in which case the filter excludes all frequencies, that is, \( c_t = 0 \). On the other extreme, as \( \lambda \to \infty \), the solution approaches the least-squares fit to the line \( \tau_t = \beta_0 + \beta_1 t \); see Hodrick and Prescott (1997) for a discussion.

For a fixed \( \lambda \), it can be shown that the cyclical component \( c' = (c_1, c_2, \ldots, c_T) \) is calculated by

\[
c = (I_T - M^{-1})y
\]

where \( y \) is the column vector \( y' = (y_1, y_2, \ldots, y_T) \), \( I_T \) is the \( T \times T \) identity matrix, and \( M \) is the \( T \times T \) matrix:

\[
M = \begin{pmatrix}
(1 + \lambda) & -2\lambda & \lambda & 0 & 0 & 0 & \ldots & 0 \\
-2\lambda & (1 + 5\lambda) & -4\lambda & \lambda & 0 & 0 & \ldots & 0 \\
\lambda & -4\lambda & (1 + 6\lambda) & -4\lambda & \lambda & 0 & \ldots & 0 \\
0 & \lambda & -4\lambda & (1 + 6\lambda) & -4\lambda & \lambda & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \ldots & \lambda & -4\lambda & (1 + 6\lambda) & -4\lambda & \lambda & 0 \\
0 & \ldots & 0 & \lambda & -4\lambda & (1 + 6\lambda) & -4\lambda & \lambda \\
0 & \ldots & 0 & 0 & \lambda & -4\lambda & (1 + 5\lambda) & -2\lambda \\
0 & \ldots & 0 & 0 & 0 & \lambda & -2\lambda & (1 + \lambda)
\end{pmatrix}
\]

The gain of the HP filter is given by (see King and Rebelo [1993], Maravall and del Rio [2007], or Harvey and Trimbur [2008])

\[
\psi(\omega) = \frac{4\lambda(1 - \cos(\omega))^2}{1 + 4\lambda(1 - \cos(\omega))^2}
\]
As discussed in [TS] tsfilter, there are two approaches to selecting $\lambda$. One method, based on the heuristic argument of Hodrick and Prescott (1997), is used to compute the default values for $\lambda$. The method sets $\lambda$ to 1,600 for quarterly data and to the rescaled values worked out by Ravn and Uhlig (2002). The rescaled default values for $\lambda$ are 6.25 for yearly data, 100 for half-yearly data, 129,600 for monthly data, $1600 \times 12^4$ for weekly data, and $1600 \times (365/4)^4$ for daily data.

The second method for selecting $\lambda$ uses the recommendations of Pollock (2000, 324), who uses the gain function of the filter to identify a value for $\lambda$.

Additional literature critiques the HP filter by pointing out that the HP filter corresponds to a specific model. Harvey and Trimbur (2008) show that the cyclical component estimated by the HP filter is equivalent to one estimated by a particular unobserved-components model. Harvey and Jaeger (1993), Gómez (1999), Pollock (2000), and Gómez (2001) also show this result and provide interesting comparisons of estimating $c_t$ by filtering and model-based methods.

References


Also see

[TS] tsfilter — Filter a time series for cyclical components
[TS] tsset — Declare data to be time-series data
[TS] tssmooth — Smooth and forecast univariate time-series data
[D] format — Set variables’ output format
[XT] xtset — Declare data to be panel data
tsline — Time-series line plots

Description


tsline draws line plots for time-series data.

tsrline draws a range plot with lines for time-series data.

Quick start

Line plot for the time series y1 using tsset data

\texttt{tsline y1}

Add plots of time series y2 and y3

\texttt{tsline y1 y2 y3}

Range plot with lines for the lower and upper values of time series y1 stored in y1_lower and y1_upper, respectively

\texttt{tsrline y1_lower y1_upper}

Overlay a range plot of the lower and upper values of time series y1 stored in y1_lower and y1_upper, respectively, on a plot of y1

\texttt{tsline y1 || tsrline y1_lower y1_upper}

Menu

Statistics > Time series > Graphs > Line plots
Syntax

Time-series line plot

\[
\text{twoway} \ \text{tsline} \ \text{varlist} \ [\text{if}] \ [\text{in}] \ [, \ tsline\_options]
\]

Time-series range plot with lines

\[
\text{twoway} \ \text{tsrline} \ y_1 \ y_2 \ [\text{if}] \ [\text{in}] \ [, \ tsrline\_options]
\]

where the time variable is assumed set by `tsset` (see [TS] tsset), `varlist` has the interpretation
\[y_1, y_2, \ldots, y_k\].

<table>
<thead>
<tr>
<th>tsline_options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scatter_options</td>
<td>any options documented in [G-2] <code>graph twoway scatter</code> with the exception of <code>marker\_options</code>, <code>marker\_placement\_options</code>, and <code>marker\_label\_options</code>, which will be ignored if specified</td>
</tr>
<tr>
<td><code>Y</code> axis, <code>Time</code> axis, <code>Titles</code>, <code>Legend</code>, <code>Overall</code>, <code>By</code></td>
<td></td>
</tr>
<tr>
<td>twoway_options</td>
<td>any options documented in [G-3] <code>twoway\_options</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tsrline_options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rline_options</td>
<td>any options documented in [G-2] <code>graph twoway rline</code></td>
</tr>
<tr>
<td><code>Y</code> axis, <code>Time</code> axis, <code>Titles</code>, <code>Legend</code>, <code>Overall</code>, <code>By</code></td>
<td></td>
</tr>
<tr>
<td>twoway_options</td>
<td>any options documented in [G-3] <code>twoway\_options</code></td>
</tr>
</tbody>
</table>

Options

Plots

| `scatter\_options` | any of the options allowed by the `graph twoway scatter` command except that `marker\_options`, `marker\_placement\_option`, and `marker\_label\_options` will be ignored if specified; see [G-2] `graph twoway scatter`. |

| `rline\_options` | any of the options allowed by the `graph twoway rline` command; see [G-2] `graph twoway rline`. |

`twoway\_options` are any of the options documented in [G-3] `twoway\_options`. These include options for titling the graph (see [G-3] `title\_options`), options for saving the graph to disk (see [G-3] `saving\_option`), and the `by()` option, which will allow you to simultaneously plot different subsets of the data (see [G-3] `by\_option`).

Also see the `recast()` option discussed in [G-3] `advanced\_options` for information on how to plot spikes, bars, etc., instead of lines.
Remarks and examples

Remarks are presented under the following headings:

Basic examples
Advanced example
Video example

Basic examples

Example 1: A time-series line plot

We simulated two separate time series (each of 200 observations) and placed them in a Stata dataset, `tsline1.dta`. The first series simulates an AR(2) process with $\phi_1 = 0.8$ and $\phi_2 = 0.2$; the second series simulates an MA(2) process with $\theta_1 = 0.8$ and $\theta_2 = 0.2$. We use `tsline` to graph these two series.

```
. use https://www.stata-press.com/data/r16/tsline1
. tsset lags
time variable:  lags, 0 to 199
delta:  1 unit
. tsline ar ma
```

Example 2: Using options to highlight information

Suppose that we kept a calorie log for an entire calendar year. At the end of the year, we would have a dataset (for example, `tsline2.dta`) that contains the number of calories consumed for 365 days. We could then use `tsset` to identify the date variable and `tsline` to plot calories versus time. Knowing that we tend to eat a little more food on Thanksgiving and Christmas day, we use the `ttick()` and `ttext()` options to point out these days on the time axis.
. use https://www.stata-press.com/data/r16/tsline2
. tsset day
    time variable:  day, 01jan2002 to 31dec2002
    delta: 1 day
. tsline calories, ttick(28nov2002 25dec2002, tpos(in))
> ttext(3470 28nov2002 "thanks" 3470 25dec2002 "x-mas", orient(vert))

Options associated with the time axis allow dates (and times) to be specified in place of numeric
date (and time) values. For instance, we used

    ttick(28nov2002 25dec2002, tpos(in))

to place tick marks at the specified dates. This works similarly for tlabel(), tmlabel(), and
tmtick().

Suppose that we wanted to place vertical lines for the previously mentioned holidays. We could
specify the dates in the tline() option as follows:

    . tsline calories, tline(28nov2002 25dec2002)
Example 3: Formatting the time axis

We could also modify the format of the time axis so that the labeled ticks display only the day in the year:

```
.tsline calories, tlabel(, format(%tdmd)) ttitle("Date (2002)"
```

![Graph showing time-series line plots with modified time axis format](image)

Advanced example

tline and tsrline are both commands and plottypes as defined in [G-2] graph twoway. Thus the syntax for tline is

```
.graph twoway tline ... 
.twoway tline ... 
.tsline ...
```

and similarly for tsrline. Being plot types, these commands may be combined with other plot types in the twoway family, as in,

```
.twoway (tsrline ...) (tline ...) (lfit ...) ...
```

which can equivalently be written as

```
.tsrline ... || tline ... || lfit ... || ...
```
Example 4: Combining line and range plots

In the first plot of example 2, we were uncertain of the exact values we logged, so we also gave a range for each day. Here is a plot of the summer months.

```
.tsline lcalories ucalories if tin(1may2002,31aug2002) || tsline calories ||
> if tin(1may2002,31aug2002), ytitle(Calories)
```

Video example

Line graphs and tin()

References


Also see

[TS] *tsset* — Declare data to be time-series data

[G-2] *graph twoway* — Two-way graphs

[XT] *xtline* — Panel-data line plots
Description

treport reports time gaps in a dataset or in a subset of variables. By default, treport reports periods in which no information is recorded in the dataset; the time variable does not include these periods. When you specify varlist, treport reports periods in which either no information is recorded in the dataset or the time variable is present, but one or more variables in varlist contain a missing value.

Quick start

Report time gaps in a tset time-series dataset

treport

Report time gaps for the variable y

treport y

As above, and report the beginning and ending times of each gap

treport y, detail

Report time gaps, ignoring panel changes, using tset or xtset data

treport, panel

Menu

Statistics > Time series > Setup and utilities > Report time-series aspects of dataset
tsreport — Report time-series aspects of a dataset or estimation sample

Syntax

```
. tsreport [varlist] [if] [in] [, options]
```

### Options

**Main**

- `detail` reports the beginning and ending times of each gap.
- `casewise` specifies that a period for which any of the specified variables are missing be counted as a gap. By default, gaps are reported for each variable individually.
- `panel` specifies that panel changes not be counted as gaps. Whether panel changes are counted as gaps usually depends on how the calling command handles panels.

`varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

### Remarks and examples

Remarks are presented under the following headings:

- Basic examples
- Video example

### Basic examples

Time-series commands sometimes require that observations be on a fixed time interval with no gaps, or the command may not function properly. `tsreport` provides a tool for reporting the gaps in a sample.

#### Example 1: A simple panel-data example

The following monthly panel data have two panels and a missing month (March) in the second panel:

```
. use https://www.stata-press.com/data/r16/tsrptxmpl
. list edlevel month income in 1/6, sep(0)
```

<table>
<thead>
<tr>
<th>edlevel</th>
<th>month</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1 1998m1</td>
<td>687</td>
</tr>
<tr>
<td>2.</td>
<td>1 1998m2</td>
<td>783</td>
</tr>
<tr>
<td>3.</td>
<td>1 1998m3</td>
<td>790</td>
</tr>
<tr>
<td>4.</td>
<td>2 1998m1</td>
<td>1435</td>
</tr>
<tr>
<td>5.</td>
<td>2 1998m2</td>
<td>1522</td>
</tr>
<tr>
<td>6.</td>
<td>2 1998m4</td>
<td>1532</td>
</tr>
</tbody>
</table>
In the following example on panel data, we observe:

```
. tsreport
Panel variable: edlevel
Time variable: month

Starting period = 1998m1
Ending period = 1998m4
Observations = 6
Number of gaps = 2
(Gap count includes panel changes)
```

Two gaps are reported in the sample. We know the second panel is missing the month of March, but where is the second gap? The note at the bottom of the output is telling us something about panel changes. Let’s use the `detail` option to get more information:

```
. tsreport, detail
Panel variable: edlevel
Time variable: month

Starting period = 1998m1
Ending period = 1998m4
Observations = 6
Number of gaps = 2
(Gap count includes panel changes)

Gap report

<table>
<thead>
<tr>
<th>Obs.</th>
<th>edlevel</th>
<th>Start</th>
<th>End</th>
<th>N. Obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>1998m4</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5—6</td>
<td>2</td>
<td>1998m3</td>
<td>1998m3</td>
<td>1</td>
</tr>
</tbody>
</table>
```

We now see what is happening. `tsreport` is counting the change from the first panel to the second panel as a gap. Look at the output from the `list` command above. The value of `month` in observation 4 is not one month later than the value of `month` in observation 3, so `tsreport` reports a gap. (If we are programmers writing a procedure that does not account for panels, a change from one panel to the next represents a break in the time series just as a gap in the data does.) For the second gap, `tsreport` indicates that just one observation is missing because we are only missing the month of March. This gap is between observations 5 and 6 of the data.

In other cases, we may not care about changes in panels and not want them counted as gaps. We can use the `panel` option to specify that `tsreport` should ignore panel changes:

```
. tsreport, detail panel
Panel variable: edlevel
Time variable: month

Starting period = 1998m1
Ending period = 1998m4
Observations = 6
Number of gaps = 1

Gap report

<table>
<thead>
<tr>
<th>Obs.</th>
<th>edlevel</th>
<th>Start</th>
<th>End</th>
<th>N. Obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5—6</td>
<td>2</td>
<td>1998m3</td>
<td>1998m3</td>
<td>1</td>
</tr>
</tbody>
</table>
```

tsreport now indicates there is just one gap, corresponding to March for the second panel.
Example 2: Variables with missing data

We asked two large hotels in Las Vegas to record the prices they were quoting people who called to make reservations. Because these prices change frequently in response to promotions and market conditions, we asked the hotels to record their prices hourly. Unfortunately, the managers did not consider us a top priority, so we are missing some data. Our dataset looks like this:

```
use https://www.stata-press.com/data/r16/hotelprice
list, sep(0)
```

<table>
<thead>
<tr>
<th></th>
<th>hour</th>
<th>price1</th>
<th>price2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13feb2007</td>
<td>140</td>
<td>245</td>
</tr>
<tr>
<td>2</td>
<td>13feb2007</td>
<td>155</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>13feb2007</td>
<td>.</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>13feb2007</td>
<td>155</td>
<td>250</td>
</tr>
<tr>
<td>5</td>
<td>13feb2007</td>
<td>160</td>
<td>255</td>
</tr>
<tr>
<td>6</td>
<td>13feb2007</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>13feb2007</td>
<td>165</td>
<td>255</td>
</tr>
<tr>
<td>8</td>
<td>13feb2007</td>
<td>170</td>
<td>260</td>
</tr>
<tr>
<td>9</td>
<td>13feb2007</td>
<td>175</td>
<td>265</td>
</tr>
<tr>
<td>10</td>
<td>13feb2007</td>
<td>180</td>
<td>.</td>
</tr>
<tr>
<td>11</td>
<td>13feb2007</td>
<td>190</td>
<td>270</td>
</tr>
</tbody>
</table>

First, let's invoke `tsreport` without specifying `price1` or `price2`. We will specify the `detail` option so that we can see the periods corresponding to the gap or gaps reported:

```
.tsreport, detail
```

```
Time variable: hour
Starting period = 13feb2007 08:00:00
Ending period = 13feb2007 20:00:00
Observations = 11
Number of gaps = 1
Gap report

<table>
<thead>
<tr>
<th>Obs.</th>
<th>Start</th>
<th>End</th>
<th>N. Obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>10–11</td>
<td>13feb2007 18:00:00</td>
<td>13feb2007 19:00:00</td>
<td>2</td>
</tr>
</tbody>
</table>
```

One gap is reported, lasting two periods. We have no data corresponding to 6:00 p.m. and 7:00 p.m. on February 13, 2007.

What about observations 3, 6, and 10? We are missing data on one or both of the price variables for those observations, but the time variable itself is present for those observations. By default, `tsreport` defines gaps as periods in which no information, not even the time variable itself, is recorded.

If we instead want to obtain information about when one or more variables are missing information, then we specify those variables in our call to `tsreport`. Here we specify `price1`, first without the `detail` option:

```
.tsreport price1
```

```
Gap summary report

<table>
<thead>
<tr>
<th>Variable</th>
<th>Start</th>
<th>End</th>
<th>Number of—</th>
<th>—Number of—</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Obs.</td>
<td>Gaps</td>
</tr>
<tr>
<td>price1</td>
<td>13feb2007 08:00:00</td>
<td>13feb2007 20:00:00</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>
```
The output indicates that we have data on \texttt{price1} from 8:00 a.m. to 8:00 p.m. However, we only have 9 observations on \texttt{price1} during that span because we have 3 gaps in the data. Let's specify the \texttt{detail} option to find out where:

```
.tsreport price1, detail
Variable: price1
Time variable: hour
```

Starting period = 13feb2007 08:00:00
Ending period = 13feb2007 20:00:00
Observations = 9
Number of gaps = 3

<table>
<thead>
<tr>
<th>Obs.</th>
<th>Start</th>
<th>End</th>
<th>N. Obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2—4</td>
<td>13feb2007 10:00:00</td>
<td>13feb2007 10:00:00</td>
<td>1</td>
</tr>
<tr>
<td>5—7</td>
<td>13feb2007 13:00:00</td>
<td>13feb2007 13:00:00</td>
<td>1</td>
</tr>
<tr>
<td>10—11</td>
<td>13feb2007 18:00:00</td>
<td>13feb2007 19:00:00</td>
<td>2</td>
</tr>
</tbody>
</table>

The three gaps correspond to observations 3 and 6, for which \texttt{price1} is missing, as well as the two-period gap in the evening when not even the time variable is recorded in the dataset.

When you specify multiple variables with \texttt{tsreport}, by default, it summarizes gaps in each variable separately. Apart from combining the information into one table, typing

```
.tsreport price1 price2
```

is almost the same as typing

```
.tsreport price1
.tsreport price2
```

The only difference between the two methods is that the former stores results for both variables in r-class macros for later use, whereas if you were to type the latter two commands in succession, r-class macros would only contain results for \texttt{price2}.

In many types of analyses, including linear regression, you can only use observations for which all the variables contain nonmissing data. Similarly, you can have \texttt{tsreport} report as gaps periods in which any of the specified variables contain missing values. To do that, you use the \texttt{casewise} option.

\begin{itemize}
\item \textbf{Example 3: Casewise analyses}
\end{itemize}

Continuing with our hotel data, we specify both \texttt{price1} and \texttt{price2} in the variable list of \texttt{tsreport}. We request \texttt{casewise} analysis, and we specify the \texttt{detail} option to get information on each gap \texttt{tsreport} finds.
. tsreport price1 price2, casewise detail
Variables: price1 and price2
Time variable: hour

Starting period = 13feb2007 08:00:00
Ending period = 13feb2007 20:00:00
Observations = 8
Number of gaps = 3

Gap report

<table>
<thead>
<tr>
<th>Obs.</th>
<th>Start</th>
<th>End</th>
<th>N. Obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2—4</td>
<td>13feb2007 10:00:00</td>
<td>13feb2007 10:00:00</td>
<td>1</td>
</tr>
<tr>
<td>5—7</td>
<td>13feb2007 13:00:00</td>
<td>13feb2007 13:00:00</td>
<td>1</td>
</tr>
<tr>
<td>9—11</td>
<td>13feb2007 17:00:00</td>
<td>13feb2007 19:00:00</td>
<td>3</td>
</tr>
</tbody>
</table>

The first gap reported by tsreport corresponds to observation 3, when price1 is missing, and the second gap corresponds to observation 6, when both price1 and price2 are missing. The third gap spans 3 observations: the 5:00 p.m. observation is missing for price2, and as we discovered earlier, not even the time variable is present at 6:00 p.m. and 7:00 p.m.

Video example

Formatting and managing dates

Stored results

treport, when no varlist is specified or when casewise is specified, stores the following in r():

Scalars
- r(N_gaps) number of gaps
- r(N_obs) number of observations
- r(start) first time in series
- r(end) last time in series

Macros
- r(panelvar) name of panel variable
- r(timevar) name of time variable
- r(tsfmt) %fmt of time variable

Matrices
- r(table) matrix containing start and end times of each gap, if detail is specified
tsreport, when a varlist is specified and casewise is not specified, stores the following in r():

Scalars
- \( r(N_{-}gaps#) \): number of gaps for variable #
- \( r(N_{-}obs#) \): number of observations for variable #
- \( r(start#) \): first time in series for variable #
- \( r(end#) \): last time in series for variable #

Macros
- \( r(panelvar) \): name of panel variable
- \( r(timevar) \): name of time variable
- \( r(tsfmt) \): %fmt of time variable
- \( r(var#) \): name of variable #

Matrices
- \( r(table#) \): matrix containing start and end times of each gap for variable #, if detail is specified

When \( k \) variables are specified in varlist, # ranges from 1 to \( k \).

Also see

[TS] tsset — Declare data to be time-series data
**tsrevar — Time-series operator programming command**

### Description

**tsrevar, substitute** takes a *varlist* that might contain *op.varname* combinations and substitutes equivalent temporary variables for the combinations.

**tsrevar, list** creates no new variables. It returns in `r(varlist)` the list of base variables corresponding to *varlist*.

### Quick start

Create temporary variables containing the first lag and difference of y using *tsset* data, and store the temporary variable names in `r(varlist)`

```stata
  tsrevar l.y d.y
```

Store the name of the base variable, y, in `r(varlist)`, and do not create any temporary variables

```stata
  tsrevar l.y d.y, list
```

### Syntax

```
  tsrevar [varlist] [if] [in] [ , substitute list ]
```

You must *tsset* your data before using *tsrevar*; see [TS] *tsset*.

### Options

- `substitute` specifies that *tsrevar* resolve *op.varname* combinations by creating temporary variables as described above. `substitute` is the default action taken by *tsrevar*; you do not need to specify the option.
- `list` specifies that *tsrevar* return a list of base variable names.

### Remarks and examples

*tsrevar* substitutes temporary variables for any *op.varname* combinations in a variable list. For instance, the original *varlist* might be “`gnp L.gnp r`”, and *tsrevar, substitute* would create `newvar = L.gnp` and create the equivalent varlist “`gnp newvar r`”. This new varlist could then be used with commands that do not otherwise support time-series operators, or it could be used in a program to make execution faster at the expense of using more memory.
tsrevar, substitute might create no new variables, one new variable, or many new variables, depending on the number of \textit{op.varname} combinations appearing in \textit{varlist}. Any new variables created are temporary. The new, equivalent \textit{varlist} is returned in \texttt{r(varlist)}. The new \textit{varlist} corresponds one to one with the original \textit{varlist}.

\texttt{tsrevar, list} returns in \texttt{r(varlist)} the list of base variable names of \textit{varlist} with the time-series operators removed. \texttt{tsrevar, list} creates no new variables. For instance, if the original \textit{varlist} were \texttt{"gnp 1.gnp 12.gnp r l.cd"}, then \texttt{r(varlist)} would contain \texttt{"gnp r cd"}. This is useful for programmers who might want to create programs to keep only the variables corresponding to \textit{varlist}.

\begin{itemize}
\item \textbf{Example 1}
\end{itemize}

\begin{verbatim}
. use https://www.stata-press.com/data/r16/tsrevar\textbackslash ex
. tsrevar l.gnp d.gnp r
\end{verbatim}

creates two temporary variables containing the values for \texttt{l.gnp} and \texttt{d.gnp}. The variable \texttt{r} appears in the new variable list but does not require a temporary variable.

The resulting variable list is

\begin{verbatim}
. display "\textquoteleft r(varlist)\textquoteright"
  __00014P __00014Q r
\end{verbatim}

(Your temporary variable names may be different, but that is of no consequence.)

We can see the results by listing the new variables alongside the original value of \texttt{gnp}.

\begin{verbatim}
. list gnp \textquoteleft r(varlist)\textquoteright\ in 1/5
\end{verbatim}

\begin{table}[h]
\centering
\begin{tabular}{c c c c}
\hline
\texttt{gnp} & \texttt{\_00014P} & \texttt{\_00014Q} & \texttt{r} \\
\hline
1. & 128 & . & 3.2 \\
2. & 135 & 128 & 7 3.8 \\
3. & 132 & 135 & -3 2.6 \\
4. & 138 & 132 & 6 3.9 \\
5. & 145 & 138 & 7 4.2 \\
\hline
\end{tabular}
\end{table}

Temporary variables automatically vanish when the program concludes.

If we had needed only the base variable names, we could have specified

\begin{verbatim}
. tsrevar l.gnp d.gnp r, list
. display "\textquoteleft r(varlist)\textquoteright"
  gnp r
\end{verbatim}

The order of the list will probably differ from that of the original list; base variables are listed only once and are listed in the order that they appear in the dataset.
Technical note

trevar, substitute avoids creating duplicate variables. Consider

```
    . tsrevar gnp l.gnp r cd l.cd l.gnp
```

1.gnp appears twice in the varlist. tsrevar will create only one new variable for 1.gnp and use that new variable twice in the resulting r(varlist). Moreover, tsrevar will even do this across multiple calls:

```
    . tsrevar gnp l.gnp cd l.cd
    . tsrevar cpi l.gnp
```

1.gnp appears in two separate calls. At the first call, tsrevar creates a temporary variable corresponding to 1.gnp. At the second call, tsrevar remembers what it has done and uses that same temporary variable for 1.gnp again.

Stored results

trevar stores the following in r():

Macros

```
    r(varlist) the modified variable list or list of base variable names
```

Also see

[P] syntax — Parse Stata syntax

[P] unab — Unabbreviate variable list

[R] fvrevar — Factor-variables operator programming command

[U] 11 Language syntax

[U] 11.4.4 Time-series varlists

[U] 18 Programming Stata
**tsset — Declare data to be time-series data**

**Description**

taxset manages the time-series settings of a dataset. tsset *timevar* declares the data in memory to be a time series. This allows you to use Stata’s time-series operators and to analyze your data with the *ts* commands. tsset *panelvar* timevar declares the data to be panel data, also known as cross-sectional time-series data, which contain one time series for each value of *panelvar*. This allows you to also analyze your data with the *xt* commands without having to xtset your data.

taxset without arguments displays how the data are currently set and sorts the data on *timevar* or *panelvar*.

taxset, clear is a rarely used programmer’s command to declare that the data are no longer a time series.

**Quick start**

Declare data to be a time series with time variable *tvar*

```
tsset tvar
```

As above, but specify that *tvar* records time for a weekly time series

```
tsset tvar, weekly
```

As above, but specify that observations occur every two weeks

```
tsset tvar, weekly delta(2)
```

Declare a panel dataset with panel identifier *pvar* and time variable *tvar*

```
tsset pvar tvar
```

As above, but specify that observations on each panel are made daily

```
tsset pvar tvar, daily
```

As above, but specify that observations on each panel are made every three days

```
tsset pvar tvar, daily delta(3 days)
```

Display current time-series settings, and sort data by *pvar* and *tvar* if they are sorted differently

```
tsset
```

**Menu**

Statistics  >  Time series  >  Setup and utilities  >  Declare dataset to be time-series data
## Syntax

Declare data to be time series

```
   tsset timevar [, options]
   tsset panelvar timevar [, options]
```

Display how data are currently tsset

```
   tsset
```

Clear time-series settings

```
   tsset, clear
```

In the declare syntax, `panelvar` identifies the panels and `timevar` identifies the times.

### options

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| `unitoptions` | specify units of `timevar`
| `deltaloption` | specify length of period of `timevar`
| `noquery` | suppress summary calculations and output

`noquery` is not shown in the dialog box.

### unitoptions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| `(default)` | `timevar`’s units from `timevar`’s display format
| `clocktime` | `timevar` is `%tc: 0 = 1jan1960 00:00:00.000, 1 = 1jan1960 00:00:00.001, ...`
| `daily` | `timevar` is `%td: 0 = 1jan1960, 1 = 2jan1960, ...`
| `weekly` | `timevar` is `%tw: 0 = 1960w1, 1 = 1960w2, ...`
| `monthly` | `timevar` is `%tm: 0 = 1960m1, 1 = 1960m2, ...`
| `quarterly` | `timevar` is `%tq: 0 = 1960q1, 1 = 1960q2, ...`
| `halfyearly` | `timevar` is `%th: 0 = 1960h1, 1 = 1960h2, ...`
| `yearly` | `timevar` is `%ty: 1960 = 1960, 1961 = 1961, ...`
| `generic` | `timevar` is `%tg: 0 = ?, 1 = ?, ...`
| `format(%)fmt` | specify `timevar`’s format and then apply default rule

In all cases, negative `timevar` values are allowed.
*deltaoption* specifies the period between observations in *timevar* units and may be specified as

<table>
<thead>
<tr>
<th>deltaoption</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta(#)</td>
<td>delta(1) or delta(2)</td>
</tr>
<tr>
<td>delta((exp))</td>
<td>delta((7*24))</td>
</tr>
<tr>
<td>delta(# units)</td>
<td>delta(7 days) or delta(15 minutes) or delta(7 days 15 minutes)</td>
</tr>
<tr>
<td>delta((exp) units)</td>
<td>delta((2+3) weeks)</td>
</tr>
</tbody>
</table>

Allowed units for %tc and %tC *timevars* are

seconds  second  secs  sec
minutes  minute  mins  min
hours    hour    
days     day      
weeks    week    

and for all other %t *timevars*, units specified must match the frequency of the data; for example, for %ty, units must be year or years.

**Options**

*unitoptions* clocktime, daily, weekly, monthly, quarterly, halfyearly, yearly, generic, and format(%fmt) specify the units in which *timevar* is recorded.

*timevar* will usually be a %t variable; see [D] Datetime. If *timevar* already has a %t display format assigned to it, you do not need to specify a *unitoption*; *tsset* will obtain the units from the format. If you have not yet bothered to assign the appropriate %t format, however, you can use the *unitoptions* to tell *tsset* the units. Then *tsset* will set *timevar*’s display format for you. Thus, the *unitoptions* are convenience options; they allow you to skip formatting the time variable. The following all have the same net result:

<table>
<thead>
<tr>
<th>Alternative 1</th>
<th>Alternative 2</th>
<th>Alternative 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>format t %td</td>
<td>(t not formatted)</td>
<td>(t not formatted)</td>
</tr>
<tr>
<td>tsset t</td>
<td>tsset t, daily</td>
<td>tsset t, format(%td)</td>
</tr>
</tbody>
</table>

*timevar* is not required to be a %t variable; it can be any variable of your own concocting so long as it takes on only integer values. In such cases, it is called generic and considered to be %tg. Specifying the *unitoption* generic or attaching a special format to *timevar*, however, is not necessary because *tsset* will assume that the variable is generic if it has any numerical format other than a %t format (or if it has a %tg format).

*clear*—used in *tsset*, *clear*—makes Stata forget that the data ever were *tsset*. This is a rarely used programmer’s option.

*Delta* 

*delta()* specifies the period between observations in *timevar* and is commonly used when *timevar* is %tc. *delta()* is only sometimes used with the other %t formats or with generic time variables.
If \texttt{delta()} is not specified, \texttt{delta(1)} is assumed. This means that at \textit{timevar} = 5, the previous time is \textit{timevar} = 5 − 1 = 4 and the next time would be \textit{timevar} = 5 + 1 = 6. Lag and lead operators, for instance, would work this way. This would be assumed regardless of the units of \textit{timevar}.

If you specified \texttt{delta(2)}, then at \textit{timevar} = 5, the previous time would be \textit{timevar} = 5 − 2 = 3 and the next time would be \textit{timevar} = 5 + 2 = 7. Lag and lead operators would work this way. In the observation with \textit{timevar} = 5, \textit{L.price} would be the value of \textit{price} in the observation for which \textit{timevar} = 3 and \textit{F.price} would be the value of \textit{price} in the observation for which \textit{timevar} = 7. If you then add an observation with \textit{timevar} = 4, the operators will still work appropriately; that is, at \textit{timevar} = 5, \textit{L.price} will still have the value of \textit{price} at \textit{timevar} = 3.

There are two aspects of \textit{timevar}: its units and its length of period. The \texttt{unitoptions} set the units. \texttt{delta()} sets the length of period.

We mentioned that \texttt{delta()} is commonly used with \%tc \textit{timevars} because Stata’s \%tc variables have units of milliseconds. If \texttt{delta()} is not specified and in some model you refer to \textit{L.price}, you will be referring to the value of \textit{price} 1 ms ago. Few people have data with periodicity of a millisecond. Perhaps your data are hourly. You could specify \texttt{delta(3600000)}. Or you could specify \texttt{delta((60*60*1000))}, because \texttt{delta()} will allow expressions if you include an extra pair of parentheses. Or you could specify \texttt{delta(1 hour)}. They all mean the same thing: \textit{timevar} has periodicity of 3,600,000 ms. In an observation for which \textit{timevar} = 1,489,572,000,000 (corresponding to 15mar2007 10:00:00), \textit{L.price} would be the observation for which \textit{timevar} = 1,489,572,000,000 − 3,600,000 = 1,489,568,400,000 (corresponding to 15mar2007 9:00:00).

When you \texttt{tsset} the data and specify \texttt{delta()}, \texttt{tsset} verifies that all the observations follow the specified periodicity. For instance, if you specified \texttt{delta(2)}, then \textit{timevar} could contain any subset of \{... −4, −2, 0, 2, 4,...\} or it could contain any subset of \{... −3, −1, 1, 3,...\}. If \textit{timevar} contained a mix of values, \texttt{tsset} would issue an error message. If you also specify \texttt{panelvar}—you type \texttt{tsset panelvar timevar, delta(2)}—the check is made on each panel independently. One panel might contain \textit{timevar} values from one set and the next, another, and that would be fine.

The following option is available with \texttt{tsset} but is not shown in the dialog box:

\texttt{noquery} prevents \texttt{tsset} from performing most of its summary calculations and suppresses output.

With this option, only the following results are posted:

\begin{verbatim}
 r(tdelta)   r(tsfmt) 
 r(panelvar) r(unit) 
 r(timevar)  r(unit1) 
\end{verbatim}

\section*{Remarks and examples}

Remarks are presented under the following headings:

\begin{itemize}
  \item \textit{Overview}
  \item \textit{Panel data}
  \item \textit{Video example}
\end{itemize}
Overview

`tset` sets `timevar` so that Stata’s time-series operators are understood in varlists and expressions. The time-series operators are

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.</td>
<td>lag ( x_{t-1} )</td>
</tr>
<tr>
<td>L2.</td>
<td>2-period lag ( x_{t-2} )</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>F.</td>
<td>lead ( x_{t+1} )</td>
</tr>
<tr>
<td>F2.</td>
<td>2-period lead ( x_{t+2} )</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>D.</td>
<td>difference ( x_t - x_{t-1} )</td>
</tr>
<tr>
<td>D2.</td>
<td>difference of difference ( x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2} )</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>S.</td>
<td>“seasonal” difference ( x_t - x_{t-1} )</td>
</tr>
<tr>
<td>S2.</td>
<td>lag-2 (seasonal) difference ( x_t - x_{t-2} )</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
</tbody>
</table>

Time-series operators may be repeated and combined. \( L3.\text{gnp} \) refers to the third lag of variable \text{gnp}, as do \( LLL.\text{gnp}, LL2.\text{gnp}, \) and \( L2L.\text{gnp} \). \( LF.\text{gnp} \) is the same as \text{gnp}. \( DS12.\text{gnp} \) refers to the one-period difference of the 12-period difference. \( LDS12.\text{gnp} \) refers to the same concept, lagged once.

\( D1. = S1. \), but \( D2. \neq S2. \), \( D3. \neq S3. \), and so on. \( D2. \) refers to the difference of the difference. \( S2. \) refers to the two-period difference. If you wanted the difference of the difference of the 12-period difference of \text{gnp}, you would write \( D2S12.\text{gnp} \).

Operators may be typed in uppercase or lowercase. Most users would type \( d2s12.\text{gnp} \) instead of \( D2S12.\text{gnp} \).

You may type operators however you wish; Stata internally converts operators to their canonical form. If you typed \( ld21s12d.\text{gnp} \), Stata would present the operated variable as \( L2D3S12.\text{gnp} \).

Stata also understands `operator(numlist)` to mean a set of operated variables; thus, typing \( L(1/3).\text{gnp} \) in a varlist is the same as typing \( L.\text{gnp} \) \( L2.\text{gnp} \) \( L3.\text{gnp} \). Here are some sample data:

```
. list year gnp L(1/3).gnp
```

<table>
<thead>
<tr>
<th>year</th>
<th>L.</th>
<th>L2.</th>
<th>L3.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gnp</td>
<td>gnp</td>
<td>gnp</td>
</tr>
<tr>
<td>1989</td>
<td>5452.8</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1990</td>
<td>5764.9</td>
<td>5452.8</td>
<td>.</td>
</tr>
<tr>
<td>1991</td>
<td>5932.4</td>
<td>5764.9</td>
<td>5452.8</td>
</tr>
<tr>
<td>1992</td>
<td>6229.1</td>
<td>5932.4</td>
<td>5764.9</td>
</tr>
<tr>
<td>1993</td>
<td>6519.1</td>
<td>6229.1</td>
<td>5932.4</td>
</tr>
<tr>
<td>1994</td>
<td>6892.2</td>
<td>6519.1</td>
<td>6229.1</td>
</tr>
<tr>
<td>1995</td>
<td>7330.1</td>
<td>6892.2</td>
<td>6519.1</td>
</tr>
<tr>
<td>1996</td>
<td>7453.9</td>
<td>7330.1</td>
<td>6892.2</td>
</tr>
</tbody>
</table>
```

The first two columns show variables in the dataset: `year` and `gnp`. The remaining columns show lagged values of `gnp` that are not in the data but become available when using time-series operators. The third column shows the lag of `gnp` (`L.gnp`). The first value of `L.gnp` is missing because the lag...
of gnp in 1989 is the gnp of 1988, which occurs before the initial period of our time series. The missing values in the remaining columns follow similar logic. When using estimators with time-series operators, it is important to remember that the size of the estimation sample decreases because of the missing values for the initial time periods.

The operators can also be applied to a list of variables by enclosing the variables in parentheses; for example,

```
    . list year gnp cpi L(1/3).(gnp cpi)
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>year</td>
<td>1989</td>
<td>5452.8</td>
<td></td>
<td>100</td>
<td>5452.8</td>
<td></td>
<td>100</td>
<td>5452.8</td>
</tr>
<tr>
<td>2</td>
<td>1990</td>
<td>5764.9</td>
<td>105</td>
<td>5452.8</td>
<td>100</td>
<td>5452.8</td>
<td>105</td>
<td>100</td>
<td>5452.8</td>
</tr>
<tr>
<td>3</td>
<td>1991</td>
<td>5932.4</td>
<td>108</td>
<td>5764.9</td>
<td>5452.8</td>
<td>105</td>
<td>100</td>
<td>108</td>
<td>5452.8</td>
</tr>
<tr>
<td>4</td>
<td>1992</td>
<td>6229.1</td>
<td>110</td>
<td>5932.4</td>
<td>5764.9</td>
<td>5452.8</td>
<td>105</td>
<td>108</td>
<td>5452.8</td>
</tr>
<tr>
<td>5</td>
<td>1993</td>
<td>6519.1</td>
<td>112</td>
<td>6229.1</td>
<td>5932.4</td>
<td>5764.9</td>
<td>110</td>
<td>108</td>
<td>5452.8</td>
</tr>
<tr>
<td>6</td>
<td>1994</td>
<td>6892.2</td>
<td>119</td>
<td>6519.1</td>
<td>6229.1</td>
<td>5932.4</td>
<td>112</td>
<td>110</td>
<td>5452.8</td>
</tr>
<tr>
<td>7</td>
<td>1995</td>
<td>7330.1</td>
<td>122</td>
<td>6892.2</td>
<td>6519.1</td>
<td>6229.1</td>
<td>119</td>
<td>112</td>
<td>6519.1</td>
</tr>
<tr>
<td>8</td>
<td>1996</td>
<td>7453.9</td>
<td>126</td>
<td>7330.1</td>
<td>6892.2</td>
<td>6519.1</td>
<td>122</td>
<td>119</td>
<td>6519.1</td>
</tr>
</tbody>
</table>

In operator#, making # zero returns the variable itself. L0.gnp is gnp. Thus, you can type list year l(0/3).gnp to mean list year gnp L.gnp L2.gnp L3.gnp.

The parenthetical notation may be used with any operator. Typing D(1/3).gnp would return the first through third differences.

The parenthetical notation may be used in operator lists with multiple operators, such as L(0/3)D2S12.gnp.

Operator lists may include up to one set of parentheses, and the parentheses may enclose a numlist; see [U] 11.1.8 numlist.

Before you can use these time-series operators, however, the dataset must satisfy two requirements:
1. the dataset must be tsset and
2. the dataset must be sorted by timevar or, if it is a cross-sectional time-series dataset, by panelvar timevar.

tsset handles both requirements. As you use Stata, however, you may later use a command that re-sorts that data, and if you do, the time-series operators will not work:

```
    . tsset time  
        (output omitted)
    . regress y x l.x  
        (output omitted)
    . (you continue to use Stata and, sometime later:)
    . regress y x l.x  
        not sorted
        r(5);
```

Then typing tsset without arguments will reestablish the sort order:

```
    . tsset  
        (output omitted)
    . regress y x l.x  
        (output omitted)
```
Here typing `tsset` is the same as typing `sort time`. Had we previously `tsset country time`, however, typing `tsset` would be the same as typing `sort country time`. You can type the `sort` command or type `tsset` without arguments; it makes no difference.

There are two syntaxes for setting your data:

```
tsset timevar
ntsset panelvar timevar
```

In both, `timevar` must contain integer values. If `panelvar` is specified, it too must contain integer values, and the dataset is declared to be a cross-section of time series, such as a collection of time series for different countries. Such datasets can be analyzed with `xt` commands as well as `ts` commands. If you `tsset panelvar timevar`, you do not need to `xtset` the data to use the `xt` commands.

If you save the data after typing `tsset`, the data will be remembered to be time series, and you will not have to `tsset` the data again.

### Example 1: Numeric time variable

You have monthly data on personal income. Variable `t` records the time of an observation, but there is nothing special about the name of the variable. There is nothing special about the values of the variable, either. `t` is not required to be `%tm` variable—perhaps you do not even know what that means. `t` is just a numeric variable containing integer values that represent the month, and we will imagine that `t` takes on the values 1, 2, ..., 9, although it could just as well be −3, −2 ..., 5, or 1,023, 1,024, ..., 1,031. What is important is that the values are dense: adjacent months have a time value that differs by 1.

```
. use https://www.stata-press.com/data/r16/tssetxmpl
. list t income
    +--------+---------+
    | t | income |
    |---|--------|
    | 1 | 1153   |
    | 2 | 1181   |
    | 9 | 1282   |
    +--------+---------+
```

```
. tsset t
   time variable: t, 1 to 9
   delta: 1 unit
. regress income l.income
   (output omitted)
```

### Example 2: Adjusting the starting date

In the example above, that `t` started at 1 was not important. As we said, the `t` variable could just as well be recorded −3, −2 ..., 5, or 1,023, 1,024, ..., 1,031. What is important is that the difference in `t` between observations be `delta()` when there are no gaps.

Although how time is measured makes no difference, Stata has formats to display time nicely if it is recorded in certain ways; you can learn about the formats by seeing `[D] Datetime`. Stata likes time variables in which `1jan1960` is recorded as 0. In our previous example, if `t = 1` corresponds to July 1995, then we could make a variable that fits Stata’s preference by typing

```
. generate newt = tm(1995m7) + t - 1
```
tm() is the function that returns a month equivalent; tm(1995m7) evaluates to the constant 426, meaning 426 months after January 1960. We now have variable newt containing

```
. list t newt income

<table>
<thead>
<tr>
<th>t</th>
<th>newt</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>426</td>
<td>1153</td>
</tr>
<tr>
<td>2</td>
<td>427</td>
<td>1181</td>
</tr>
<tr>
<td>3</td>
<td>428</td>
<td>1208</td>
</tr>
<tr>
<td>9</td>
<td>434</td>
<td>1282</td>
</tr>
</tbody>
</table>
```

If we put a %tm format on newt, it will display more cleanly:

```
. format newt %tm
. list t newt income

<table>
<thead>
<tr>
<th>t</th>
<th>newt</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>2</td>
<td>1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>3</td>
<td>1995m9</td>
<td>1208</td>
</tr>
<tr>
<td>9</td>
<td>1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>
```

We could now tsset newt rather than t:

```
. tsset newt
    time variable: newt, 1995m7 to 1996m3
delta: 1 month
```

⚠️ Technical note

In addition to monthly, Stata understands clock times (to the millisecond level) as well as daily, weekly, quarterly, half-yearly, and yearly data. See [D] Datetime for a description of these capabilities.

Let’s reconsider the previous example, but rather than monthly, let’s assume the data are daily, weekly, etc. The only thing to know is that, corresponding to function tm(), there are functions td(), tw(), tq(), th(), and ty() and that, corresponding to format %tm, there are formats %td, %tw, %tq, %th, and %ty. Here is what we would have typed had our data been on a different time scale:
Daily: if your \( t \) variable had \( t=1 \) corresponding to 15mar1993
\[
\text{generate newt = td(15mar1993) + t - 1}
\]
\[
\text{tsset newt, daily}
\]

Weekly: if your \( t \) variable had \( t=1 \) corresponding to 1994w1:
\[
\text{generate newt = tw(1994w1) + t - 1}
\]
\[
\text{tsset newt, weekly}
\]

Monthly: if your \( t \) variable had \( t=1 \) corresponding to 2004m7:
\[
\text{generate newt = tm(2004m7) + t - 1}
\]
\[
\text{tsset newt, monthly}
\]

Quarterly: if your \( t \) variable had \( t=1 \) corresponding to 1994q1:
\[
\text{generate newt = tq(1994q1) + t - 1}
\]
\[
\text{tsset newt, quarterly}
\]

Half-yearly: if your \( t \) variable had \( t=1 \) corresponding to 1921h2:
\[
\text{generate newt = th(1921h2) + t - 1}
\]
\[
\text{tsset newt, halfyearly}
\]

Yearly: if your \( t \) variable had \( t=1 \) corresponding to 1842:
\[
\text{generate newt = 1842 + t - 1}
\]
\[
\text{tsset newt, yearly}
\]

In each example above, we subtracted one from our time variable in constructing the new time variable `newt` because we assumed that our starting time value was 1. For the quarterly example, if our starting time value were 5 and that corresponded to 1994q1, we would type
\[
\text{generate newt = tq(1994q1) + t - 5}
\]

Had our initial time value been \( t = 742 \) and that corresponded to 1994q1, we would have typed
\[
\text{generate newt = tq(1994q1) + t - 742}
\]

---

Example 3: Time-series data but no time variable

Perhaps we have the same time-series data but no time variable:
\[
\text{use https://www.stata-press.com/data/r16/tssetxmpl2, clear}
\]
\[
\text{list income}
\]

<table>
<thead>
<tr>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 1153</td>
</tr>
<tr>
<td>2. 1181</td>
</tr>
<tr>
<td>3. 1208</td>
</tr>
<tr>
<td>4. 1272</td>
</tr>
<tr>
<td>5. 1236</td>
</tr>
<tr>
<td>6. 1297</td>
</tr>
<tr>
<td>7. 1265</td>
</tr>
<tr>
<td>8. 1230</td>
</tr>
<tr>
<td>9. 1282</td>
</tr>
</tbody>
</table>

Say that we know that the first observation corresponds to July 1995 and continues without gaps. We can create a monthly time variable and format it by typing
\[
\text{generate t = tm(1995m7) + _n - 1}
\]
\[
\text{format t %tm}
\]
We can now `tsset` our dataset and list it:

```stata
.tsset t
  time variable: t, 1995m7 to 1996m3
delta: 1 month
.list t income
```

<table>
<thead>
<tr>
<th>t</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>1995m9</td>
<td>1208</td>
</tr>
<tr>
<td>1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>

Example 4: Time variable as a string

Your data might include a time variable that is encoded into a string. In the example below each monthly observation is identified by string variable `yrmo` containing the month and year of the observation, sometimes with punctuation between:

```stata
.use https://www.stata-press.com/data/r16/tssetxmpl, clear
.list yrmo income
```

<table>
<thead>
<tr>
<th>yrmo</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/1995</td>
<td>1153</td>
</tr>
<tr>
<td>8/1995</td>
<td>1181</td>
</tr>
<tr>
<td>9-1995</td>
<td>1208</td>
</tr>
<tr>
<td>10,1995</td>
<td>1272</td>
</tr>
<tr>
<td>11 1995</td>
<td>1236</td>
</tr>
<tr>
<td>12 1995</td>
<td>1297</td>
</tr>
<tr>
<td>1/1996</td>
<td>1265</td>
</tr>
<tr>
<td>2.1996</td>
<td>1230</td>
</tr>
<tr>
<td>3- 1996</td>
<td>1282</td>
</tr>
</tbody>
</table>

The first step is to convert the string to a numeric representation. Doing so is easy using the `monthly()` function; see [D] Datetime.

```stata
.generate mdate = monthly(yrmo, "MY")
.list yrmo mdate income
```

<table>
<thead>
<tr>
<th>yrmo</th>
<th>mdate</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/1995</td>
<td>426</td>
<td>1153</td>
</tr>
<tr>
<td>8/1995</td>
<td>427</td>
<td>1181</td>
</tr>
<tr>
<td>9-1995</td>
<td>428</td>
<td>1208</td>
</tr>
<tr>
<td>3- 1996</td>
<td>434</td>
<td>1282</td>
</tr>
</tbody>
</table>

Our new variable, `mdate`, contains the number of months from January 1960. Now that we have numeric variable `mdate`, we can `tsset` the data:
. format mdate %tm
. tsset mdate
    time variable: mdate, 1995m7 to 1996m3
    delta: 1 month

In fact, we can combine the two and type

. tsset mdate, format(%tm)
    time variable: mdate, 1995m7 to 1996m3
    delta: 1 month

or type

. tsset mdate, monthly
    time variable: mdate, 1995m7 to 1996m3
    delta: 1 month

In all cases, we obtain

. list yrmo mdate income

<table>
<thead>
<tr>
<th>yrmo</th>
<th>mdate</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/1995</td>
<td>1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>8/1995</td>
<td>1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>9-1995</td>
<td>1995m9</td>
<td>1208</td>
</tr>
<tr>
<td>10,1995</td>
<td>1995m10</td>
<td>1272</td>
</tr>
<tr>
<td>11 1995</td>
<td>1995m11</td>
<td>1236</td>
</tr>
<tr>
<td>12 1995</td>
<td>1995m12</td>
<td>1297</td>
</tr>
<tr>
<td>1/1996</td>
<td>1996m1</td>
<td>1265</td>
</tr>
<tr>
<td>2.1996</td>
<td>1996m2</td>
<td>1230</td>
</tr>
<tr>
<td>3- 1996</td>
<td>1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>

Stata can translate many different date formats, including strings like 12jan2009; January 12, 2009; 12-01-2009; 01/12/2009; 01/12/09; 12jan2009 8:14; 12-01-2009 13:12; 01/12/09 1:12 pm; Wed Jan 31 13:03:25 CST 2009; 1998q1; and more. See [D] Datetime.

Example 5: Time-series data with gaps

Gaps in the time series cause no difficulties:

. use https://www.stata-press.com/data/r16/tssetxmpl3, clear
. list yrmo income

<table>
<thead>
<tr>
<th>yrmo</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/1995</td>
<td>1153</td>
</tr>
<tr>
<td>8/1995</td>
<td>1181</td>
</tr>
<tr>
<td>11 1995</td>
<td>1236</td>
</tr>
<tr>
<td>12 1995</td>
<td>1297</td>
</tr>
<tr>
<td>1/1996</td>
<td>1265</td>
</tr>
<tr>
<td>2.1996</td>
<td>1230</td>
</tr>
<tr>
<td>3- 1996</td>
<td>1282</td>
</tr>
</tbody>
</table>

. generate mdate = monthly(yrmo, "MY")
. tsset mdate, monthly
    time variable: mdate, 1995m7 to 1996m3, but with gaps
    delta: 1 month
Once the dataset has been \texttt{tsset}, we can use the time-series operators. The \texttt{D} operator specifies first differences:

\begin{verbatim}
. list mdate income d.income

\begin{tabular}{|l|l|l|}
\hline
 mdate & income & D.income \\
\hline
1. & 1995m7 & 1153 . \\
2. & 1995m8 & 1181 28  \\
3. & 1995m11 & 1236 . \\
4. & 1995m12 & 1297 61  \\
5. & 1996m1 & 1265 -32  \\
6. & 1996m3 & 1282 . \\
\hline
\end{tabular}
\end{verbatim}

We can use the operators in an expression or varlist context; we do not have to create a new variable to hold \texttt{D.income}. We can use \texttt{D.income} with the \texttt{list} command, with \texttt{regress} or any other Stata command that allows time-series varlists.

\section*{Example 6: Clock times}

We have data from a large hotel in Las Vegas that changes the reservation prices for its rooms hourly. A piece of the data looks like

\begin{verbatim}
. use https://www.stata-press.com/data/r16/tssetxmpl4, clear
. list in 1/5

\begin{tabular}{|l|l|}
\hline
 time & price \\
\hline
1. & 02.13.2007 08:00 140 \\
2. & 02.13.2007 09:00 155 \\
3. & 02.13.2007 10:00 160 \\
4. & 02.13.2007 11:00 155 \\
5. & 02.13.2007 12:00 160 \\
\hline
\end{tabular}
\end{verbatim}

Variable \texttt{time} is a string variable. The first step in making this dataset a time-series dataset is to translate the string to a numeric variable:

\begin{verbatim}
. generate double t = clock(time, "MDY hm")
. list in 1/5

\begin{tabular}{|l|l|l|}
\hline
 time & price & t \\
\hline
1. & 02.13.2007 08:00 & 140 1.487e+12 \\
2. & 02.13.2007 09:00 & 155 1.487e+12 \\
3. & 02.13.2007 10:00 & 160 1.487e+12 \\
4. & 02.13.2007 11:00 & 155 1.487e+12 \\
5. & 02.13.2007 12:00 & 160 1.487e+12 \\
\hline
\end{tabular}
\end{verbatim}

See \cite{D} \textbf{Datetime translation} for an explanation of what is going on here. \texttt{clock()} is the function that converts strings to datetime (\texttt{%tc}) values. We typed \texttt{clock(time, "MDY hm")} to convert string variable \texttt{time}, and we told \texttt{clock()} that the values in \texttt{time} were in the order month, day, year, hour, and minute. We stored new variable \texttt{t} as a \texttt{double} because time values are large and that is required to prevent rounding. Even so, the resulting values 1.487e+12 look rounded, but that is only
because of the default display format for new variables. We can see the values better if we change the format:

```
. format t %20.0gc
. list in 1/5
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007 08:00</td>
<td>140</td>
<td>1,486,972,800,000</td>
</tr>
<tr>
<td>02.13.2007 09:00</td>
<td>155</td>
<td>1,486,976,400,000</td>
</tr>
<tr>
<td>02.13.2007 10:00</td>
<td>160</td>
<td>1,486,980,000,000</td>
</tr>
<tr>
<td>02.13.2007 11:00</td>
<td>155</td>
<td>1,486,983,600,000</td>
</tr>
<tr>
<td>02.13.2007 12:00</td>
<td>160</td>
<td>1,486,987,200,000</td>
</tr>
</tbody>
</table>

Even better would be to change the format to %tc—Stata’s clock-time format:

```
. format t %tc
. list in 1/5
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007 08:00</td>
<td>140</td>
<td>13feb2007 08:00:00</td>
</tr>
<tr>
<td>02.13.2007 09:00</td>
<td>155</td>
<td>13feb2007 09:00:00</td>
</tr>
<tr>
<td>02.13.2007 10:00</td>
<td>160</td>
<td>13feb2007 10:00:00</td>
</tr>
<tr>
<td>02.13.2007 11:00</td>
<td>155</td>
<td>13feb2007 11:00:00</td>
</tr>
<tr>
<td>02.13.2007 12:00</td>
<td>160</td>
<td>13feb2007 12:00:00</td>
</tr>
</tbody>
</table>

We could drop variable `time`. New variable `t` contains the same information as `time` and `t` is better because it is a Stata time variable, the most important property of which being that it is numeric rather than string. We can `tsset` it. Here, however, we also need to specify the length of the periods with `tsset`’s `delta()` option. Stata’s time variables are numeric, but they record milliseconds since 01jan1960 00:00:00. By default, `tsset` uses `delta(1)`, and that means the time-series operators would not work as we want them to work. For instance, `L.price` would look back only 1 ms (and find nothing). We want `L.price` to look back 1 hour (3,600,000 ms):

```
. tsset t, delta(1 hour)
time variable: t,
    13feb2007 08:00:00.000 to 13feb2007 14:00:00.000
delta: 1 hour
```

```
. list t price l.price in 1/5
```

<table>
<thead>
<tr>
<th>t</th>
<th>price</th>
<th>L.price</th>
</tr>
</thead>
<tbody>
<tr>
<td>13feb2007 08:00:00</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>13feb2007 09:00:00</td>
<td>155</td>
<td>140</td>
</tr>
<tr>
<td>13feb2007 10:00:00</td>
<td>160</td>
<td>155</td>
</tr>
<tr>
<td>13feb2007 11:00:00</td>
<td>155</td>
<td>160</td>
</tr>
<tr>
<td>13feb2007 12:00:00</td>
<td>160</td>
<td>155</td>
</tr>
</tbody>
</table>
Example 7: Clock times must be double

In the previous example, it was of vital importance that when we generated the \%tc variable \textit{t},

\begin{verbatim}
    . generate double t = clock(time, "MDY hm")
\end{verbatim}

we generated it as a double. Let’s see what would have happened had we forgotten and just typed

\begin{verbatim}
    . generate t = clock(time, "MDY hm")
\end{verbatim}

Let’s go back and start with the same original data:

\begin{verbatim}
    . use https://www.stata-press.com/data/r16/tssetxmpl4, clear
    . list in 1/5
\end{verbatim}

\begin{verbatim}
+--------------+---------+--------+
|    time      |   price |     t  |
|--------------+---------+--------+
| 02.13.2007 08:00 |    140  | 1.49e+12|
| 02.13.2007 09:00 |    155  | 1.49e+12|
| 02.13.2007 10:00 |    160  | 1.49e+12|
| 02.13.2007 11:00 |    155  | 1.49e+12|
| 02.13.2007 12:00 |    160  | 1.49e+12|
+--------------+---------+--------+
\end{verbatim}

Remember, variable \textit{time} is a string variable, and we need to translate it to numeric. So we translate, but this time we forget to make the new variable a double:

\begin{verbatim}
    . generate t = clock(time, "MDY hm")
    . list in 1/5
\end{verbatim}

\begin{verbatim}
+--------------+---------+--------+--------+
|    time      |   price |    t   |
|--------------+---------+--------+--------+
| 02.13.2007 08:00 |    140  | 1,486,972,780,544|
| 02.13.2007 09:00 |    155  | 1,486,976,450,560|
| 02.13.2007 10:00 |    160  | 1,486,979,989,504|
| 02.13.2007 11:00 |    155  | 1,486,983,659,520|
| 02.13.2007 12:00 |    160  | 1,486,987,198,464|
+--------------+---------+--------+--------+
\end{verbatim}

We see the first difference—\textit{t} now lists as 1.49e+12 rather than 1.487e+12 as it did previously—but this is nothing that would catch our attention. We would not even know that the value is different. Let’s continue.

We next put a \%20.0gc format on \textit{t} to better see the numerical values. In fact, that is not something we would usually do in an analysis. We did that in the example to emphasize to you that the \textit{t} values were really big numbers. We will repeat the exercise just to be complete, but in real analysis, we would not bother.

\begin{verbatim}
    . format t %20.0gc
    . list in 1/5
\end{verbatim}

\begin{verbatim}
+--------------+---------+--------+--------+
|    time      |   price |   t    |
|--------------+---------+--------+--------+
| 02.13.2007 08:00 |    140  | 1,486,972,780,544|
| 02.13.2007 09:00 |    155  | 1,486,976,450,560|
| 02.13.2007 10:00 |    160  | 1,486,979,989,504|
| 02.13.2007 11:00 |    155  | 1,486,983,659,520|
| 02.13.2007 12:00 |    160  | 1,486,987,198,464|
+--------------+---------+--------+--------+
\end{verbatim}

Okay, we see big numbers in \textit{t}. Let’s continue.

Next we put a \%tc format on \textit{t}, and that is something we would usually do, and you should always do. You should also list a bit of the data, as we did:

\begin{verbatim}
    . format t %tc
    . list in 1/5
\end{verbatim}

\begin{verbatim}
+--------------+---------+--------+--------+
|    time      |   price |     t  |
|--------------+---------+--------+--------+
| 02.13.2007 08:00 |    140  | 1,486,972,780,544|
| 02.13.2007 09:00 |    155  | 1,486,976,450,560|
| 02.13.2007 10:00 |    160  | 1,486,979,989,504|
| 02.13.2007 11:00 |    155  | 1,486,983,659,520|
| 02.13.2007 12:00 |    160  | 1,486,987,198,464|
+--------------+---------+--------+--------+
\end{verbatim}
. format t %tc
. list in 1/5

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007</td>
<td>140</td>
<td>13feb2007 07:59:40</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
<td>13feb2007 09:00:50</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
<td>13feb2007 09:59:49</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
<td>13feb2007 11:00:59</td>
</tr>
</tbody>
</table>

By now, you should see a problem: the translated datetime values are off by a second or two. That was caused by rounding. Dates and times should be the same, not approximately the same, and when you see a difference like this, you should say to yourself, “The translation is off a little. Why is that?” and then you should think, “Of course, rounding. I bet that I did not create t as a double.”

Let us assume, however, that you do not do this. You instead plow ahead:

```
. tsset t, delta(1 hour)
```

```
<table>
<thead>
<tr>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>time values with period less than delta() found</td>
</tr>
<tr>
<td>r(451);</td>
</tr>
</tbody>
</table>
```

And that is what will happen when you forget to create t as a double. The rounding will cause uneven period, and tsset will complain.

By the way, it is important only that clock times (\%tc and \%tC variables) be stored as doubles. The other date values \%td, \%tw, \%tm, \%tq, \%th, and \%ty are small enough that they can safely be stored as floats, although forgetting and storing them as doubles does no harm.

---

**Technical note**

Stata provides two clock-time formats, \%tc and \%tC. \%tC provides a clock with leap seconds. Leap seconds are occasionally inserted to account for randomness of the earth’s rotation, which gradually slows. Unlike the extra day inserted in leap years, the timing of when leap seconds will be inserted cannot be foretold. The authorities in charge of such matters announce a leap second approximately 6 months before insertion. Leap seconds are inserted at the end of the day, and the leap second is called 23:59:60 (that is, 11:59:60 p.m.), which is then followed by the usual 00:00:00 (12:00:00 a.m.). Most nonastronomers find these leap seconds vexing. The added seconds cause problems because of their lack of predictability—knowing how many seconds there will be between 01jan2012 and 01jan2013 is not possible—and because there are not necessarily 24 hours in a day. If you use a leap second–adjusted clock, most days have 24 hours, but a few have 24 hours and 1 second. You must look at a table to find out.

From a time-series analysis point of view, the nonconstant day causes the most problems. Let’s say that you have data on blood pressure for a set of patients, taken hourly at 1:00, 2:00, . . . , and that you have tsset your data with delta(1 hour). On most days, L24.bp would be blood pressure at the same time yesterday. If the previous day had a leap second, however, and your data were recorded using a leap second–adjusted clock, there would be no observation L24.bp because 86,400 seconds before the current reading does not correspond to an on-the-hour time; 86,401 seconds before the current reading corresponds to yesterday’s time. Thus, whenever possible, using Stata’s \%tc encoding rather than \%tC is better.

When times are recorded by computers using leap second–adjusted clocks, however, avoiding \%tC is not possible. For performing most time-series analysis, the recommended procedure is to map the
%tC values to %tc and then \texttt{tsset} those. You must ask yourself whether the process you are studying is based on the clock—the nurse does something at 2 o’clock every day—or the true passage of time—the emitter spits out an electron every 86,400,000 ms.

When dealing with computer-recorded times, first find out whether the computer (and its time-recording software) use a leap second–adjusted clock. If it does, translate that to a \%tC value. Then use function \texttt{cofC()} to convert to a \%tc value and \texttt{tsset} that. If variable \texttt{T} contains the \%tC value,

\begin{verbatim}
. generate double t = cofC(T)
. format t %tc
. tsset t, delta(...)
\end{verbatim}

Function \texttt{cofC()} moves leap seconds forward: 23:59:60 becomes 00:00:00 of the next day.

Panel data

\textbf{Example 8: Time-series data for multiple groups}

Assume that we have a time series on average annual income and that we have the series for two groups: individuals who have not completed high school (\texttt{edlevel = 1}) and individuals who have (\texttt{edlevel = 2}).

\begin{verbatim}
. use https://www.stata-press.com/data/r16/tssetxmpl5, clear
. list edlevel year income, sep(0)
\end{verbatim}

<table>
<thead>
<tr>
<th>edlevel</th>
<th>year</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1988</td>
<td>14500</td>
</tr>
<tr>
<td>2</td>
<td>1989</td>
<td>14750</td>
</tr>
<tr>
<td>3</td>
<td>1990</td>
<td>14950</td>
</tr>
<tr>
<td>4</td>
<td>1991</td>
<td>15100</td>
</tr>
<tr>
<td>5</td>
<td>1989</td>
<td>22100</td>
</tr>
<tr>
<td>6</td>
<td>1990</td>
<td>22200</td>
</tr>
<tr>
<td>7</td>
<td>1992</td>
<td>22800</td>
</tr>
</tbody>
</table>

We declare the data to be a panel by typing

\begin{verbatim}
. tsset edlevel year, yearly
    panel variable: edlevel, (unbalanced)
    time variable: year, 1988 to 1992, but with a gap
    delta: 1 year
\end{verbatim}

Having \texttt{tsset} the data, we can now use time-series operators. The difference operator, for example, can be used to list annual changes in income:

\begin{verbatim}
. list edlevel year income d.income, sep(0)
\end{verbatim}

<table>
<thead>
<tr>
<th>edlevel</th>
<th>year</th>
<th>income</th>
<th>D.income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1988</td>
<td>14500</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1989</td>
<td>14750</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>1990</td>
<td>14950</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>1991</td>
<td>15100</td>
<td>150</td>
</tr>
<tr>
<td>5</td>
<td>1989</td>
<td>22100</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1990</td>
<td>22200</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>1992</td>
<td>22800</td>
<td></td>
</tr>
</tbody>
</table>
We see that in addition to producing missing values due to missing times, the difference operator correctly produced a missing value at the start of each panel. Once we have `tsset` our panel data, we can use time-series operators and be assured that they will handle missing time periods and panel changes correctly.

**Video example**

Formatting and managing dates

**Stored results**

`tsset` stores the following in `r()`:

Scalars

- `r(imin)` minimum panel ID
- `r(imax)` maximum panel ID
- `r(tmin)` minimum time
- `r(tmax)` maximum time
- `r(tdelta)` delta
- `r(gaps)` 1 if there are gaps, 0 otherwise

Macros

- `r(panelvar)` name of panel variable
- `r(timevar)` name of time variable
- `r(tdeltas)` formatted delta
- `r(tmins)` formatted minimum time
- `r(tmaxs)` formatted maximum time
- `r(tsfmt)` `ym` of time variable
- `r(unit)` units of time variable: Clock, clock, daily, weekly, monthly, quarterly, halfyearly, yearly, or generic
- `r(unit1)` units of time variable: C, c, d, w, m, q, h, y, or ""
- `r(balanced)` unbalanced, weakly balanced, or strongly balanced; panels are strongly balanced if they all have the same time values, weakly balanced if same number of observations but different time values, otherwise unbalanced

**References**


**Also see**

[TS] `tsfill` — Fill in gaps in time variable
**tssmooth** — Smooth and forecast univariate time-series data

**Description**

tssmooth creates new variable *newvar* and fills it in by passing the specified expression (usually a variable name) through the requested smoother.

**Syntax**

tssmooth *smoother* [*type*] *newvar* = *exp* [*if*] [*in*] [, ...]

<table>
<thead>
<tr>
<th>Smoother category</th>
<th><em>smoother</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Moving average</td>
<td></td>
</tr>
<tr>
<td>with uniform weights</td>
<td><em>ma</em></td>
</tr>
<tr>
<td>with specified weights</td>
<td><em>ma</em></td>
</tr>
<tr>
<td>Recursive</td>
<td></td>
</tr>
<tr>
<td>exponential</td>
<td><em>exponential</em></td>
</tr>
<tr>
<td>double exponential</td>
<td><em>dexponential</em></td>
</tr>
<tr>
<td>nonseasonal Holt–Winters</td>
<td><em>hwinters</em></td>
</tr>
<tr>
<td>seasonal Holt–Winters</td>
<td><em>shwinters</em></td>
</tr>
<tr>
<td>Nonlinear filter</td>
<td><em>nl</em></td>
</tr>
</tbody>
</table>


**Remarks and examples**

The recursive smoothers may also be used for forecasting univariate time series; indeed, the Holt–Winters methods are used almost exclusively for this. All can perform dynamic out-of-sample forecasts, and the smoothing parameters may be chosen to minimize the in-sample sum-of-squared prediction errors.

The moving-average and nonlinear smoothers are generally used to extract the trend—or signal—from a time series while omitting the high-frequency or noise components.

All smoothers work both with time-series data and panel data. When used with panel data, the calculation is performed separately within panel.

Several texts provide good introductions to the methods available in tssmooth. Chatfield (2004) discusses how these methods fit into time-series analysis in general. Abraham and Ledolter (1983); Montgomery, Johnson, and Gardiner (1990); Bowerman, O’Connell, and Koehler (2005); and Chatfield (2001) discuss using these methods for modern time-series forecasting. Becketti (2013) includes a Stata-centric discussion of these techniques. As he emphasizes, these methods often work as well as more complicated methods and are easier to explain to lay audiences. Do not dismiss these techniques as being too simplistic or inferior.

673
References
Becketti, S. 2013. Introduction to Time Series Using Stata. College Station, TX: Stata Press.

Also see

[TS] arima — ARIMA, ARMAX, and other dynamic regression models
[TS] sspace — State-space models
[TS] tsfilter — Filter a time series for cyclical components
[TS] tsset — Declare data to be time-series data
[R] smooth — Robust nonlinear smoother
**tssmooth dexponential — Double-exponential smoothing**

**Description**

`tssmooth dexponential` models the trend of a variable whose difference between changes from the previous values is serially correlated. More precisely, it models a variable whose second difference follows a low-order, moving-average process.

**Quick start**

Create `smooth` using a double-exponential smoother over `y` with `tsset` data

`tssmooth dexponential smooth=y`

As above, but forecast 10 periods out of sample

`tssmooth dexponential smooth=y, forecast(10)`

As above, but use 111 and 112 as the initial values for the recursion

`tssmooth dexponential smooth=y, forecast(10) s0(111 112)`

As above, but use 0.5 as the smoothing parameter

`tssmooth dexponential smooth=y, forecast(10) s0(111 112) parms(.5)`

Note: The above commands can also be used to apply the smoother separately to each panel of a panel dataset when a `panelvar` has been specified using `tsset` or `xtset`.

**Menu**

Statistics > Time series > Smoothers/univariate forecasters > Double-exponential smoothing
Syntax

tssmooth dexponential [type] newvar = exp [if] [in] [ , options]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>replace replaces newvar if it already exists</td>
</tr>
<tr>
<td>parms(#α)</td>
<td>use #α as smoothing parameter</td>
</tr>
<tr>
<td>samp0(#)</td>
<td>use # observations to obtain initial values for recursions</td>
</tr>
<tr>
<td>s0(#1 #2)</td>
<td>use #1 and #2 as initial values for recursions</td>
</tr>
<tr>
<td>forecast(#)</td>
<td>use # periods for the out-of-sample forecast</td>
</tr>
</tbody>
</table>

You must tsset your data before using tssmooth dexponential; see [TS] tsset.
exp may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

replace replaces newvar if it already exists.
parms(#α) specifies the parameter α for the double-exponential smoothers; 0 < #α < 1. If parms(#α) is not specified, the smoothing parameter is chosen to minimize the in-sample sum-of-squared forecast errors.
samp0(#) and s0(#1 #2) are mutually exclusive ways of specifying the initial values for the recursion.

By default, initial values are obtained by fitting a linear regression with a time trend, using the first half of the observations in the dataset; see Remarks and examples.
samp0(#) specifies that the first # be used in that regression.
s0(#1 #2) specifies that #1 #2 be used as initial values.
forecast(#) specifies the number of periods for the out-of-sample prediction; 0 ≤ # ≤ 500. The default is forecast(0), which is equivalent to not performing an out-of-sample forecast.

Remarks and examples

The double-exponential smoothing procedure is designed for series that can be locally approximated as

\[ \hat{x}_t = m_t + b_t t \]

where \( \hat{x}_t \) is the smoothed or predicted value of the series \( x \), and the terms \( m_t \) and \( b_t \) change over time. Abraham and Ledolter (1983), Bowerman, O’Connell, and Koehler (2005), and Montgomery, Johnson, and Gardiner (1990) all provide good introductions to double-exponential smoothing. Chatfield (2001, 2004) provides helpful discussions of how double-exponential smoothing relates to modern time-series methods.

The double-exponential method has been used both as a smoother and as a prediction method. [TS] tssmooth exponential shows that the single-exponential smoothed series is given by

\[ S_t = \alpha x_t + (1 - \alpha) S_{t-1} \]
where $\alpha$ is the smoothing constant and $x_t$ is the original series. The double-exponential smoother is obtained by smoothing the smoothed series,

$$S_t^{[2]} = \alpha S_t + (1 - \alpha)S_{t-1}^{[2]}$$

Values of $S_0$ and $S_0^{[2]}$ are necessary to begin the process. Per Montgomery, Johnson, and Gardiner (1990), the default method is to obtain $S_0$ and $S_0^{[2]}$ from a regression of the first $N_{\text{pre}}$ values of $x_t$ on $\tilde{t} = (1, \ldots, N_{\text{pre}} - t_0)'$. By default, $N_{\text{pre}}$ is equal to one-half the number of observations in the sample. $N_{\text{pre}}$ can be specified using the `samp0()` option.

The values of $S_0$ and $S_0^{[2]}$ can also be specified using the option `s0()`.

**Example 1: Smoothing a locally trending series**

Suppose that we had some data on the monthly sales of a book and that we wanted to smooth this series. The graph below illustrates that this series is locally trending over time, so we would not want to use single-exponential smoothing.
The following example illustrates that double-exponential smoothing is simply smoothing the smoothed series. Because the starting values are treated as time-zero values, we actually lose 2 observations when smoothing the smoothed series.

```
. use https://www.stata-press.com/data/r16/sales2
. tssmooth exponential double sm1=sales, p(.7) s0(1031)
exponential coefficient = 0.7000
sum-of-squared residuals = 13923
root mean squared error = 13.192
. tssmooth exponential double sm2=sm1, p(.7) s0(1031)
exponential coefficient = 0.7000
sum-of-squared residuals = 7698.6
root mean squared error = 9.8098
. tssmooth dexponential double sm2b=sales, p(.7) s0(1031 1031)
double-exponential coefficient = 0.7000
sum-of-squared residuals = 3724.4
root mean squared error = 6.8231
. generate double sm2c = f2.sm2
(2 missing values generated)
. list sm2b sm2c in 1/10

     sm2b    sm2c
   -------------
1.      1031      1031
2.  1028.3834  1028.3834
3.  1030.6306  1030.6306
4.  1017.8182  1017.8182
5.  1022.9380  1022.9380
6.  1026.0752  1026.0752
7.  1041.8587  1041.8587
8.  1042.8341  1042.8341
9.  1035.9571  1035.9571
10. 1030.6651  1030.6651
```

The double-exponential method can also be viewed as a forecasting mechanism. The exponential forecast method is a constrained version of the Holt–Winters method implemented in [TS] tssmooth hwinters (as discussed by Gardner [1985] and Chatfield [2001]). Chatfield (2001) also notes that the double-exponential method arises when the underlying model is an ARIMA(0,2,2) with equal roots.

This method produces predictions $\hat{x}_t$ for $t = t_1, \ldots, T + \text{forecast()}$. These predictions are obtained as a function of the smoothed series and the smoothed-smoothed series. For $t \in [t_0, T]$,

$$\hat{x}_t = \left(2 + \frac{\alpha}{1 - \alpha}\right) S_t - \left(1 + \frac{\alpha}{1 - \alpha}\right) S_t^{[2]}$$

where $S_t$ and $S_t^{[2]}$ are as given above.

The out-of-sample predictions are obtained as a function of the constant term, the linear term of the smoothed series at the last observation in the sample, and time. The constant term is $a_T = 2S_T - S_T^{[2]}$, and the linear term is $b_T = \frac{\alpha}{1 - \alpha} (S_T - S_T^{[2]})$. The $\tau$th-step-ahead out-of-sample prediction is given by

$$\hat{x}_t = a_t + \tau b_T$$
Example 2: Forecasting a locally trending series

Specifying the forecast option puts the double-exponential forecast into the new variable instead of the double-exponential smoothed series. The code given below uses the smoothed series sm1 and sm2 that were generated above to illustrate how the double-exponential forecasts are computed.

```stata
. tssmooth dexponential double f1=sales, p(.7) s0(1031 1031) forecast(4)
double-exponential coefficient = 0.7000
sum-of-squared residuals = 20737
root mean squared error = 16.1

. generate double xhat = (2 + .7/.3) * sm1 - (1 + .7/.3)* f.sm2
   (5 missing values generated)

. list xhat f1 in 1/10
```

<table>
<thead>
<tr>
<th>xhat</th>
<th>f1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1031</td>
<td>1031</td>
</tr>
<tr>
<td>1031</td>
<td>1031</td>
</tr>
<tr>
<td>1023.524</td>
<td>1023.524</td>
</tr>
<tr>
<td>1034.8039</td>
<td>1034.8039</td>
</tr>
<tr>
<td>994.0237</td>
<td>994.0237</td>
</tr>
<tr>
<td>1032.4463</td>
<td>1032.4463</td>
</tr>
<tr>
<td>1031.9015</td>
<td>1031.9015</td>
</tr>
<tr>
<td>1071.1709</td>
<td>1071.1709</td>
</tr>
<tr>
<td>1044.6454</td>
<td>1044.6454</td>
</tr>
<tr>
<td>1023.1855</td>
<td>1023.1855</td>
</tr>
</tbody>
</table>

Example 3: Choosing an optimal parameter to forecast

Generally, when you are forecasting, you do not know the smoothing parameter. tssmooth dexponential computes the double-exponential forecasts of a series and obtains the optimal smoothing parameter by finding the smoothing parameter that minimizes the in-sample sum-of-squared forecast errors.

```stata
. tssmooth dexponential f2=sales, forecast(4)
computing optimal double-exponential coefficient (0,1)
optimal double-exponential coefficient = 0.3631
sum-of-squared residuals = 16075.805
root mean squared error = 14.175598
```

The following graph describes the fit that we obtained by applying the double-exponential forecast method to our sales data. The out-of-sample dynamic predictions are not constant, as in the single-exponential case.
. line f2 sales t, title("Double exponential forecast with optimal alpha")
> ytitle(Sales) xtitle(time)

`tssmooth dexponential` automatically detects panel data from the information provided when
the dataset was `tsset`. The starting values are chosen separately for each series. If the smoothing
parameter is chosen to minimize the sum-of-squared prediction errors, the optimization is performed
separately on each panel. The stored results contain the results from the last panel. Missing values at
the beginning of the sample are excluded from the sample. After at least one value has been found,
missing values are filled in using the one-step-ahead predictions from the previous period.

**Stored results**

`tssmooth dexponential` stores the following in `r()`:

**Scalars**

- `r(N)` number of observations
- `r(alpha)` smoothing parameter
- `r(rss)` sum-of-squared errors
- `r(rmse)` root mean squared error
- `r(N_pre)` number of observations used in calculating starting values, if starting values calculated
- `r(s2_0)` initial value for linear term, i.e., $S_0^{[2]}$
- `r(s1_0)` initial value for constant term, i.e., $S_0$
- `r(linear)` final value of linear term
- `r(constant)` final value of constant term
- `r(period)` period, if filter is seasonal

**Macros**

- `r(method)` smoothing method
- `r(exp)` expression specified
- `r(timevar)` time variable specified in `tsset`
- `r(panelvar)` panel variable specified in `tsset`
Methods and formulas

A truncated description of the specified double-exponential filter is used to label the new variable. See [D] label for more information on labels.

An untruncated description of the specified double-exponential filter is saved in the characteristic tssmooth for the new variable. See [P] char for more information on characteristics.

The updating equations for the smoothing and forecasting versions are as given previously.

The starting values for both the smoothing and forecasting versions of double-exponential are obtained using the same method, which begins with the model

$$x_t = \beta_0 + \beta_1 t$$

where $x_t$ is the series to be smoothed and $t$ is a time variable that has been normalized to equal 1 in the first period included in the sample. The regression coefficient estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are obtained via OLS. The sample is determined by the option samp0(). By default, samp0() includes the first half of the observations. Given the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, the starting values are

$$S_0 = \hat{\beta}_0 - \frac{(1 - \alpha)}{\alpha} \hat{\beta}_1$$
$$S_0^{[2]} = \hat{\beta}_0 - 2 \frac{(1 - \alpha)}{\alpha} \hat{\beta}_1$$

References


Also see

[TS] tsset — Declare data to be time-series data

[TS] tssmooth — Smooth and forecast univariate time-series data
tssmooth exponential — Single-exponential smoothing

Description

tssmooth exponential models the trend of a variable whose change from the previous value is serially correlated. More precisely, it models a variable whose first difference follows a low-order, moving-average process.

Quick start

Create smooth using a single-exponential smoother over y with tsset data
   tssmooth exponential smooth=y

As above, but forecast 10 periods out of sample
   tssmooth exponential smooth=y, forecast(10)

As above, but use 111 as the initial value for the recursion
   tssmooth exponential smooth=y, forecast(10) s0(111)

As above, but use 0.5 as the smoothing parameter
   tssmooth exponential smooth=y, forecast(10) s0(111) parms(.5)

Note: The above commands can also be used to apply the smoother separately to each panel of a panel dataset when a panelvar has been specified using tsset or xtset.

Menu

Statistics > Time series > Smoothers/univariate forecasters > Single-exponential smoothing
tssmooth exponential

Syntax

tssmooth exponential [type] newvar = exp [if] [in] [ , options ]

options       Description

Main
replace        replace newvar if it already exists

parms(#,α)    use #,α as smoothing parameter

samp0(#)      use # observations to obtain initial value for recursion

s0(#)         use # as initial value for recursion

forecast(#)   use # periods for the out-of-sample forecast

You must tsset your data before using tssmooth exponential; see [TS] tsset.

exp may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

replace replaces newvar if it already exists.

parms(#,α) specifies the parameter α for the exponential smoother; 0 < #,α < 1. If parms(#,α) is not specified, the smoothing parameter is chosen to minimize the in-sample sum-of-squared forecast errors.

samp0(#) and s0(#) are mutually exclusive ways of specifying the initial value for the recursion.

samp0(#) specifies that the initial value be obtained by calculating the mean over the first # observations of the sample.

s0(#) specifies the initial value to be used.

If neither option is specified, the default is to use the mean calculated over the first half of the sample.

forecast(#) gives the number of observations for the out-of-sample prediction; 0 ≤ # ≤ 500. The default is forecast(0) and is equivalent to not forecasting out of sample.

Remarks and examples

Introduction

Exponential smoothing can be viewed either as an adaptive-forecasting algorithm or, equivalently, as a geometrically weighted moving-average filter. Exponential smoothing is most appropriate when used with time-series data that exhibit no linear or higher-order trends but that do exhibit low-velocity, aperiodic variation in the mean. Abraham and Ledolter (1983), Bowerman, O’Connell, and Koehler (2005), and Montgomery, Johnson, and Gardiner (1990) all provide good introductions to single-exponential smoothing. Chatfield (2001, 2004) discusses how single-exponential smoothing relates to modern time-series methods. For example, simple exponential smoothing produces optimal forecasts for several underlying models, including ARIMA(0,1,1) and the random-walk-plus-noise state-space model. (See Chatfield [2001, sec. 4.3.1].)
The exponential filter with smoothing parameter $\alpha$ creates the series $S_t$, where

$$S_t = \alpha X_t + (1 - \alpha) S_{t-1} \quad \text{for } t = 1, \ldots, T$$

and $S_0$ is the initial value. This is the adaptive forecast-updating form of the exponential smoother. This implies that

$$S_t = \alpha \sum_{k=0}^{T-1} (1 - \alpha)^k X_{T-k} + (1 - \alpha)^T S_0$$

which is the weighted moving-average representation, with geometrically declining weights. The choice of the smoothing constant $\alpha$ determines how quickly the smoothed series or forecast will adjust to changes in the mean of the unfiltered series. For small values of $\alpha$, the response will be slow because more weight is placed on the previous estimate of the mean of the unfiltered series, whereas larger values of $\alpha$ will put more emphasis on the most recently observed value of the unfiltered series.

**Examples**

Example 1: Smoothing a series for specified parameters

Let’s consider some examples using sales data. Here we forecast sales for three periods with a smoothing parameter of 0.4:

```stata
use https://www.stata-press.com/data/r16/sales1
.tssmooth exponential sm1=sales, parms(.4) forecast(3)
```

exponential coefficient = 0.4000
sum-of-squared residuals = 8345
root mean squared error = 12.919

To compare our forecast with the actual data, we graph the series and the forecasted series over time.

```stata
.line sm1 sales t, title("Single exponential forecast")
> ytitle(Sales) xtitle(Time)
```

![Single exponential forecast graph](image)
The graph indicates that our forecasted series may not be adjusting rapidly enough to the changes in the actual series. The smoothing parameter $\alpha$ controls the rate at which the forecast adjusts. Smaller values of $\alpha$ adjust the forecasts more slowly. Thus we suspect that our chosen value of 0.4 is too small. One way to investigate this suspicion is to ask \texttt{tssmooth exponential} to choose the smoothing parameter that minimizes the sum-of-squared forecast errors.

\texttt{. tssmooth exponential sm2=sales, forecast(3)}

\begin{verbatim}
computing optimal exponential coefficient (0,1)
optimal exponential coefficient = 0.7815
sum-of-squared residuals = 6727.7056
root mean squared error = 11.599746
\end{verbatim}

The output suggests that the value of $\alpha = 0.4$ is too small. The graph below indicates that the new forecast tracks the series much more closely than the previous forecast.

\texttt{. line sm2 t, title("Single exponential forecast with optimal alpha")}
\texttt{> ytitle(sales) xtitle(Time)}

![Graph showing single exponential forecast with optimal alpha](image)

We noted above that simple exponential forecasts are optimal for an ARIMA (0,1,1) model. (See \texttt{arima} for fitting ARIMA models in Stata.) Chatfield (2001, 90) gives the following useful derivation that relates the MA coefficient in an ARIMA (0,1,1) model to the smoothing parameter in single-exponential smoothing. An ARIMA (0,1,1) is given by

\begin{equation}
x_t - x_{t-1} = \epsilon_t + \theta \epsilon_{t-1}
\end{equation}

where $\epsilon_t$ is an independent and identically distributed white-noise error term. Thus given $\hat{\theta}$, an estimate of $\theta$, an optimal one-step prediction of $\hat{x}_{t+1}$ is $\hat{x}_{t+1} = x_t + \hat{\theta} \epsilon_t$. Because $\epsilon_t$ is not observable, it can be replaced by

\begin{equation}
\hat{\epsilon}_t = x_t - \hat{x}_{t-1}
\end{equation}

yielding

\begin{equation}
\hat{x}_{t+1} = x_t + \hat{\theta}(x_t - \hat{x}_{t-1})
\end{equation}

Letting $\hat{\alpha} = 1 + \hat{\theta}$ and doing more rearranging implies that

\begin{align*}
\hat{x}_{t+1} &= (1 + \hat{\theta})x_t - \hat{\theta}\hat{x}_{t-1} \\
\hat{x}_{t+1} &= \hat{\alpha}x_t - (1 - \hat{\alpha})\hat{x}_{t-1}
\end{align*}
Example 2: Comparing ARIMA to exponential smoothing

Let's compare the estimate of the optimal smoothing parameter of 0.7815 with the one we could obtain using \texttt{TS arima}. Below we fit an ARIMA(0,1,1) to the sales data and then remove the estimate of $\alpha$. The two estimates of $\alpha$ are quite close, given the large estimated standard error of $\hat{\theta}$.

```
    . arima sales, arima(0,1,1)
      (setting optimization to BHHH)
    Iteration 0:  log likelihood =  -189.91037
    Iteration 1:  log likelihood =  -189.62405
    Iteration 2:  log likelihood =  -189.60468
    Iteration 3:  log likelihood =  -189.60352
    Iteration 4:  log likelihood =  -189.60343
      (switching optimization to BFGS)
    Iteration 5:  log likelihood =  -189.60342
     ARIMA regression
    Sample:  2 - 50    Number of obs =  49
             Wald chi2(1) =  1.41
    Log likelihood =  -189.6034    Prob > chi2 =  0.2347

          |            OPG
    D.sales | Coef.  Std. Err.    z  P>|z|    [95% Conf. Interval]
    sales   |      .5025469  1.382727   0.36  0.716   -2.207548  3.212641
     _cons  |      .80134387
    ARMA    |                          
      ma   |     -.1986561  .1671699  -1.19  0.235   -.5263031  .1289908
     L1.    |                            
     /sigma|      11.58992  1.240607   9.34  0.000     9.158378  14.02147
```

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

```
    . di 1 + _b[ARMA:L.ma]
    .80134387
```

Example 3: Handling panel data

tssmooth exponential automatically detects panel data. Suppose that we had sales figures for five companies in long form. Running tssmooth exponential on the variable that contains all five series puts the smoothed series and the predictions in one variable in long form. When the smoothing parameter is chosen to minimize the squared prediction error, an optimal value for the smoothing parameter is chosen separately for each panel.
. use https://www.stata-press.com/data/r16/sales_cert, clear
. tsset
    panel variable:  id (strongly balanced)
    time variable:  t, 1 to 100
    delta: 1 unit
. tssmooth exponential sm5=sales, forecast(3)

-> id = 1
computing optimal exponential coefficient (0,1)
optimal exponential coefficient = 0.8702
sum-of-squared residuals = 16070.567
root mean squared error = 12.676974

-> id = 2
computing optimal exponential coefficient (0,1)
optimal exponential coefficient = 0.7003
sum-of-squared residuals = 20792.393
root mean squared error = 14.419568

-> id = 3
computing optimal exponential coefficient (0,1)
optimal exponential coefficient = 0.6927
sum-of-squared residuals = 21629
root mean squared error = 14.706801

-> id = 4
computing optimal exponential coefficient (0,1)
optimal exponential coefficient = 0.3866
sum-of-squared residuals = 22321.334
root mean squared error = 14.940326

-> id = 5
computing optimal exponential coefficient (0,1)
optimal exponential coefficient = 0.4540
sum-of-squared residuals = 20714.095
root mean squared error = 14.392392

tssmooth exponential computed starting values and chose an optimal $\alpha$ for each panel individually.

Treatment of missing values

Missing values in the middle of the data are filled in with the one-step-ahead prediction using the previous values. Missing values at the beginning or end of the data are treated as if the observations were not there.


tssmooth exponential treats observations excluded from the sample by if and in just as if they were missing.
Example 4: Handling missing data in the middle of a sample

Here the 28th observation is missing. The prediction for the 29th observation is repeated in the new series.

```
. use https://www.stata-press.com/data/r16/sales1, clear
. tssmooth exponential sm1=sales, parms(.7) forecast(3)
(output omitted)
. generate sales2=sales if t!=28
(4 missing values generated)
. tssmooth exponential sm3=sales2, parms(.7) forecast(3)
exponential coefficient  =  0.7000
sum-of-squared residuals =  6842.4
root mean squared error   =  11.817
. list t sales2 sm3 if t>25 & t<31
    +-------+--------+-------+
    | t     | sales2 | sm3   |
    +-------+--------+-------+
    | 26.   |  1011.5|  1007.5|
    | 27.   |  1028.3|  1010.3|
    | 28.   |       .|  1022.9|
    | 29.   |  1028.4|  1022.9|
    | 30.   |  1054.8|  1026.75|
    +-------+--------+-------+
```

Because the data for \( t = 28 \) are missing, the prediction for period 28 has been used in its place. This implies that the updating equation for period 29 is

\[
S_{29} = \alpha S_{28} + (1 - \alpha)S_{28} = S_{28}
\]

which explains why the prediction for \( t = 28 \) is repeated.

Because this is a single-exponential procedure, the loss of that one observation will not be noticed several periods later.

```
. generate diff = sm3-sm1 if t>28
(28 missing values generated)
. list t diff if t>28 & t<39
    +-------+-------+
    | t     | diff   |
    +-------+-------+
    | 29.   | -3.5   |
    | 30.   | -1.050049|
    | 31.   | -.3150635|
    | 32.   | -.0946045|
    | 33.   | -.0283203|
    | 34.   | -.0085449|
    | 35.   | -.0025635|
    | 36.   | -.0008545|
    | 37.   | -.0003662|
    | 38.   | -.0001221|
    +-------+-------+
```
Example 5: Handling missing data at the beginning and end of a sample

Now consider an example in which there are data missing at the beginning and end of the sample.

```
. generate sales3=sales if t>2 & t<49
(7 missing values generated)
. tssmooth exponential sm4=sales3, parms(.7) forecast(3)
  exponential coefficient = 0.7000
  sum-of-squared residuals = 6215.3
  root mean squared error = 11.624
. list t sales sales3 sm4 if t<5 | t>45

<table>
<thead>
<tr>
<th>t</th>
<th>sales</th>
<th>sales3</th>
<th>sm4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1031</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>1022.1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>1005.6</td>
<td>1005.6</td>
<td>1016.787</td>
</tr>
<tr>
<td>4</td>
<td>1025</td>
<td>1025</td>
<td>1008.956</td>
</tr>
<tr>
<td>46</td>
<td>1055.2</td>
<td>1055.2</td>
<td>1057.2</td>
</tr>
<tr>
<td>47</td>
<td>1056.8</td>
<td>1056.8</td>
<td>1055.8</td>
</tr>
<tr>
<td>48</td>
<td>1034.5</td>
<td>1034.5</td>
<td>1056.5</td>
</tr>
<tr>
<td>49</td>
<td>1041.1</td>
<td>.</td>
<td>1041.1</td>
</tr>
<tr>
<td>50</td>
<td>1056.1</td>
<td>.</td>
<td>1041.1</td>
</tr>
<tr>
<td>51</td>
<td>.</td>
<td>.</td>
<td>1041.1</td>
</tr>
<tr>
<td>52</td>
<td>.</td>
<td>.</td>
<td>1041.1</td>
</tr>
<tr>
<td>53</td>
<td>.</td>
<td>.</td>
<td>1041.1</td>
</tr>
</tbody>
</table>
```

The output above illustrates that missing values at the beginning or end of the sample cause the sample to be truncated. The new series begins with nonmissing data and begins predicting immediately after it stops.

One period after the actual data concludes, the exponential forecast becomes a constant. After the actual end of the data, the forecast at period \( t \) is substituted for the missing data. This also illustrates why the forecasted series is a constant.

Stored results

tssmooth exponential stores the following in `r()`:

Scalars
- `r(N)`: number of observations
- `r(alpha)`: \( \alpha \) smoothing parameter
- `r(rss)`: sum-of-squared prediction errors
- `r(rmse)`: root mean squared error
- `r(N_pre)`: number of observations used in calculating starting values
- `r(s1_0)`: initial value for \( S_t \)

Macros
- `r(method)`: smoothing method
- `r(exp)`: expression specified
- `r(timevar)`: time variable specified in `tsset`
- `r(panelvar)`: panel variable specified in `tsset`
Methods and formulas

The formulas for deriving smoothed series are as given in the text. When the value of $\alpha$ is not specified, an optimal value is found that minimizes the mean squared forecast error. A method of bisection is used to find the solution to this optimization problem.

A truncated description of the specified exponential filter is used to label the new variable. See [D] `label` for more information about labels.

An untruncated description of the specified exponential filter is saved in the characteristic `tssmooth` for the new variable. See [P] `char` for more information about characteristics.

References


Also see

[TS] `tsset` — Declare data to be time-series data

[TS] `tssmooth` — Smooth and forecast univariate time-series data
**tssmooth hwinters** — Holt–Winters nonseasonal smoothing

### Description

`tssmooth hwinters` is used in smoothing or forecasting a series that can be modeled as a linear trend in which the intercept and the coefficient on time vary over time.

### Quick start

Create smooth using Holt–Winters nonseasonal smoothing over y with `tsset` data

```
tssmooth hwinters smooth=y
```

As above, but forecast 10 periods out of sample

```
tssmooth hwinters smooth=y, forecast(10)
```

As above, but use 111 and 112 as the initial values for the recursion

```
tssmooth hwinters smooth=y, forecast(10) s0(111 112)
```

As above, but use 0.5 and 0.3 as the smoothing parameters

```
tssmooth hwinters smooth=y, forecast(10) s0(111 112) parms(.5 .3)
```

Note: The above commands can also be used to apply the smoother separately to each panel of a panel dataset when a `panelvar` has been specified using `tsset` or `xtset`.

### Menu

Statistics > Time series > Smoothers/univariate forecasters > Holt-Winters nonseasonal smoothing

---

691
tssmooth hwinters  

Syntax

`tssmooth hwinters [type] newvar = exp [if] [in] [ , options ]`

**options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
</tr>
<tr>
<td>replace</td>
</tr>
<tr>
<td><code>parms(#_α #_β)</code></td>
</tr>
<tr>
<td><code>samp0(#)</code></td>
</tr>
<tr>
<td><code>s0(#_cons #_lt)</code></td>
</tr>
<tr>
<td><code>forecast(#)</code></td>
</tr>
<tr>
<td><strong>Options</strong></td>
</tr>
<tr>
<td><code>diff</code></td>
</tr>
<tr>
<td><strong>Maximization</strong></td>
</tr>
<tr>
<td><code>maximize_options</code></td>
</tr>
<tr>
<td><code>from(#_α #_β)</code></td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `tssmooth hwinters`; see [TS] `tsset`.

`exp` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

**Options**

- `replace` replaces `newvar` if it already exists.
- `parms(#_α #_β)`, `0 ≤ #_α ≤ 1` and `0 ≤ #_β ≤ 1`, specifies the parameters. If `parms()` is not specified, the values are chosen by an iterative process to minimize the in-sample sum-of-squared prediction errors.

  If you experience difficulty converging (many iterations and “not concave” messages), try using `from()` to provide better starting values.

- `samp0(#)` and `s0(#_cons #_lt)` specify how the initial values `#_cons` and `#_lt` for the recursion are obtained.

  By default, initial values are obtained by fitting a linear regression with a time trend using the first half of the observations in the dataset.

  `samp0(#)` specifies that the first `#` observations be used in that regression.

  `s0(#_cons #_lt)` specifies that `#_cons` and `#_lt` be used as initial values.

- `forecast(#)` specifies the number of periods for the out-of-sample prediction; `0 ≤ # ≤ 500`. The default is `forecast(0)`, which is equivalent to not performing an out-of-sample forecast.

- `diff` specifies that the linear term is obtained by averaging the first difference of `exp_t` and the intercept is obtained as the difference of `exp` in the first observation and the mean of `D.exp_t`.

  If the `diff` option is not specified, a linear regression of `exp_t` on a constant and `t` is fit.
maximize_options controls the process for solving for the optimal \( \alpha \) and \( \beta \) when \texttt{parms()} is not specified.

\texttt{maximize_options: nodifficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), and nonrtolerance; see [R] Maximize.}\ These options are seldom used.

\texttt{from(#_alpha, #_beta, 0 < #_alpha < 1 and 0 < #_beta < 1, specifies starting values from which the optimal values of \alpha and \beta will be obtained. If from() is not specified, from(.5 .5) is used.}

**Remarks and examples**

The Holt–Winters method forecasts series of the form

\[ \tilde{x}_{t+1} = a_t + b_t t \]

where \( \tilde{x}_t \) is the forecast of the original series \( x_t \), \( a_t \) is a mean that drifts over time, and \( b_t \) is a coefficient on time that also drifts. In fact, as Gardner (1985) has noted, the Holt–Winters method produces optimal forecasts for an \texttt{ARIMA}(0,2,2) model and some local linear models. See [TS] \texttt{arima} and the references in that entry for \texttt{ARIMA} models, and see Harvey (1989) for a discussion of the local linear model and its relationship to the Holt–Winters method. Abraham and Ledolter (1983), Bowerman, O’Connell, and Koehler (2005), and Montgomery, Johnson, and Gardiner (1990) all provide good introductions to the Holt–Winters method. Chatfield (2001, 2004) provides helpful discussions of how this method relates to modern time-series analysis.

The Holt–Winters method can be viewed as an extension of double-exponential smoothing with two parameters, which may be explicitly set or chosen to minimize the in-sample sum-of-squared forecast errors. In the latter case, as discussed in \textit{Methods and formulas}, the smoothing parameters are chosen to minimize the in-sample sum-of-squared forecast errors plus a penalty term that helps to achieve convergence when one of the parameters is too close to the boundary.

Given the series \( x_t \), the smoothing parameters \( \alpha \) and \( \beta \), and the starting values \( a_0 \) and \( b_0 \), the updating equations are

\[ a_t = \alpha x_t + (1 - \alpha) (a_{t-1} + b_{t-1}) \]
\[ b_t = \beta (a_t - a_{t-1}) + (1 - \beta) b_{t-1} \]

After computing the series of constant and linear terms, \( a_t \) and \( b_t \), respectively, the \( \tau \)-step-ahead prediction of \( x_t \) is given by

\[ \tilde{x}_{t+\tau} = a_t + b_t \tau \]

---

**Example 1: Smoothing a series for specified parameters**

Below we show how to use \texttt{tssmooth h winters} with specified smoothing parameters. This example also shows that the Holt–Winters method can closely follow a series in which both the mean and the time coefficient drift over time.
Suppose that we have data on the monthly sales of a book and that we want to forecast this series with the Holt–Winters method.

```stata
use https://www.stata-press.com/data/r16/bsales
tssmooth hwinters hw1=sales, parms(.7 .3) forecast(3)
```

Specified weights:

- **alpha** = 0.7000
- **beta** = 0.3000
- sum-of-squared residuals = 2301.046
- root mean squared error = 6.192799

```stata
line sales hw1 t, title("Holt-Winters Forecast with alpha=.7 and beta=.3")
> ytitle(Sales) xtitle(Time)
```

The graph indicates that the forecasts are for linearly decreasing sales. Given $a_T$ and $b_T$, the out-of-sample predictions are linear functions of time. In this example, the slope appears to be too steep, probably because our choice of $\alpha$ and $\beta$.

### Example 2: Choosing the initial values

The graph in the previous example illustrates that the starting values for the linear and constant series can affect the in-sample fit of the predicted series for the first few observations. The previous example used the default method for obtaining the initial values for the recursion. The output below illustrates that, for some problems, the differenced-based initial values provide a better in-sample fit for the first few observations. However, the differenced-based initial values do not always outperform the regression-based initial values. Furthermore, as shown in the output below, for series of reasonable length, the predictions produced are nearly identical.
. tssmooth hwinters hw2=sales, parms(.7 .3) forecast(3) diff

Specified weights:
   alpha = 0.7000
   beta = 0.3000

sum-of-squared residuals = 2261.173
root mean squared error = 6.13891

. list hw1 hw2 if _n<6 | _n>57

<table>
<thead>
<tr>
<th>hw1</th>
<th>hw2</th>
</tr>
</thead>
<tbody>
<tr>
<td>93.31973</td>
<td>97.80807</td>
</tr>
<tr>
<td>98.40002</td>
<td>98.11447</td>
</tr>
<tr>
<td>100.8845</td>
<td>99.2267</td>
</tr>
<tr>
<td>98.50404</td>
<td>96.78276</td>
</tr>
<tr>
<td>93.62408</td>
<td>92.2452</td>
</tr>
<tr>
<td>116.5771</td>
<td>116.5771</td>
</tr>
<tr>
<td>119.2146</td>
<td>119.2146</td>
</tr>
<tr>
<td>119.2608</td>
<td>119.2608</td>
</tr>
<tr>
<td>111.0299</td>
<td>111.0299</td>
</tr>
<tr>
<td>109.2815</td>
<td>109.2815</td>
</tr>
<tr>
<td>107.5331</td>
<td>107.5331</td>
</tr>
</tbody>
</table>

When the smoothing parameters are chosen to minimize the in-sample sum-of-squared forecast errors, changing the initial values can affect the choice of the optimal $\alpha$ and $\beta$. When changing the initial values results in different optimal values for $\alpha$ and $\beta$, the predictions will also differ.

When the Holt–Winters model fits the data well, finding the optimal smoothing parameters generally proceeds well. When the model fits poorly, finding the $\alpha$ and $\beta$ that minimize the in-sample sum-of-squared forecast errors can be difficult.

Example 3: Forecasting with optimal parameters

In this example, we forecast the book sales data using the $\alpha$ and $\beta$ that minimize the in-sample squared forecast errors.
. tssmooth hwinters hw3=sales, forecast(3)  
computing optimal weights
Iteration 0:  penalized RSS = -2632.2073  (not concave)
Iteration 1:  penalized RSS = -1982.8431
Iteration 2:  penalized RSS = -1976.4236
Iteration 3:  penalized RSS = -1975.9172
Iteration 4:  penalized RSS = -1975.9036
Iteration 5:  penalized RSS = -1975.9036
Optimal weights:
alpha = 0.8209
beta = 0.0067
penalized sum-of-squared residuals = 1975.904
sum-of-squared residuals = 1975.904
root mean squared error = 5.738617

The following graph contains the data and the forecast using the optimal $\alpha$ and $\beta$. Comparing this graph with the one above illustrates how different choices of $\alpha$ and $\beta$ can lead to very different forecasts. Instead of linearly decreasing sales, the new forecast is for linearly increasing sales.

. line sales hw3 t, title("Holt-Winters Forecast with optimal alpha and beta")  
> ytitle(Sales) xtitle(Time)

 Stored results

**tssmooth hwinters** stores the following in r():

Scalars
- r(N) number of observations  
- r(alpha) $\alpha$ smoothing parameter  
- r(beta) $\beta$ smoothing parameter  
- r(rss) sum-of-squared errors  
- r(prss) penalized sum-of-squared errors, if parms() not specified  
- r(rmse) root mean squared error  
- r(N_pre) number of observations used in calculating starting values  
- r(s2_0) initial value for linear term  
- r(s1_0) initial value for constant term  
- r(linear) final value of linear term  
- r(constant) final value of constant term

Macros
- r(method) smoothing method  
- r(exp) expression specified  
- r(timevar) time variables specified in tsset  
- r(panelvar) panel variables specified in tsset
Methods and formulas

A truncated description of the specified Holt–Winters filter is used to label the new variable. See \[D\] label for more information on labels.

An untruncated description of the specified Holt–Winters filter is saved in the characteristic named tssmooth for the new variable. See \[P\] char for more information on characteristics.

Given the series, \(x_t\); the smoothing parameters, \(\alpha\) and \(\beta\); and the starting values, \(a_0\) and \(b_0\), the updating equations are

\[
\begin{align*}
a_t &= \alpha x_t + (1 - \alpha) (a_{t-1} + b_{t-1}) \\
b_t &= \beta (a_t - a_{t-1}) + (1 - \beta) b_{t-1}
\end{align*}
\]

By default, the initial values are found by fitting a linear regression with a time trend. The time variable in this regression is normalized to equal one in the first period included in the sample. By default, one-half of the data is used in this regression, but this sample can be changed using samp0().\(a_0\) is then set to the estimate of the constant, and \(b_0\) is set to the estimate of the coefficient on the time trend. Specifying the diff option sets \(b_0\) to the mean of \(D.x\) and \(a_0\) to \(x_1 - b_0\). s0() can also be used to specify the initial values directly.

Sometimes, one or both of the optimal parameters may lie on the boundary of \([0, 1]\). To keep the estimates inside \([0, 1]\), tssmooth hwinters parameterizes the objective function in terms of their inverse logits, that is, in terms of \(\exp(\alpha)/\{1 + \exp(\alpha)\}\) and \(\exp(\beta)/\{1 + \exp(\beta)\}\). When one of these parameters is actually on the boundary, this can complicate the optimization. For this reason, tssmooth hwinters optimizes a penalized sum-of-squared forecast errors. Let \(\hat{x}_t(\tilde{\alpha}, \tilde{\beta})\) be the forecast for the series \(x_t\), given the choices of \(\tilde{\alpha}\) and \(\tilde{\beta}\). Then the in-sample penalized sum-of-squared prediction errors is

\[
P = \sum_{t=1}^{T} \left[ x_t - \hat{x}_t(\tilde{\alpha}, \tilde{\beta}) \right]^2 + I_{|f(\tilde{\alpha})|>12} (|f(\tilde{\alpha})| - 12)^2 + I_{|f(\tilde{\beta})|>12} (|f(\tilde{\beta})| - 12)^2
\]

where \(f(x) = \ln \{x(1-x)\}\). The penalty term is zero unless one of the parameters is close to the boundary. When one of the parameters is close to the boundary, the penalty term will help to obtain convergence.

Acknowledgment

We thank Nicholas J. Cox of the Department of Geography at Durham University, UK, and coeditor of the Stata Journal and author of Speaking Stata Graphics for his helpful comments.

References


**Also see**

[TS] **tsset** — Declare data to be time-series data

[TS] **tssmooth** — Smooth and forecast univariate time-series data
tssmooth ma — Moving-average filter

**Description**

`tssmooth ma` creates a new series in which each observation is an average of nearby observations in the original series. The moving average may be calculated with uniform or user-specified weights. Missing periods are excluded from calculations.

**Quick start**

Create `ma1` using a second-order moving average of `y1` with `tsset` data

```
tssmooth ma ma1 = y1, window(2)
```

Also include the current observation in the average

```
tssmooth ma ma1 = y1, window(2 1)
```

Also include 4 forward terms in the average

```
tssmooth ma ma1 = y1, window(2 1 4)
```

Create `ma2` using a moving average of `y2` with weight 2 for the first lag of `y2`, 3 for its current value, 5 for its first forward value, and 4 for its second forward value

```
tssmooth ma ma2 = y2, weights(2 <3> 5 4)
```

Note: The above commands can also be used to apply the smoother separately to each panel of a panel dataset when a `panelvar` has been specified using `tsset` or `xtset`.

**Menu**

Statistics > Time series > Smoothers/univariate forecasters > Moving-average filter
Syntax

Moving average with uniform weights

```plaintext
tssmooth ma [type] newvar = exp [if] [in], window(#l[#c[#f]]) [replace]
```

Moving average with specified weights

```plaintext
tssmooth ma [type] newvar = exp [if] [in], weights([numlistl] <#c> [numlistf]) [replace]
```

You must `tsset` your data before using `tssmooth ma`; see [TS] `tsset`.
`exp` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

`window(#l[#c[#f]])` describes the span of the uniformly weighted moving average.
- `#l` specifies the number of lagged terms to be included, `0 ≤ #l ≤ one-half the number of observations in the sample`.
- `#c` is optional and specifies whether to include the current observation in the filter. A 0 indicates exclusion and 1, inclusion. The current observation is excluded by default.
- `#f` is optional and specifies the number of forward terms to be included, `0 ≤ #f ≤ one-half the number of observations in the sample`.

`weights([numlistl] <#c> [numlistf])` is required for the weighted moving average and describes the span of the moving average, as well as the weights to be applied to each term in the average. The middle term literally is surrounded by `<` and `>`, so you might type `weights(1/2 <3> 2/1)`. `numlistl` is optional and specifies the weights to be applied to the lagged terms when computing the moving average.
- `#c` is required and specifies the weight to be applied to the current term.
- `numlistf` is optional and specifies the weights to be applied to the forward terms when computing the moving average.

The number of elements in each `numlist` is limited to one-half the number of observations in the sample.

`replace` replaces `newvar` if it already exists.

Remarks and examples

Remarks are presented under the following headings:

- Overview
- Video example
Overview

Moving averages are simple linear filters of the form

\[ \hat{x}_t = \frac{\sum_{i=-l}^{f} w_i x_{t+i}}{\sum_{i=-l}^{f} w_i} \]

where

- \( \hat{x}_t \) is the moving average
- \( x_t \) is the variable or expression to be smoothed
- \( w_i \) are the weights being applied to the terms in the filter
- \( l \) is the longest lag in the span of the filter
- \( f \) is the longest lead in the span of the filter

Moving averages are used primarily to reduce noise in time-series data. Using moving averages to isolate signals is problematic, however, because the moving averages themselves are serially correlated, even when the underlying data series is not. Still, Chatfield (2004) discusses moving-average filters and provides several specific moving-average filters for extracting certain trends.

Example 1: A symmetric moving-average filter with uniform weights

Suppose that we have a time series of sales data, and we want to separate the data into two components: signal and noise. To eliminate the noise, we apply a moving-average filter. In this example, we use a symmetric moving average with a span of 5. This means that we will average the first two lagged values, the current value, and the first two forward terms of the series, with each term in the average receiving a weight of 1.

```
use https://www.stata-press.com/data/r16/sales1
tset time variable: t, 1 to 50 delta: 1 unit
tsmooth ma sm1 = sales, window(2 1 2)
The smoother applied was
(1/5)*[x(t-2) + x(t-1) + 1*x(t) + x(t+1) + x(t+2)]; x(t) = sales
```

We would like to smooth our series so that there is no autocorrelation in the noise. Below we compute the noise as the difference between the smoothed series and the series itself. Then we use `ac` (see [TS] corregram) to check for autocorrelation in the noise.
Example 2: A symmetric moving-average filter with nonuniform weights

In the previous example, there is some evidence of negative second-order autocorrelation, possibly due to the uniform weighting or the length of the filter. We are going to specify a shorter filter in which the weights decline as the observations get farther away from the current observation.

The weighted moving-average filter requires that we supply the weights to apply to each element with the weights() option. In specifying the weights, we implicitly specify the span of the filter.

Below we use the filter

\[ \hat{x}_t = \frac{1}{9}(1x_{t-2} + 2x_{t-1} + 3x_t + 2x_{t+1} + 1x_{t+2}) \]

In what follows, 1/2 does not mean one-half, it means the numlist 1 2:

```
. tsmooth ma sm2 = sales, weights( 1/2 <3> 2/1)
```

The smoother applied was

\[(1/9)*[1*x(t-2) + 2*x(t-1) + 3*x(t) + 2*x(t+1) + 1*x(t+2)]; x(t)= sales\]

. generate noise2 = sales-sm2

We compute the noise and use ac to check for autocorrelation.
The graph shows no significant evidence of autocorrelation in the noise from the second filter.

**Technical note**

`tssmooth ma` gives any missing observations a coefficient of zero in both the uniformly weighted and weighted moving-average filters. This simply means that missing values or missing periods are excluded from the moving average.

Sample restrictions, via `if` and `in`, cause the expression smoothed by `tssmooth ma` to be missing for the excluded observations. Thus sample restrictions have the same effect as missing values in a variable that is filtered in the expression. Also, gaps in the data that are longer than the span of the filter will generate missing values in the filtered series.

Because the first `l` observations and the last `f` observations will be outside the span of the filter, those observations will be set to missing in the moving-average series.

**Video example**

Moving-average smoothers
Stored results

tssmooth ma stores the following in r():

Scalars
- \( r(N) \)  number of observations
- \( r(w0) \)  weight on the current observation
- \( r(wlead\#) \)  weight on lead \#, if leads are specified
- \( r(wlag\#) \)  weight on lag \#, if lags are specified

Macros
- \( r(method) \)  smoothing method
- \( r(exp) \)  expression specified
- \( r(timevar) \)  time variable specified in tsset
- \( r(panelvar) \)  panel variable specified in tsset

Methods and formulas

The formula for moving averages is the same as previously given.

A truncated description of the specified moving-average filter labels the new variable. See \([D]\) label for more information on labels.

An untruncated description of the specified moving-average filter is saved in the characteristic tssmooth for the new variable. See \([P]\) char for more information on characteristics.

Reference


Also see

\([TS]\) tset  — Declare data to be time-series data
\([TS]\) tssmooth  — Smooth and forecast univariate time-series data
**tssmooth nl — Nonlinear filter**

### Description

`tssmooth nl` uses nonlinear smoothers to identify the underlying trend in a series.

### Quick start

Create `nly` as a running median smoother of `y` of span 5 using `tsset` data

```
tssmooth nl nly=y, smoother(3)
```

As above, but use a Hanning linear smoother

```
tssmooth nl nly=y, smoother(H)
```

As above, but smooth over `y` and then over the part of `y` that is not smooth and add the smooth components of the two steps

```
tssmooth nl nly=y, smoother(H, twice)
```

Note: The above commands can also be used to apply the smoother separately to each panel of a panel dataset when a `panelvar` has been specified using `tsset` or `xtset`.

### Menu

Statistics > Time series > Smoothers/univariate forecasters > Nonlinear filter
Syntax

\[
\text{tssmooth nl } [\text{type}] \text{ newvar} = \text{exp } [\text{if}] [\text{in}], \text{ smoother(smooth\text{er}[\text{, twice}]})
\]

[replace]

where smoother is specified as \text{Sm[ Sm[]{...}] } and Sm is one of:

\[
\{1|2|3|4|5|6|7|8|9\}[R]\n3[R]S[S|R][S|R]...\nE\nH
\]

The numbers specified in smoother represent the span of a running median smoother. For example, a number 3 specifies that each value be replaced by the median of the point and the two adjacent data values. The letter H indicates that a Hanning linear smoother, which is a span-3 smoother with binomial weights, be applied.

The letters E, S, and R are three refinements that can be combined with the running median and Hanning smoothers. First, the end points of a smooth can be given special treatment. This is specified by the E operator. Second, smoothing by 3, the span-3 running median, tends to produce flat-topped hills and valleys. The splitting operator, S, “splits” these repeated values, applies the end-point operator to them, and then “rejoins” the series. Third, it is sometimes useful to repeat an odd-span median smoother or the splitting operator until the smooth no longer changes. Following a digit or an S with an R specifies this type of repetition.

Finally, the twice operator specifies that after smoothing, the smoother be reapplied to the resulting rough, and any recovered signal be added back to the original smooth.

Letters may be specified in lowercase, if preferred. Examples of smoother[, twice] include:

\[
3RSSH \quad 3RSSH,twice \quad 4253H \quad 4253H,twice \quad 43RSR2H,twice
\]

You must \text{tsset} your data before using \text{tssmooth nl}; see [TS] \text{tsset}.

\text{exp} may contain time-series operators; see [U] 11.4.4 Time-series \text{varlists}.

Options

\underline{Main}

smoother(smooth\text{er}[\text{, twice}]}) is required; it specifies the nonlinear smoother to be used.

\text{replace} replaces \text{newvar} if it already exists.

Remarks and examples

\text{tssmooth nl} works as a front end to smooth. See [R] smooth for details.
Stored results

tssmooth nl stores the following in r():

Scalars
   r(N)         number of observations

Macros
   r(method)    nl
   r(smooth)    specified smoother
   r(timevar)   time variable specified in tsset
   r(panelvar)  panel variable specified in tsset

Methods and formulas

The methods are documented in [R] smooth.

A truncated description of the specified nonlinear filter labels the new variable. See [D] label for more information on labels.

An untruncated description of the specified nonlinear filter is saved in the characteristic tssmooth for the new variable. See [P] char for more information on characteristics.

Also see

[TS] tsset — Declare data to be time-series data
[TS] tssmooth — Smooth and forecast univariate time-series data
**tssmooth shwinters** — Holt–Winters seasonal smoothing

### Description

`tssmooth shwinters` performs the seasonal Holt–Winters method on a user-specified expression, which is usually just a variable name, and generates a new variable containing the forecasted series.

### Quick start

Create `smooth` using Holt–Winters seasonal smoothing over `y` with `tsset` data

```
tssmooth shwinters smooth=y
```

As above, but forecast 10 periods out of sample

```
tssmooth shwinters smooth=y, forecast(10)
```

As above, but use 111 and 112 as the initial values for the recursion

```
tssmooth shwinters smooth=y, forecast(10) s0(111 112)
```

As above but use 0.5, 0.3, and 0.7 as the smoothing parameters

```
tssmooth shwinters smooth=y, forecast(10) s0(111 112) ///
    parms(.5 .3 .7)
```

As above, but normalize seasonal values

```
tssmooth shwinters smooth=y, forecast(10) s0(111 112) ///
    parms(.5 .3 .7) normalize
```

Note: The above commands can also be used to apply the smoother separately to each panel of a panel dataset when a `panelvar` has been specified using `tsset` or `xtset`.

### Menu

Statistics > Time series > Smoothers/univariate forecasters > Holt-Winters seasonal smoothing
tssmooth shwinters — Holt–Winters seasonal smoothing 709

Syntax

tssmooth shwinters [ type ] newvar = exp [ if ] [ in ] [ , options ]

options Description

Main
replace replace newvar if it already exists
parm(#α #β #γ) use #α, #β, and #γ as smoothing parameters
samp0(#) use # observations to obtain initial values for recursion
s0(#cons #lt) use #cons and #lt as initial values for recursion
forecast(#) use # periods for the out-of-sample forecast
period(#) use # for period of the seasonality
additive use additive seasonal Holt–Winters method

Options
sn0_0(varname) use initial seasonal values in varname
sn0_v(newvar) store estimated initial values for seasonal terms in newvar
snt_v(newvar) store final year’s estimated seasonal terms in newvar
normalize normalize seasonal values
altstart use alternative method for computing the starting values

Maximization
maximize_options control the maximization process; seldom used
from(#α #β #γ) use #α, #β, and #γ as starting values for the parameters

You must tsset your data before using tssmooth shwinters; see [TS] tsset.
exp may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

Main
replace replaces newvar if it already exists.
parm(#α #β #γ), 0 ≤ #α ≤ 1, 0 ≤ #β ≤ 1, and 0 ≤ #γ ≤ 1, specifies the parameters. If
parm() is not specified, the values are chosen by an iterative process to minimize the in-sample
sum-of-squared prediction errors.

If you experience difficulty converging (many iterations and “not concave” messages), try using
from() to provide better starting values.
samp0(#) and s0(#cons #lt) have to do with how the initial values #cons and #lt for the recursion
are obtained.
s0(#cons #lt) specifies the initial values to be used.
samp0(#) specifies that the initial values be obtained using the first # observations of the sample.
This calculation is described under Methods and formulas and depends on whether the altstart
and additive options are also specified.

If neither option is specified, the first half of the sample is used to obtain initial values.
forecast(#) specifies the number of periods for the out-of-sample prediction; 0 ≤ # ≤ 500. The
default is forecast(0), which is equivalent to not performing an out-of-sample forecast.
period(#) specifies the period of the seasonality. If period() is not specified, the seasonality is obtained from the tsset options daily, weekly, ..., yearly; see [TS] tsset. If you did not specify one of those options when you tsset the data, you must specify the period() option. For instance, if your data are quarterly and you did not specify tsset’s quarterly option, you must now specify period(4).

By default, seasonal values are calculated, but you may specify the initial seasonal values to be used via the sn0_0(varname) option. The first period() observations of varname are to contain the initial seasonal values.

additive uses the additive seasonal Holt–Winters method instead of the default multiplicative seasonal Holt–Winters method.

Options

sn0_0(varname) specifies the initial seasonal values to use. varname must contain a complete year’s worth of seasonal values, beginning with the first observation in the estimation sample. For example, if you have monthly data, the first 12 observations of varname must contain nonmissing data. sn0_0() cannot be used with sn0_v().

sn0_v(newvar) stores in newvar the initial seasonal values after they have been estimated. sn0_v() cannot be used with sn0_0().

snt_v(newvar) stores in newvar the seasonal values for the final year’s worth of data.

normalize specifies that the seasonal values be normalized. In the multiplicative model, they are normalized to sum to one. In the additive model, the seasonal values are normalized to sum to zero.

altstarts uses an alternative method to compute the starting values for the constant, the linear, and the seasonal terms. The default and the alternative methods are described in Methods and formulas. altstarts may not be specified with s0().

Maximization

maximize_options controls the process for solving for the optimal \( \alpha \), \( \beta \), and \( \gamma \) when the parms() option is not specified.

maximize_options: nodifficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ltolerance(#), nrtolerance(#), and nonrtolerance; see [R] Maximize. These options are seldom used.

from(#\( \alpha \) #\( \beta \) #\( \gamma \)), 0 < #\( \alpha \) < 1, 0 < #\( \beta \) < 1, and 0 < #\( \gamma \) < 1, specifies starting values from which the optimal values of \( \alpha \), \( \beta \), and \( \gamma \) will be obtained. If from() is not specified, from(.5 .5 .5) is used.

Remarks and examples

Remarks are presented under the following headings:

- Introduction
- Holt–Winters seasonal multiplicative method
- Holt–Winters seasonal additive method
Introduction

The seasonal Holt–Winters methods forecast univariate series that have a seasonal component. If the amplitude of the seasonal component grows with the series, the Holt–Winters multiplicative method should be used. If the amplitude of the seasonal component is not growing with the series, the Holt–Winters additive method should be used. Abraham and Ledolter (1983), Bowerman, O’Connell, and Koehler (2005), and Montgomery, Johnson, and Gardiner (1990) provide good introductions to the Holt–Winters methods in recursive univariate forecasting methods. Chatfield (2001, 2004) provides introductions in the broader context of modern time-series analysis.

Like the other recursive methods in tssmooth, tssmooth shwinters uses the information stored by tsset to detect panel data. When applied to panel data, each series is smoothed separately, and the starting values are computed separately for each panel. If the smoothing parameters are chosen to minimize the in-sample sum-of-squared forecast errors, the optimization is performed separately on each panel.

When there are missing values at the beginning of the series, the sample begins with the first nonmissing observation. Missing values after the first nonmissing observation are filled in with forecasted values.

Holt–Winters seasonal multiplicative method

This method forecasts seasonal time series in which the amplitude of the seasonal component grows with the series. Chatfield (2001) notes that there are some nonlinear state-space models whose optimal prediction equations correspond to the multiplicative Holt–Winters method. This procedure is best applied to data that could be described by

\[ x_{t+j} = (\mu_t + \beta j)S_{t+j} + \epsilon_{t+j} \]

where \( x_t \) is the series, \( \mu_t \) is the time-varying mean at time \( t \), \( \beta \) is a parameter, \( S_t \) is the seasonal component at time \( t \), and \( \epsilon_t \) is an idiosyncratic error. See Methods and formulas for the updating equations.

Example 1: Forecasting from the multiplicative model

We have quarterly data on turkey sales by a new producer in the 1990s. The data have a strong seasonal component and an upward trend. We use the multiplicative Holt–Winters method to forecast sales for the year 2000. Because we have already tsset our data to the quarterly format, we do not need to specify the period() option.
. use https://www.stata-press.com/data/r16/turksales
. tssmooth shwinters shw1 = sales, forecast(4)

computing optimal weights

Iteration 0:  penalized RSS = -189.34609  (not concave)
Iteration 1:  penalized RSS = -108.68038  (not concave)
Iteration 2:  penalized RSS = -106.2277
Iteration 3:  penalized RSS = -106.14103
Iteration 4:  penalized RSS = -106.14093
Iteration 5:  penalized RSS = -106.14093

Optimal weights:

\[
\begin{align*}
\alpha &= 0.1310 \\
\beta &= 0.1428 \\
\gamma &= 0.2999
\end{align*}
\]

penalized sum-of-squared residuals = 106.1409
sum-of-squared residuals = 106.1409
root mean squared error = 1.628964

The graph below describes the fit and the forecast that was obtained.

. line sales shw1 t, title("Multiplicative Holt-Winters forecast")
> xtitle(Time) ytitle(Sales)

Holt–Winters seasonal additive method

This method is similar to the previous one, but the seasonal effect is assumed to be additive rather than multiplicative. This method forecasts series that can be described by the equation

\[ x_{t+j} = (\mu_t + \beta j) + S_{t+j} + \epsilon_{t+j} \]

See Methods and formulas for the updating equations.

Example 2: Forecasting from the additive model

In this example, we fit the data from the previous example to the additive model to forecast sales in the coming year. We use the \texttt{snt\_v()} option to save the last year’s seasonal terms in the new variable \texttt{seas}. 
. tssmooth shwinters shwa = sales, forecast(4) snt_v(seas) normalize additive
computing optimal weights
Iteration 0:  penalized RSS = -190.90242 (not concave)
Iteration 1:  penalized RSS = -108.8357
Iteration 2:  penalized RSS = -108.25359
Iteration 3:  penalized RSS = -107.68187
Iteration 4:  penalized RSS = -107.66444
Iteration 5:  penalized RSS = -107.66442
Iteration 6:  penalized RSS = -107.66442
Optimal weights:
alpha = 0.1219
beta = 0.1580
gamma = 0.3340
penalized sum-of-squared residuals = 107.6644
sum-of-squared residuals = 107.6644
root mean squared error = 1.640613

The output reveals that the multiplicative model has a better in-sample fit, and the graph below shows that the forecast from the multiplicative model is higher than that of the additive model.

. line shw1 shwa t if t>=tq(2000q1), title("Multiplicative and additive"
> "Holt-Winters forecasts") xtitle("Time") ytitle("Sales") legend(cols(1))

To check whether the estimated seasonal components are intuitively sound, we list the last year's seasonal components.

. list t seas if seas < .

<table>
<thead>
<tr>
<th>t</th>
<th>seas</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>-2.7533393</td>
</tr>
<tr>
<td>38</td>
<td>-0.91752573</td>
</tr>
<tr>
<td>39</td>
<td>1.8082417</td>
</tr>
<tr>
<td>40</td>
<td>1.8626233</td>
</tr>
</tbody>
</table>

The output indicates that the signs of the estimated seasonal components agree with our intuition.
Stored results

tssmooth shwinters stores the following in r():

Scalars

<table>
<thead>
<tr>
<th>r(N)</th>
<th>number of observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>r(alpha)</td>
<td>smoothing parameter</td>
</tr>
<tr>
<td>r(beta)</td>
<td>β</td>
</tr>
<tr>
<td>r(gamma)</td>
<td>γ smoothing parameter</td>
</tr>
<tr>
<td>r(prss)</td>
<td>penalized sum-of-squared errors</td>
</tr>
<tr>
<td>r(rss)</td>
<td>sum-of-squared errors</td>
</tr>
<tr>
<td>r(rmse)</td>
<td>root mean squared error</td>
</tr>
<tr>
<td>r(exp)</td>
<td>expression specified</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>r(method)</th>
<th>shwinters, additive or multiplicative</th>
</tr>
</thead>
<tbody>
<tr>
<td>r(normalize)</td>
<td>normalize, if specified</td>
</tr>
<tr>
<td>r(timevar)</td>
<td>time variable specified in tsset</td>
</tr>
<tr>
<td>r(panelvar)</td>
<td>panel variable specified in tsset</td>
</tr>
<tr>
<td>r(N_pre)</td>
<td>number of seasons used in calculating starting values</td>
</tr>
<tr>
<td>r(s2_0)</td>
<td>initial value for linear term</td>
</tr>
<tr>
<td>r(s1_0)</td>
<td>initial value for constant term</td>
</tr>
<tr>
<td>r(linear)</td>
<td>final value of linear term</td>
</tr>
<tr>
<td>r(constant)</td>
<td>final value of constant term</td>
</tr>
<tr>
<td>r(period)</td>
<td>period, if filter is seasonal</td>
</tr>
</tbody>
</table>
| r(prf)          | forecast for the series x_t given the choices of α, β, and γ. Then the in-sample penalized sum-of-squared prediction errors is

\[
P = \sum_{t=1}^{T} \left[ \left( x_t - \tilde{x}_t(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) \right)^2 + I_{|f(\tilde{\alpha})|>12}(|f(\tilde{\alpha})| - 12)^2 + I_{|f(\tilde{\beta})|>12}(|f(\tilde{\beta})| - 12)^2 + I_{|f(\tilde{\gamma})|>12}(|f(\tilde{\gamma})| - 12)^2 \right]
\]

Methods and formulas

A truncated description of the specified seasonal Holt–Winters filter labels the new variable. See [D] label for more information on labels.

An untruncated description of the specified seasonal Holt–Winters filter is saved in the characteristic named tssmooth for the new variable. See [P] char for more information on characteristics.

When the **parms** () option is not specified, the smoothing parameters are chosen to minimize the in-sample sum of penalized squared-forecast errors. Sometimes, one or more of the three optimal parameters lies on the boundary [0, 1]. To keep the estimates inside [0, 1], tssmooth shwinters parameterizes the objective function in terms of their inverse logits, that is, \(\exp(\alpha)/\{1 + \exp(\alpha)\}\), \(\exp(\beta)/\{1 + \exp(\beta)\}\), and \(\exp(\gamma)/\{1 + \exp(\gamma)\}\). When one of these parameters is actually on the boundary, this can complicate the optimization. For this reason, tssmooth shwinters optimizes a penalized sum-of-squared forecast errors. Let \(\tilde{x}_t(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})\) be the forecast for the series \(x_t\) given the choices of \(\tilde{\alpha}, \tilde{\beta}, \text{and} \tilde{\gamma}\). Then the in-sample penalized sum-of-squared prediction errors is

\[
P = \sum_{t=1}^{T} \left[ \left( x_t - \tilde{x}_t(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) \right)^2 + I_{|f(\tilde{\alpha})|>12}(|f(\tilde{\alpha})| - 12)^2 + I_{|f(\tilde{\beta})|>12}(|f(\tilde{\beta})| - 12)^2 + I_{|f(\tilde{\gamma})|>12}(|f(\tilde{\gamma})| - 12)^2 \right]
\]

where \(f(x) = \ln \left( \frac{x}{1-x} \right)\). The penalty term is zero unless one of the parameters is close to the boundary. When one of the parameters is close to the boundary, the penalty term will help to obtain convergence.

Holt–Winters seasonal multiplicative procedure

As with the other recursive methods in tssmooth, there are three aspects to implementing the Holt–Winters seasonal multiplicative procedure: the forecasting equation, the initial values, and the updating equations. Unlike in the other methods, the data are now assumed to be seasonal with period \(L\).

Given the estimates \(a(t), b(t), \text{and} s(t + \tau - L)\), a \(\tau\) step-ahead point forecast of \(x_t\), denoted by \(\hat{y}_{t+\tau}\), is

\[
\hat{y}_{t+\tau} = \{a(t) + b(t)\} s(t + \tau - L)
\]
Given the smoothing parameters $\alpha$, $\beta$, and $\gamma$, the updating equations are

\[ a(t) = \alpha \frac{x_t}{s(t-L)} + (1-\alpha) \{a(t-1) + b(t-1)\} \]

\[ b(t) = \beta \{a(t) - a(t-1)\} + (1-\beta) b(t-1) \]

and

\[ s(t) = \gamma \left\{ \frac{x_t}{a(t)} \right\} + (1-\gamma) s(t-L) \]

To restrict the seasonal terms to sum to 1 over each year, specify the `normalize` option.

The updating equations require the $L+2$ initial values $a(0)$, $b(0)$, $s(1-L)$, $s(2-L)$, \ldots, $s(0)$. Two methods calculate the initial values with the first $m$ years, each of which contains $L$ seasons. By default, $m$ is set to the number of seasons in half the sample.

The initial value of the trend component, $b(0)$, can be estimated by

\[ b(0) = \frac{\bar{x}_m - \bar{x}_1}{(m-1)L} \]

where $\bar{x}_m$ is the average level of $x_t$ in year $m$ and $\bar{x}_1$ is the average level of $x_t$ in the first year.

The initial value for the linear term, $a(0)$, is then calculated as

\[ a(0) = \bar{x}_1 - \frac{L}{2} b(0) \]

To calculate the initial values for the seasons $1, 2, \ldots, L$, we first calculate the deviation-adjusted values,

\[ S(t) = \frac{x_t}{\bar{x}_i - \left\{ \frac{(L+1)}{2} - j \right\} b(0)} \]

where $i$ is the year that corresponds to time $t$, $j$ is the season that corresponds to time $t$, and $\bar{x}_i$ is the average level of $x_t$ in year $i$.

Next, for each season $l = 1, 2, \ldots, L$, we define $\bar{s}_l$ as the average $S_t$ over the years. That is,

\[ \bar{s}_l = \frac{1}{m} \sum_{k=0}^{m-1} S_{l+kL} \quad \text{for} \ l = 1, 2, \ldots, L \]

Then the initial seasonal estimates are

\[ s_{0l} = \bar{s}_l \left( \frac{L}{\sum_{l=1}^{L} \bar{s}_l} \right) \quad \text{for} \ l = 1, 2, \ldots, L \]

and these values are used to fill in $s(1-L), \ldots, s(0)$. 

If the `altstarts` option is specified, the starting values are computed based on a regression with seasonal indicator variables. Specifically, the series $x_t$ is regressed on a time variable normalized to equal one in the first period in the sample and on a constant. Then $b(0)$ is set to the estimated coefficient on the time variable, and $a(0)$ is set to the estimated constant term. To calculate the seasonal starting values, $x_t$ is regressed on a set of $L$ seasonal dummy variables. The $l$th seasonal starting value is set to $(\frac{1}{\mu})\hat{\beta}_l$, where $\mu$ is the mean of $x_t$ and $\hat{\beta}_l$ is the estimated coefficient on the $l$th seasonal dummy variable. The sample used in both regressions and the mean computation is restricted to include the first `samp0()` years. By default, `samp0()` includes half the data.

## Technical note

If there are missing values in the first few years, a small value of $m$ can cause the starting value methods for seasonal term to fail. Here you should either specify a larger value of $m$ by using `samp0()` or directly specify the seasonal starting values by using the `snt0_0()` option.

### Holt–Winters seasonal additive procedure

This procedure is similar to the previous one, except that the data are assumed to be described by

$$x_t = (\beta_0 + \beta_1 t) + s_t + \epsilon_t$$

As in the multiplicative case, there are three smoothing parameters, $\alpha$, $\beta$, and $\gamma$, which can either be set or chosen to minimize the in-sample sum-of-squared forecast errors.

The updating equations are

$$a(t) = \alpha \{x_t - s(t - L)\} + (1 - \alpha) \{a(t - 1) + b(t - 1)\}$$

$$b(t) = \beta \{a(t) - a(t - 1)\} + (1 - \beta)b(t - 1)$$

and

$$s(t) = \gamma \{x_t - a(t)\} + (1 - \gamma)s(t - L)$$

To restrict the seasonal terms to sum to 0 over each year, specify the `normalize` option. A $\tau$-step-ahead forecast, denoted by $\hat{y}_{t+\tau}$, is given by

$$\hat{x}_{t+\tau} = a(t) + b(t)\tau + s(t + \tau - L)$$

As in the multiplicative case, there are two methods for setting the initial values. The default method is to obtain the initial values for $a(0), b(0), s(1 - L), \ldots, s(0)$ from the regression

$$x_t = a(0) + b(0)t + \beta_{s,1-L}D_1 + \beta_{s,2-L}D_2 + \cdots + \beta_{s,0}D_L + \epsilon_t$$

where the $D_1, \ldots, D_L$ are dummy variables with

$$D_i = \begin{cases} 1 & \text{if } t \text{ corresponds to season } i \\ 0 & \text{otherwise} \end{cases}$$
When `altstarts` is specified, an alternative method is used that regresses the \( x_t \) series on a time variable that has been normalized to equal one in the first period in the sample and on a constant term. \( b(0) \) is set to the estimated coefficient on the time variable, and \( a(0) \) is set to the estimated constant term. Then the demeaned series \( \tilde{x}_t = x_t - \mu \) is created, where \( \mu \) is the mean of the \( x_t \). The \( \tilde{x}_t \) are regressed on \( L \) seasonal dummy variables. The \( l \)th seasonal starting value is then set to \( \beta_l \), where \( \beta_l \) is the estimated coefficient on the \( l \)th seasonal dummy variable. The sample in both the regression and the mean calculation is restricted to include the first \( \text{samp0} \) years, where, by default, \( \text{samp0}() \) includes half the data.

**Acknowledgment**

We thank Nicholas J. Cox of the Department of Geography at Durham University, UK, and coeditor of the *Stata Journal* and author of *Speaking Stata Graphics* for his helpful comments.

**References**


**Also see**

[TS] `tsset` — Declare data to be time-series data

[TS] `tssmooth` — Smooth and forecast univariate time-series data
Unobserved-components models (UCMs) decompose a time series into trend, seasonal, cyclical, and idiosyncratic components and allow for exogenous variables. *ucm* estimates the parameters of UCMs by maximum likelihood.

All the components are optional. The trend component may be first-order deterministic or it may be first-order or second-order stochastic. The seasonal component is stochastic; the seasonal effects at each time period sum to a zero-mean finite-variance random variable. The cyclical component is modeled by the stochastic-cycle model derived by Harvey (1989).

**Quick start**

Random-walk model for \( y \) using *tsset* data

\[
\text{ucm y}
\]

Add a cyclical component of order 2

\[
\text{ucm y, cycle(2)}
\]

Add a seasonal component arising every 3 periods

\[
\text{ucm y, cycle(2) seasonal(3)}
\]

Random-walk model for \( y \) with a drift component and a cyclical component of order 1

\[
\text{ucm y, model(rwdrift) cycle(1)}
\]

Smooth-trend model for \( y \) with cyclical and seasonal components of order 2

\[
\text{ucm y, model(strend) cycle(2) seasonal(2)}
\]
Syntax

```
ucm  depvar  [ indepvars ]  [ if ]  [ in ]  [ ,  options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>model</code></td>
<td>specify trend and idiosyncratic components</td>
</tr>
<tr>
<td><code>seasonal(#)</code></td>
<td>include a seasonal component with a period of # time units</td>
</tr>
<tr>
<td><code>cycle(# [, frequency(#f)])</code></td>
<td>include a cycle component of order # and optionally set initial frequency to #f, 0 &lt; #f &lt; π; cycle() may be specified up to three times</td>
</tr>
<tr>
<td><code>constraints(constraints)</code></td>
<td>apply specified linear constraints</td>
</tr>
<tr>
<td><code>vce(vcetype)</code></td>
<td>vcetype may be oim or robust</td>
</tr>
<tr>
<td><code>level(#)</code></td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td><code>nocnsreport</code></td>
<td>do not display constraints</td>
</tr>
<tr>
<td><code>display_options</code></td>
<td>control columns and column formats, row spacing, display of omitted variables and base and empty cells, and factor-variable labeling</td>
</tr>
<tr>
<td><code>maximize_options</code></td>
<td>control the maximization process</td>
</tr>
<tr>
<td><code>collinear</code></td>
<td>keep collinear variables</td>
</tr>
<tr>
<td><code>coeflegend</code></td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rwalk</code></td>
<td>random-walk model; the default</td>
</tr>
<tr>
<td><code>none</code></td>
<td>no trend or idiosyncratic component</td>
</tr>
<tr>
<td><code>ntrend</code></td>
<td>no trend component but include idiosyncratic component</td>
</tr>
<tr>
<td><code>dconstant</code></td>
<td>deterministic constant with idiosyncratic component</td>
</tr>
<tr>
<td><code>llevel</code></td>
<td>local-level model</td>
</tr>
<tr>
<td><code>dtrend</code></td>
<td>deterministic-trend model with idiosyncratic component</td>
</tr>
<tr>
<td><code>lldtrend</code></td>
<td>local-level model with deterministic trend</td>
</tr>
<tr>
<td><code>rwdrift</code></td>
<td>random-walk-with-drift model</td>
</tr>
<tr>
<td><code>lltrend</code></td>
<td>local-linear-trend model</td>
</tr>
<tr>
<td><code>strend</code></td>
<td>smooth-trend model</td>
</tr>
<tr>
<td><code>rtrend</code></td>
<td>random-trend model</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `ucm`; see [TS] `tsset`.

`indepvars` may contain factor variables; see [U] 11.4.3 Factor variables.

`indepvars` and `depvar` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, fp, rolling, and statsby are allowed; see [U] 11.1.10 Prefix commands.

collinear and coeflegend do not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
model(model) specifies the trend and idiosyncratic components. The default is model(rwalk). The available models are listed in Syntax and discussed in detail in Models for the trend and idiosyncratic components under Remarks and examples below.

seasonal(#) adds a stochastic-seasonal component to the model. # is the period of the season, that is, the number of time-series observations required for the period to complete.

cycle(#) adds a stochastic-cycle component of order # to the model. The order # must be 1, 2, or 3. Multiple cycles are added by repeating the cycle(#) option with up to three cycles allowed.

cycle(#, frequency(#f)) specifies #f as the initial value for the central-frequency parameter in the stochastic-cycle component of order #. #f must be in the interval (0, π).

constraints(constraints); see [R] Estimation options.

vce(vcetype) specifies the estimator for the variance–covariance matrix of the estimator.

vce(oim), the default, causes ucm to use the observed information matrix estimator.

vce(robust) causes ucm to use the Huber/White/sandwich estimator.

level(#), nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nolabel, fvwrap(#) , fvwrapon(style), cformat(%,fmt), pformat(%,fmt), and sformat(%,fmt); see [R] Estimation options.

maximize_options: difficult, technique(algorithm_spec), iterate(#), [no]log, trace, gradient, showstep, hessian, showtolerance, tolerance(#), ftolerance(#), nrtolerance(#), and from(matname); see [R] Maximize for all options except from(), and see below for information on from().

from(matname) specifies initial values for the maximization process. from(b0) causes ucm to begin the maximization algorithm with the values in b0. b0 must be a row vector; the number of columns must equal the number of parameters in the model; and the values in b0 must be in the same order as the parameters in e(b).

If you model fails to converge, try using the difficult option. Also see the technical note below example 5.

The following options are available with ucm but are not shown in the dialog box:

collinear, coeflegend; see [R] Estimation options.
Remarks and examples

Remarks are presented under the following headings:

- An introduction to UCMs
- A random-walk model example
- Frequency-domain concepts used in the stochastic-cycle model
- Another random-walk model example
- Comparing UCM and ARIMA
- A local-level model example
- Comparing UCM and ARIMA, revisited
- Models for the trend and idiosyncratic components
- Seasonal component

An introduction to UCMs

UCMs decompose a time series into trend, seasonal, cyclical, and idiosyncratic components and allow for exogenous variables. Formally, UCMs can be written as

\[ y_t = \tau_t + \gamma_t + \psi_t + \beta x_t + \epsilon_t \]  

where \( y_t \) is the dependent variable, \( \tau_t \) is the trend component, \( \gamma_t \) is the seasonal component, \( \psi_t \) is the cyclical component, \( \beta \) is a vector of fixed parameters, \( x_t \) is a vector of exogenous variables, and \( \epsilon_t \) is the idiosyncratic component.

By placing restrictions on \( \tau_t \) and \( \epsilon_t \), Harvey (1989) derived a series of models for the trend and the idiosyncratic components. These models are briefly described in Syntax and are further discussed in Models for the trend and idiosyncratic components. To these models, Harvey (1989) added models for the seasonal and cyclical components, and he also allowed for the presence of exogenous variables.

It is rare that a UCM contains all the allowed components. For instance, the seasonal component is rarely needed when modeling deseasonalized data.

Harvey (1989) and Durbin and Koopman (2012) show that UCMs can be written as state-space models that allow the parameters of a UCM to be estimated by maximum likelihood. In fact, ucm uses sspace (see [TS] sspace) to perform the estimation calculations; see Methods and formulas for details.

After estimating the parameters, predict can produce in-sample predictions or out-of-sample forecasts; see [TS] ucm postestimation. After estimating the parameters of a UCM that contains a cyclical component, estat period converts the estimated central frequency to an estimated central period and psdensity estimates the spectral density implied by the model; see [TS] ucm postestimation and the examples below.

We illustrate the basic approach of analyzing data with UCMs, and then we discuss the details of the different trend models in Models for the trend and idiosyncratic components.

Although the methods implemented in ucm have been widely applied by economists, they are general time-series techniques and may be of interest to researchers from other disciplines. In example 8, we analyze monthly data on the reported cases of mumps in New York City.
A random-walk model example

Example 1

We begin by plotting monthly data on the U.S. civilian unemployment rate.

```
use https://www.stata-press.com/data/r16/unrate
tsline unrate, name(unrate)
```

This series looks like it might be well approximated by a random-walk model. Formally, a random-walk model is given by

\[
y_t = \mu_t \\
\mu_t = \mu_{t-1} + \eta_t
\]
The random-walk is so frequently applied, at least as a starting model, that it is the default model for ucm. In the output below, we fit the random-walk model to the unemployment data.

```
    . ucm unrate
    searching for initial values .............
    (setting technique to bhhh)
    Iteration 0:  log likelihood =  84.272992
    Iteration 1:  log likelihood =  84.394942
    Iteration 2:  log likelihood =  84.400923
    Iteration 3:  log likelihood =  84.401282
    Iteration 4:  log likelihood =  84.401305
    (switching technique to nr)
    Iteration 5:  log likelihood =  84.401306
    Refining estimates:
    Iteration 0:  log likelihood =  84.401306
    Iteration 1:  log likelihood =  84.401307
    Unobserved-components model
    Components: random walk
    Sample:  1948m1 - 2011m1 Number of obs   =   757
    Log likelihood =  84.401307

                    OIM
  unrate          Coef.     Std. Err.     z   P>|z|     [95% Conf. Interval]
    var(level)   .0467196   .0024030   19.44   0.000   .0420098   .0514294
```

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The output indicates that the model is nonstationary, as all random-walk models are.

We consider a richer model in the next example.

Example 2

We suspect that there should be a stationary cyclical component that produces serially correlated shocks around the random-walk trend. Harvey (1989) derived a stochastic-cycle model for these stationary cyclical components.

The stochastic-cycle model has three parameters: the frequency at which the random components are centered, a damping factor that parameterizes the dispersion of the random components around the central frequency, and the variance of the stochastic-cycle process that acts as a scale factor.
Fitting this model to unemployment data yields

```
. ucm unrate, cycle(1)
searching for initial values ......................
(setting technique to bhhh)
Iteration 0:  log likelihood =  84.273579
Iteration 1:  log likelihood =  87.852115
Iteration 2:  log likelihood =  88.253422
Iteration 3:  log likelihood =  89.191311
Iteration 4:  log likelihood =  94.675898
(switching technique to nr)
Iteration 5:  log likelihood =  98.394691  (not concave)
Iteration 6:  log likelihood =  98.983093
Iteration 7:  log likelihood =  104.8309
Iteration 8:  log likelihood =  114.27142
Iteration 9:  log likelihood =  116.4741
Iteration 10: log likelihood =  118.45816
Iteration 11: log likelihood =  118.88056
Iteration 12: log likelihood =  118.88421
Iteration 13: log likelihood =  118.88421
Iteration 14: log likelihood =  118.88421
Refining estimates:
Iteration 0:  log likelihood =  118.88421
Iteration 1:  log likelihood =  118.88421

Unobserved-components model
Components: random walk, order 1 cycle
Sample: 1948m1 - 2011m1  Number of obs   =   757
Wald chi2(2)   = 26650.81
Log likelihood = 118.88421  Prob > chi2   = 0.0000

+-----------------+-----------------+-----------------+-----------------+-----------------+
|                | Coef.         | Std. Err.      |     z     | P>|z|    | [95% Conf. Interval]   |
|-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| unrate          |                |                |              |              |                |
| frequency       | 0.0933466      | 0.0103609       | 9.01        | 0.000    | 0.0730397 1136535  |
| damping         | 0.9820003      | 0.0061121       | 160.66      | 0.000    | 0.9700207 9939798   |
| var(level)      | 0.0143786      | 0.0051392       | 2.80        | 0.003    | 0.004306 0244511   |
| var(cycle1)     | 0.0270339      | 0.0043433       | 4.97        | 0.000    | 0.0163829 0376848   |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
```

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The estimated central frequency for the cyclical component is small, implying that the cyclical component is centered on low-frequency components. The high-damping factor indicates that all the components from this cyclical component are close to the estimated central frequency. The estimated variance of the stochastic-cycle process is small but significant.

We use `estat period` to convert the estimate of the central frequency to an estimated central period.

```
. estat period

+-----------------+-----------------+-----------------+-----------------+-----------------+
| cycle1          | Coef.         | Std. Err.      | [95% Conf. Interval]   |
|-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| period          | 67.31029        | 7.471004        | 52.6674 81.95319 |
| frequency       | 0.0933466       | 0.0103609       | 0.0730397 1136535  |
| damping         | 0.9820003       | 0.0061121       | 0.9700207 9939798   |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
```

Note: Cycle time unit is monthly.
Because we have monthly data, the estimated central period of 67.31 implies that the cyclical component is composed of random components that occur around a central periodicity of about 5.61 years. This estimate falls within the conventional Burns and Mitchell (1946) definition of business-cycle shocks occurring between 1.5 and 8 years.

We can convert the estimated parameters of the cyclical component to an estimated spectral density of the cyclical component, as described by Harvey (1989). The spectral density of the cyclical component describes the relative importance of the random components at different frequencies; see Frequency-domain concepts used in the stochastic-cycle model for details. We use psdensity (see [TS] psdensity) to obtain the spectral density of the cyclical component implied by the estimated parameters, and we use twoway line (see [G-2] graph twoway line) to plot the estimated spectral density.

```
. psdensity sdensity omega
. line sdensity omega
```

The estimated spectral density shows that the cyclical component is composed of random components that are tightly distributed at the low-frequency peak.

Frequency-domain concepts used in the stochastic-cycle model

The parameters of the stochastic-cycle model are easiest to interpret in the frequency domain. We now provide a review of the useful concepts from the frequency domain. Crucial to understanding the stochastic-cycle model is the frequency-domain concept that a stationary process can be decomposed into random components that occur at the frequencies in the interval $[0, \pi]$.

We need some concepts from the frequency-domain approach to interpret the parameters in the stochastic-cycle model of the cyclical component. Here we provide a simple, intuitive explanation. More technical presentations can be found in Priestley (1981), Harvey (1989, 1993), Hamilton (1994), Fuller (1996), and Wei (2006).

As with much time-series analysis, the basic results are for covariance-stationary processes with additional results handling some nonstationary cases. We present some useful results for covariance-stationary processes. These results provide what we need to interpret the stochastic-cycle model for the stationary cyclical component.
The autocovariances $\gamma_j, j \in \{0, 1, \ldots, \infty\}$, of a covariance-stationary process $y_t$ specify its variance and dependence structure. In the frequency-domain approach to time-series analysis, the spectral density describes the importance of the random components that occur at frequency $\omega$ relative to the components that occur at other frequencies.

The frequency-domain approach focuses on the relative contributions of random components that occur at the frequencies $[0, \pi]$.

The spectral density can be written as a weighted average of the autocorrelations of $y_t$. Like autocorrelations, the spectral density is normalized by $\gamma_0$, the variance of $y_t$. Multiplying the spectral density by $\gamma_0$ yields the power-spectrum of $y_t$.

In an independent and identically distributed (i.i.d.) process, the components at all frequencies are equally important, so the spectral density is a flat line.

In common parlance, we speak of high-frequency noise making a series look more jagged and of low-frequency components causing smoother plots. More formally, we say that a process composed primarily of high-frequency components will have fewer runs above or below the mean than an i.i.d. process and that a process composed primarily of low-frequency components will have more runs above or below the mean than an i.i.d. process.

To further formalize these ideas, consider the first-order autoregressive (AR(1)) process given by

$$y_t = \phi y_{t-1} + \epsilon_t$$

where $\epsilon_t$ is a zero-mean, covariance-stationary process with finite variance $\sigma^2$, and $|\phi| < 1$ so that $y_t$ is covariance stationary. The first-order autocorrelation of this AR(1) process is $\phi$.

Below are plots of simulated data when $\phi$ is set to 0, −0.8, and 0.8. When $\phi = 0$, the data are i.i.d. When $\phi = −0.8$, the value today is strongly negatively correlated with the value yesterday, so this case should be a prototypical high-frequency noise example. When $\phi = 0.8$, the value today is strongly positively correlated with the value yesterday, so this case should be a prototypical low-frequency shock example.

The plots above confirm our conjectures. The plot when $\phi = −0.8$ contains fewer runs above or below the mean, and it is more jagged than the i.i.d. plot. The plot when $\phi = 0.8$ contains more runs above or below the mean, and it is smoother than the i.i.d. plot.
Below we plot the spectral densities for the AR(1) model with $\phi = 0$, $\phi = -0.8$, and $\phi = 0.8$.

The high-frequency components are much more important to the AR(1) process with $\phi = -0.8$ than to the i.i.d. process with $\phi = 0$. The low-frequency components are much more important to the AR(1) process with $\phi = 0.8$ than to the i.i.d. process.

**Technical note**

Autoregressive moving-average (ARMA) models parameterize the autocorrelation in a time series by allowing today’s value to be a weighted average of past values and a weighted average of past i.i.d. shocks; see Hamilton (1994), Wei (2006), and [TS arima] for introductions and a Stata implementation. The intuitive ARMA parameterization has many nice features, including that one can easily rewrite the ARMA model as a weighted average of past i.i.d. shocks to trace how a shock feeds through the system.

Although it is easy to obtain the spectral density of an ARMA process, the parameters themselves provide limited information about the underlying spectral density.

In contrast, the parameters of the stochastic-cycle parameterization of autocorrelation in a time series directly provide information about the underlying spectral density. The parameter $\omega_0$ is the central frequency at which the random components are clustered. If $\omega_0$ is small, then the model is centered on low-frequency components. If $\omega_0$ is close to $\pi$, then the model is centered on high-frequency components. The parameter $\rho$ is the damping factor that indicates how tightly clustered the random components are at the central frequency $\omega_0$. If $\rho$ is close to 0, there is no clustering of the random components. If $\rho$ is close to 1, the random components are tightly distributed at the central frequency $\omega_0$.

In the graph below, we draw the spectral densities implied by stochastic-cycle models with four sets of parameters: $\omega_0 = \pi/4, \rho = 0.8$; $\omega_0 = \pi/4, \rho = 0.9$; $\omega_0 = 4\pi/5, \rho = 0.8$; and $\omega_0 = 4\pi/5, \rho = 0.9$. The graph below illustrates that $\omega_0$ is the central frequency at which the other important random components are distributed. It also illustrates that the damping parameter $\rho$ controls the dispersion of the important components at the central frequency.
Another random-walk model example

Example 3

Now let’s reconsider example 2. Although we might be happy with how our model has identified a stationary cyclical component that we could interpret in business-cycle terms, we suspect that there should also be a high-frequency cyclical component. It is difficult to estimate the parameters of a UCM with two or more stochastic-cycle models. Providing starting values for the central frequencies can be a crucial help to the optimization procedure. Below we estimate a UCM with two cyclical components.

We use the `frequency()` suboption to provide starting values for the central frequencies; we specified the values below because we suspect one model will pick up the low-frequency components and the other will pick up the high-frequency components. We specified the low-frequency model to be order 2 to make it less peaked for any given damping factor. (Trimbur [2006] provides a nice introduction and some formal results for higher-order stochastic-cycle models.)

```
. ucm unrate, cycle(1, frequency(2.9)) cycle(2, frequency(.09))
searching for initial values ......................
(setting technique to bhhh)
Iteration 0:  log likelihood = 115.98563
Iteration 1:  log likelihood = 125.04043
Iteration 2:  log likelihood = 127.69387
Iteration 3:  log likelihood = 134.50864
Iteration 4:  log likelihood = 136.91353
(switching technique to nr)
Iteration 0:  log likelihood = 146.28326
Iteration 1:  log likelihood = 146.28326
Refining estimates:
Iteration 0:  log likelihood = 146.28326
Iteration 1:  log likelihood = 146.28326
```

```
Unobserved-components model
Components: random walk, 2 cycles of order 1 2
Sample: 1948m1 - 2011m1  
Number of obs = 757
Wald chi2(4) = 7681.33
Log likelihood = 146.28326  Prob > chi2 = 0.0000

| unrate | OIM                  | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|--------|----------------------|-------|-----------|------|-----|---------------------|
| cycle1 | frequency            | 2.882382 | .0668017 | 43.15 | 0.000 | 2.751453 3.013311 |
|        | damping              | .7004295 | .1251571 | 5.60 | 0.000 | .4551261 .9457329 |
| cycle2 | frequency            | .0667929 | .0206849 | 3.23 | 0.001 | .0262513 .1073346 |
|        | damping              | .9074708 | .0142273 | 63.78 | 0.000 | .8795858 .9353559 |
| var(level) |                  | .0207704 | .0039669 | 5.24 | 0.000 | .0129953 .0285454 |
| var(cycle1) |               | .0027886 | .0014363 | 1.94 | 0.026 | 0 .0056037 |
| var(cycle2) |              | .002714  | .0010281 | 2.64 | 0.004 | .0006991 .004729 |

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The output provides some support for the existence of a second, high-frequency cycle. The high-frequency components are centered at 2.88, whereas the low-frequency components are centered at 0.067. That the estimated damping factor is 0.70 for the high-frequency cycle whereas the estimated damping factor for the low-frequency cycle is 0.91 indicates that the high-frequency components are more diffusely distributed at 2.88 than the low-frequency components are at 0.067.

We obtain and plot the estimated spectral densities to get another look at these results.

```
pdensity sdensity2a omega2a
psdensity sdensity2b omega2b, cycle(2)
line sdensity2a sdensity2b omega2a, legend(col(1))
```

The estimated spectral densities indicate that we have found two distinct cyclical components.
It does not matter whether we specify $\omega_2a$ or $\omega_2b$ to be the $x$-axis variable, because they are equal to each other.

Technical note

That the estimated spectral densities in the previous example do not overlap is important for parameter identification. Although the parameters are identified in large-sample theory, we have found it difficult to estimate the parameters of two cyclical components when the spectral densities overlap. When the spectral densities of two cyclical components overlap, the parameters may not be well identified and the optimization procedure may not converge.

Comparing UCM and ARIMA

Example 4

This example provides some insight for readers familiar with autoregressive integrated moving-average (ARIMA) models but not with UCMs. If you are not familiar with ARIMA models, you may wish to skip this example. See [TS] arima for an introduction to ARIMA models in Stata.

UCMs provide an alternative to ARIMA models implemented in [TS] arima. Neither set of models is nested within the other, but there are some cases in which instructive comparisons can be made.

The random-walk model corresponds to an ARIMA model that is first-order integrated and has an i.i.d. error term. In other words, the random-walk UCM and the ARIMA(0,1,0) are asymptotically equivalent. Thus

\[ \text{ucm unrate} \]

and

\[ \text{arima unrate, arima(0,1,0) noconstant} \]

produce asymptotically equivalent results.

The stochastic-cycle model for the stationary cyclical component is an alternative functional form for stationary processes to stationary autoregressive moving-average (ARMA) models. Which model is preferred depends on the application and which parameters a researchers wants to interpret. Both the functional forms and the parameter interpretations differ between the stochastic-cycle model and the ARMA model. See Trimbur (2006, eq. 25) for some formal comparisons of the two models.

That both models can be used to estimate the stationary cyclical components for the random-walk model implies that we can compare the results in this case by comparing their estimated spectral densities. Below we estimate the parameters of an ARIMA(2,1,1) model and plot the estimated spectral density of the stationary component.
. arima unrate, noconstant arima(2,1,1)
(setting optimization to BHHH)
Iteration 0:  log likelihood =  129.8801
Iteration 1:  log likelihood =  134.61953
Iteration 2:  log likelihood =  137.04909
Iteration 3:  log likelihood =  137.71386
Iteration 4:  log likelihood =  138.25255
(switching optimization to BFGS)
Iteration 5:  log likelihood =  138.51924
Iteration 6:  log likelihood =  138.81638
Iteration 7:  log likelihood =  138.83615
Iteration 8:  log likelihood =  138.8364
Iteration 9:  log likelihood =  138.83642
Iteration 10: log likelihood =  138.83642

ARIMA regression
Sample:  1948m2 - 2011m1  Number of obs  =  756
Log likelihood =  138.8364   Prob > chi2     =  0.0000

Wald chi2(3)  =  683.34

OPG

D.unrate      Coef.   Std. Err.    z     P>|z|     [95% Conf. Interval]
ARMA
   ar
L1.     .5398016    .0586304    9.21     0.000     .4248882    .6547151
L2.     .2468148    .0359396    6.87     0.000     .1763744    .3172551
   ma
L1.    -.5146506    .0632838   -8.13     0.000    -.6386845   -.3906167
   /sigma    .2013332    .0032644   61.68     0.000     .1949351    .2077313

Note: The test of the variance against zero is one sided, and the two-sided
confidence interval is truncated at zero.

. psdensity sdensity_arma omega_arma
. line sdensity_arma omega_arma

The estimated spectral density from the ARIMA(2,1,1) has a similar shape to the plot obtained by combining the two spectral densities estimated from the stochastic-cycle model in example 3. For this particular application, the estimated central frequencies of the two cyclical components from the
stochastic-cycle model provide information about the business-cycle component and the high-frequency component that is not easily obtained from the ARIMA(2,1,1) model. On the other hand, it is easier to work out the impulse–response function for the ARMA model than for the stochastic-cycle model, implying that the ARMA model is easier to use when tracing the effect of a shock feeding through the system.

A local-level model example

We now consider the weekly series of initial claims for unemployment insurance in the United States, which is plotted below.

Example 5

```
. use https://www.stata-press.com/data/r16/icsa1, clear
. tsline icsa
```

This series looks like it was generated by a random walk with extra noise, so we want to use a random-walk model that includes an additional random term. This structure causes the model to be occasionally known as the random-walk-plus-noise model, but it is more commonly known as the local-level model in the UCM literature.

The local-level model models the trend as a random walk and models the idiosyncratic components as independent and identically distributed components. Formally, the local-level model specifies the observed time-series $y_t$, for $t = 1, \ldots, T$, as

\[
y_t = \mu_t + \epsilon_t
\]

\[
\mu_t = \mu_{t-1} + \eta_t
\]

where $\epsilon_t \sim \text{i.i.d. } N(0, \sigma_\epsilon^2)$ and $\eta_t \sim \text{i.i.d. } N(0, \sigma_\eta^2)$ and are mutually independent.
We fit the local-level model in the output below:

```
.ucm icsa, model(llevel)
searching for initial values ...........
(setting technique to bhhh)
Iteration 0: log likelihood = -9954.8223
Iteration 1: log likelihood = -9917.406
Iteration 2: log likelihood = -9905.6679
Iteration 3: log likelihood = -9897.7588
Iteration 4: log likelihood = -9894.2015
(setting technique to nr)
Iteration 5: log likelihood = -9893.4337
Iteration 6: log likelihood = -9893.2469
Iteration 7: log likelihood = -9893.2469
Refining estimates:
Iteration 0: log likelihood = -9893.2469
Iteration 1: log likelihood = -9893.2469
Unobserved-components model
Components: local level
Sample: 07jan1967 - 19feb2011 Number of obs = 2,303
Log likelihood = -9893.2469

<table>
<thead>
<tr>
<th></th>
<th>OIM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coef.</td>
</tr>
<tr>
<td>var(level)</td>
<td>116.558</td>
</tr>
<tr>
<td>var(icsa)</td>
<td>124.2715</td>
</tr>
</tbody>
</table>
```

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
Note: Time units are in 7 days.

The output indicates that both components are statistically significant.

### Technical note

The estimation procedure will not always converge when estimating the parameters of the local-level model. If the series does not vary enough in the random level, modeled by the random walk, and in the stationary shocks around the random level, the estimation procedure will not converge because it will be unable to set the variance of one of the two components to 0.

Take another look at the graphs of `unrate` and `icsa`. The extra noise around the random level that can be seen in the graph of `icsa` allows us to estimate both variances.

A closely related point is that it is difficult to estimate the parameters of a local-level model with a stochastic-cycle component because the series must have enough variation to identify the variance of the random-walk component, the variance of the idiosyncratic term, and the parameters of the stochastic-cycle component. In some cases, series that look like candidates for the local-level model are best modeled as random-walk models with stochastic-cycle components.

In fact, convergence can be a problem for most of the models in `ucm`. Convergence problems occur most often when there is insufficient variation to estimate the variances of the components in the model. When there is insufficient variation to estimate the variances of the components in the model, the optimization routine will fail to converge as it attempts to set the variance equal to 0. This usually shows up in the iteration log when the log likelihood gets stuck at a particular value and the message (not concave) or (backed up) is displayed repeatedly. When this happens, use the
iterate() option to limit the number of iterations, look to see which of the variances is being driven to 0, and drop that component from the model. (This technique is a method to obtain convergence to interpretable estimates, not a model-selection method.)

Example 6

We might suspect that there is some serial correlation in the idiosyncratic shock. Alternatively, we could include a cyclical component to model the stationary time-dependence in the series. In the example below, we add a stochastic-cycle model for the stationary cyclical process, but we drop the idiosyncratic term and use a random-walk model instead of the local-level model. We change the model because it is difficult to estimate the variance of the idiosyncratic term along with the parameters of a stationary cyclical component.

```
. ucm icsa, model(rwalk) cycle(1)
searching for initial values ........................
(setting technique to bhhh)
Iteration 0: log likelihood = -10055.453
Iteration 1: log likelihood = -10047.163
Iteration 2: log likelihood = -10047.146 (backed up)
Iteration 3: log likelihood = -10047.146 (backed up)
Iteration 4: log likelihood = -10047.145 (backed up)
(switching technique to nr)
Iteration 5: log likelihood = -10047.142 (not concave)
Iteration 6: log likelihood = -9889.8038
Iteration 7: log likelihood = -9883.967
Iteration 8: log likelihood = -9883.381 (not concave)
Iteration 9: log likelihood = -9883.381 (not concave)
Iteration 10: log likelihood = -9883.381 (not concave)
Iteration 11: log likelihood = -9883.379 (not concave)
Iteration 12: log likelihood = -9883.376 (not concave)
Iteration 13: log likelihood = -9883.3684 (not concave)
Iteration 14: log likelihood = -9882.0687 (not concave)
Iteration 15: log likelihood = -9881.6615
Iteration 16: log likelihood = -9881.4511
Iteration 17: log likelihood = -9881.4441
Iteration 18: log likelihood = -9881.4441
Refining estimates:
Iteration 0: log likelihood = -9881.4441
Iteration 1: log likelihood = -9881.4441
```

Unobserved-components model
Components: random walk, order 1 cycle
Sample: 07jan1967 - 19feb2011
Number of obs = 2,303
Wald chi2(2) = 23.04
Log likelihood = -9881.4441
Prob > chi2 = 0.0000

```

|                | Coef.    | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|----------------|----------|-----------|-------|------|----------------------|
| frequency      | 1.469633 | .3855657  | 3.81  | 0.000| .7139385 2.225328    |
| damping        | .1644576 | .0349537  | 4.71  | 0.000| .0959495 .2329656    |
| var(level)     | 97.90982 | 8.320047  | 11.77 | 0.000| 81.60282 114.2168    |
| var(cycle1)    | 149.7323 | 9.980798  | 15.00 | 0.000| 130.1703 169.2943    |
```

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
Note: Time units are in 7 days.
Although the output indicates that the model fits well, the small estimate of the damping parameter indicates that the random components will be widely distributed at the central frequency. To get a better idea of the dispersion of the components, we look at the estimated spectral density of the stationary cyclical component.

```
psdensity sdensity3 omega3
line sdensity3 omega3
```

The graph shows that the random components that make up the cyclical component are diffusely distributed at a central frequency.

---

Comparing UCM and ARIMA, revisited

Example 7

Including lags of the dependent variable is an alternative method for modeling serially correlated errors. The estimated coefficients on the lags of the dependent variable estimate the coefficients in an autoregressive model for the stationary cyclical component; see Harvey (1989, 47–48) for a discussion. Including lags of the dependent variable should be viewed as an alternative to the stochastic-cycle model for the stationary cyclical component. In this example, we use the large-sample equivalence of the random-walk model with $p$th order autoregressive errors and an ARIMA($p$, 1, 0) to illustrate this point.
In the output below, we include 2 lags of the dependent variable in the random-walk UCM.

```
. ucm icsa L(1/2).icsa, model(rwalk)
searching for initial values .......... 
(setting technique to bhhh)  
Iteration 0: log likelihood = -10026.649
Iteration 1: log likelihood = -9947.9671
Iteration 2: log likelihood = -9896.4778
Iteration 3: log likelihood = -9890.8199
Iteration 4: log likelihood = -9890.3202
(switching technique to nr)  
Iteration 5: log likelihood = -9890.1546
Iteration 6: log likelihood = -9889.561
Iteration 7: log likelihood = -9889.5608
Refining estimates:  
Iteration 0: log likelihood = -9889.5608
Iteration 1: log likelihood = -9889.5608
```

Unobserved-components model
Components: random walk
Sample: 21jan1967 - 19feb2011  
Number of obs = 2,301
Wald chi2(2) = 271.88
Log likelihood = -9889.5608  
Prob > chi2 = 0.0000

<table>
<thead>
<tr>
<th></th>
<th>OIM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coef.  Std. Err.  z  P&gt;</td>
</tr>
<tr>
<td>icsa</td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td>-.3250633  .0205148  -15.85  0.000  -.3652715  -.2848551</td>
</tr>
<tr>
<td>L2.</td>
<td>-.1794686  .0205246  -8.74  0.000  -.2196961  -.1392411</td>
</tr>
<tr>
<td>icsa</td>
<td></td>
</tr>
<tr>
<td>L1.</td>
<td></td>
</tr>
<tr>
<td>L2.</td>
<td></td>
</tr>
<tr>
<td>var(level)</td>
<td>317.6474  9.36691  33.91  0.000  299.2886  336.0062</td>
</tr>
</tbody>
</table>

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
Note: Time units are in 7 days.

Now we use `arima` to estimate the parameters of an asymptotically equivalent ARIMA(2,1,0) model. (We specify the `technique(nr)` option so that `arima` will compute the observed information matrix standard errors that `ucm` computes.) We use `nlcom` to compute a point estimate and a standard error for the variance, which is directly comparable to the one produced by `ucm`. 
. arima icsa, noconstant arima(2,1,0) technique(nr)
Iteration 0:  log likelihood = -9896.4584
Iteration 1:  log likelihood = -9896.458

ARIMA regression
Sample:  14jan1967 - 19feb2011 Number of obs = 2302
Wald chi2(2) = 271.95
Log likelihood = -9896.458  Prob > chi2 = 0.0000

| D.icsa | Coef.  | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|--------|--------|-----------|------|------|---------------------|
|        | GIM    |           |      |      |                     |
| ARMA   |        |           |      |      |                     |
| ar     | -0.3249383 | 0.0205036 | -15.85 | 0.000 | -0.3651246 -0.284752 |
| L1.    | -0.1793353 | 0.0205088 | -8.74  | 0.000 | -0.2195317 -0.1391388 |
| /sigma | 17.81606  | 0.2625695 | 67.85  | 0.000 | 17.30143 18.33068 |

Note: The test of the variance against zero is one sided, and the two-sided confidence interval is truncated at zero.

. nlcom _b[σ:_cons]^2
   _nl_1: _b[σ:_cons]^2

| D.icsa | Coef.  | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|--------|--------|-----------|------|------|---------------------|
| _nl_1  | 317.4119 | 9.355904 | 33.93 | 0.000 | 299.0746 335.7491 |

It is no accident that the parameter estimates and the standard errors from the two estimators are so close. As the sample size grows the differences in the parameter estimates and the estimated standard errors will go to 0, because the two estimators are equivalent in large samples.

Models for the trend and idiosyncratic components

A general model that allows for fixed or stochastic trends in \( \tau_t \) is given by

\[
\tau_t = \tau_{t-1} + \beta_{t-1} + \eta_t \\
\beta_t = \beta_{t-1} + \xi_t
\]  

(2)  
(3)

Following Harvey (1989), we define 11 flexible models for \( y_t \) that specify both \( \tau_t \) and \( \epsilon_t \) in (1). These models place restrictions on the general model specified in (2) and (3) and on \( \epsilon_t \) in (1). In other words, these models jointly specify \( \tau_t \) and \( \epsilon_t \).

To any of these models, a cyclical component, a seasonal component, or exogenous variables may be added.
The majority of the models available in `ucm` are designed for nonstationary time series. The deterministic-trend model incorporates a first-order deterministic time-trend in the model. The local-level, random-walk, local-level-with-deterministic-trend, and random-walk-with-drift models are for modeling series with first-order stochastic trends. A series with a $d$th-order stochastic trend must be differenced $d$ times to be stationary. The local-linear-trend, smooth-trend, and random-trend models are for modeling series with second-order stochastic trends.

The no-trend-or-idsyncratic-component model is useful for using `ucm` to model stationary series with cyclical components or seasonal components and perhaps exogenous variables. The no-trend and the deterministic-constant models are useful for using `ucm` to model stationary series with seasonal components or exogenous variables.

### Table 1. Models for the trend and idiosyncratic components

<table>
<thead>
<tr>
<th>Model name</th>
<th>Syntax option</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>No trend or idiosyncratic component</td>
<td>model(none)</td>
<td>$y_t = \epsilon_t$</td>
</tr>
<tr>
<td>No trend</td>
<td>model(ntrend)</td>
<td>$y_t = \mu + \epsilon_t$</td>
</tr>
<tr>
<td>Deterministic constant</td>
<td>model(dconstant)</td>
<td>$y_t = \mu \quad \mu = \mu$</td>
</tr>
<tr>
<td>Local level</td>
<td>model(llevel)</td>
<td>$y_t = \mu_t + \epsilon_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \eta_t$</td>
</tr>
<tr>
<td>Random walk</td>
<td>model(rwalk)</td>
<td>$y_t = \mu_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \eta_t$</td>
</tr>
<tr>
<td>Deterministic trend</td>
<td>model(dtrend)</td>
<td>$y_t = \mu_t + \epsilon_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \beta$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta = \beta$</td>
</tr>
<tr>
<td>Local level with deterministic trend</td>
<td>model(lldtrtend)</td>
<td>$y_t = \mu_t + \epsilon_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \beta + \eta_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta = \beta$</td>
</tr>
<tr>
<td>Random walk with drift</td>
<td>model(rwdrift)</td>
<td>$y_t = \mu_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \beta + \eta_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta = \beta$</td>
</tr>
<tr>
<td>Local linear trend</td>
<td>model(lltrend)</td>
<td>$y_t = \mu_t + \epsilon_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \beta_{t-1} + \eta_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta_t = \beta_{t-1} + \xi_t$</td>
</tr>
<tr>
<td>Smooth trend</td>
<td>model(strend)</td>
<td>$y_t = \mu_t + \epsilon_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \beta_{t-1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta_t = \beta_{t-1} + \xi_t$</td>
</tr>
<tr>
<td>Random trend</td>
<td>model(rtrend)</td>
<td>$y_t = \mu_t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu_t = \mu_{t-1} + \beta_{t-1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta_t = \beta_{t-1} + \xi_t$</td>
</tr>
</tbody>
</table>
Seasonal component

A seasonal component models cyclical behavior in a time series that occurs at known seasonal periodicities. A seasonal component is modeled in the time domain; the period of the cycle is specified as the number of time periods required for the cycle to complete.

Example 8

Let’s begin by considering a series that displays a seasonal effect. Below we plot a monthly series containing the number of new cases of mumps in New York City between January 1928 and December 1972. (See Hipel and McLeod [1994] for the source and further discussion of this dataset.)

```
use https://www.stata-press.com/data/r16/mumps, clear
tslineline mumps
```

The graph reveals recurring spikes at regular intervals, which we suspect to be seasonal effects. The series may or may not be stationary; the graph evidence is not definitive.

Deterministic seasonal effects are a standard method of incorporating seasonality into a model. In a model with a constant term, the $s$ deterministic seasonal effects are modeled as $s$ parameters subject to the constraint that they sum to zero; formally, $\gamma_t + \gamma_{t-1} + \cdots + \gamma_{t-(s-1)} = 0$. A stochastic-seasonal model is a more flexible alternative that allows the seasonal effects at time $t$ to sum to $\zeta_t$, a zero-mean, finite-variance, i.i.d. random variable; formally, $\gamma_t + \gamma_{t-1} + \cdots + \gamma_{t-(s-1)} = \zeta_t$.

In the output below, we model the seasonal effects by a stochastic-seasonal model, we allow for the series to follow a random walk, and we include a stationary cyclical component.
740 ucm — Unobserved-components model

. ucm mumps, seasonal(12) cycle(1)
searching for initial values ....................
(setting technique to bhhh)
 Iteration 0:  log likelihood = -3270.1579
 Iteration 1:  log likelihood = -3257.7346
 Iteration 2:  log likelihood = -3257.1819
 Iteration 3:  log likelihood = -3249.8570
 Iteration 4:  log likelihood = -3249.5035
(switching technique to nr)
 Iteration 5:  log likelihood = -3248.9152
 Iteration 6:  log likelihood = -3248.7244
 Iteration 7:  log likelihood = -3248.7138
 Iteration 8:  log likelihood = -3248.7138
Refining estimates:
 Iteration 0:  log likelihood = -3248.7138
 Iteration 1:  log likelihood = -3248.7138

Unobserved-components model
Components: random walk, seasonal(12), order 1 cycle
Sample: 1928m1 - 1972m6  Number of obs = 534
 Wald chi2(2) = 2141.69
Log likelihood = -3248.7138  Prob > chi2 = 0.0000

| Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|-------|-----------|------|------|----------------------|
| mumps |           |      |      |                      |
| frequency | .3863607 | .0282037 | 13.70 | 0.000 | .3310824 | .4416389 |
| damping  | .8405622 | .0197933 | 42.47 | 0.000 | .8017681 | .8793563 |
| var(level) | 221.2131 | 140.5179 | 1.57  | 0.058 | 0 | 496.6231 |
| var(seasonal) | 4.151639 | 4.383442 | 0.95  | 0.172 | 0 | 12.74303 |
| var(cycle1) | 12228.17 | 813.8394 | 15.03 | 0.000 | 10633.08 | 13823.27 |

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The output indicates that the trend and seasonal variances may not be necessary. When the variance of the seasonal component is zero, the seasonal component becomes deterministic. Below we estimate the parameters of a model that includes deterministic seasonal effects and a stationary cyclical component.

. ucm mumps ibn.month, model(none) cycle(1)
searching for initial values ..........
(setting technique to bhhh)
 Iteration 0:  log likelihood = -3934.0178
 Iteration 1:  log likelihood = -3615.0098
 Iteration 2:  log likelihood = -3502.5223
 Iteration 3:  log likelihood = -3407.9644
 Iteration 4:  log likelihood = -3368.2264
(switching technique to nr)
 Iteration 5:  log likelihood = -3352.1077
 Iteration 6:  log likelihood = -3284.5218
 Iteration 7:  log likelihood = -3283.0588
 Iteration 8:  log likelihood = -3283.0284
 Iteration 9:  log likelihood = -3283.0284
Refining estimates:
 Iteration 0:  log likelihood = -3283.0284
 Iteration 1:  log likelihood = -3283.0284
Unobserved-components model
Components: order 1 cycle
Sample: 1928m1 - 1972m6
Number of obs = 534
Wald chi2(14) = 3404.29
Prob > chi2 = 0.0000
Log likelihood = -3283.0284

| mumps | Coef.  | Std. Err. | z      | P>|z| | [95% Conf. Interval] |
|--------|--------|-----------|--------|------|---------------------|
| cycle1 |        |           |        |      |                     |
| frequency | 0.3272753 | 0.0262922 | 12.45  | 0.000 | 0.2757436 - 0.3788071 |
| damping  | 0.844874  | 0.0184994 | 45.67  | 0.000 | 0.8086157 - 0.8811322 |

<table>
<thead>
<tr>
<th>mumps</th>
<th>month</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>480.5095</td>
<td>32.67128</td>
<td>14.71</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>561.9174</td>
<td>32.66999</td>
<td>17.20</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>832.8666</td>
<td>32.67696</td>
<td>25.49</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>894.0747</td>
<td>32.64568</td>
<td>27.39</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>869.6568</td>
<td>32.56282</td>
<td>26.71</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>770.1562</td>
<td>32.48587</td>
<td>23.71</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>433.839</td>
<td>32.50165</td>
<td>13.35</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>218.2394</td>
<td>32.56712</td>
<td>6.70</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>140.686</td>
<td>32.64138</td>
<td>4.31</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>148.5876</td>
<td>32.69067</td>
<td>4.55</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>215.0958</td>
<td>32.70311</td>
<td>6.58</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>330.2232</td>
<td>32.68906</td>
<td>10.10</td>
<td>0.000</td>
</tr>
</tbody>
</table>

| var(cycle1) | 13031.53 | 798.2719 | 16.32 | 0.000 | 11466.95 - 14596.11 |

Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.

The output indicates that each of these components is statistically significant.

Technical note

In a stochastic model for the seasonal component, the seasonal effects sum to the random variable $\zeta_t \sim$ i.i.d. $N(0, \sigma_\zeta^2)$:

$$\gamma_t = - \sum_{j=1}^{s-1} \gamma_{t-j} + \zeta_t$$

Stored results

Because ucm is estimated using **sspace**, most of the **sspace** stored results appear after ucm. Not all of these results are relevant for ucm; programmers wishing to treat ucm results as **sspace** results should see **Stored results** of [TS] **sspace**. See **Methods and formulas** for the state-space representation of UCMs, and see [TS] **sspace** for more documentation that relates to all the stored results.
ucm stores the following in e():

Scalars

- **e(N)**: number of observations
- **e(k)**: number of parameters
- **e(k_eq)**: number of equations in e(b)
- **e(k_dv)**: number of dependent variables
- **e(k_cycles)**: number of stochastic cycles
- **e(df_m)**: model degrees of freedom
- **e(ll)**: log likelihood
- **e(chi2)**: $\chi^2$ value
- **e(p)**: $p$-value for model test
- **e(tmin)**: minimum time in sample
- **e(tmax)**: maximum time in sample
- **e(stationary)**: 1 if the estimated parameters indicate a stationary model, 0 otherwise
- **e(rank)**: rank of VCE
- **e(ic)**: number of iterations
- **e(rc)**: return code
- **e(converged)**: 1 if converged, 0 otherwise

Macros

- **e(cmd)**: ucm
- **e(cmdline)**: command as typed
- **e(depvar)**: unoperated names of dependent variables in observation equations
- **e(covariates)**: list of covariates
- **e(tvar)**: variable denoting time within groups
- **e(eqnames)**: names of equations
- **e(model)**: type of model
- **e(title)**: title in estimation output
- **e(tmins)**: formatted minimum time
- **e(tmaxs)**: formatted maximum time
- **e(chi2type)**: Wald; type of model $\chi^2$ test
- **e(vcetype)**: vcetype specified in vce()
- **e(opt)**: type of optimization
- **e(initial_values)**: type of initial values
- **e(technique)**: maximization technique
- **e(tech_steps)**: iterations taken in maximization technique
- **e(properties)**: b V
- **e(estat_cmd)**: program used to implement estat
- **e(predict)**: program used to implement predict
- **e(marginsnotok)**: predictions disallowed by margins
- **e(asbalanced)**: factor variables fvset as asbalanced
- **e(asobserved)**: factor variables fvset as asobserved

Matrices

- **e(b)**: parameter vector
- **e(Cns)**: constraints matrix
- **e(llog)**: iteration log (up to 20 iterations)
- **e(gradient)**: gradient vector
- **e(V)**: variance–covariance matrix of the estimators
- **e(V_modelbased)**: model-based variance

Functions

- **e(sample)**: marks estimation sample
Methods and formulas

Methods and formulas are presented under the following headings:

Introduction
State-space formulation
Cyclical component extensions

Introduction

The general form of UCMs can be expressed as

\[ y_t = \tau_t + \gamma_t + \psi_t + x_t\beta + \epsilon_t \]

where \( \tau_t \) is the trend, \( \gamma_t \) is the seasonal component, \( \psi_t \) is the cycle, \( \beta \) is the regression coefficients for regressors \( x_t \), and \( \epsilon_t \) is the idiosyncratic error with variance \( \sigma^2_\epsilon \).

We can decompose the trend as

\[ \tau_t = \mu_t \]
\[ \mu_t = \mu_{t-1} + \alpha_{t-1} + \eta_t \]
\[ \alpha_t = \alpha_{t-1} + \xi_t \]

where \( \mu_t \) is the local level, \( \alpha_t \) is the local slope, and \( \eta_t \) and \( \xi_t \) are i.i.d. normal errors with mean 0 and variance \( \sigma^2_\eta \) and \( \sigma^2_\xi \), respectively.

Next consider the seasonal component, \( \gamma_t \), with a period of \( s \) time units. Ignoring a seasonal disturbance term, the seasonal effects will sum to zero, \( \sum_{j=0}^{s-1} \gamma_{t-j} = 0 \). Adding a normal error term, \( \omega_t \), with mean 0 and variance \( \sigma^2_\omega \), we express the seasonal component as

\[ \gamma_t = -\sum_{j=1}^{s-1} \gamma_{t-j} + \omega_t \]

Finally, the cyclical component, \( \psi_t \), is a function of the frequency \( \lambda \), in radians, and a unit-less scaling variable \( \rho \), termed the damping effect, \( 0 < \rho < 1 \). We require two equations to express the cycle:

\[ \psi_t = \psi_{t-1}\rho \cos \lambda + \tilde{\psi}_{t-1}\rho \sin \lambda + \kappa_t \]
\[ \tilde{\psi}_t = -\psi_{t-1}\rho \sin \lambda + \tilde{\psi}_{t-1}\rho \cos \lambda + \tilde{\kappa}_t \]

where the \( \kappa_t \) and \( \tilde{\kappa}_t \) disturbances are normally distributed with mean 0 and variance \( \sigma^2_\kappa \).

The disturbance terms \( \epsilon_t, \eta_t, \xi_t, \omega_t, \kappa_t, \) and \( \tilde{\kappa}_t \) are independent.

State-space formulation

ucm is an easy-to-use implementation of the state-space command \texttt{sspace}, with special modifications, where the local linear trend components, seasonal components, and cyclical components are states of the state-space model. The state-space model can be expressed in matrix form as

\[ y_t = Dz_t + Fx_t + \epsilon_t \]
\[ z_t = Az_{t-1} + C\zeta_t \]
where $y_t$, $t = 1, \ldots, T$, are the observations and $z_t$ are the unobserved states. The number of states, $m$, depends on the model specified. The $k \times 1$ vector $x_t$ contains the exogenous variables specified as indepvars, and the $1 \times k$ vector $F$ contains the regression coefficients to be estimated. $\epsilon_t$ is the observation equation disturbance, and the $m_0 \times 1$ vector $\zeta_t$ contains the state equation disturbances, where $m_0 \leq m$. Finally, $C$ is a $m \times m_0$ matrix of zeros and ones. These recursive equations are evaluated using the diffuse Kalman filter of De Jong (1991).

Below we give the state-space matrix structures for a local linear trend with a stochastic seasonal component, with a period of 4 time units, and an order-2 cycle. The state vector, $z_t$, and its transition matrix, $A$, have the structure

$$
A = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
$$

$$
\begin{pmatrix}
\mu_t \\
\alpha_t \\
\gamma_t \\
\gamma_{t-1} \\
\gamma_{t-2} \\
\psi_{t,1} \\
\tilde{\psi}_{t,1} \\
\psi_{t,2} \\
\tilde{\psi}_{t,2} \\
\end{pmatrix}
$$

$$
\begin{pmatrix}
\mu_t \\
\alpha_t \\
\gamma_t \\
\gamma_{t-1} \\
\gamma_{t-2} \\
\psi_{t,1} \\
\tilde{\psi}_{t,1} \\
\psi_{t,2} \\
\tilde{\psi}_{t,2} \\
\end{pmatrix}
$$

$$
C = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
$$

$$
\begin{pmatrix}
\eta_t \\
\xi_t \\
\omega_t \\
\kappa_t \\
\tilde{\kappa}_t \\
\end{pmatrix}
$$

$$
D = \begin{pmatrix}
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{pmatrix}
$$

**Cyclical component extensions**

Recall that the stochastic cyclical model is given by

$$
\psi_t = \rho(\psi_{t-1} \cos \lambda_c + \psi_{t-1}^* \sin \lambda_c) + \kappa_{t,1}
$$

$$
\psi_t^* = \rho(-\psi_{t-1} \sin \lambda_c + \psi_{t-1}^* \cos \lambda_c) + \kappa_{t,2}
$$

where $\kappa_{t,j} \sim \text{i.i.d.} \ N(0, \sigma^2_\kappa)$ and $0 < \rho < 1$ is a damping effect. The cycle is variance-stationary when $\rho < 1$ because $\text{Var}(\psi_t) = \sigma^2_\kappa / (1 - \rho)$. We will express a UCM with a cyclical component added to a trend as

$$
y_t = \mu_t + \psi_t + \epsilon_t
$$

where $\mu_t$ can be any of the trend parameterizations discussed earlier.
Higher-order cycles, \( k = 2 \) or \( k = 3 \), are defined as

\[
\psi_{t,j} = \rho (\psi_{t-1,j} \cos \lambda_c + \psi_{t-1,j+1}^* \sin \lambda_c) + \psi_{t-1,j+1}
\]

\[
\psi_{t,j}^* = \rho (-\psi_{t-1,j} \sin \lambda_c + \psi_{t-1,j}^* \cos \lambda_c) + \psi_{t-1,j+1}
\]

for \( j < k \), and

\[
\psi_{t,k} = \rho (\psi_{t-1,k} \cos \lambda_c + \psi_{t-1,k}^* \sin \lambda_c) + \kappa_{t,1}
\]

\[
\psi_{t,k}^* = \rho (-\psi_{t-1,k} \sin \lambda_c + \psi_{t-1,k}^* \cos \lambda_c) + \kappa_{t,2}
\]

Harvey and Trimbur (2003) discuss the properties of this model and its state-space formulation.

Andrew Charles Harvey (1947– ) is a British econometrician. After receiving degrees in economics and statistics from the University of York and the London School of Economics and working for a period in Kenya, he has worked as a teacher and researcher at the University of Kent, the London School of Economics, and now the University of Cambridge. Harvey’s interests are centered on time series, especially state-space models, signal extraction, volatility, and changes in quantiles.

References


Also see

[TS] ucm postestimation — Postestimation tools for ucm

[TS] arima — ARIMA, ARMAX, and other dynamic regression models

[TS] sspace — State-space models

[TS] tsfilter — Filter a time series for cyclical components

[TS] tsset — Declare data to be time-series data

[TS] tssmooth — Smooth and forecast univariate time-series data

[TS] var — Vector autoregressive models

[U] 20 Estimation and postestimation commands
Postestimation commands

The following postestimation commands are of special interest after `ucm`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat period</code></td>
<td>display cycle periods in time units</td>
</tr>
<tr>
<td><code>psdensity</code></td>
<td>estimate the spectral density</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaïke’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

    predict creates a new variable containing predictions such as linear predictions, trend components, seasonal components, cyclical components, and standardized and unstandardized residuals. The root mean squared error is available for all predictions. All predictions are also available as static one-step-ahead predictions or as dynamic multistep predictions, and you can control when dynamic predictions begin.

Menu for predict

    Statistics  >  Postestimation

Syntax for predict

    predict [type] { [stub** | newvarlist] [ if ] [ in ] [, statistic options ]

statistic                      Description

Main

    xb                        linear prediction using exogenous variables
    trend                      trend component
    seasonal                   seasonal component
    cycle                      cyclical component
    residuals                  residuals
    rstandard                  standardized residuals

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for the estimation sample.

options                      Description

Options

    rmse( stub* | newvarlist )  put estimated root mean squared errors of predicted statistics in the new variable
    dynamic( time_constant )    begin dynamic forecast at specified time

Advanced

    smethod( method )           method for predicting unobserved components

method                      Description

    onestep                    predict using past information
    smooth                     predict using all sample information
    filter                     predict using past and contemporaneous information
Options for predict

Main

xb, trend, seasonal, cycle, residuals, and rstandard specify the statistic to be predicted.

xb, the default, calculates the linear predictions using the exogenous variables. xb may not be used with the smethod(filter) option.

trend estimates the unobserved trend component.

seasonal estimates the unobserved seasonal component.

cycle estimates the unobserved cyclical component.

residuals calculates the residuals in the equation for the dependent variable. residuals may not be specified with dynamic().

rstandard calculates the standardized residuals, which are the residuals normalized to have unit variances. rstandard may not be specified with the smethod(filter), smethod(smooth), or dynamic() option.

Options

rmse(stub* | newvarlist) puts the root mean squared errors of the predicted statistic into the specified new variable. Multiple variables are only required for predicting cycles of a model that has more than one cycle. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error. The stub* syntax is for models with multiple cycles, where you provide the prefix and predict will add a numeric suffix for each predicted cycle.

dynamic(time_constant) specifies when predict should start producing dynamic forecasts. The specified time_constant must be in the scale of the time variable specified in tsset, and the time_constant must be inside a sample for which observations on the dependent variable are available. For example, dynamic(tq(2008q4)) causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] Datetime. If the model contains exogenous variables, they must be present for the whole predicted sample. dynamic() may not be specified with the rstandard, residuals, or smethod(smooth) option.

Advanced

smethod(method) specifies the method for predicting the unobserved components. smethod() causes different amounts of information on the dependent variable to be used in predicting the components at each time period.

smethod(onestep), the default, causes predict to estimate the components at each time period using previous information on the dependent variable. The Kalman filter is performed on previous periods, but only the one-step predictions are made for the current period.

smethod(smooth) causes predict to estimate the components at each time period using all the sample data by the Kalman smoother. smethod(smooth) may not be specified with the rstandard option.

smethod(filter) causes predict to estimate the components at each time period using previous and contemporaneous data by the Kalman filter. The Kalman filter is performed on previous periods and the current period. smethod(filter) may not be specified with the xb option.
estat

Description for estat

`estat period` transforms an estimated central frequency to an estimated period after `ucm`.

Menu for estat

Statistics > Postestimation

Syntax for estat

```
estat period [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>level(#)</em></td>
<td>set confidence level; default is <em>level(95)</em></td>
</tr>
<tr>
<td><em>cformat(</em>%fmt_)_</td>
<td>numeric format</td>
</tr>
</tbody>
</table>

Options for estat period

- _level(#)_ specifies the confidence level, as a percentage, for confidence intervals. The default is _level(95)_ or as set by _set level_; see [U] 20.8 Specifying the width of confidence intervals.
- _cformat(_%fmt_)_ sets the display format for the table numeric values. The default is _cformat(_%9.0g)_.

Remarks and examples

We assume that you have already read [TS] ucm. In this entry, we illustrate some features of `predict` after using `ucm` to estimate the parameters of an unobserved-components model.

All predictions after `ucm` depend on the unobserved components, which are estimated recursively using a Kalman filter. Changing the sample can alter the state estimates, which can change all other predictions.
Example 1

We begin by modeling monthly data on the median duration of employment spells in the United States. We include a stochastic-seasonal component because the data have not been seasonally adjusted.

```
use https://www.stata-press.com/data/r16/uduration2
(BLS data, not seasonally adjusted)
.ucm duration, seasonal(12) cycle(1) difficult
searching for initial values ....................
(setting technique to bhhh)
Iteration 0: log likelihood = -409.79557
Iteration 1: log likelihood = -403.3831
Iteration 2: log likelihood = -403.37373 (backed up)
Iteration 3: log likelihood = -403.36901 (backed up)
Iteration 4: log likelihood = -403.36663 (backed up)
(switching technique to nr)
Iteration 5: log likelihood = -403.36656 (backed up)
Iteration 6: log likelihood = -397.87756 (not concave)
Iteration 7: log likelihood = -396.47272 (not concave)
Iteration 8: log likelihood = -394.50895
Iteration 9: log likelihood = -392.9516 (not concave)
Iteration 10: log likelihood = -389.38063 (not concave)
Iteration 11: log likelihood = -388.59491
Iteration 12: log likelihood = -388.30078
Iteration 13: log likelihood = -388.25666
Iteration 14: log likelihood = -388.25675
Iteration 15: log likelihood = -388.25675
Refining estimates:
Iteration 0: log likelihood = -388.25675
Iteration 1: log likelihood = -388.25675
Unobserved-components model
Components: random walk, seasonal(12), order 1 cycle
Sample: 1967m7 - 2008m12 Number of obs = 498
Wald chi2(2) = 7.17
Log likelihood = -388.25675 Prob > chi2 = 0.0277

| OIM             | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-----------------|--------|-----------|-------|------|----------------------|
| duration        |        |           |       |      |                      |
| frequency       | 1.641531 | .7250323  | 2.26  | 0.024 | .2204938            | 3.062568         |
| damping         | .2671232 | .1050168  | 2.54  | 0.011 | .0612939            | .4729524          |
| var(level)      | .1262922 | .0221428  | 5.70  | 0.000 | .0828932            | .1696912          |
| var(seasonal)   | .0017289 | .0009647  | 1.79  | 0.037 | 0                    | .0036196          |
| var(cycle1)     | .0641496 | .0211839  | 3.03  | 0.001 | .0226299            | .1056693          |
```

Note: Model is not stationary.
Note: Tests of variances against zero are one sided, and the two-sided confidence intervals are truncated at zero.
Below we predict the trend and the seasonal components to get a look at the model fit.

- `predict strend, trend`
- `predict season, seasonal`
- `tsline duration strend, name(trend) nodraw legend(rows(1))`
- `tsline season, name(season) yline(0,lwidth(vthin)) nodraw`
- `graph combine trend season, rows(2)`

The trend tracks the data well. That the seasonal component appears to change over time indicates that the stochastic-seasonal component might fit better than a deterministic-seasonal component.
Example 2

In this example, we use the model to forecast the median unemployment duration. We use the root mean squared error of the prediction to compute a confidence interval of our dynamic predictions. Recall that the root mean squared error accounts for variances due to the disturbances but not due to the estimation error.

```
.tsappend, add(12)
.predict duration_f, dynamic(tm(2009m1)) rmse(rmse)
.scalar z = invnormal(0.95)
generate lbound = duration_f - z*rmse if tm>=tm(2008m12)  
(497 missing values generated)
generate ubound = duration_f + z*rmse if tm>=tm(2008m12)  
(497 missing values generated)
.label variable lboun "90% forecast interval"
.twoway (tsline duration duration_f if tm>=tm(2006m1))
> (tsrline lbound ubound if tm>=tm(2008m12)),
> ysize(2) xtitle("") legend(cols(1))
```

The model forecasts a large temporary increase in the median duration of unemployment.

### Methods and formulas

For details on the ucm postestimation methods, see \[\text{TS}\] sspace postestimation.

See \[\text{TS}\] psdensity for the methods used to estimate the spectral density.

### Also see

- \[\text{TS}\] ucm — Unobserved-components model
- \[\text{TS}\] psdensity — Parametric spectral density estimation after arima, arfima, and ucm
- \[\text{TS}\] sspace postestimation — Postestimation tools for sspace
- \[\text{U}\] 20 Estimation and postestimation commands
var intro — Introduction to vector autoregressive models

Description

Stata has a suite of commands for fitting, forecasting, interpreting, and performing inference on vector autoregressive (VAR) models and structural vector autoregressive (SVAR) models. The suite includes several commands for estimating and interpreting impulse–response functions (IRFs), dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs). The table below describes the available commands.

Fitting a VAR or SVAR

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Fit vector autoregressive models</td>
</tr>
<tr>
<td>svar</td>
<td>Fit structural vector autoregressive models</td>
</tr>
<tr>
<td>varbasic</td>
<td>Fit a simple VAR and graph IRFs or FEVDs</td>
</tr>
</tbody>
</table>

Model diagnostics and inference

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varstable</td>
<td>Check the stability condition of VAR or SVAR estimates</td>
</tr>
<tr>
<td>varsoc</td>
<td>Obtain lag-order selection statistics for ARs and VECMs</td>
</tr>
<tr>
<td>varwle</td>
<td>Obtain Wald lag-exclusion statistics after var or svar</td>
</tr>
<tr>
<td>vargranger</td>
<td>Perform pairwise Granger causality tests after var or svar</td>
</tr>
<tr>
<td>varlmar</td>
<td>Perform LM test for residual autocorrelation after var or svar</td>
</tr>
<tr>
<td>varnorm</td>
<td>Test for normally distributed disturbances after var or svar</td>
</tr>
</tbody>
</table>

Forecasting after fitting a VAR or SVAR

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcast compute</td>
<td>Compute dynamic forecasts after var, svar, or vec</td>
</tr>
<tr>
<td>fcast graph</td>
<td>Graph forecasts after fcast compute</td>
</tr>
</tbody>
</table>

Working with IRFs, dynamic-multiplier functions, and FEVDs

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irf</td>
<td>Create and analyze IRFs, dynamic-multiplier functions, and FEVDs</td>
</tr>
</tbody>
</table>

This entry provides an overview of vector autoregressions and structural vector autoregressions. More rigorous treatments can be found in Hamilton (1994), Lütkepohl (2005), and Amisano and Giannini (1997). Stock and Watson (2001) provide an excellent nonmathematical treatment of vector autoregressions and their role in macroeconomics. Becketti (2013) provides an excellent introduction to VAR analysis with an emphasis on how it is done in practice.
Introduction to VARs

A VAR is a model in which $K$ variables are specified as linear functions of $p$ of their own lags, $p$ lags of the other $K - 1$ variables, and possibly additional exogenous variables. Algebraically, a $p$-order VAR model, written $\text{VAR}(p)$, with exogenous variables $x_t$ is given by

$$y_t = v + A_1 y_{t-1} + \cdots + A_p y_{t-p} + B_0 x_t + B_1 x_{t-1} + \cdots + B_s x_{t-s} + u_t \quad t \in \{-\infty, \infty\} \quad (1)$$

where

- $y_t = (y_{1t}, \ldots, y_{Kt})'$ is a $K \times 1$ random vector,
- $A_1$ through $A_p$ are $K \times K$ matrices of parameters,
- $x_t$ is an $M \times 1$ vector of exogenous variables,
- $B_0$ through $B_s$ are $K \times M$ matrices of coefficients,
- $v$ is a $K \times 1$ vector of parameters, and
- $u_t$ is assumed to be white noise; that is,
  - $E(u_t) = 0$,
  - $E(u_t u'_s) = \Sigma$, and
  - $E(u_t u'_s) = 0$ for $t \neq s$.

There are $K^2 \times p + K \times (M(s + 1) + 1)$ parameters in the equation for $y_t$, and there are $\{K \times (K + 1)\}/2$ parameters in the covariance matrix $\Sigma$. One way to reduce the number of parameters is to specify an incomplete VAR, in which some of the $A$ or $B$ matrices are set to zero. Another way is to specify linear constraints on some of the coefficients in the VAR.

A VAR can be viewed as the reduced form of a system of dynamic simultaneous equations. Consider the system

$$W_0 y_t = a + W_1 y_{t-1} + \cdots + W_p y_{t-p} + \overset{\text{hat}}{W}_1 x_t + \overset{\text{hat}}{W}_2 x_{t-2} + \cdots + \overset{\text{hat}}{W}_s x_{t-s} + e_t \quad (2)$$

where $a$ is a $K \times 1$ vector of parameters, each $W_i$, $i = 0, \ldots, p$, is a $K \times K$ matrix of parameters, and $e_t$ is a $K \times 1$ disturbance vector. In the traditional dynamic simultaneous equations approach, sufficient restrictions are placed on the $W_i$ to obtain identification. Assuming that $W_0$ is nonsingular, (2) can be rewritten as

$$y_t = W_0^{-1} a + W_0^{-1} W_1 y_{t-1} + \cdots + W_0^{-1} W_p y_{t-p}$$

$$+ W_0^{-1} \overset{\text{hat}}{W}_1 x_t + W_0^{-1} \overset{\text{hat}}{W}_2 x_{t-2} + \cdots + W_0^{-1} \overset{\text{hat}}{W}_s x_{t-s} + W_0^{-1} e_t \quad (3)$$

which is a VAR with

- $v = W_0^{-1} a$
- $A_i = W_0^{-1} W_i$
- $B_i = W_0^{-1} \overset{\text{hat}}{W}_i$
- $u_t = W_0^{-1} e_t$
The cross-equation error variance–covariance matrix $\Sigma$ contains all the information about contemporaneous correlations in a VAR and may be the VAR’s greatest strength and its greatest weakness. Because no questionable a priori assumptions are imposed, fitting a VAR allows the dataset to speak for itself. However, without imposing some restrictions on the structure of $\Sigma$, we cannot make a causal interpretation of the results.

If we make additional technical assumptions, we can derive another representation of the VAR in (1). If the VAR is stable (see [TS] varstable), we can rewrite $y_t$ as

$$y_t = \mu + \sum_{i=0}^{\infty} D_i x_{t-i} + \sum_{i=0}^{\infty} \Phi_i u_{t-i}$$

(4)

where $\mu$ is the $K \times 1$ time-invariant mean of the process and $D_i$ and $\Phi_i$ are $K \times M$ and $K \times K$ matrices of parameters, respectively. Equation (4) states that the process by which the variables in $y_t$ fluctuate about their time-invariant means, $\mu$, is completely determined by the parameters in $D_i$ and $\Phi_i$ and the (infinite) past history of the exogenous variables $x_t$ and the independent and identically distributed (i.i.d.) shocks or innovations, $u_{t-1}, u_{t-2}, \ldots$. Equation (4) is known as the vector moving-average representation of the VAR. The $D_i$ are the dynamic-multiplier functions, or transfer functions. The moving-average coefficients $\Phi_i$ are also known as the simple IRFs at horizon $i$. The precise relationships between the VAR parameters and the $D_i$ and $\Phi_i$ are derived in Methods and formulas of [TS] irf create.

The joint distribution of $y_t$ is determined by the distributions of $x_t$ and $u_t$ and the parameters $v$, $B_i$, and $A_i$. Estimating the parameters in a VAR requires that the variables in $y_t$ and $x_t$ be covariance stationary, meaning that their first two moments exist and are time invariant. If the $y_t$ are not covariance stationary, but their first differences are, a vector error-correction model (VECM) can be used. See [TS] vec intro and [TS] vec for more information about those models.

If the $u_t$ form a zero mean, i.i.d. vector process, and $y_t$ and $x_t$ are covariance stationary and are not correlated with the $u_t$, consistent and efficient estimates of the $B_i$, the $A_i$, and $v$ are obtained via seemingly unrelated regression, yielding estimators that are asymptotically normally distributed. When the equations for the variables $y_t$ have the same set of regressors, equation-by-equation OLS estimates are the conditional maximum likelihood estimates.

Much of the interest in VAR models is focused on the forecasts, IRFs, dynamic-multiplier functions, and the FEVDs, all of which are functions of the estimated parameters. Estimating these functions is straightforward, but their asymptotic standard errors are usually obtained by assuming that $u_t$ forms a zero mean, i.i.d. Gaussian (normal) vector process. Also, some of the specification tests for VARS have been derived using the likelihood-ratio principle and the stronger Gaussian assumption.

In the absence of contemporaneous exogenous variables, the disturbance variance–covariance matrix contains all the information about contemporaneous correlations among the variables. VARS are sometimes classified into three types by how they account for this contemporaneous correlation. (See Stock and Watson [2001] for one derivation of this taxonomy.) A reduced-form VAR, aside from estimating the variance–covariance matrix of the disturbance, does not try to account for contemporaneous correlations. In a recursive VAR, the $K$ variables are assumed to form a recursive dynamic structural equation model in which the first variable is a function of lagged variables, the second is a function of contemporaneous values of the first variable and lagged values, and so on. In a structural VAR, the theory you are working with places restrictions on the contemporaneous correlations that are not necessarily recursive.

Stata has two commands for fitting reduced-form VARS: var and varbasic. var allows for constraints to be imposed on the coefficients. varbasic allows you to fit a simple VAR quickly without constraints and graph the IRFs.
Because fitting a VAR of the correct order can be important, \texttt{varsoc} offers several methods for choosing the lag order $p$ of the VAR to fit. After fitting a VAR, and before proceeding with inference, interpretation, or forecasting, checking that the VAR fits the data is important. \texttt{varlmar} can be used to check for autocorrelation in the disturbances. \texttt{varwle} performs Wald tests to determine whether certain lags can be excluded. \texttt{varnorm} tests the null hypothesis that the disturbances are normally distributed. \texttt{varstable} checks the eigenvalue condition for stability, which is needed to interpret the IRFs and IRFs.

**Introduction to SVARs**

As discussed in [TS] \texttt{irf create}, a problem with VAR analysis is that, because $\Sigma$ is not restricted to be a diagonal matrix, an increase in an innovation to one variable provides information about the innovations to other variables. This implies that no causal interpretation of the simple IRFs is possible: there is no way to determine whether the shock to the first variable caused the shock in the second variable or vice versa.

However, suppose that we had a matrix $P$ such that $\Sigma = PP'$. We can then show that the variables in $P^{-1}u_t$ have zero mean and that $E\{P^{-1}u_t(P^{-1}u_t)'\} = I_K$. We could rewrite (4) as

$$\begin{align*}
y_t &= \mu + \sum_{s=0}^{\infty} \Phi_s PP^{-1}u_{t-s} \\
     &= \mu + \sum_{s=0}^{\infty} \Theta_s P^{-1}u_{t-s} \\
     &= \mu + \sum_{s=0}^{\infty} \Theta_s w_{t-s}
\end{align*}$$

(5)

where $\Theta_s = \Phi_s P$ and $w_t = P^{-1}u_t$. If we had such a $P$, the $w_k$ would be mutually orthogonal, and the $\Theta_s$ would allow the causal interpretation that we seek.

SVAR models provide a framework for estimation of and inference about a broad class of $P$ matrices. As described in [TS] \texttt{irf create}, the estimated $P$ matrices can then be used to estimate structural IRFs and structural FEVDs. There are two types of SVAR models. Short-run SVAR models identify a $P$ matrix by placing restrictions on the contemporaneous correlations between the variables. Long-run SVAR models, on the other hand, do so by placing restrictions on the long-term accumulated effects of the innovations.

**Short-run SVAR models**

A short-run SVAR model without exogenous variables can be written as

$$A(I_K - A_1 L - A_2 L^2 - \cdots - A_p L^p)y_t = A\epsilon_t = Be_t$$

(6)

where $L$ is the lag operator; $A$, $B$, and $A_1, \ldots, A_p$ are $K \times K$ matrices of parameters; $\epsilon_t$ is a $K \times 1$ vector of innovations with $\epsilon_t \sim N(0, \Sigma)$ and $E[\epsilon_t \epsilon_t'] = 0_K$ for all $s \neq t$; and $e_t$ is a $K \times 1$ vector of orthogonalized disturbances; that is, $e_t \sim N(0, I_K)$ and $E[e_t e_s'] = 0_K$ for all $s \neq t$. These transformations of the innovations allow us to analyze the dynamics of the system in terms of a change to an element of $e_t$. In a short-run SVAR model, we obtain identification by placing restrictions on $A$ and $B$, which are assumed to be nonsingular.
Equation (6) implies that $\mathbf{P}_{sr} = \mathbf{A}^{-1}\mathbf{B}$, where $\mathbf{P}_{sr}$ is the $\mathbf{P}$ matrix identified by a particular short-run SVAR model. The latter equality in (6) implies that

$$\mathbf{A}\mathbf{e}_t\mathbf{e}'_t\mathbf{A}' = \mathbf{B}\mathbf{e}_t\mathbf{e}'_t\mathbf{B}'$$

Taking the expectation of both sides yields

$$\Sigma = \mathbf{P}_{sr}\mathbf{P}'_{sr}$$

Assuming that the underlying VAR is stable (see [TS] varstable for a discussion of stability), we can invert the autoregressive representation of the model in (6) to an infinite-order, moving-average representation of the form

$$\mathbf{y}_t = \mu + \sum_{s=0}^{\infty} \Theta^s_{sr} \mathbf{e}_{t-s}$$  \hspace{1cm} (7)

whereby $\mathbf{y}_t$ is expressed in terms of the mutually orthogonal, unit-variance structural innovations $\mathbf{e}_t$. The $\Theta^s_{sr}$ contain the structural IRFs at horizon $s$.

In a short-run SVAR model, the $\mathbf{A}$ and $\mathbf{B}$ matrices model all the information about contemporaneous correlations. The $\mathbf{B}$ matrix also scales the innovations $\mathbf{u}_t$ to have unit variance. This allows the structural IRFs constructed from (7) to be interpreted as the effect on variable $i$ of a one-time unit increase in the structural innovation to variable $j$ after $s$ periods.

$\mathbf{P}_{sr}$ identifies the structural IRFs by defining a transformation of $\Sigma$, and $\mathbf{P}_{sr}$ is identified by the restrictions placed on the parameters in $\mathbf{A}$ and $\mathbf{B}$. Because there are only $K(K+1)/2$ free parameters in $\Sigma$, only $K(K+1)/2$ parameters may be estimated in an identified $\mathbf{P}_{sr}$. Because there are $2K^2$ total parameters in $\mathbf{A}$ and $\mathbf{B}$, the order condition for identification requires that at least $2K^2 - K(K+1)/2$ restrictions be placed on those parameters. Just as in the simultaneous-equations framework, this order condition is necessary but not sufficient. Amisano and Giannini (1997) derive a method to check that an SVAR model is locally identified near some specified values for $\mathbf{A}$ and $\mathbf{B}$.

Before moving on to models with long-run constraints, consider these limitations. We cannot place constraints on the elements of $\mathbf{A}$ in terms of the elements of $\mathbf{B}$, or vice versa. This limitation is imposed by the form of the check for identification derived by Amisano and Giannini (1997). As noted in Methods and formulas of [TS] var svar, this test requires separate constraint matrices for the parameters in $\mathbf{A}$ and $\mathbf{B}$. Also, we cannot mix short-run and long-run constraints.

### Long-run restrictions

A general short-run SVAR has the form

$$\mathbf{A}(\mathbf{I}_K - \mathbf{A}_1\mathbf{L} - \mathbf{A}_2\mathbf{L}^2 - \cdots - \mathbf{A}_p\mathbf{L}^p)\mathbf{y}_t = \mathbf{Be}_t$$

To simplify the notation, let $\tilde{\mathbf{A}} = (\mathbf{I}_K - \mathbf{A}_1\mathbf{L} - \mathbf{A}_2\mathbf{L}^2 - \cdots - \mathbf{A}_p\mathbf{L}^p)$. The model is assumed to be stable (see [TS] varstable), so $\tilde{\mathbf{A}}^{-1}$, the matrix of estimated long-run effects of the reduced-form VAR shocks, is well defined. Constraining $\mathbf{A}$ to be an identity matrix allows us to rewrite this equation as

$$\mathbf{y}_t = \tilde{\mathbf{A}}^{-1}\mathbf{Be}_t$$

which implies that $\Sigma = \mathbf{B}\mathbf{B}'$. Thus $\mathbf{C} = \tilde{\mathbf{A}}^{-1}\mathbf{B}$ is the matrix of long-run responses to the orthogonalized shocks, and

$$\mathbf{y}_t = \mathbf{Ce}_t$$
In long-run models, the constraints are placed on the elements of $C$, and the free parameters are estimated. These constraints are often exclusion restrictions. For instance, constraining $C[1,2]$ to be zero can be interpreted as setting the long-run response of variable 1 to the structural shocks driving variable 2 to be zero.

Stata’s \texttt{svar} command estimates the parameters of structural VARs. See \texttt{[TS] var svar} for more information and examples.

\textbf{IRFs and FEVDs}

IRFs describe how the $K$ endogenous variables react over time to a one-time shock to one of the $K$ disturbances. Because the disturbances may be contemporaneously correlated, these functions do not explain how variable $i$ reacts to a one-time increase in the innovation to variable $j$ after $s$ periods, holding everything else constant. To explain this, we must start with orthogonalized innovations so that the assumption to hold everything else constant is reasonable. Recursive VARs use a Cholesky decomposition to orthogonalize the disturbances and thereby obtain structurally interpretable IRFs. Structural VARs use theory to impose sufficient restrictions, which need not be recursive, to decompose the contemporaneous correlations into orthogonal components.

FEVDs are another tool for interpreting how the orthogonalized innovations affect the $K$ variables over time. The FEVD from $j$ to $i$ gives the fraction of the $s$-step forecast-error variance of variable $i$ that can be attributed to the $j$th orthogonalized innovation.

Dynamic-multiplier functions describe how the endogenous variables react over time to a unit change in an exogenous variable. This is a different experiment from that in IRFs and FEVDs because dynamic-multiplier functions consider a change in an exogenous variable instead of a shock to an endogenous variable.

\texttt{irf create} estimates IRFs, Cholesky orthogonalized IRFs, dynamic-multiplier functions, and structural IRFs and their standard errors. It also estimates Cholesky and structural FEVDs. The \texttt{irf graph}, \texttt{irf cgraph}, \texttt{irf ograph}, \texttt{irf table}, and \texttt{irf ctable} commands graph and tabulate these estimates. Stata also has several other commands to manage IRF and FEVD results. See \texttt{[TS] irf} for a description of these commands.

\texttt{fcast compute} computes dynamic forecasts and their standard errors from VARs. \texttt{fcast graph} graphs the forecasts that are generated using \texttt{fcast compute}.

VARs allow researchers to investigate whether one variable is useful in predicting another variable. A variable $x$ is said to Granger-cause a variable $y$ if, given the past values of $y$, past values of $x$ are useful for predicting $y$. The Stata command \texttt{vargranger} performs Wald tests to investigate Granger causality between the variables in a VAR.

\textbf{References}


Becketti, S. 2013. \textit{Introduction to Time Series Using Stata}. College Station, TX: Stata Press.


Also see

[TS] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] var — Vector autoregressive models
[TS] var svar — Structural vector autoregressive models
[TS] vec — Vector error-correction models
[TS] vec intro — Introduction to vector error-correction models
var — Vector autoregressive models

Description

var fits a multivariate time-series regression of each dependent variable on lags of itself and on lags of all the other dependent variables. var also fits a variant of vector autoregressive (VAR) models known as the VARX model, which also includes exogenous variables. See [TS] var intro for a list of commands that are used in conjunction with var.

Quick start

Vector autoregressive model for dependent variables $y_1$, $y_2$, and $y_3$ and their first and second lags using \texttt{tsset} data

\begin{verbatim}
  var y1 y2 y3
\end{verbatim}

As above, but include second and third lags instead of first and second

\begin{verbatim}
  var y1 y2 y3, lags(2 3)
\end{verbatim}

Add exogenous variables $x_1$ and $x_2$

\begin{verbatim}
  var y1 y2 y3, lags(2 3) exog(x1 x2)
\end{verbatim}

As above, but make a small-sample degrees-of-freedom adjustment

\begin{verbatim}
  var y1 y2 y3, lags(2 3) exog(x1 x2) dfk
\end{verbatim}

Menu

Statistics $>$ Multivariate time series $>$ Vector autoregression (VAR)
**Syntax**

```
var depvarlist [if] [in] [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noconstant</td>
<td>suppress constant term</td>
</tr>
<tr>
<td>lags(numlist)</td>
<td>use lags numlist in the VAR</td>
</tr>
<tr>
<td>exog(varlist)</td>
<td>use exogenous variables varlist</td>
</tr>
<tr>
<td>constraints(numlist)</td>
<td>apply specified linear constraints</td>
</tr>
<tr>
<td>[no]log</td>
<td>display or suppress SURE iteration log; default is to display</td>
</tr>
<tr>
<td>iterate(#)</td>
<td>set maximum number of iterations for SURE; default is iterate(1600)</td>
</tr>
<tr>
<td>tolerance(#)</td>
<td>set convergence tolerance of SURE</td>
</tr>
<tr>
<td>noisure</td>
<td>use one-step SURE</td>
</tr>
<tr>
<td>dfk</td>
<td>make small-sample degrees-of-freedom adjustment</td>
</tr>
<tr>
<td>small</td>
<td>report small-sample t and F statistics</td>
</tr>
<tr>
<td>nobigf</td>
<td>do not compute parameter vector for coefficients implicitly set to zero</td>
</tr>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>lutstats</td>
<td>report Lütkepohl lag-order selection statistics</td>
</tr>
<tr>
<td>nocnsreport</td>
<td>do not display constraints</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats, row spacing, and line width</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `var`; see [TS] `tsset`.
`depvarlist` and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.
by, `fp`, `rolling`, `statsby`, and `xi` are allowed; see [U] 11.1.10 Prefix commands.
`coeflegend` does not appear in the dialog box.
See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

**Options**

- **Model**

  - `noconstant`; see [R] Estimation options.

  - `lags(numlist)` specifies the lags to be included in the model. The default is `lags(1 2)`. This option takes a `numlist` and not simply an integer for the maximum lag. For example, `lags(2)` would include only the second lag in the model, whereas `lags(1/2)` would include both the first and second lags in the model. See [U] 11.1.8 `numlist` and [U] 11.4.4 Time-series varlists for more discussion of numlists and lags.

  - `exog(varlist)` specifies a list of exogenous variables to be included in the VAR.
Model 2

constraints(numlist); see [R] Estimation options.

log and nolog specify whether to display the log from the iterated seemingly unrelated regression algorithm. By default, the iteration log is displayed when the coefficients are estimated through iterated seemingly unrelated regression. When the constraints() option is not specified, the estimates are obtained via OLS, and nolog has no effect. For this reason, log and nolog can be specified only when constraints() is specified. Similarly, nolog cannot be combined with noisure. You can also control the iteration log using set iterlog; see [R] set iter.

iterate(#) specifies an integer that sets the maximum number of iterations when the estimates are obtained through iterated seemingly unrelated regression. By default, the limit is 1,600. When constraints() is not specified, the estimates are obtained using OLS, and iterate() has no effect. For this reason, iterate() can be specified only when constraints() is specified. Similarly, iterate() cannot be combined with noisure.

tolerance(#) specifies a number greater than zero and less than 1 for the convergence tolerance of the iterated seemingly unrelated regression algorithm. By default, the tolerance is 1e-6. When constraints() is not specified, the estimates are obtained using OLS, and tolerance() has no effect. For this reason, tolerance() can be specified only when constraints() is specified. Similarly, tolerance() cannot be combined with noisure.

noisure specifies that the estimates in the presence of constraints be obtained through one-step seemingly unrelated regression. By default, var obtains estimates in the presence of constraints through iterated seemingly unrelated regression. When constraints() is not specified, the estimates are obtained using OLS, and noisure has no effect. For this reason, noisure can be specified only when constraints() is specified. Similarly, noisure cannot be combined with noisure.

dfk specifies that a small-sample degrees-of-freedom adjustment be used when estimating Σ, the error variance–covariance matrix. Specifically, \( 1/(T - \bar{m}) \) is used instead of the large-sample divisor \( 1/T \), where \( \bar{m} \) is the average number of parameters in the functional form for \( y_t \) over the \( K \) equations.

small causes var to report small-sample \( t \) and \( F \) statistics instead of the large-sample normal and chi-squared statistics.

nobigf requests that var not save the estimated parameter vector that incorporates coefficients that have been implicitly constrained to be zero, such as when some lags have been omitted from a model. e(bf) is used for computing asymptotic standard errors in the postestimation commands irf create and fcast compute; see [TS] irf create and [TS] fcast compute. Therefore, specifying nobigf implies that the asymptotic standard errors will not be available from irf create and fcast compute. See Fitting models with some lags excluded.

level(#); see [R] Estimation options.

lutstats specifies that the Lütkepohl (2005) versions of the lag-order selection statistics be reported. See Methods and formulas in [TS] varsoc for a discussion of these statistics.

nocnsreport; see [R] Estimation options.

display_options: noci, nopvalues, vsquish, cformat(%,fmt), pformat(%,fmt), sformat(%,fmt), and nolstretch; see [R] Estimation options.

The following option is available with var but is not shown in the dialog box:

coefflegend; see [R] Estimation options.
Remarks and examples

Remarks are presented under the following headings:

- Introduction
- Fitting models with some lags excluded
- Fitting models with exogenous variables
- Fitting models with constraints on the coefficients

Introduction

A VAR is a model in which $K$ variables are specified as linear functions of $p$ of their own lags, $p$ lags of the other $K - 1$ variables, and possibly exogenous variables. A VAR with $p$ lags is usually denoted a VAR($p$). For more information, see [TS] var intro.

Example 1: VAR model

To illustrate the basic usage of var, we replicate the example in Lütkepohl (2005, 77–78). The data consists of three variables: the first difference of the natural log of investment, dln inv; the first difference of the natural log of income, dln inc; and the first difference of the natural log of consumption, dln consump. The dataset contains data through the fourth quarter of 1982, though Lütkepohl uses only the observations through the fourth quarter of 1978.

```
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. tsset
time variable: qtr, 1960q1 to 1982q4
delta: 1 quarter
```
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lutstats dfk

Vector autoregression

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>RMSE</th>
<th>R-sq</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>7</td>
<td>0.046148</td>
<td>0.1286</td>
<td>9.736909</td>
<td>0.1362</td>
</tr>
<tr>
<td>dln_inc</td>
<td>7</td>
<td>0.011719</td>
<td>0.1142</td>
<td>8.508289</td>
<td>0.2032</td>
</tr>
<tr>
<td>dln_consump</td>
<td>7</td>
<td>0.009445</td>
<td>0.2513</td>
<td>22.15096</td>
<td>0.0011</td>
</tr>
</tbody>
</table>

| Coef. | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|-------|-----------|------|-----|-------------------------|
| dln_inv     |           |      |     |                         |
| L1.  | -.3196318 | .1254564 | -2.55| 0.011 | -.5655218 | -.0737419 |
| L2.  | -.1608508 | .1249066 | -1.29| 0.199 | -.4053633 | .0842616  |
| dln_inc     |           |      |     |                         |
| L1.  | .1459851  | .5456664 | 0.27 | 0.789 | -.9235013 | 1.215472 |
| L2.  | .1146009  | .5345709 | 0.21 | 0.830 | -.9331388 | 1.162341 |
| dln_consump |           |      |     |                         |
| L1.  | .9612288  | .6643086 | 1.45 | 0.148 | -.3407922 | 2.26325  |
| L2.  | .9344001  | .6650949 | 1.40 | 0.160 | -.369162  | 2.237962 |
| _cons | -.0167221 | .0172264 | -0.97| 0.332 | -.0504852 | .0170409 |
| dln_inc     |           |      |     |                         |
| L1.  | .0439309  | .0318592 | 1.38 | 0.168 | -.018512  | .1063739 |
| L2.  | .0500302  | .0317196 | 1.58 | 0.115 | -.0121391 | .1121995 |
| dln_consump |           |      |     |                         |
| L1.  | .1527311  | .1385702 | -1.10| 0.270 | -.4243237 | .1188615 |
| L2.  | .0191634  | .1357525 | 0.14 | 0.888 | -.2469067 | .2852334 |
| _cons | .0157672  | .0043746 | 3.60 | 0.000 | .0071932  | .0243412 |
| dln_consump |           |      |     |                         |
| L1.  | -.002423 | .0256763 | -0.09| 0.925 | -.0527476 | .0479016 |
| L2.  | .0338806  | .0255638 | 1.33 | 0.185 | -.0162235 | .0839847 |
| dln_inc     |           |      |     |                         |
| L1.  | .2248134  | .1116778 | 2.01 | 0.044 | .005929   | .4436978 |
| L2.  | .3549135  | .1094069 | 3.24 | 0.001 | .1404798  | .5693471 |
| dln_consump |           |      |     |                         |
| L1.  | -.2639695 | .1359595 | -1.94| 0.052 | -.5304451 | .0025062 |
| L2.  | -.0222264 | .1361204 | -0.16| 0.870 | -.2890175 | .2445646 |
| _cons | .0129258  | .0035256 | 3.67 | 0.000 | .0060157  | .0198358 |
The output has two parts: a header and the standard Stata output table for the coefficients, standard errors, and confidence intervals. The header contains summary statistics for each equation in the VAR and statistics used in selecting the lag order of the VAR. Although there are standard formulas for all the lag-order statistics, Lütkepohl (2005) gives different versions of the three information criteria that drop the constant term from the likelihood. To obtain the Lütkepohl (2005) versions, we specified the \texttt{lutstats} option. The formulas for the standard and Lütkepohl versions of these statistics are given in \textit{Methods and formulas} of \texttt{[TS] varsoc}.

The \texttt{dfk} option specifies that the small-sample divisor $1/(T - \bar{m})$ be used in estimating $\Sigma$ instead of the maximum likelihood (ML) divisor $1/T$, where $\bar{m}$ is the average number of parameters included in each of the $K$ equations. All the lag-order statistics are computed using the ML estimator of $\Sigma$. Thus, specifying \texttt{dfk} will not change the computed lag-order statistics, but it will change the estimated variance–covariance matrix. Also, when \texttt{dfk} is specified, a \texttt{dfk}-adjusted log likelihood is computed and stored in $\texttt{e(ll\_dfk)}$.

The \texttt{lag()} option takes a \textit{numlist} of lags. To specify a model that includes the first and second lags, type

\verbatim
. var y1 y2 y3, lags(1/2)
\endverbatim

not

\verbatim
. var y1 y2 y3, lags(2)
\endverbatim

because the latter specification would fit a model that included only the second lag.

\section*{Fitting models with some lags excluded}

To fit a model that has only a fourth lag, that is,

$$y_t = \nu + A_4 y_{t-4} + u_t$$

you would specify the \texttt{lags(4)} option. Doing so is equivalent to fitting the more general model

$$y_t = \nu + A_1 y_{t-1} + A_2 y_{t-2} + A_3 y_{t-3} + A_4 y_{t-4} + u_t$$

with $A_1$, $A_2$, and $A_3$ constrained to be 0. When you fit a model with some lags excluded, \texttt{var} estimates the coefficients included in the specification ($A_4$ here) and stores these estimates in $\texttt{e(b)}$. To obtain the asymptotic standard errors for impulse–response functions and other postestimation statistics, Stata needs the complete set of parameter estimates, including those that are constrained to be zero; \texttt{var} stores them in $\texttt{e(bf)}$. Because you can specify models for which the full set of parameter estimates exceeds Stata’s limit on the size of matrices, the \texttt{nobigf} option specifies that \texttt{var} not compute and store $\texttt{e(bf)}$. This means that the asymptotic standard errors of the postestimation functions cannot be obtained, although bootstrap standard errors are still available. Building $\texttt{e(bf)}$ can be time consuming, so if you do not need this full matrix, and speed is an issue, use \texttt{nobigf}.
Example 2: VAR model with exogenous variables

We use the `exog()` option to include exogenous variables in a VAR.

```
. var dln_inc dln_consump if qtr<=tq(1978q4), dfk exog(dln_inv)
```

```
Vector autoregression
Sample: 1960q4 - 1978q4  Number of obs = 73
Log likelihood = 478.5663  AIC = -12.78264
FPE = 9.64e-09  HQIC = -12.63259
Det(Sigma_ml) = 6.93e-09  SBIC = -12.40612
```

```
Equation Parms RMSE R-sq chi2 P>chi2
```

```
  dln_inc   6   .011917  0.0702  5.059587  0.4087
  dln_consump   6   .009197  0.2794  25.97262  0.0001
```

```
  Coef. Std. Err.    z  P>|z|      [95% Conf. Interval]
  dln_inc
     dln_inc
       L1.  -.1343345  .1391074 -0.97 0.334  -.4069801 -.1383111
       L2.   .0120331  .1380346  0.09 0.931  -.2585097 .2825759
  dln_consump
       L1.   .3235342  .1652769  1.96 0.050   -.0004027 .647471
       L2.   .0754177  .1648624  0.46 0.647  -.2477066 .398542
  dln_inv
     _cons   .0151546  .0302319  0.50 0.616  -.0440987 .074408
     _cons   .0145136  .0043815  3.31 0.001    .0059259 .0231012
```

All the postestimation commands for analyzing VARs work when exogenous variables are included in a model, but the asymptotic standard errors for the $h$-step-ahead forecasts are not available.

Fitting models with constraints on the coefficients

`var` permits model specifications that include constraints on the coefficient, though `var` does not allow for constraints on $\Sigma$. See `[TS] var intro` and `[TS] var svar` for ways to constrain $\Sigma$. 
Example 3: VAR model with constraints

In the first example, we fit a full VAR(2) to a three-equation model. The coefficients in the equation for `dln_inv` were jointly insignificant, as were the coefficients in the equation for `dln_inc`; and many individual coefficients were not significantly different from zero. In this example, we constrain the coefficient on `L2.dln_inc` in the equation for `dln_inv` and the coefficient on `L2.dln_consump` in the equation for `dln_inc` to be zero.

```
constraint 1 [dln_inv]L2.dln_inc = 0
constraint 2 [dln_inc]L2.dln_consump = 0
var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lutstats dfk
> constraints(1 2)
```

Estimating VAR coefficients

```
Iteration 1:  tolerance = .00737681
Iteration 2:  tolerance = 3.998e-06
Iteration 3:  tolerance = 2.730e-09
```

Vector autoregression

```
Sample:  1960q4 - 1978q4 Number of obs   =   73
Log likelihood =  606.2804 (lutstats) AIC  =  -31.69254
FPE =  1.77e-14   HQIC  =  -31.46747
Det(Sigma_ml) =  1.05e-14   SBIC  =  -31.12777
```

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>RMSE</th>
<th>R-sq</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>6</td>
<td>.043985</td>
<td>0.1280</td>
<td>9.842338</td>
<td>0.0798</td>
</tr>
<tr>
<td>dln_inc</td>
<td>6</td>
<td>.011143</td>
<td>0.1141</td>
<td>8.584446</td>
<td>0.1268</td>
</tr>
<tr>
<td>dln_consump</td>
<td>7</td>
<td>.008981</td>
<td>0.2512</td>
<td>22.86958</td>
<td>0.0008</td>
</tr>
</tbody>
</table>

( 1) [dln_inv]L2.dln_inc = 0
( 2) [dln_inc]L2.dln_consump = 0
|                | Coef.  | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|----------------|--------|-----------|-------|------|----------------------|
| **dln_inv**    |        |           |       |      |                      |
| dln_inv        |        |           |       |      |                      |
| L1.            | -.320713 | .1247512  | -2.57 | 0.010 | -.5652208 -.0762051 |
| L2.            | -.1607084 | .124261  | -1.29 | 0.196 | -.4042555 .0828386  |
| dln_inc        |        |           |       |      |                      |
| L1.            | .1195448 | .5295669  | 0.23  | 0.821 | -.9183873 1.157477   |
| L2.            | 5.66e-19 | 9.33e-18  | 0.06  | 0.952 | -1.77e-17 1.89e-17   |
| dln_consump    |        |           |       |      |                      |
| L1.            | 1.009281 | .623501   | 1.62  | 0.106 | -.2127586 2.231321   |
| L2.            | 1.008079 | .5713486  | 1.76  | 0.078 | -.1117438 2.127902   |
| _cons          | -.0162102 | .016893  | -0.96 | 0.337 | -.0493199 0.0168995  |
| **dln_inc**    |        |           |       |      |                      |
| dln_inv        |        |           |       |      |                      |
| L1.            | .0435712 | .0309078  | 1.41  | 0.159 | -.017007 .1041495   |
| L2.            | .0496788 | .0306455  | 1.62  | 0.105 | -.0103852 .1097428  |
| dln_inc        |        |           |       |      |                      |
| L1.            | -.1555119 | .1315854 | -1.18 | 0.237 | -.4134146 .1023908  |
| L2.            | .0122353 | .1165811  | 0.10  | 0.916 | -.2162595 .2407301  |
| dln_consump    |        |           |       |      |                      |
| L1.            | .29286 | .1568345  | 1.87  | 0.062 | -.01453  .6002501   |
| L2.            | -1.53e-18 | 1.89e-17  | -0.08 | 0.935 | -3.85e-17 3.55e-17   |
| _cons          | .015689 | .003819   | 4.11  | 0.000 | .0082039 0.0231741  |
| **dln_consump**|        |           |       |      |                      |
| dln_inv        |        |           |       |      |                      |
| L1.            | -.0026229 | .0253538 | -0.10 | 0.918 | -.0523154  .0470696|
| L2.            | .0337245 | .0252113  | 1.34  | 0.181 | -.0156888  .0831378 |
| dln_inc        |        |           |       |      |                      |
| L1.            | .2224798 | .1094349  | 2.03  | 0.042 | .0079912  .4369683  |
| L2.            | .3469758 | .1006026  | 3.45  | 0.001 | .1497984  .5441532  |
| dln_consump    |        |           |       |      |                      |
| L1.            | -.2600227 | .1321622 | -1.97 | 0.049 | -.519056  -.0009895 |
| L2.            | -.0146825 | .1117618 | -0.13 | 0.895 | -.2337315  .2043666 |
| _cons          | .0129149 | .003376   | 3.83  | 0.000 | .0062981 0.0195317  |

None of the free parameter estimates changed by much. Whereas the coefficients in the equation dln_inv are now significant at the 10% level, the coefficients in the equation for dln_inc remain jointly insignificant.
Stored results

`var` stores the following in `e()`:

Scalars:
- `e(N)` number of observations
- `e(N_gaps)` number of gaps in sample
- `e(k)` number of parameters
- `e(k_eq)` number of equations in `e(b)`
- `e(k_dv)` number of dependent variables
- `e(df_eq)` average number of parameters in an equation
- `e(df_m)` model degrees of freedom
- `e(df_r)` residual degrees of freedom (small only)
- `e(ll)` log likelihood
- `e(ll_dfk)` dfk adjusted log likelihood (dfk only)
- `e(obs_#)` number of observations on equation #
- `e(k_#)` number of parameters in equation #
- `e(df_m#)` model degrees of freedom for equation #
- `e(df_r#)` residual degrees of freedom for equation # (small only)
- `e(r2_#)` $R^2$ for equation #
- `e(ll_#)` log likelihood for equation #
- `e(chi2_#)` $\chi^2$ for equation #
- `e(F_#)` $F$ statistic for equation # (small only)
- `e(rmse_#)` root mean squared error for equation #
- `e(aic)` Akaike information criterion
- `e(hqic)` Hannan–Quinn information criterion
- `e(sbic)` Schwarz–Bayesian information criterion
- `e(fpe)` final prediction error
- `e(mlag)` highest lag in VAR
- `e(tmin)` first time period in sample
- `e(tmax)` maximum time
- `e(detsig)` determinant of `e(Sigma)`
- `e(detsig_ml)` determinant of $\hat{\Sigma}_{ml}$
- `e(rank)` rank of `e(V)`
Macros

- `e(cmd)` var
- `e(cmdline)` command as typed
- `e(depvar)` names of dependent variables
- `e(endog)` names of endogenous variables, if specified
- `e(exog)` names of exogenous variables, and their lags, if specified
- `e(exogvars)` names of exogenous variables, if specified
- `e(eqnames)` names of equations
- `e(lags)` lags in model
- `e(exlags)` lags of exogenous variables in model, if specified
- `e(title)` title in estimation output
- `e(eqnames)` eqnames, if noconstant is specified
- `e(constraints)` constraints, if constraints() is specified
- `e(cnslist_var)` list of specified constraints
- `e(small)` small, if specified
- `e(lutstats)` lutstats, if specified
- `e(timevar)` time variable specified in tsset
- `e(tsfmt)` format for the current time variable
- `e(dfk)` dfk, if specified
- `e(properties)` b V
- `e(predict)` program used to implement predict
- `e(marginsok)` predictions allowed by margins
- `e(marginsnotok)` predictions disallowed by margins
- `e(marginsdefault)` default predict() specification for margins

Matrices

- `e(b)` coefficient vector
- `e(Cns)` constraints matrix
- `e(Sigma)` $\Sigma$ matrix
- `e(V)` variance–covariance matrix of the estimators
- `e(bf)` constrained coefficient vector
- `e(exlagsm)` matrix mapping lags to exogenous variables
- `e(G)` Gamma matrix; see Methods and formulas

Functions

- `e(sample)` marks estimation sample

Methods and formulas

When there are no constraints placed on the coefficients, the $\text{VAR}(p)$ is a seemingly unrelated regression model with the same explanatory variables in each equation. As discussed in Lütkepohl (2005) and Greene (2008, 696), performing linear regression on each equation produces the maximum likelihood estimates of the coefficients. The estimated coefficients can then be used to calculate the residuals, which in turn are used to estimate the cross-equation error variance–covariance matrix $\Sigma$.

Per Lütkepohl (2005), we write the $\text{VAR}(p)$ with exogenous variables as

$$y_t = AY_{t-1} + B_0x_t + u_t$$

where

$y_t$ is the $K \times 1$ vector of endogenous variables,

$A$ is a $K \times Kp$ matrix of coefficients,

$B_0$ is a $K \times M$ matrix of coefficients,
\[ x_t \] is the \( M \times 1 \) vector of exogenous variables, \\
\[ u_t \] is the \( K \times 1 \) vector of white noise innovations, and \\
\[ Y_t \] is the \( Kp \times 1 \) matrix given by \\
\[ Y_t = \begin{pmatrix} Y_t \\ \vdots \\ Y_{t-p+1} \end{pmatrix} \]

Although (5) is easier to read, the formulas are much easier to manipulate if it is instead written as

\[ Y = BZ + U \]

where

\[ Y = (y_1, \ldots, y_T) \] \( \text{Y is } K \times T \) \\
\[ B = (A, B_0) \] \( \text{B is } K \times (Kp + M) \) \\
\[ Z = \begin{pmatrix} Y_0 \ldots, Y_{T-1} \\ x_1 \ldots, x_T \end{pmatrix} \] \( \text{Z is } (Kp + M) \times T \) \\
\[ U = (u_1, \ldots, u_T) \] \( \text{U is } K \times T \)

Intercept terms in the model are included in \( x_t \). If there are no exogenous variables and no intercept terms in the model, \( x_t \) is empty.

The coefficients are estimated by iterated seemingly unrelated regression. Because the estimation is actually performed by \texttt{reg3}, the methods are documented in [R] \texttt{reg3}. See [P] \texttt{makecns} for more on estimation with constraints.

Let \( \hat{U} \) be the matrix of residuals that are obtained via \( Y - \hat{B}Z \), where \( \hat{B} \) is the matrix of estimated coefficients. Then the estimator of \( \Sigma \) is 

\[ \hat{\Sigma} = \frac{1}{\tilde{T}} \hat{U}' \hat{U} \]

By default, the maximum likelihood divisor of \( \tilde{T} = T \) is used. When \texttt{dfk} is specified, a small-sample degrees-of-freedom adjustment is used; then, \( \tilde{T} = T - m \) where \( m \) is the average number of parameters per equation in the functional form for \( y_t \) over the \( K \) equations.

\texttt{small} specifies that Wald tests after \texttt{var} be assumed to have \( F \) or \( t \) distributions instead of chi-squared or standard normal distributions. The standard errors from each equation are computed using the degrees of freedom for the equation.

The “gamma” matrix stored in \texttt{e(G)} referred to in \textit{Stored results} is the \( (Kp + 1) \times (Kp + 1) \) matrix given by

\[
\frac{1}{\tilde{T}} \sum_{t=1}^{T} (1, \ Y'_t)(1, \ Y'_t)'
\]

The formulas for the lag-order selection criteria and the log likelihood are discussed in [TS] \texttt{varsoc}.
Acknowledgment

We thank Christopher F. Baum of the Department of Economics at Boston College and author of the Stata Press books *An Introduction to Modern Econometrics Using Stata* and *An Introduction to Stata Programming* for his helpful comments.

References


Also see

- [TS] var postestimation — Postestimation tools for var
- [TS] dfactor — Dynamic-factor models
- [TS] forecast — Econometric model forecasting
- [TS] mgarch — Multivariate GARCH models
- [TS] sspace — State-space models
- [TS] tsset — Declare data to be time-series data
- [TS] var intro — Introduction to vector autoregressive models
- [TS] var svar — Structural vector autoregressive models
- [TS] varbasic — Fit a simple VAR and graph IRFs or FEVDs
- [TS] vec — Vector error-correction models
- [U] 20 Estimation and postestimation commands

*Stata Dynamic Stochastic General Equilibrium Models Reference Manual*
Title

**var postestimation — Postestimation tools for var**

Postestimation commands predict margins Remarks and examples
Methods and formulas Also see

**Postestimation commands**

The following postestimation commands are of special interest after `var`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fcast compute</code></td>
<td>obtain dynamic forecasts</td>
</tr>
<tr>
<td><code>fcast graph</code></td>
<td>graph dynamic forecasts obtained from <code>fcast compute</code></td>
</tr>
<tr>
<td><code>irf</code></td>
<td>create and analyze IRFs and FEVDs</td>
</tr>
<tr>
<td><code>vargranger</code></td>
<td>Granger causality tests</td>
</tr>
<tr>
<td><code>varlmar</code></td>
<td>LM test for autocorrelation in residuals</td>
</tr>
<tr>
<td><code>varnorm</code></td>
<td>test for normally distributed residuals</td>
</tr>
<tr>
<td><code>varsoc</code></td>
<td>lag-order selection criteria</td>
</tr>
<tr>
<td><code>varstable</code></td>
<td>check stability condition of estimates</td>
</tr>
<tr>
<td><code>varwle</code></td>
<td>Wald lag-exclusion statistics</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>

774
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

```
predict [type] newvar [if] [in] [ , statistic equation(eqno|eqname) ]
```

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>xb, the default, calculates the linear prediction for the specified equation.</td>
</tr>
<tr>
<td></td>
<td>stdp calculates the standard error of the linear prediction for the specified equation.</td>
</tr>
<tr>
<td></td>
<td>residuals calculates the residuals.</td>
</tr>
<tr>
<td></td>
<td>equation(eqno</td>
</tr>
<tr>
<td></td>
<td>equation() is filled in with one eqno or eqname for options xb, stdp, and residuals. For example, equation(#1) would mean that the calculation is to be made for the first equation, equation(#2) would mean the second, and so on. You could also refer to the equation by its name; thus, equation(income) would refer to the equation named income and equation(hours), to the equation named hours.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify equation(), the results are the same as if you specified equation(#1). For more information on using predict after multiple-equation estimation commands, see [R] predict.</td>
</tr>
</tbody>
</table>
margins

Description for margins

margins estimates margins of response for linear predictions.

Menu for margins

Statistics > Postestimation

Syntax for margins

margins [marginlist] [, options]
margins [marginlist], predict(statistic ...) [predict(statistic ...) ...] [options]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td>stdp</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

xb defaults to the first equation.

Statistics not allowed with margins are functions of stochastic quantities other than e(b).

For the full syntax, see [R] margins.

Remarks and examples

Remarks are presented under the following headings:

Model selection and inference
Forecasting

Model selection and inference

See the following sections for information on model selection and inference after var.

[TS] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] vargranger — Pairwise Granger causality tests after var or svar
[TS] varlmar — LM test for residual autocorrelation after var or svar
[TS] varnorm — Test for normally distributed disturbances after var or svar
[TS] varsoc — Obtain lag-order selection statistics for VARs and VECMs
[TS] varstable — Check the stability condition of VAR or SVAR estimates
[TS] varwle — Obtain Wald lag-exclusion statistics after var or svar
Two types of forecasts are available after you fit a VAR($p$): a one-step-ahead forecast and a dynamic $h$-step-ahead forecast.

The one-step-ahead forecast produces a prediction of the value of an endogenous variable in the current period by using the estimated coefficients, the past values of the endogenous variables, and any exogenous variables. If you include contemporaneous values of exogenous variables in your model, you must have observations on the exogenous variables that are contemporaneous with the period in which the prediction is being made to compute the prediction. In Stata terms, these one-step-ahead predictions are just the standard linear predictions available after any estimation command. Thus predict, xb eq(eqno | eqname) produces one-step-ahead forecasts for the specified equation. predict, stdp eq(eqno | eqname) produces the standard error of the linear prediction for the specified equation. The standard error of the forecast includes an estimate of the variability due to innovations, whereas the standard error of the linear prediction does not.

The dynamic $h$-step-ahead forecast begins by using the estimated coefficients, the lagged values of the endogenous variables, and any exogenous variables to predict one step ahead for each endogenous variable. Then the one-step-ahead forecast produces two-step-ahead forecasts for each endogenous variable. The process continues for $h$ periods. Because each step uses the predictions of the previous steps, these forecasts are known as dynamic forecasts. See the following sections for information on obtaining forecasts after svar:

- [TS] fcast compute — Compute dynamic forecasts after var, svar, or vec
- [TS] fcast graph — Graph forecasts after fcast compute

Methods and formulas

Formulas for predict

predict with the xb option provides the one-step-ahead forecast. If exogenous variables are specified, the forecast is conditional on the exogenous $x_t$ variables. Specifying the residuals option causes predict to calculate the errors of the one-step-ahead forecasts. Specifying the stdp option causes predict to calculate the standard errors of the one-step-ahead forecasts.

Also see

- [TS] var — Vector autoregressive models
- [U] 20 Estimation and postestimation commands
Description

`svar` fits a vector autoregressive model subject to short- or long-run constraints you place on the resulting impulse–response functions (IRFs). Economic theory typically motivates the constraints, allowing a causal interpretation of the IRFs to be made. See `[TS] var intro` for a list of commands that are used in conjunction with `svar`.

Quick start

Structural VAR for `y1`, `y2`, and `y3` using `tsset` data with short-run constraints on impulse responses given by predefined matrices `A` and `B`

```
svar y1 y2 y3, aeq(A) beq(B)
```

Structural VAR for `y1`, `y2`, and `y3` with long-run constraint on impulse responses given by the predefined matrix `C`

```
svar y1 y2 y3, lreq(C)
```

Add exogenous variables `x1` and `x2`

```
svar y1 y2 y3, lreq(C) exog(x1 x2)
```

As above, but include third and fourth lags of the dependent variables instead of first and second

```
svar y1 y2 y3, lreq(C) exog(x1 x2) lags(3 4)
```

Menu

Statistics > Multivariate time series > Structural vector autoregression (SVAR)

Syntax

**Short-run constraints**

```
svar depvarlist [if] [in], { aconstrained(constraints_a) aeq(matrix_aeq) acns(matrix_acns) bconstrained(constraints_b) beq(matrix_beq) bcns(matrix_bcns) } [short_run_options]
```

**Long-run constraints**

```
svar depvarlist [if] [in], { lrcns(matrix_lrcns) lreq(matrix_lreq) lreq(constraints_lr) lreq(matrix_lreq) lrcns(matrix_lrcns) } [long_run_options]
```
### short-run_options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noconstant</td>
<td>suppress constant term</td>
</tr>
<tr>
<td><em>aconstraints(constraints_a)</em></td>
<td>apply previously defined <code>constraints_a</code> to <code>A</code></td>
</tr>
<tr>
<td><em>aeq(matrix_aeq)</em></td>
<td>define and apply to <code>A</code> equality constraint matrix <code>matrix_aeq</code></td>
</tr>
<tr>
<td><em>acns(matrix_acns)</em></td>
<td>define and apply to <code>A</code> cross-parameter constraint matrix <code>matrix_acns</code></td>
</tr>
<tr>
<td><em>bconstraints(constraints_b)</em></td>
<td>apply previously defined <code>constraints_b</code> to <code>B</code></td>
</tr>
<tr>
<td><em>beq(matrix_beq)</em></td>
<td>define and apply to <code>B</code> equality constraint matrix <code>matrix_beq</code></td>
</tr>
<tr>
<td><em>bcns(matrix_bcns)</em></td>
<td>define and apply to <code>B</code> cross-parameter constraint <code>matrix_bcns</code></td>
</tr>
<tr>
<td>lags(numlist)</td>
<td>use lags <code>numlist</code> in the underlying VAR</td>
</tr>
</tbody>
</table>

**Model 2**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exog(varlist_exog)</td>
<td>use exogenous variables <code>varlist</code></td>
</tr>
<tr>
<td>varconstraints(constraints_v)</td>
<td>apply <code>constraints_v</code> to underlying VAR</td>
</tr>
<tr>
<td>noislog</td>
<td>suppress SURE iteration log</td>
</tr>
<tr>
<td>isiterate(#)</td>
<td>set maximum number of iterations for SURE; default is <code>isiterate(1600)</code></td>
</tr>
<tr>
<td>istolerance(#)</td>
<td>set convergence tolerance of SURE</td>
</tr>
<tr>
<td>noisure</td>
<td>use one-step SURE</td>
</tr>
<tr>
<td>dfk</td>
<td>make small-sample degrees-of-freedom adjustment</td>
</tr>
<tr>
<td>small</td>
<td>report small-sample <code>t</code> and <code>F</code> statistics</td>
</tr>
<tr>
<td>noidencheck</td>
<td>do not check for local identification</td>
</tr>
<tr>
<td>nobigf</td>
<td>do not compute parameter vector for coefficients implicitly set to zero</td>
</tr>
</tbody>
</table>

**Reporting**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level(#)</td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td>full</td>
<td>show constrained parameters in table</td>
</tr>
<tr>
<td>var</td>
<td>display underlying <code>var</code> output</td>
</tr>
<tr>
<td>lutstats</td>
<td>report Lütkepohl lag-order selection statistics</td>
</tr>
<tr>
<td>nocnsreport</td>
<td>do not display constraints</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats</td>
</tr>
</tbody>
</table>

**Maximization**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize_options</td>
<td>control the maximization process; seldom used</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

*aconstraints(constraints_a), aeq(matrix_aeq), acns(matrix_acns), bconstraints(constraints_b), beq(matrix_beq), bcns(matrix_bcns): at least one of these options must be specified.

coeflegend does not appear in the dialog box.
### long_run_options Description

**Model**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noconstant</td>
<td>suppress constant term</td>
</tr>
<tr>
<td>lrconstraints(constraints lr)</td>
<td>apply previously defined constraints lr to C</td>
</tr>
<tr>
<td>lreq(matrix lreq)</td>
<td>define and apply to C equality constraint matrix matrix lreq</td>
</tr>
<tr>
<td>lrcns(matrix lrcns)</td>
<td>define and apply to C cross-parameter constraint matrix matrix lrcns</td>
</tr>
<tr>
<td>lags(numlist)</td>
<td>use lags numlist in the underlying VAR</td>
</tr>
</tbody>
</table>

**Model 2**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exog(varlist exog)</td>
<td>use exogenous variables varlist</td>
</tr>
<tr>
<td>varconstraints(constraints v)</td>
<td>apply constraints v to underlying VAR</td>
</tr>
<tr>
<td>noislog</td>
<td>suppress SURE iteration log</td>
</tr>
<tr>
<td>isiterate(#)</td>
<td>set maximum number of iterations for SURE; default is isiterate(1600)</td>
</tr>
<tr>
<td>istolerance(#)</td>
<td>set convergence tolerance of SURE</td>
</tr>
<tr>
<td>noise</td>
<td>use one-step SURE</td>
</tr>
<tr>
<td>dfk</td>
<td>make small-sample degrees-of-freedom adjustment</td>
</tr>
<tr>
<td>small</td>
<td>report small-sample $t$ and $F$ statistics</td>
</tr>
<tr>
<td>noidencheck</td>
<td>do not check for local identification</td>
</tr>
<tr>
<td>nobigf</td>
<td>do not compute parameter vector for coefficients implicitly set to zero</td>
</tr>
</tbody>
</table>

**Reporting**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level(#)</td>
<td>set confidence level; default is level(95)</td>
</tr>
<tr>
<td>full</td>
<td>show constrained parameters in table</td>
</tr>
<tr>
<td>var</td>
<td>display underlying var output</td>
</tr>
<tr>
<td>lutstats</td>
<td>report Lütkepohl lag-order selection statistics</td>
</tr>
<tr>
<td>nocnsreport</td>
<td>do not display constraints</td>
</tr>
<tr>
<td>display_options</td>
<td>control columns and column formats</td>
</tr>
</tbody>
</table>

**Maximization**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize_options</td>
<td>control the maximization process; seldom used</td>
</tr>
<tr>
<td>coeflegend</td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

* lrconstraints(constraints lr), lreq(matrix lreq), lrcns(matrix lrcns): at least one of these options must be specified.

coeflegend does not appear in the dialog box.

You must tsset your data before using svar; see [TS] tsset.

depvarlist and varlist exog may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, fp, rolling, statsby, and xi are allowed; see [U] 11.1.10 Prefix commands.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
Options

noconstant; see [R] Estimation options.

acconstraints(constraints$_a$), aeq(matrix$_{aeq}$), acns(matrix$_{acns}$)

bconstraints(constraints$_b$), beq(matrix$_{beq}$), bcns(matrix$_{bcns}$)

These options specify the short-run constraints in an SVAR. To specify a short-run SVAR model, you must specify at least one of these options. The first list of options specifies constraints on the parameters of the $A$ matrix; the second list specifies constraints on the parameters of the $B$ matrix (see Short-run SVAR models). If at least one option is selected from the first list and none are selected from the second list, svar sets $B$ to the identity matrix. Similarly, if at least one option is selected from the second list and none are selected from the first list, svar sets $A$ to the identity matrix.

None of these options may be specified with any of the options that define long-run constraints.

acconstraints(constraints$_a$) specifies a numlist of previously defined Stata constraints to be applied to $A$ during estimation.

aeq(matrix$_{aeq}$) specifies a matrix that defines a set of equality constraints. This matrix must be square with dimension equal to the number of equations in the underlying VAR. The elements of this matrix must be missing or real numbers. A missing value in the $(i, j)$ element of this matrix specifies that the $(i, j)$ element of $A$ is a free parameter. A real number in the $(i, j)$ element of this matrix constrains the $(i, j)$ element of $A$ to this real number. For example,

$$ A = \begin{bmatrix} 1 & 0 \\ \cdot & 1.5 \end{bmatrix} $$


acns(matrix$_{acns}$) specifies a matrix that defines a set of exclusion or cross-parameter equality constraints on $A$. This matrix must be square with dimension equal to the number of equations in the underlying VAR. Each element of this matrix must be missing, 0, or a positive integer. A missing value in the $(i, j)$ element of this matrix specifies that no constraint be placed on this element of $A$. A zero in the $(i, j)$ element of this matrix constrains the $(i, j)$ element of $A$ to be zero. Any strictly positive integers must be in two or more elements of this matrix. A strictly positive integer in the $(i, j)$ element of this matrix constrains the $(i, j)$ element of $A$ to be equal to all the other elements of $A$ that correspond to elements in this matrix that contain the same integer. For example, consider the matrix

$$ A = \begin{bmatrix} \cdot & 1 \\ 1 & 0 \end{bmatrix} $$


bconstraints(constraints$_b$) specifies a numlist of previously defined Stata constraints to be applied to $B$ during estimation.

beq(matrix$_{beq}$) specifies a matrix that defines a set of equality constraints. This matrix must be square with dimension equal to the number of equations in the underlying VAR. The elements of this matrix must be either missing or real numbers. The syntax of implied constraints is analogous to the one described in aeq(), except that it applies to $B$ rather than to $A$. 

bcns(matrixbcns) specifies a matrix that defines a set of exclusion or cross-parameter equality constraints on B. This matrix must be square with dimension equal to the number of equations in the underlying VAR. Each element of this matrix must be missing, 0, or a positive integer. The format of the implied constraints is the same as the one described in the acns() option above.

lrconstraints(constraintslr), lreq(matrixlreq), lrcns(matrixlrcns)
These options specify the long-run constraints in an SVAR. To specify a long-run SVAR model, you must specify at least one of these options. The list of options specifies constraints on the parameters of the long-run C matrix (see Long-run SVAR models for the definition of C). None of these options may be specified with any of the options that define short-run constraints.

lrconstraints(constraintslr) specifies a numlist of previously defined Stata constraints to be applied to C during estimation.
lreq(matrixlreq) specifies a matrix that defines a set of equality constraints on the elements of C. This matrix must be square with dimension equal to the number of equations in the underlying VAR. The elements of this matrix must be either missing or real numbers. The syntax of implied constraints is analogous to the one described in option aeq(), except that it applies to C.
lrcns(matrixlrcns) specifies a matrix that defines a set of exclusion or cross-parameter equality constraints on C. This matrix must be square with dimension equal to the number of equations in the underlying VAR. Each element of this matrix must be missing, 0, or a positive integer. The syntax of the implied constraints is the same as the one described for the acns() option above.

lags(numlist) specifies the lags to be included in the underlying VAR model. The default is lags(1 2). This option takes a numlist and not simply an integer for the maximum lag. For instance, lags(2) would include only the second lag in the model, whereas lags(1/2) would include both the first and second lags in the model. See [U] 11.1.8 numlist and [U] 11.4.4 Time-series varlists for further discussion of numlists and lags.

exog(varlistexog) specifies a list of exogenous variables to be included in the underlying VAR.

varconstraints(constraintsv) specifies a list of constraints to be applied to the coefficients in the underlying VAR. Because svar estimates multiple equations, the constraints must specify the equation name for all but the first equation.

noislog prevents svar from displaying the iteration log from the iterated seemingly unrelated regression algorithm. When the varconstraints() option is not specified, the VAR coefficients are estimated via OLS, a noniterative procedure. As a result, noislog may be specified only with varconstraints(). Similarly, noislog may not be combined with noisure.

isiterate(#) sets the maximum number of iterations for the iterated seemingly unrelated regression algorithm. The default limit is 1,600. When the varconstraints() option is not specified, the VAR coefficients are estimated via OLS, a noniterative procedure. As a result, isiterate() may be specified only with varconstraints(). Similarly, isiterate() may not be combined with noisure.

istolerance(#) specifies the convergence tolerance of the iterated seemingly unrelated regression algorithm. The default tolerance is 1e-6. When the varconstraints() option is not specified, the VAR coefficients are estimated via OLS, a noniterative procedure. As a result, istolerance() may be specified only with varconstraints(). Similarly, istolerance() may not be combined with noisure.
noisure specifies that the VAR coefficients be estimated via one-step seemingly unrelated regression when \texttt{varconstraints()} is specified. By default, \texttt{svar} estimates the coefficients in the VAR via iterated seemingly unrelated regression when \texttt{varconstraints()} is specified. When the \texttt{varconstraints()} option is not specified, the VAR coefficient estimates are obtained via OLS, a noniterative procedure. As a result, \texttt{noisure} may be specified only with \texttt{varconstraints()}.

dfk specifies that a small-sample degrees-of-freedom adjustment be used when estimating $\Sigma$, the covariance matrix of the VAR disturbances. Specifically, $\frac{1}{(T - \bar{m})}$ is used instead of the large-sample divisor $\frac{1}{T}$, where $\bar{m}$ is the average number of parameters in the functional form for $y_t$ over the $K$ equations.

small causes \texttt{svar} to calculate and report small-sample $t$ and $F$ statistics instead of the large-sample normal and chi-squared statistics.

noidencheck requests that the Amisano and Giannini (1997) check for local identification not be performed. This check is local to the starting values used. Because of this dependence on the starting values, you may wish to suppress this check by specifying the \texttt{noidencheck} option. However, be careful in specifying this option. Models that are not structurally identified can still converge, thereby producing meaningless results that only appear to have meaning.

nobigf requests that \texttt{svar} not compute the estimated parameter vector that incorporates coefficients that have been implicitly constrained to be zero, such as when some lags have been omitted from a model. \texttt{e(bf)} is used for computing asymptotic standard errors in the postestimation commands \texttt{irf create} and \texttt{fcast compute}. Therefore, specifying \texttt{nobigf} implies that the asymptotic standard errors will not be available from \texttt{irf create} and \texttt{fcast compute}. See Fitting models with some lags excluded in [TS] \texttt{var}.

\section{Reporting}

\texttt{level(#)}; see \texttt{[R] Estimation options}.

\texttt{full} shows constrained parameters in table.

\texttt{var} specifies that the output from \texttt{var} also be displayed. By default, the underlying VAR is fit quietly.

\texttt{lutstats} specifies that the Lütkepohl versions of the lag-order selection statistics be computed. See Methods and formulas in [TS] \texttt{varsoc} for a discussion of these statistics.

\texttt{nocnsreport}; see \texttt{[R] Estimation options}.

display_options: \texttt{noci}, \texttt{nopvalues}, \texttt{cformat(\%fmt)}, \texttt{pformat(\%fmt)}, and \texttt{sformat(\%fmt)}; see \texttt{[R] Estimation options}.

\section{Maximization}

\texttt{maximize_options: difficult}, \texttt{technique(algorithm_spec)}, \texttt{iterate(#), [no]log}, \texttt{trace, gradient}, \texttt{showstep}, \texttt{hessian}, \texttt{showtolerance, tolerance(#), itolerance(#), nrtolerance(#), nonrtolerance, and from(init_specs)}; see \texttt{[R] Maximize}. These options are seldom used.

The following option is available with \texttt{svar} but is not shown in the dialog box:

\texttt{coeflegend}; see \texttt{[R] Estimation options}.
Remarks and examples

Remarks are presented under the following headings:

Introduction
Short-run SVAR models
Long-run SVAR models

Introduction

This entry assumes that you have already read [TS] var intro and [TS] var; if not, please do. Here we illustrate how to fit SVARs in Stata subject to short-run and long-run restrictions. For more detailed information on SVARs, see Amisano and Giannini (1997) and Hamilton (1994). For good introductions to VARs, see Lütkepohl (2005), Hamilton (1994), Stock and Watson (2001), and Becketti (2013).

Short-run SVAR models

A short-run SVAR model without exogenous variables can be written as

$$A(I_K - A_1 L - A_2 L^2 - \cdots - A_p L^p)y_t = A \epsilon_t = B e_t$$

where $L$ is the lag operator, $A$, $B$, and $A_1, \ldots, A_p$ are $K \times K$ matrices of parameters, $\epsilon_t$ is a $K \times 1$ vector of innovations with $\epsilon_t \sim N(0, \Sigma)$ and $E[\epsilon_t \epsilon_s'] = 0_K$ for all $s \neq t$, and $e_t$ is a $K \times 1$ vector of orthogonalized disturbances; that is, $e_t \sim N(0, I_K)$ and $E[e_t e_s'] = 0_K$ for all $s \neq t$. These transformations of the innovations allow us to analyze the dynamics of the system in terms of a change to an element of $e_t$. In a short-run SVAR model, we obtain identification by placing restrictions on $A$ and $B$, which are assumed to be nonsingular.

Example 1: Short-run just-identified SVAR model

Following Sims (1980), the Cholesky decomposition is one method of identifying the impulse–response functions in a VAR; thus, this method corresponds to an SVAR. There are several sets of constraints on $A$ and $B$ that are easily manipulated back to the Cholesky decomposition, and the following example illustrates this point.

One way to impose the Cholesky restrictions is to assume an SVAR model of the form

$$\tilde{A}(I_K - A_1 - A_2 L^2 - \cdots - A_p L^p)y_t = \tilde{B} e_t$$

where $\tilde{A}$ is a lower triangular matrix with ones on the diagonal and $\tilde{B}$ is a diagonal matrix. Because the $P$ matrix for this model is $P_{sr} = \tilde{A}^{-1} \tilde{B}$, its estimate, $\hat{P}_{sr}$, obtained by plugging in estimates of $\tilde{A}$ and $\tilde{B}$, should equal the Cholesky decomposition of $\hat{\Sigma}$.

To illustrate, we use the German macroeconomic data discussed in Lütkepohl (2005) and used in [TS] var. In this example, $y_t = (\text{dln_inv}, \text{dln_inc}, \text{dln_consump})$, where dln_inv is the first difference of the log of investment, dln_inc is the first difference of the log of income, and dln_consump is the first difference of the log of consumption. Because the first difference of the natural log of a variable can be treated as an approximation of the percentage change in that variable, we will refer to these variables as percentage changes in inv, inc, and consump, respectively.

We will impose the Cholesky restrictions on this system by applying equality constraints with the constraint matrices

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} . & 0 & 0 \\ 0 & . & 0 \\ 0 & 0 \end{bmatrix}$$
With these structural restrictions, we assume that the percentage change in inv is not contemporaneously affected by the percentage changes in either inc or consump. We also assume that the percentage change of inc is affected by contemporaneous changes in inv but not consump. Finally, we assume that percentage changes in consump are affected by contemporaneous changes in both inv and inc.

The following commands fit an SVAR model with these constraints.

```
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. matrix A = (1,0,0\,1,0,\,0,1)
. matrix B = (.\,0,0\,0,0,0)
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), aeq(A) beq(B)
```

```
Estimating short-run parameters
(output omitted)
```

```
Structural vector autoregression

(1) [/A]1_1 = 1
(2) [/A]1_2 = 0
(3) [/A]1_3 = 0
(4) [/A]2_2 = 1
(5) [/A]2_3 = 0
(6) [/A]3_3 = 1
(7) [/B]1_2 = 0
(8) [/B]1_3 = 0
(9) [/B]2_1 = 0
(10) [/B]2_3 = 0
(11) [/B]3_1 = 0
(12) [/B]3_2 = 0

Sample: 1960q4 – 1978q4  Number of obs = 73
Exactly identified model  Log likelihood = 606.307

|     | Coef. | Std. Err. | z    | P>|z|   | [95% Conf. Interval] |
|-----|-------|-----------|------|-------|---------------------|
| /A  |       |           |      |       |                     |
| 1_1 | 1     | (constrained) |
| 2_1 | -.0336288 | .0294605 | -1.14 | .254   | -.0913702 -.0241126 |
| 3_1 | -.0435846 | .0194408 | -2.24 | .025   | -.0816879 -.0054812 |
| 1_2 | 0     | (constrained) |
| 2_2 | 1     | (constrained) |
| 3_2 | -.424774 | .0765548 | -5.55 | .000   | -.5748187 -.2747293 |
| 1_3 | 0     | (constrained) |
| 2_3 | 0     | (constrained) |
| 3_3 | 1     | (constrained) |

| /B  |       |           |      |       |                     |
| 1_1 | .0438796 | .0036315 | 12.08 | .000  | .036762 .0509972   |
| 2_1 | 0     | (constrained) |
| 3_1 | 0     | (constrained) |
| 1_2 | 0     | (constrained) |
| 2_2 | .0110449 | .0009141 | 12.08 | .000  | .0092534 .0128365 |
| 3_2 | 0     | (constrained) |
| 1_3 | 0     | (constrained) |
| 2_3 | 0     | (constrained) |
| 3_3 | .0072243 | .0005979 | 12.08 | .000  | .0060525 .0083962 |
```

The SVAR output has four parts: an iteration log, a display of the constraints imposed, a header with sample and SVAR log-likelihood information, and a table displaying the estimates of the parameters from the A and B matrices. From the output above, we can see that the equality constraint matrices...
supplied to `svar` imposed the intended constraints and that the SVAR header informs us that the model we fit is just identified. The estimates of \( A_{2,1} \), \( A_{3,1} \), and \( A_{3,2} \) are all negative. Because the off-diagonal elements of the \( A \) matrix contain the negative of the actual contemporaneous effects, the estimated effects are positive, as expected.

The estimates \( \hat{A} \) and \( \hat{B} \) are stored in `e(A)` and `e(B)`, respectively, allowing us to compute the estimated Cholesky decomposition.

```
. matrix Aest = e(A)
. matrix Best = e(B)
. matrix chol_est = inv(Aest)*Best
. matrix list chol_est
```

```
<table>
<thead>
<tr>
<th></th>
<th>dln_inv</th>
<th>dln_inc</th>
<th>dln_consump</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>0.04387957</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dln_inc</td>
<td>0.00147562</td>
<td>0.01104494</td>
<td>0</td>
</tr>
<tr>
<td>dln_consump</td>
<td>0.00253928</td>
<td>0.0046916</td>
<td>0.00722432</td>
</tr>
</tbody>
</table>
```

`svar` stores the estimated \( \Sigma \) from the underlying `var` in `e(Sigma)`. The output below illustrates the computation of the Cholesky decomposition of `e(Sigma)`. It is the same as the output computed from the `SVAR` estimates.

```
. matrix sig_var = e(Sigma)
. matrix chol_var = cholesky(sig_var)
. matrix list chol_var
```

```
<table>
<thead>
<tr>
<th></th>
<th>dln_inv</th>
<th>dln_inc</th>
<th>dln_consump</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>0.04387957</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dln_inc</td>
<td>0.00147562</td>
<td>0.01104494</td>
<td>0</td>
</tr>
<tr>
<td>dln_consump</td>
<td>0.00253928</td>
<td>0.0046916</td>
<td>0.00722432</td>
</tr>
</tbody>
</table>
```

We might now wonder why we bother obtaining parameter estimates via nonlinear estimation if we can obtain them simply by a transform of the estimates produced by `var`. When the model is just identified, as in the previous example, the SVAR parameter estimates can be computed via a transform of the VAR estimates. However, when the model is overidentified, such is not the case.

**Example 2: Short-run overidentified SVAR model**

The Cholesky decomposition example above fit a just-identified model. This example considers an overidentified model. In example 1, the \( A_{2,1} \) parameter was not significant, which is consistent with a theory in which changes in our measure of investment affect only changes in income with a lag. We can impose the restriction that \( A_{2,1} \) is zero and then test this overidentifying restriction. Our \( A \) and \( B \) matrices are now

\[
A = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
. & . & 1
\end{bmatrix}
\quad\text{and}\quad
B = \begin{bmatrix}
. & 0 & 0 \\
0 & . & 0 \\
0 & 0 & .
\end{bmatrix}
\]

The output below contains the commands and results we obtained by fitting this model on the Lütkepohl data.

```
. matrix B = (.,0,0\0,0,0\0,0,.)
. matrix A = (1,0,0\0,1,0\.,.,1)
```
Estimating short-run parameters

(output omitted)

Structural vector autoregression

(1) \[ A_{11} = 1 \]
(2) \[ A_{12} = 0 \]
(3) \[ A_{13} = 0 \]
(4) \[ A_{21} = 0 \]
(5) \[ A_{22} = 1 \]
(6) \[ A_{23} = 0 \]
(7) \[ A_{33} = 1 \]
(8) \[ B_{12} = 0 \]
(9) \[ B_{13} = 0 \]
(10) \[ B_{21} = 0 \]
(11) \[ B_{23} = 0 \]
(12) \[ B_{31} = 0 \]
(13) \[ B_{32} = 0 \]

Sample: 1960q4 - 1978q4 Number of obs = 73
Overidentified model Log likelihood = 605.6613

|       | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|-------|-------|-----------|------|------|----------------------|
| \( A \) |       |           |      |      |                      |
| 1_1   | 1     | (constrained) |
| 2_1   | 0     | (constrained) |
| 3_1   | -.0435911 | .0192696   | -2.26 | 0.024 | -.0813589 -.0058233 |
| 1_2   | 0     | (constrained) |
| 2_2   | 1     | (constrained) |
| 3_2   | -.4247741 | .0758806   | -5.60 | 0.000 | -.5734973 -.2760508 |
| 1_3   | 0     | (constrained) |
| 2_3   | 0     | (constrained) |
| 3_3   | 1     | (constrained) |
| \( B \) |       |           |      |      |                      |
| 1_1   | .0438796 | .0036315   | 12.08 | 0.000 | .036762 .0509972    |
| 2_1   | 0     | (constrained) |
| 3_1   | 0     | (constrained) |
| 1_2   | 0     | (constrained) |
| 2_2   | .0111431 | .0009222   | 12.08 | 0.000 | .0093356 .0129506   |
| 3_2   | 0     | (constrained) |
| 1_3   | 0     | (constrained) |
| 2_3   | 0     | (constrained) |
| 3_3   | .0072243 | .0005979   | 12.08 | 0.000 | .0060525 .0083962   |

LR test of identifying restrictions: chi2(1) = 1.292 Prob > chi2 = 0.256

The footer in this example reports a test of the overidentifying restriction. The null hypothesis of this test is that any overidentifying restrictions are valid. In the case at hand, we cannot reject this null hypothesis at any of the conventional levels.

Example 3: Short-run SVAR model with constraints

`svar` also allows us to place constraints on the parameters of the underlying VAR. We begin by looking at the underlying VAR for the SVARs that we have used in the previous examples.
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4)

Vector autoregression

Sample: 1960q4 - 1978q4  Number of obs = 73
Log likelihood = 606.307  AIC = -16.03581
FPE = 2.18e-11  HQIC = -15.77323
Det(Sigma_ml) = 1.23e-11  SBIC = -15.37691

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>RMSE</th>
<th>R-sq</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>7</td>
<td>.046148</td>
<td>0.1286</td>
<td>10.76961</td>
<td>0.0958</td>
</tr>
<tr>
<td>dln_inc</td>
<td>7</td>
<td>.011719</td>
<td>0.1142</td>
<td>9.410683</td>
<td>0.1518</td>
</tr>
<tr>
<td>dln_consump</td>
<td>7</td>
<td>.009445</td>
<td>0.2513</td>
<td>24.50031</td>
<td>0.0004</td>
</tr>
</tbody>
</table>

| dln_inv       | Coef. | Std. Err. | z  | P>|z|   | [95% Conf. Interval] |
|---------------|-------|-----------|----|-------|----------------------|
| dln_inv       |       |           |    |        |                      |
| L1.           | -.3196318 | .1192898 | -2.68 | 0.007 | -.5534355 -.0858282 |
| L2.           | -.1605508 | .118767  | -1.35 | 0.176 | -.39333 .0722283   |

| dln_inc       | Coef. | Std. Err. | z  | P>|z|   | [95% Conf. Interval] |
|---------------|-------|-----------|----|-------|----------------------|
| dln_inc       |       |           |    |        |                      |
| L1.           | .1459851 | .5188451 | 0.28 | 0.778 | -.8709326 1.162903  |
| L2.           | .1146009 | .508295  | 0.23 | 0.822 | -.881639 1.110841  |

| dln_consump   | Coef. | Std. Err. | z  | P>|z|   | [95% Conf. Interval] |
|---------------|-------|-----------|----|-------|----------------------|
| dln_consump   |       |           |    |        |                      |
| L1.           | .9612288 | .6316575 | 1.52 | 0.128 | -.2767936 2.199251  |
| L2.           | .9344001 | .6324034 | 1.48 | 0.140 | -.3050877 2.173888  |

| _cons         | Coef. | Std. Err. | z  | P>|z|   | [95% Conf. Interval] |
|---------------|-------|-----------|----|-------|----------------------|
| _cons         | -.0167221 | .0163796 | -1.02 | 0.307 | -.0488257 .0153814  |
The equation-level model tests reported in the header indicate that we cannot reject the null hypotheses that all the coefficients in the first equation are zero, nor can we reject the null that all the coefficients in the second equation are zero at the 5% significance level. We use a combination of theory and the \( p \)-values from the output above to place some exclusion restrictions on the underlying VAR(2). Specifically, in the equation for the percentage change of \( \text{inv} \), we constrain the coefficients on \( L_2.dln_{\text{inv}}, L.dln_{\text{inc}}, L_2.dln_{\text{inc}}, \) and \( L_2.dln_{\text{consump}} \) to be zero. In the equation for \( dln_{\text{inc}} \), we constrain the coefficients on \( L_2.dln_{\text{inv}}, L_2.dln_{\text{inc}}, \) and \( L_2.dln_{\text{consump}} \) to be zero. Finally, in the equation for \( dln_{\text{consump}} \), we constrain \( L.dln_{\text{inv}} \) and \( L_2.dln_{\text{consump}} \) to be zero. We then refit the SVAR from the previous example.

. constraint 1 [dln_inv]L2.dln_inv = 0
. constraint 2 [dln_inv]L.dln_inc = 0
. constraint 3 [dln_inv]L2.dln_inc = 0
. constraint 4 [dln_inv]L2.dln_consump = 0
. constraint 5 [dln_inc]L2.dln_inv = 0
. constraint 6 [dln_inc]L2.dln_inc = 0
. constraint 7 [dln_inc]L2.dln_consump = 0
. constraint 8 [dln_consump]L.dln_inv = 0
. constraint 9 [dln_consump]L2.dln_consump = 0
.svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), aeq(A) beq(B) > varconst(1/9) noislog
Estimating short-run parameters
(output omitted)

Structural vector autoregression
( 1)  [A]1_1 = 1
( 2)  [A]1_2 = 0
( 3)  [A]1_3 = 0
( 4)  [A]2_1 = 0
( 5)  [A]2_2 = 1
( 6)  [A]2_3 = 0
( 7)  [A]3_3 = 1
( 8)  [B]1_2 = 0
( 9)  [B]1_3 = 0
(10)  [B]2_1 = 0
(11)  [B]2_2 = 0
(12)  [B]3_1 = 0
(13)  [B]3_2 = 0
Sample: 1960q4 - 1978q4  Number of obs    =    73
Overidentified model  Log likelihood =  601.8591

| Coef.  | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|--------|-----------|-------|------|----------------------|
| /A     |           |       |      |                      |
| 1_1    | 1         |       |      |                      |
| 2_1    | 0         |       |      |                      |
| 3_1    | -.0418708 | .0187579 | -2.23 | .026 | -.0786356 | -.0051061 |
| 1_2    | 0         |       |      |                      |
| 2_2    | 1         |       |      |                      |
| 3_2    | -.4255808 | .0745298 | -5.71 | .000 | -.5716565 | -.2795051 |
| 1_3    | 0         |       |      |                      |
| 2_3    | 0         |       |      |                      |
| 3_3    | 1         |       |      |                      |
| /B     |           |       |      |                      |
| 1_1    | .0451851  | .0037395 | 12.08 | .000 | .0378557 | .0525145 |
| 2_1    | 0         |       |      |                      |
| 3_1    | 0         |       |      |                      |
| 1_2    | 0         |       |      |                      |
| 2_2    | .0113723  | .0009412 | 12.08 | .000 | .0095276 | .013217  |
| 3_2    | 0         |       |      |                      |
| 1_3    | 0         |       |      |                      |
| 2_3    | 0         |       |      |                      |
| 3_3    | .0072417  | .0005993 | 12.08 | .000 | .006067  | .0084164 |

LR test of identifying restrictions: chi2(1) = 0.8448  Prob > chi2 = 0.358

If we displayed the underlying VAR(2) results by using the var option, we would see that most of the unconstrained coefficients are now significant at the 10% level and that none of the equation-level model statistics fail to reject the null hypothesis at the 10% level. The svar output reveals that the p-value of the overidentification test rose and that the coefficient on /A:3_1 is still insignificant at the 1% level but not at the 5% level.

Before moving on to models with long-run constraints, consider these limitations. We cannot place constraints on the elements of A in terms of the elements of B, or vice versa. This limitation is imposed by the form of the check for identification derived by Amisano and Giannini (1997). As noted in Methods and formulas, this test requires separate constraint matrices for the parameters in A and B. Another limitation is that we cannot mix short-run and long-run constraints.

Long-run SVAR models

As discussed in [TS] var intro, a long-run SVAR has the form

\[ y_t = C e_t \]

In long-run models, the constraints are placed on the elements of C, and the free parameters are estimated. These constraints are often exclusion restrictions. For instance, constraining \( C[1,2] \) to be zero can be interpreted as setting the long-run response of variable 1 to the structural shocks driving variable 2 to be zero.

Similar to the short-run model, the \( P_{lr} \) matrix such that \( P_{lr} P_{lr}' = \Sigma \) identifies the structural impulse–response functions. \( P_{lr} = C \) is identified by the restrictions placed on the parameters in C. There are \( K^2 \) parameters in C, and the order condition for identification requires that there be
at least $K^2 - K(K + 1)/2$ restrictions placed on those parameters. As in the short-run model, this order condition is necessary but not sufficient, so the Amisano and Giannini (1997) check for local identification is performed by default.

### Example 4: Long-run SVAR model

Suppose that we have a theory in which unexpected changes to the money supply have no long-run effects on changes in output and, similarly, that unexpected changes in output have no long-run effects on changes in the money supply. The $C$ matrix implied by this theory is

$$C = \begin{bmatrix} . & 0 \\ 0 & . \end{bmatrix}$$

```
. use https://www.stata-press.com/data/r16/m1gdp
. matrix lr = (.,0\0,.)
. svar d.ln_m1 d.ln_gdp, lreq(lr)
```

```
Estimating long-run parameters
(output omitted)
```

```
Structural vector autoregression
(1) [C]1_2 = 0
(2) [C]2_1 = 0

Sample: 1959q4 - 2002q2    Number of obs = 171
Overidentified model    Log likelihood = 1151.614

|     | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-----|--------|-----------|-------|------|----------------------|
| /C  |        |           |       |      |                      |
| 1_1 | 0.0301007 | 0.0016277 | 18.49 | 0.000 | 0.0269106 - 0.0332909 |
| 2_1 | 0 (constrained) |       |       |      |                      |
| 1_2 | 0 (constrained) |       |       |      |                      |
| 2_2 | 0.0129691 | 0.0007013 | 18.49 | 0.000 | 0.0115946 - 0.0143436 |

LR test of identifying restrictions: chi2(1) = .1368  Prob > chi2 = 0.712
```

We have assumed that the underlying VAR has 2 lags; four of the five selection-order criteria computed by `varsoc` (see `[TS] varsoc`) recommended this choice. The test of the overidentifying restrictions provides no indication that it is not valid.
**svar** stores the following in e():

Scalars
- \( e(N) \): number of observations
- \( e(N_{\text{cns}}) \): number of constraints
- \( e(k_{\text{eq}}) \): number of equations in e(b)
- \( e(k_{\text{dv}}) \): number of dependent variables
- \( e(ll) \): log likelihood from svar
- \( e(N_{\text{gaps \_var}}) \): number of gaps in the sample
- \( e(k_{\text{var}}) \): number of coefficients in VAR
- \( e(k_{\text{eq \_var}}) \): number of equations in underlying VAR
- \( e(k_{\text{dv \_var}}) \): number of dependent variables in underlying VAR
- \( e(df_{eq \_var}) \): average number of parameters in an equation
- \( e(df_{r \_var}) \): if small, residual degrees of freedom
- \( e(\text{obs \_# \_var}) \): number of observations on equation #
- \( e(k_{\text{# \_var}}) \): number of coefficients in equation #
- \( e(df_{m\# \_var}) \): model degrees of freedom for equation #
- \( e(df_{r\# \_var}) \): residual degrees of freedom for equation # (small only)
- \( e(r2_{\# \_var}) \): R-squared for equation #
- \( e(ll_{\# \_var}) \): log likelihood for equation # VAR
- \( e(chi2_{\# \_var}) \): \( \chi^2 \) statistic for equation #
- \( e(F_{\# \_var}) \): F statistic for equation # (small only)
- \( e(rmse_{\# \_var}) \): root mean squared error for equation #
- \( e(mlag \_var) \): highest lag in VAR
- \( e(tparms \_var) \): number of parameters in all equations
- \( e(aic \_var) \): Akaike information criterion
- \( e(hbic \_var) \): Hannan–Quinn information criterion
- \( e(sbic \_var) \): Schwarz–Bayesian information criterion
- \( e(fpe \_var) \): final prediction error
- \( e(ll \_var) \): log likelihood from var
- \( e(detsig \_var) \): determinant of e(Sigma)
- \( e(detsig_{ml \_var}) \): determinant of \( \hat{\Sigma}_{ml} \)
- \( e(tmin) \): first time period in the sample
- \( e(tmax) \): maximum time
- \( e(chi2_{oid}) \): overidentification test
- \( e(oid_{df}) \): number of overidentifying restrictions
- \( e(rank) \): rank of e(V)
- \( e(ic_{ml}) \): number of iterations
- \( e(rc_{ml}) \): return code from ml
Methods and formulas

The log-likelihood function for models with short-run constraints is

\[ L(A, B) = -\frac{NK}{2} \ln(2\pi) + \frac{N}{2} \ln(|W|^2) - \frac{N}{2} \text{tr}(W'W\hat{\Sigma}) \]

where \( W = B^{-1}A \).

When there are long-run constraints, because \( C = \bar{A}^{-1}B \) and \( A = I_K \), \( W = B^{-1} = C^{-1}\bar{A}^{-1} = (\bar{A}C)^{-1} \). Substituting the last term for \( W \) in the short-run log likelihood produces the long-run log likelihood
\[
L(C) = -\frac{NK}{2} \ln(2\pi) + \frac{N}{2} \ln(|\tilde{W}|^2) - \frac{N}{2} \text{tr}(\tilde{W}'\tilde{W}\hat{\Sigma})
\]

where \(\tilde{W} = (\tilde{A}C)^{-1}\).

For both the short-run and the long-run models, the maximization is performed by the scoring method. See Harvey (1990) for a discussion of this method.

Based on results from Amisano and Giannini (1997), the score vector for the short-run model is

\[
\frac{\partial L(A, B)}{\partial [\text{vec}(A), \text{vec}(B)]} = N \left[ \{\text{vec}(W'^{-1})\}' - \{\text{vec}(W)\}'(\hat{\Sigma} \otimes I_K) \right] \times
\]

\[
[(I_K \otimes B^{-1}), -(A'B'^{-1} \otimes B^{-1})]
\]

and the expected information matrix is

\[
I[\text{vec}(A), \text{vec}(B)] = N \left[ (W^{-1} \otimes B'^{-1})' \right] (I_K^2 + \oplus) \left[ (W'^{-1} \otimes B^{-1}), -(I_K \otimes B^{-1}) \right]
\]

where \(\oplus\) is the commutation matrix defined in Magnus and Neudecker (1999, 46–48).

Using results from Amisano and Giannini (1997), we can derive the score vector and the expected information matrix for the case with long-run restrictions. The score vector is

\[
\frac{\partial L(C)}{\partial \text{vec}(C)} = N \left[ \{\text{vec}(W'^{-1})\}' - \{\text{vec}(W)\}'(\hat{\Sigma} \otimes I_K) \right] [-&(A'^{-1}C'^{-1} \otimes C^{-1})]
\]

and the expected information matrix is

\[
I[\text{vec}(C)] = N(I_K \otimes C'^{-1})(I_K^2 + \oplus)(I_K \otimes C'^{-1})
\]

### Checking for identification

This section describes the methods used to check for identification of models with short-run or long-run constraints. Both methods depend on the starting values. By default, \texttt{svar} uses starting values constructed by taking a vector of appropriate dimension and applying the constraints. If there are \(m\) parameters in the model, the \(j\)th element of the \(1 \times m\) vector is \(1 + m/100\). \texttt{svar} also allows the user to provide starting values.

For the short-run case, the model is identified if the matrix

\[
V_{sr}^* = \begin{bmatrix}
N_K & N_K \\
N_K & N_K \\
R_a(W' \otimes B) & 0_{K^2} \\
0_{K^2} & R_a(I_K \otimes B)
\end{bmatrix}
\]

has full column rank of \(2K^2\), where \(N_K = (1/2)(I_K^2 + \oplus)\), \(R_a\) is the constraint matrix for the parameters in \(A\) (that is, \(R_a\text{vec}(A) = r_a\)), and \(R_b\) is the constraint matrix for the parameters in \(B\) (that is, \(R_b\text{vec}(B) = r_b\)).
For the long-run case, based on results from the C model in Amisano and Giannini (1997), the model is identified if the matrix

\[ V_{lr}^* = \begin{bmatrix} (I \otimes C'^{-1})(2N_K) & (I \otimes C^{-1}) \\ R_c \end{bmatrix} \]

has full column rank of \( K^2 \), where \( R_c \) is the constraint matrix for the parameters in \( C \); that is, \( R_c \vec(C) = r_c \).

The test of the overidentifying restrictions is computed as

\[ LR = 2(LL_{var} - LL_{svar}) \]

where \( LR \) is the value of the test statistic against the null hypothesis that the overidentifying restrictions are valid, \( LL_{var} \) is the log likelihood from the underlying VAR(\( p \)) model, and \( LL_{svar} \) is the log likelihood from the SVAR model. The test statistic is asymptotically distributed as \( \chi^2(q) \), where \( q \) is the number of overidentifying restrictions. Amisano and Giannini (1997, 38–39) emphasize that, because this test of the validity of the overidentifying restrictions is an omnibus test, it can be interpreted as a test of the null hypothesis that all the restrictions are valid.

Because constraints might not be independent either by construction or because of the data, the number of restrictions is not necessarily equal to the number of constraints. The rank of \( e(V) \) gives the number of parameters that were independently estimated after applying the constraints. The maximum number of parameters that can be estimated in an identified short-run or long-run SVAR is \( K(K+1)/2 \). This implies that the number of overidentifying restrictions, \( q \), is equal to \( K(K+1)/2 \) minus the rank of \( e(V) \).

The number of overidentifying restrictions is also linked to the order condition for each model. In a short-run SVAR model, there are \( 2K^2 \) parameters. Because no more than \( K(K+1)/2 \) parameters may be estimated, the order condition for a short-run SVAR model is that at least \( 2K^2 - K(K+1)/2 \) restrictions be placed on the model. Similarly, there are \( K^2 \) parameters in long-run SVAR model. Because no more than \( K(K+1)/2 \) parameters may be estimated, the order condition for a long-run SVAR model is that at least \( K^2 - K(K+1)/2 \) restrictions be placed on the model.

Acknowledgment

We thank Gianni Amisano of the Board of Governors of the Federal Reserve System for his helpful comments.

References


Becketti, S. 2013. *Introduction to Time Series Using Stata*. College Station, TX: Stata Press.


Also see

[TS] **var svar postestimation** — Postestimation tools for svar

[TS] **tsset** — Declare data to be time-series data

[TS] **var** — Vector autoregressive models

[TS] **var intro** — Introduction to vector autoregressive models

[TS] **varbasic** — Fit a simple VAR and graph IRFs or FEVDs

[TS] **vec** — Vector error-correction models

[U] **20 Estimation and postestimation commands**

*Stata Dynamic Stochastic General Equilibrium Models Reference Manual*
### Postestimation commands

The following postestimation commands are of special interest after `svar`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fcast compute</code></td>
<td>obtain dynamic forecasts</td>
</tr>
<tr>
<td><code>fcast graph</code></td>
<td>graph dynamic forecasts obtained from <code>fcast compute</code></td>
</tr>
<tr>
<td><code>irf</code></td>
<td>create and analyze IRFs and FEVDs</td>
</tr>
<tr>
<td><code>vargranger</code></td>
<td>Granger causality tests</td>
</tr>
<tr>
<td><code>varlmar</code></td>
<td>LM test for autocorrelation in residuals</td>
</tr>
<tr>
<td><code>varnorm</code></td>
<td>test for normally distributed residuals</td>
</tr>
<tr>
<td><code>varsoc</code></td>
<td>lag-order selection criteria</td>
</tr>
<tr>
<td><code>varstable</code></td>
<td>check stability condition of estimates</td>
</tr>
<tr>
<td><code>varwle</code></td>
<td>Wald lag-exclusion statistics</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>

797
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

predict [type] newvar [if] [in] [, statistic equation(eqno|eqname)]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>standard error of the linear prediction</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for the estimation sample.

Options for predict

xb, the default, calculates the linear prediction for the specified equation.

studp calculates the standard error of the linear prediction for the specified equation.

residuals calculates the residuals.

equation(eqno|eqname) specifies the equation to which you are referring.

equation() is filled in with one eqno or eqname for options xb, stdp, and residuals. For example, equation(#1) would mean that the calculation is to be made for the first equation, equation(#2) would mean the second, and so on. You could also refer to the equation by its name; thus, equation(income) would refer to the equation named income and equation(hours), to the equation named hours.

If you do not specify equation(), the results are the same as if you specified equation(#1).

For more information on using predict after multiple-equation commands, see [R] predict.

Remarks and examples

Remarks are presented under the following headings:

Model selection and inference
Forecasting
Model selection and inference

See the following sections for information on model selection and inference after `var`:

- [TS] `irf` — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
- [TS] `vargranger` — Pairwise Granger causality tests after `var` or `svar`
- [TS] `varlmar` — LM test for residual autocorrelation after `var` or `svar`
- [TS] `varnorm` — Test for normally distributed disturbances after `var` or `svar`
- [TS] `varsoc` — Obtain lag-order selection statistics for VARs and VECMs
- [TS] `varstable` — Check the stability condition of VAR or SVAR estimates
- [TS] `varwle` — Obtain Wald lag-exclusion statistics after `var` or `svar`

Forecasting

See the following sections for information on obtaining forecasts after `svar`:

- [TS] `fcast compute` — Compute dynamic forecasts after `var`, `svar`, or `vec`
- [TS] `fcast graph` — Graph forecasts after `fcast compute`

Also see

- [TS] `var svar` — Structural vector autoregressive models
- [U] 20 Estimation and postestimation commands
Description

`varbasic` fits a basic vector autoregressive (VAR) model and graphs the impulse–response functions (IRFs), the orthogonalized impulse–response functions (OIRFs), or the forecast-error variance decompositions (FEVDs).

Quick start

Fit vector autoregressive model for dependent variables $y_1$, $y_2$, and $y_3$ and their first and second lags, and graph orthogonalized impulse responses using `tsset` data

```
varbasic y1 y2 y3
```

As above, but include second and third lags instead of first and second lags

```
var y1 y2 y3, lags(2 3)
```

As above, but produce forecast-error variance decompositions instead of impulse responses

```
var y1 y2 y3, lags(2 3) fevd
```

As above, but set the forecast horizon for the forecast-error variance decompositions to be 12 periods

```
var y1 y2 y3, lags(2 3) fevd step(12)
```

Menu

Statistics > Multivariate time series > Basic VAR

Syntax

```
varbasic depvarlist [if] [in] [ , options ]
```

`options` Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lags</code></td>
<td>use lags <code>numlist</code> in the model; default is <code>lags(1 2)</code></td>
</tr>
<tr>
<td><code>irf</code></td>
<td>produce matrix graph of IRFs</td>
</tr>
<tr>
<td><code>fevd</code></td>
<td>produce matrix graph of FEVDs</td>
</tr>
<tr>
<td><code>nograph</code></td>
<td>do not produce a graph</td>
</tr>
<tr>
<td><code>step(#)</code></td>
<td>set forecast horizon <code>#</code> for estimating the OIRFs, IRFs, and FEVDs; default is <code>step(8)</code></td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `varbasic`; see [TS] `tsset`.

`depvarlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

`rolling`, `statsby`, and `xi` are allowed; see [U] 11.1.10 Prefix commands.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
**Options**

*lags*(numlist) specifies the lags to be included in the model. The default is *lags*(1 2). This option takes a numlist and not simply an integer for the maximum lag. For instance, *lags*(2) would include only the second lag in the model, whereas *lags*(1/2) would include both the first and second lags in the model. See [U] 11.1.8 numlist and [U] 11.4.4 Time-series varlists for more discussion of numlists and lags.

*irf* causes *varbasic* to produce a matrix graph of the IRFs instead of a matrix graph of the OIRFs, which is produced by default.

*fevd* causes *varbasic* to produce a matrix graph of the FEVDs instead of a matrix graph of the OIRFs, which is produced by default.

*nograph* specifies that no graph be produced. The IRFs, OIRFs, and FEVDs are still estimated and saved in the IRF file *varbasic.irf*.

*step(#)* specifies the forecast horizon for estimating the IRFs, OIRFs, and FEVDs. The default is eight periods.

**Remarks and examples**

*varbasic* simplifies fitting simple VARs and graphing the IRFs, the OIRFs, or the FEVDs. See [TS] var and [TS] var svar for fitting more advanced VAR models and structural vector autoregressive (SVAR) models. All the postestimation commands discussed in [TS] var postestimation work after *varbasic*.

This entry does not discuss the methods for fitting a VAR or the methods surrounding the IRFs, OIRFs, and FEVDs. See [TS] var and [TS] irf create for more on these methods. This entry illustrates how to use *varbasic* to easily obtain results. It also illustrates how *varbasic* serves as an entry point to further analysis.

> **Example 1**

We fit a three-variable VAR with two lags to the German macro data used by Lütkepohl (2005). The three variables are the first difference of natural log of investment, dln_inv; the first difference of the natural log of income, dln_inc; and the first difference of the natural log of consumption, dln_consump. In addition to fitting the VAR, we want to see the OIRFs. Below we use *varbasic* to fit a VAR(2) model on the data from the second quarter of 1961 through the fourth quarter of 1978. By default, *varbasic* produces graphs of the OIRFs.

. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. varbasic dln_inv dln_inc dln_consump if qtr<=tq(1978q4)

Vector autoregression

Sample: 1960q4 - 1978q4
Number of obs = 73

Log likelihood =  606.307
AIC = -16.03581
FPE =  2.18e-11
HQIC = -15.77323
Det(Sigma_ml) =  1.23e-11
SBIC = -15.37691

Equation Parms RMSE R-sq chi2 P>chi2

dln_inv 7 .046148 0.1286 10.76961 0.0958

dln_inc 7 .011719 0.1142 9.410683 0.1518

dln_consump 7 .009445 0.2513 24.50031 0.0004

| Coef. | Std. Err. | z | P>|z| | [95% Conf. Interval] |
|-------|-----------|---|-----|---------------------------|
| dln_inv | | | | |
| dln_inv | | | | |
| L1. | -.3196318 0.1192898 | -2.68 | 0.007 | -.5534355 -.0858282 |
| L2. | -.1605508 0.118767 | -1.35 | 0.176 | -.39333 .0722283 |
| dln_inc | | | | |
| dln_inc | | | | |
| L1. | .1459851 0.5188451 | 0.28 | 0.778 | -.8709326 1.162903 |
| L2. | .1146009 0.508295 | 0.23 | 0.822 | -.881639 1.110841 |
| dln_consump | | | | |
| dln_consump | | | | |
| L1. | .9612888 0.6316557 | 1.52 | 0.128 | -.2767936 2.199251 |
| L2. | .9344001 0.6324034 | 1.48 | 0.140 | -.3050877 2.173888 |
| _cons | | | | |
| _cons | | | | |
| -.0167221 0.0163796 | -1.02 | 0.307 | -.0488257 .0153814 |
| dln_inc | | | | |
| dln_inc | | | | |
| L1. | .0439309 0.0302933 | 1.45 | 0.147 | -.0154427 .1033046 |
| L2. | .0500302 0.0301605 | 1.66 | 0.097 | -.0090833 .1091437 |
| dln_consump | | | | |
| dln_consump | | | | |
| L1. | -.1527311 0.131759 | -1.16 | 0.246 | -.4109741 .1055118 |
| L2. | .0191634 0.1290799 | 0.15 | 0.882 | -.2338285 .2721552 |
| _cons | | | | |
| _cons | | | | |
| .0157672 0.0041596 | 3.79 | 0.000 | .0076146 .0239198 |
| dln_consump | | | | |
| dln_consump | | | | |
| L1. | -.002423 0.0244142 | -0.10 | 0.921 | -.050274 .045428 |
| L2. | .0338806 0.0243072 | 1.39 | 0.163 | -.0137607 .0815219 |
| dln_inc | | | | |
| dln_inc | | | | |
| L1. | .2248134 0.1061884 | 2.12 | 0.034 | .0166879 .4329389 |
| L2. | .3549135 0.1040292 | 3.41 | 0.001 | .1510199 .558807 |
| dln_consump | | | | |
| dln_consump | | | | |
| L1. | -.2639695 0.1292766 | -2.04 | 0.041 | -.517347 -.010592 |
| L2. | -.0222264 0.1294296 | -0.17 | 0.864 | -.2759039 .231451 |
| _cons | | | | |
| _cons | | | | |
| .0129258 0.0033523 | 3.86 | 0.000 | .0063554 .0194962 |
Because we are also interested in looking at the FEVDs, we can use \texttt{irf graph} to obtain the graphs. Although the details are available in \texttt{[TS] irf} and \texttt{[TS] irf graph}, the command below produces what we want after the call to \texttt{varbasic}.

\begin{verbatim}
. irf graph fevd, lstep(1)
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{irf_graphs.png}
\caption{Graphs by irfname, impulse variable, and response variable}
\end{figure}

\section*{Technical note}

Stata stores the estimated IRFs, OIRFs, and FEVDs in a IRF file called \texttt{_varbasic.irf} in the current working directory. \texttt{varbasic} replaces any \texttt{_varbasic.irf} that already exists. Finally, \texttt{varbasic} makes \texttt{_varbasic.irf} the active IRF file. This means that the graph and table commands \texttt{irf graph},
irf cgraph, irf ograph, irf table, and irf ctable will all display results that correspond to the VAR fit by varbasic.

Stored results

See Stored results in [TS] var.

Methods and formulas

varbasic uses var and irf graph to obtain its results. See [TS] var and [TS] irf graph for a discussion of how those commands obtain their results.

References


Also see

[TS] varbasic postestimation — Postestimation tools for varbasic
[TS] tsset — Declare data to be time-series data
[TS] var — Vector autoregressive models
[TS] var intro — Introduction to vector autoregressive models
[TS] var svar — Structural vector autoregressive models
[U] 20 Estimation and postestimation commands
Postestimation commands

The following postestimation commands are of special interest after `varbasic`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fcast compute</code></td>
<td>obtain dynamic forecasts</td>
</tr>
<tr>
<td><code>fcast graph</code></td>
<td>graph dynamic forecasts obtained from <code>fcast compute</code></td>
</tr>
<tr>
<td><code>irf</code></td>
<td>create and analyze IRFs and FEVDs</td>
</tr>
<tr>
<td><code>vargranger</code></td>
<td>Granger causality tests</td>
</tr>
<tr>
<td><code>varlmar</code></td>
<td>LM test for autocorrelation in residuals</td>
</tr>
<tr>
<td><code>varnorm</code></td>
<td>test for normally distributed residuals</td>
</tr>
<tr>
<td><code>varsoc</code></td>
<td>lag-order selection criteria</td>
</tr>
<tr>
<td><code>varstable</code></td>
<td>check stability condition of estimates</td>
</tr>
<tr>
<td><code>varwle</code></td>
<td>Wald lag-exclusion statistics</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>
predict

Description for predict

predict creates a new variable containing predictions such as linear predictions and residuals.

Menu for predict

Statistics > Postestimation

Syntax for predict

predict [type] newvar [if] [in] [, statistic equation(eqno|eqname)]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>standard error of the linear prediction</td>
</tr>
<tr>
<td>residuals</td>
<td>residuals</td>
</tr>
</tbody>
</table>

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for the estimation sample.

Options for predict

xb, the default, calculates the linear prediction for the specified equation.

stdp calculates the standard error of the linear prediction for the specified equation.

residuals calculates the residuals.

equation(eqno|eqname) specifies the equation to which you are referring.

equation() is filled in with one eqno or eqname for the xb, stdp, and residuals options. For example, equation(#1) would mean that the calculation is to be made for the first equation, equation(#2) would mean the second, and so on. You could also refer to the equation by its name; thus, equation(income) would refer to the equation named income and equation(hours), to the equation named hours.

If you do not specify equation(), the results are the same as if you specified equation(#1).

For more information on using predict after multiple-equation estimation commands, see [R] predict.
margins

Description for margins

margins estimates margins of response for linear predictions.

Menu for margins

Statistics  >  Postestimation

Syntax for margins

margins [marginlist] [, options]
margins [marginlist], predict(statistic ...) [options]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td>stdp</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

xb defaults to the first equation.

Statistics not allowed with margins are functions of stochastic quantities other than e(b).

For the full syntax, see [R] margins.
Remarks and examples

Example 1

All the postestimation commands discussed in [TS] var postestimation work after varbasic. Suppose that we are interested in testing the hypothesis that there is no autocorrelation in the VAR disturbances. Continuing example 1 from [TS] varbasic, we now use varlmar to test this hypothesis.

```
use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
varbasic dln_inv dln_inc dln_consump if qtr<=tq(1978q4)
(output omitted)
varlmar
```

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.5871</td>
<td>9</td>
<td>0.78043</td>
</tr>
<tr>
<td>2</td>
<td>6.3189</td>
<td>9</td>
<td>0.70763</td>
</tr>
</tbody>
</table>

H0: no autocorrelation at lag order

Because we cannot reject the null hypothesis of no autocorrelation in the residuals, this test does not indicate any model misspecification.

Also see

[TS] varbasic — Fit a simple VAR and graph IRFs or FEVDs
[U] 20 Estimation and postestimation commands
Description

vargranger performs a set of Granger causality tests for each equation in a VAR, providing a convenient alternative to test; see [R] test.

Quick start

Perform a Granger causality test after var or svar
vargranger

Perform a Granger causality test on vector autoregression estimation results stored as myest
vargranger, estimates(myest)

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > Granger causality tests
Syntax

```
vargranger [, estimates(estname) separator(#)]
```

vargranger can be used only after var or svar; see [TS] var and [TS] var svar.

Options

- `estimates(estname)`: requests that vargranger use the previously obtained set of var or svar estimates stored as `estname`. By default, vargranger uses the active results. See [R] estimates for information on manipulating estimation results.

- `separator(#)`: specifies how often separator lines should be drawn between rows. By default, separator lines appear every $K$ lines, where $K$ is the number of equations in the VAR under analysis. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on. `separator(0)` specifies that lines not appear in the table.

Remarks and examples

After fitting a VAR, we may want to know whether one variable “Granger-causes” another (Granger 1969). A variable $x$ is said to Granger-cause a variable $y$ if, given the past values of $y$, past values of $x$ are useful for predicting $y$. A common method for testing Granger causality is to regress $y$ on its own lagged values and on lagged values of $x$ and test the null hypothesis that the estimated coefficients on the lagged values of $x$ are jointly zero. Failure to reject the null hypothesis is equivalent to failing to reject the hypothesis that $x$ does not Granger-cause $y$.

For each equation and each endogenous variable that is not the dependent variable in that equation, vargranger computes and reports Wald tests that the coefficients on all the lags of an endogenous variable are jointly zero. For each equation in a VAR, vargranger tests the hypotheses that each of the other endogenous variables does not Granger-cause the dependent variable in that equation.

Because it may be interesting to investigate these types of hypotheses by using the VAR that underlies an SVAR, vargranger can also produce these tests by using the `e()` results from an svar. When vargranger uses svar `e()` results, the hypotheses concern the underlying var estimates.

See [TS] var and [TS] var svar for information about fitting VARS and SVARS in Stata. See Lütkepohl (2005), Hamilton (1994), and Amisano and Giannini (1997) for information about Granger causality and on VARS and SVARS in general.
Example 1: After var

Here we refit the model with German data described in [TS] var and then perform Granger causality tests with vargranger.

```
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk small
(output omitted)
. vargranger
Granger causality Wald tests
```

<table>
<thead>
<tr>
<th>Equation</th>
<th>Excluded</th>
<th>F</th>
<th>df</th>
<th>df_r</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>dln_inc</td>
<td>.04847</td>
<td>2</td>
<td>66</td>
<td>0.9527</td>
</tr>
<tr>
<td>dln_inv</td>
<td>dln_consump</td>
<td>1.5004</td>
<td>2</td>
<td>66</td>
<td>0.2306</td>
</tr>
<tr>
<td>dln_inv</td>
<td>ALL</td>
<td>1.5917</td>
<td>4</td>
<td>66</td>
<td>0.1869</td>
</tr>
<tr>
<td>dln_inc</td>
<td>dln_inv</td>
<td>1.7683</td>
<td>2</td>
<td>66</td>
<td>0.1786</td>
</tr>
<tr>
<td>dln_inc</td>
<td>dln_consump</td>
<td>1.7184</td>
<td>2</td>
<td>66</td>
<td>0.1873</td>
</tr>
<tr>
<td>dln_inc</td>
<td>ALL</td>
<td>1.9466</td>
<td>4</td>
<td>66</td>
<td>0.1130</td>
</tr>
<tr>
<td>dln_consump</td>
<td>dln_inv</td>
<td>.97147</td>
<td>2</td>
<td>66</td>
<td>0.3839</td>
</tr>
<tr>
<td>dln_consump</td>
<td>dln_inc</td>
<td>6.1465</td>
<td>2</td>
<td>66</td>
<td>0.0036</td>
</tr>
<tr>
<td>dln_consump</td>
<td>ALL</td>
<td>3.7746</td>
<td>4</td>
<td>66</td>
<td>0.0080</td>
</tr>
</tbody>
</table>

Because the estimates() option was not specified, vargranger used the active e() results. Consider the results of the three tests for the first equation. The first is a Wald test that the coefficients on the two lags of dln_inc that appear in the equation for dln_inv are jointly zero. The null hypothesis that dln_inc does not Granger-cause dln_inv cannot be rejected. Similarly, we cannot reject the null hypothesis that the coefficients on the two lags of dln_consump in the equation for dln_inv are jointly zero, so we cannot reject the hypothesis that dln_consump does not Granger-cause dln_inv. The third test is with respect to the null hypothesis that the coefficients on the two lags of all the other endogenous variables are jointly zero. Because this cannot be rejected, we cannot reject the null hypothesis that dln_inc and dln_consump, jointly, do not Granger-cause dln_inv.

Because we failed to reject most of these null hypotheses, we might be interested in imposing some constraints on the coefficients. See [TS] var for more on fitting VAR models with constraints on the coefficients.
Example 2: Using test instead of vargranger

We could have used `test` to compute these Wald tests, but `vargranger` saves a great deal of typing. Still, seeing how to use `test` to obtain the results reported by `vargranger` is useful.

```stata
. test [dln_inv]L.dln_inc [dln_inv]L2.dln_inc
 ( 1) [dln_inv]L.dln_inc = 0
 ( 2) [dln_inv]L2.dln_inc = 0
    F(  2,  66) =  0.05
    Prob > F =  0.9527

. test [dln_inv]L.dln_consump [dln_inv]L2.dln_consump, accumulate
 ( 1) [dln_inv]L.dln_inc = 0
 ( 2) [dln_inv]L2.dln_inc = 0
 ( 3) [dln_inv]L.dln_consump = 0
 ( 4) [dln_inv]L2.dln_consump = 0
    F(  4,  66) =  1.59
    Prob > F =  0.1869

. test [dln_inv]L.dln_inv [dln_inv]L2.dln_inv, accumulate
 ( 1) [dln_inv]L.dln_inc = 0
 ( 2) [dln_inv]L2.dln_inc = 0
 ( 3) [dln_inv]L.dln_consump = 0
 ( 4) [dln_inv]L2.dln_consump = 0
 ( 5) [dln_inv]L.dln_inv = 0
 ( 6) [dln_inv]L2.dln_inv = 0
    F(  6,  66) =  1.62
    Prob > F =  0.1547
```

The first two calls to `test` show how `vargranger` obtains its results. The first test reproduces the first test reported for the `dln_inv` equation. The second test reproduces the ALL entry for the first equation. The third test reproduces the standard $F$ statistic for the `dln_inv` equation, reported in the header of the `var` output in the previous example. The standard $F$ statistic also includes the lags of the dependent variable, as well as any exogenous variables in the equation. This illustrates that the test performed by `vargranger` of the null hypothesis that the coefficients on all the lags of all the other endogenous variables are jointly zero for a particular equation; that is, the ALL test is not the same as the standard $F$ statistic for that equation.
**Example 3: After svar**

When `vargranger` is run on `svar` estimates, the null hypotheses are with respect to the underlying `var` estimates. We run `vargranger` after using `svar` to fit an SVAR that has the same underlying VAR as our model in example 1.

```
. matrix A = (., 0,0 \ ., ., 0\ .,.,.)
. matrix B = I(3)
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk small aeq(A) beq(B)
   (output omitted)
. vargranger
```

Granger causality Wald tests

<table>
<thead>
<tr>
<th>Equation</th>
<th>Excluded</th>
<th>F</th>
<th>df</th>
<th>df_r</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>dln_inc</td>
<td>.04847</td>
<td>2</td>
<td>66</td>
<td>0.9527</td>
</tr>
<tr>
<td>dln_inv</td>
<td>dln_consump</td>
<td>1.5004</td>
<td>2</td>
<td>66</td>
<td>0.2306</td>
</tr>
<tr>
<td>dln_inventory</td>
<td>ALL</td>
<td>1.5917</td>
<td>4</td>
<td>66</td>
<td>0.1869</td>
</tr>
<tr>
<td>dln_inc</td>
<td>dln_inv</td>
<td>1.7683</td>
<td>2</td>
<td>66</td>
<td>0.1786</td>
</tr>
<tr>
<td>dln_inc</td>
<td>dln_consump</td>
<td>1.7184</td>
<td>2</td>
<td>66</td>
<td>0.1873</td>
</tr>
<tr>
<td>dln_inventory</td>
<td>ALL</td>
<td>1.9466</td>
<td>4</td>
<td>66</td>
<td>0.1130</td>
</tr>
<tr>
<td>dln_consump</td>
<td>dln_inv</td>
<td>.97147</td>
<td>2</td>
<td>66</td>
<td>0.3839</td>
</tr>
<tr>
<td>dln_consump</td>
<td>dln_inc</td>
<td>6.1465</td>
<td>2</td>
<td>66</td>
<td>0.0036</td>
</tr>
<tr>
<td>dln_consump</td>
<td>ALL</td>
<td>3.7746</td>
<td>4</td>
<td>66</td>
<td>0.0080</td>
</tr>
</tbody>
</table>

As we expected, the `vargranger` results are identical to those in the first example.

**Stored results**

`vargranger` stores the following in `r()`:

Matrices

- `r(gstats)` $\chi^2$, df, and p-values (if `e(small)=="n"`)  
- `r(gstats)` $F$, df, df_r, and p-values (if `e(small)!="n"`)

**Methods and formulas**

`vargranger` uses `test` to obtain Wald statistics of the hypothesis that all coefficients on the lags of variable $x$ are jointly zero in the equation for variable $y$. `vargranger` uses the `e()` results stored by `var` or `svar` to determine whether to calculate and report small-sample $F$ statistics or large-sample $\chi^2$ statistics.

---

**Clive William John Granger** (1934–2009) was born in Swansea, Wales, and earned degrees in mathematics and statistics at the University of Nottingham. Joining the staff there, he also worked at Princeton on the spectral analysis of economic time series before moving in 1974 to the University of California, San Diego. He was awarded the 2003 Nobel Prize in Economics for methods of analyzing economic time series with common trends (cointegration). He was knighted in 2005, thus becoming Sir Clive Granger.
References


Also see

[TS] var — Vector autoregressive models
[TS] var intro — Introduction to vector autoregressive models
[TS] var svar — Structural vector autoregressive models
[TS] varbasic — Fit a simple VAR and graph IRFs or FEVDs
varlmar — LM test for residual autocorrelation after var or svar

Description

varlmar implements a Lagrange multiplier (LM) test for autocorrelation in the residuals of VAR models, which was presented in Johansen (1995).

Quick start

Test the null hypothesis of no autocorrelation for the first two lags of the residuals after var or svar

varlmar

As above, but test the first 5 lags

varlmar, mlag(5)

Perform test for the first two lags of the residuals of a vector autoregression using estimation results stored in myest

varlmar, estimates(myest)

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > LM test for residual autocorrelation
Syntax

```
varlmar [, options]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlag(#)</td>
<td>use # for the maximum order of autocorrelation; default is mlag(2)</td>
</tr>
<tr>
<td>estimates(estname)</td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
<tr>
<td>separator(#)</td>
<td>draw separator line after every # rows</td>
</tr>
</tbody>
</table>

varlmar can be used only after var or svar; see [TS] var and [TS] var svar.
You must tsset your data before using varlmar; see [TS] tsset.

Options

- `mlag(#)` specifies the maximum order of autocorrelation to be tested. The integer specified in `mlag()` must be greater than 0; the default is 2.

- `estimates(estname)` requests that `varlmar` use the previously obtained set of var or svar estimates stored as `estname`. By default, `varlmar` uses the active results. See [R] estimates for information on manipulating estimation results.

- `separator(#)` specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Remarks and examples

Most postestimation analyses of VAR models and SVAR models assume that the disturbances are not autocorrelated. `varlmar` implements the LM test for autocorrelation in the residuals of a VAR model discussed in Johansen (1995, 21–22). The test is performed at lags $j = 1, \ldots, mlag()$. For each $j$, the null hypothesis of the test is that there is no autocorrelation at lag $j$.

`varlmar` uses the estimation results stored by var or svar. By default, `varlmar` uses the active estimation results. However, `varlmar` can use any previously stored var or svar estimation results specified in the `estimates()` option.
Example 1: After var

Here we refit the model with German data described in [TS] var and then call varlmar.

```
use https://www.stata-press.com/data/r16/lutkepohl2
(Quee, Quarterly SA West German macro data, Bi1 DM, from Lutkepohl 1993 Table E.1)

var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk
(output omitted)

varlmar, mlag(5)
```

Lagrange-multiplier test

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.5871</td>
<td>9</td>
<td>0.78043</td>
</tr>
<tr>
<td>2</td>
<td>6.3189</td>
<td>9</td>
<td>0.70763</td>
</tr>
<tr>
<td>3</td>
<td>8.4022</td>
<td>9</td>
<td>0.49418</td>
</tr>
<tr>
<td>4</td>
<td>11.8742</td>
<td>9</td>
<td>0.22049</td>
</tr>
<tr>
<td>5</td>
<td>5.2914</td>
<td>9</td>
<td>0.80821</td>
</tr>
</tbody>
</table>

H0: no autocorrelation at lag order

Because we cannot reject the null hypothesis that there is no autocorrelation in the residuals for any of the five orders tested, this test gives no hint of model misspecification. Although we fit the VAR with the dfk option to be consistent with the example in [TS] var, varlmar always uses the ML estimator of $\Sigma$. The results obtained from varlmar are the same whether or not dfk is specified.

Example 2: After svar

When varlmar is applied to estimation results produced by svar, the sequence of LM tests is applied to the underlying VAR. See [TS] var svar for a description of how an SVAR model builds on a VAR. In this example, we fit an SVAR that has an underlying VAR with two lags that is identical to the one fit in the previous example.

```
matrix A = (.,.,0\0,.,0\.,.,.)
matrix B = I(3)
svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk aeq(A) beq(B)
(output omitted)

varlmar, mlag(5)
```

Lagrange-multiplier test

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.5871</td>
<td>9</td>
<td>0.78043</td>
</tr>
<tr>
<td>2</td>
<td>6.3189</td>
<td>9</td>
<td>0.70763</td>
</tr>
<tr>
<td>3</td>
<td>8.4022</td>
<td>9</td>
<td>0.49418</td>
</tr>
<tr>
<td>4</td>
<td>11.8742</td>
<td>9</td>
<td>0.22049</td>
</tr>
<tr>
<td>5</td>
<td>5.2914</td>
<td>9</td>
<td>0.80821</td>
</tr>
</tbody>
</table>

H0: no autocorrelation at lag order

Because the underlying VAR(2) is the same as the previous example (we assure you that this is true), the output from varlmar is also the same.
Stored results

`varlmar` stores the following in `r()`:

Matrices

- `r(lm)` \( \chi^2 \), df, and \( p \)-values

Methods and formulas

The formula for the LM test statistic at lag \( j \) is

\[
LM_s = (T - d - 0.5) \ln \left( \frac{|\hat{\Sigma}|}{|\tilde{\Sigma}_s|} \right)
\]

where \( T \) is the number of observations in the VAR; \( d \) is explained below; \( \hat{\Sigma} \) is the maximum likelihood estimate of \( \Sigma \), the variance–covariance matrix of the disturbances from the VAR; and \( \tilde{\Sigma}_s \) is the maximum likelihood estimate of \( \Sigma \) from the following augmented VAR.

If there are \( K \) equations in the VAR, we can define \( e_t \) to be a \( K \times 1 \) vector of residuals. After we create the \( K \) new variables \( e_1, e_2, \ldots, e_K \) containing the residuals from the \( K \) equations, we can augment the original VAR with lags of these \( K \) new variables. For each lag \( s \), we form an augmented regression in which the new residual variables are lagged \( s \) times. Per the method of Davidson and MacKinnon (1993, 358), the missing values from these \( s \) lags are replaced with zeros. \( \tilde{\Sigma}_s \) is the maximum likelihood estimate of \( \Sigma \) from this augmented VAR, and \( d \) is the number of coefficients estimated in the augmented VAR. See [TS] var for a discussion of the maximum likelihood estimate of \( \Sigma \) in a VAR.

The asymptotic distribution of \( LM_s \) is \( \chi^2 \) with \( K^2 \) degrees of freedom.

References


Also see

[TS] var — Vector autoregressive models

[TS] var intro — Introduction to vector autoregressive models

[TS] var svar — Structural vector autoregressive models

[TS] varbasic — Fit a simple VAR and graph IRFs or FEVDs
**Description**

`varnorm` computes and reports a series of statistics against the null hypothesis that the disturbances in a VAR are normally distributed. For each equation, and for all equations jointly, up to three statistics may be computed: a skewness statistic, a kurtosis statistic, and the Jarque–Bera statistic. By default, all three statistics are reported.

**Quick start**

- Compute Jarque–Bera, skewness, and kurtosis statistics after `var` or `svar` to test the null hypothesis that the residuals are normally distributed
  
  `varnorm`

- As above, but only report the Jarque–Bera statistic
  
  `varnorm, jbera`

- After `svar`, use the Cholesky decomposition of the estimated variance–covariance matrix to compute the tests
  
  `varnorm, cholesky`

**Menu**

- Statistics > Multivariate time series > VAR diagnostics and tests > Test for normally distributed disturbances
Syntax

```
varnorm [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jbera</code></td>
<td>report Jarque–Bera statistic; default is to report all three statistics</td>
</tr>
<tr>
<td><code>skewness</code></td>
<td>report skewness statistic; default is to report all three statistics</td>
</tr>
<tr>
<td><code>kurtosis</code></td>
<td>report kurtosis statistic; default is to report all three statistics</td>
</tr>
<tr>
<td><code>estimates(estname)</code></td>
<td>use previously stored results <code>estname</code>; default is to use active results</td>
</tr>
<tr>
<td><code>cholesky</code></td>
<td>use Cholesky decomposition</td>
</tr>
<tr>
<td><code>separator(#)</code></td>
<td>draw separator line after every # rows</td>
</tr>
</tbody>
</table>

`varnorm` can be used only after `var` or `svar`; see [TS] `var` and [TS] `var svar`. You must `tsset` your data before using `varnorm`; see [TS] `tsset`.

Options

- **`jbera`**: requests that the Jarque–Bera statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.
- **`skewness`**: requests that the skewness statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.
- **`kurtosis`**: requests that the kurtosis statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.
- **`estimates(estname)`**: specifies that `varnorm` use the previously obtained set of `var` or `svar` estimates stored as `estname`. By default, `varnorm` uses the active results. See [R] `estimates` for information on manipulating estimation results.
- **`cholesky`**: specifies that `varnorm` use the Cholesky decomposition of the estimated variance–covariance matrix of the disturbances, $\hat{\Sigma}$, to orthogonalize the residuals when `varnorm` is applied to `svar` results. By default, when `varnorm` is applied to `svar` results, it uses the estimated structural decomposition $\hat{\Lambda}^{-1}\hat{B}$ on $\hat{C}$ to orthogonalize the residuals. When applied to `var e()` results, `varnorm` always uses the Cholesky decomposition of $\hat{\Sigma}$. For this reason, the `cholesky` option may not be specified when using `var` results.
- **`separator(#)`**: specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Remarks and examples

Some of the postestimation statistics for `VAR` and `SVAR` assume that the $K$ disturbances have a $K$-dimensional multivariate normal distribution. `varnorm` uses the estimation results produced by `var` or `svar` to produce a series of statistics against the null hypothesis that the $K$ disturbances in the `VAR` are normally distributed.

Per the notation in Lütkepohl (2005), call the skewness statistic $\hat{\lambda}_1$, the kurtosis statistic $\hat{\lambda}_2$, and the Jarque–Bera statistic $\hat{\lambda}_3$. The Jarque–Bera statistic is a combination of the other two statistics. The single-equation results are from tests against the null hypothesis that the disturbance for that
particular equation is normally distributed. The results for all the equations are from tests against the null hypothesis that the $K$ disturbances follow a $K$-dimensional multivariate normal distribution. Failure to reject the null hypothesis indicates a lack of model misspecification.

> Example 1: After var

We refit the model with German data described in [TS] var and then call varnorm.

```
use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)

var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk
(output omitted)

varnorm
```

<table>
<thead>
<tr>
<th>Equation</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>2.821</td>
<td>2</td>
<td>0.24397</td>
</tr>
<tr>
<td>dln_inc</td>
<td>3.450</td>
<td>2</td>
<td>0.17817</td>
</tr>
<tr>
<td>dln_consump</td>
<td>1.566</td>
<td>2</td>
<td>0.45702</td>
</tr>
<tr>
<td>ALL</td>
<td>7.838</td>
<td>6</td>
<td>0.25025</td>
</tr>
</tbody>
</table>

Skewness test

<table>
<thead>
<tr>
<th>Equation</th>
<th>Skewness</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>0.11935</td>
<td>0.173</td>
<td>1</td>
<td>0.67718</td>
</tr>
<tr>
<td>dln_inc</td>
<td>-0.38316</td>
<td>1.786</td>
<td>1</td>
<td>0.18139</td>
</tr>
<tr>
<td>dln_consump</td>
<td>-0.31275</td>
<td>1.190</td>
<td>1</td>
<td>0.27532</td>
</tr>
<tr>
<td>ALL</td>
<td>3.150</td>
<td>3</td>
<td>0.36913</td>
<td></td>
</tr>
</tbody>
</table>

Kurtosis test

<table>
<thead>
<tr>
<th>Equation</th>
<th>Kurtosis</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>3.9331</td>
<td>2.648</td>
<td>1</td>
<td>0.10367</td>
</tr>
<tr>
<td>dln_inc</td>
<td>3.7396</td>
<td>1.664</td>
<td>1</td>
<td>0.19710</td>
</tr>
<tr>
<td>dln_consump</td>
<td>2.6484</td>
<td>0.376</td>
<td>1</td>
<td>0.53973</td>
</tr>
<tr>
<td>ALL</td>
<td>4.688</td>
<td>3</td>
<td>0.19613</td>
<td></td>
</tr>
</tbody>
</table>

dfk estimator used in computations

In this example, neither the single-equation Jarque–Bera statistics nor the joint Jarque–Bera statistic come close to rejecting the null hypothesis.

The skewness and kurtosis results have similar structures.

The Jarque–Bera results use the sum of the skewness and kurtosis statistics. The skewness and kurtosis results are based on the skewness and kurtosis coefficients, respectively. See Methods and formulas.

> Example 2: After svar

The test statistics are computed on the orthogonalized VAR residuals; see Methods and formulas. When varnorm is applied to var results, varnorm uses a Cholesky decomposition of the estimated variance–covariance matrix of the disturbances, $\hat{\Sigma}$, to orthogonalize the residuals.
By default, when `varnorm` is applied to `svar` estimation results, it uses the estimated structural decomposition $\hat{A}^{-1}\hat{B}$ on $\hat{C}$ to orthogonalize the residuals of the underlying VAR. Alternatively, when `varnorm` is applied to `svar` results and the `cholesky` option is specified, `varnorm` uses the Cholesky decomposition of $\hat{\Sigma}$ to orthogonalize the residuals of the underlying VAR.

We fit an SVAR that is based on an underlying VAR with two lags that is the same as the one fit in the previous example. We impose a structural decomposition that is the same as the Cholesky decomposition, as illustrated in [TS] `var svar`.

```
. matrix a = (.,0,0\.,0\.,.)
. matrix b = I(3)
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk aeq(a) beq(b)
     (output omitted)
. varnorm
```

**Jarque-Bera test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>2.821</td>
<td>2</td>
<td>0.24397</td>
</tr>
<tr>
<td>dln_inc</td>
<td>3.450</td>
<td>2</td>
<td>0.17817</td>
</tr>
<tr>
<td>dln_consump</td>
<td>1.566</td>
<td>2</td>
<td>0.46702</td>
</tr>
<tr>
<td>ALL</td>
<td>7.838</td>
<td>6</td>
<td>0.25025</td>
</tr>
</tbody>
</table>

**Skewness test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Skewness</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>.11935</td>
<td>0.173</td>
<td>1</td>
<td>0.67718</td>
</tr>
<tr>
<td>dln_inc</td>
<td>-.38316</td>
<td>1.786</td>
<td>1</td>
<td>0.18139</td>
</tr>
<tr>
<td>dln_consump</td>
<td>-.31275</td>
<td>1.190</td>
<td>1</td>
<td>0.27532</td>
</tr>
<tr>
<td>ALL</td>
<td>3.150</td>
<td>3</td>
<td>0.36913</td>
<td></td>
</tr>
</tbody>
</table>

**Kurtosis test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Kurtosis</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dln_inv</td>
<td>3.9331</td>
<td>2.648</td>
<td>1</td>
<td>0.10367</td>
</tr>
<tr>
<td>dln_inc</td>
<td>3.7396</td>
<td>1.664</td>
<td>1</td>
<td>0.19710</td>
</tr>
<tr>
<td>dln_consump</td>
<td>2.6484</td>
<td>0.376</td>
<td>1</td>
<td>0.53973</td>
</tr>
<tr>
<td>ALL</td>
<td>4.688</td>
<td>3</td>
<td>0.19613</td>
<td></td>
</tr>
</tbody>
</table>

dfk estimator used in computations

Because the estimated structural decomposition is the same as the Cholesky decomposition, the `varnorm` results are the same as those from the previous example.

---

**Technical note**

The statistics computed by `varnorm` depend on $\hat{\Sigma}$, the estimated variance–covariance matrix of the disturbances. `var` uses the maximum likelihood estimator of this matrix by default, but the `dfk` option produces an estimator that uses a small-sample correction. Thus specifying `dfk` in the call to `var` or `svar` will affect the test results produced by `varnorm`. **
Stored results

`varnorm` stores the following in `r()`:

- **Macros**
  - `r(dfk)` dfk, if specified

- **Matrices**
  - `r(kurtosis)` kurtosis test, df, and p-values
  - `r(skewness)` skewness test, df, and p-values
  - `r(jb)` Jarque–Bera test, df, and p-values

Methods and formulas

`varnorm` is based on the derivations found in Lütkepohl (2005, 174–181). Let \( \hat{u}_t \) be the \( K \times 1 \) vector of residuals from the \( K \) equations in a previously fitted VAR or the residuals from the \( K \) equations of the VAR underlying a previously fitted SVAR. Similarly, let \( \hat{\Sigma} \) be the estimated covariance matrix of the disturbances. (Note that \( \hat{\Sigma} \) depends on whether the `dfk` option was specified.) The skewness, kurtosis, and Jarque–Bera statistics must be computed using the orthogonalized residuals.

Because

\[
\hat{\Sigma} = \hat{P}\hat{P}'
\]

implies that

\[
\hat{P}^{-1}\hat{\Sigma}\hat{P}^{-1'} = I_K
\]

premultiplying \( \hat{u}_t \) by \( \hat{P} \) is one way of performing the orthogonalization. When `varnorm` is applied to `var` results, \( \hat{P} \) is defined to be the Cholesky decomposition of \( \hat{\Sigma} \). When `varnorm` is applied to `svar` results, \( \hat{P} \) is set, by default, to the estimated structural decomposition; that is, \( \hat{P} = \hat{A}^{-1}\hat{B} \), where \( \hat{A} \) and \( \hat{B} \) are the `svar` estimates of the \( A \) and \( B \) matrices, or \( \hat{C} \), where \( \hat{C} \) is the long-run SVAR estimation of \( C \). (See [TS] var svar for more on the origin and estimation of the \( A \) and \( B \) matrices.) When `varnorm` is applied to `svar` results and the `cholesky` option is specified, \( \hat{P} \) is set to the Cholesky decomposition of \( \hat{\Sigma} \).

Define \( \hat{w}_t \) to be the orthogonalized VAR residuals given by

\[
\hat{w}_t = (\hat{w}_{1t}, \ldots, \hat{w}_{Kt})' = \hat{P}^{-1}\hat{u}_t
\]

The \( K \times 1 \) vectors of skewness and kurtosis coefficients are then computed using the orthogonalized residuals by

\[
\hat{b}_1 = (\hat{b}_{11}, \ldots, \hat{b}_{K1})'; \quad \hat{b}_{k1} = \frac{1}{T} \sum_{i=1}^{T} \hat{w}_{3kt}^3
\]

\[
\hat{b}_2 = (\hat{b}_{12}, \ldots, \hat{b}_{K2})'; \quad \hat{b}_{k2} = \frac{1}{T} \sum_{i=1}^{T} \hat{w}_{4kt}^4
\]

Under the null hypothesis of multivariate Gaussian disturbances,

\[
\hat{\lambda}_1 = \frac{T\hat{b}_1'\hat{b}_1}{6} \xrightarrow{d} \chi^2(K)
\]
\[ \hat{\lambda}_2 = \frac{T(\hat{b}_2 - 3)'(\hat{b}_2 - 3)}{24} \overset{d}{\rightarrow} \chi^2(K) \]

and
\[ \hat{\lambda}_3 = \hat{\lambda}_1 + \hat{\lambda}_2 \overset{d}{\rightarrow} \chi^2(2K) \]

\( \hat{\lambda}_1 \) is the skewness statistic, \( \hat{\lambda}_2 \) is the kurtosis statistic, and \( \hat{\lambda}_3 \) is the Jarque–Bera statistic.

\( \hat{\lambda}_1, \hat{\lambda}_2, \) and \( \hat{\lambda}_3 \) are for tests of the null hypothesis that the \( K \times 1 \) vector of disturbances follows a multivariate normal distribution. The corresponding statistics against the null hypothesis that the disturbances from the \( k \)th equation come from a univariate normal distribution are

\[ \hat{\lambda}_{1k} = \frac{T \hat{b}_{k1}^2}{6} \overset{d}{\rightarrow} \chi^2(1) \]

\[ \hat{\lambda}_{2k} = \frac{T (\hat{b}_{k2}^2 - 3)^2}{24} \overset{d}{\rightarrow} \chi^2(1) \]

and
\[ \hat{\lambda}_{3k} = \hat{\lambda}_1 + \hat{\lambda}_2 \overset{d}{\rightarrow} \chi^2(2) \]

References


Also see

[TS] var — Vector autoregressive models

[TS] var intro — Introduction to vector autoregressive models

[TS] var svar — Structural vector autoregressive models

[TS] varbasic — Fit a simple VAR and graph IRFs or FEVDs
varsoc — Obtain lag-order selection statistics for VARs and VECMs

Description

varsoc reports the final prediction error (FPE), Akaike’s information criterion (AIC), Schwarz’s Bayesian information criterion (SBIC), and the Hannan and Quinn information criterion (HQIC) lag-order selection statistics for a series of vector autoregressions of order 1 through a requested maximum lag. A sequence of likelihood-ratio test statistics for all the full VARs of order less than or equal to the highest lag order is also reported.

varsoc can be used as a preestimation or a postestimation command. The preestimation version can be used to select the lag order for a VAR or vector error-correction model (VECM). The postestimation version obtains the information needed to compute the statistics from the previous model or specified stored estimates.

Quick start

Compute AIC, SBIC, and HQIC, and final prediction error to aid in the lag-order selection before VAR or VECM estimation of y1 and y2 using tsset data

varsoc y1 y2

As above, but set the maximum lag order to be tested to 7

varsoc y1 y2, maxlag(7)

As above, but use Lütkepohl’s version of the information criteria

varsoc y1 y2, maxlag(7) lutstats

Menu

Preestimation for VARs
Statistics > Multivariate time series > VAR diagnostics and tests > Lag-order selection statistics (preestimation)

Postestimation for VARs
Statistics > Multivariate time series > VAR diagnostics and tests > Lag-order selection statistics (postestimation)

Preestimation for VECMs
Statistics > Multivariate time series > VEC diagnostics and tests > Lag-order selection statistics (preestimation)

Postestimation for VECMs
Statistics > Multivariate time series > VEC diagnostics and tests > Lag-order selection statistics (postestimation)

825
Syntax

Preestimation syntax

```
varsoc depvarlist [if] [in] [ , preestimation_options ]
```

Postestimation syntax

```
varsoc [ , estimates(estname) ]
```

**preestimation_options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
</tr>
<tr>
<td><code>maxlag(#)</code> set maximum lag order to #; default is <code>maxlag(4)</code></td>
</tr>
<tr>
<td><code>exog(varlist)</code> use <code>varlist</code> as exogenous variables</td>
</tr>
<tr>
<td><code>constraints(constraints)</code> apply constraints to exogenous variables</td>
</tr>
<tr>
<td><code>noconstant</code> suppress constant term</td>
</tr>
<tr>
<td><code>lutstats</code> use Lütkepohl’s version of information criteria</td>
</tr>
<tr>
<td><code>level(#)</code> set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td><code>separator(#)</code> draw separator line after every # rows</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `varsoc`; see [TS] `tsset`. by is allowed with the preestimation version of `varsoc`; see [U] 11.1.10 Prefix commands.

**Preestimation options**

- `maxlag(#)` specifies the maximum lag order for which the statistics are to be obtained.
- `exog(varlist)` specifies exogenous variables to include in the VARs fit by `varsoc`.
- `constraints(constraints)` specifies a list of constraints on the exogenous variables to be applied. Do not specify constraints on the lags of the endogenous variables because specifying one would mean that at least one of the VAR models considered by `varsoc` will not contain the lag specified in the constraint. Use `var` directly to obtain selection-order criteria with constraints on lags of the endogenous variables.
- `noconstant` suppresses the constant terms from the model. By default, constant terms are included.
- `lutstats` specifies that the Lütkepohl (2005) versions of the information criteria be reported. See Methods and formulas for a discussion of these statistics.
- `level(#)` specifies the confidence level, as a percentage, that is used to identify the first likelihood-ratio test that rejects the null hypothesis that the additional parameters from adding a lag are jointly zero. The default is `level(95)` or as set by `set level`; see [U] 20.8 Specifying the width of confidence intervals.
- `separator(#)` specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.
Postestimation option

estimates(estname) specifies the name of a previously stored set of var or svar estimates. When no depvarlist is specified, varsoc uses the postestimation syntax and uses the currently active estimation results or the results specified in estimates(estname). See [R] estimates for information on manipulating estimation results.

Remarks and examples

Many selection-order statistics have been developed to assist researchers in fitting a VAR of the correct order. Several of these selection-order statistics appear in the [TS] var output. The varsoc command computes these statistics over a range of lags $p$ while maintaining a common sample and option specification.

varsoc can be used as a preestimation or a postestimation command. When it is used as a preestimation command, a depvarlist is required, and the default maximum lag is 4. When it is used as a postestimation command, varsoc uses the model specification stored in estname or the previously fitted model.

varsoc computes four information criteria as well as a sequence of likelihood ratio (LR) tests. The information criteria include the FPE, AIC, the HQIC, and SBIC.

For a given lag $p$, the LR test compares a VAR with $p$ lags with one with $p-1$ lags. The null hypothesis is that all the coefficients on the $p$th lags of the endogenous variables are zero. To use this sequence of LR tests to select a lag order, we start by looking at the results of the test for the model with the most lags, which is at the bottom of the table. Proceeding up the table, the first test that rejects the null hypothesis is the lag order selected by this process. See Lütkepohl (2005, 143–144) for more information on this procedure. An ‘*’ appears next to the LR statistic indicating the optimal lag.

For the remaining statistics, the lag with the smallest value is the order selected by that criterion. An ‘*’ indicates the optimal lag. Strictly speaking, the FPE is not an information criterion, though we include it in this discussion because, as with an information criterion, we select the lag length corresponding to the lowest value; and, naturally, we want to minimize the prediction error. The AIC measures the discrepancy between the given model and the true model, which, of course, we want to minimize. Amemiya (1985) provides an intuitive discussion of the arguments in Akaike (1973). The SBIC and the HQIC can be interpreted similarly to the AIC, though the SBIC and the HQIC have a theoretical advantage over the AIC and the FPE. As Lütkepohl (2005, 148–152) demonstrates, choosing $p$ to minimize the SBIC or the HQIC provides consistent estimates of the true lag order, $p$. In contrast, minimizing the AIC or the FPE will overestimate the true lag order with positive probability, even with an infinite sample size.

Although VAR models assume that the modulus is strictly less than 1 (see [TS] varstable), VECMs do not need to satisfy this condition, and they work even if all the variables included in the model are integrated of order 1, I(1). Regardless of these differences, varsoc works for both estimation commands. As shown by Nielsen (2001), the lag-order selection statistics discussed above can be used in the presence of I(1) variables.
Example 1: Preestimation

Here we use `varsoc` as a preestimation command.

```stata
.use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
.use https://www.stata-press.com/data/r16/lutkepohl2
.varsoc dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lutstats
Selection-order criteria (lutstats)
Sample: 1961q2 - 1978q4 Number of obs = 71

<table>
<thead>
<tr>
<th>lag</th>
<th>LL</th>
<th>LR</th>
<th>df</th>
<th>p</th>
<th>FPE</th>
<th>AIC</th>
<th>HQIC</th>
<th>SBIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>564.784</td>
<td>2.7e-11</td>
<td>-24.423</td>
<td>-24.423*</td>
<td>-24.423*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>576.409</td>
<td>23.249</td>
<td>9</td>
<td>0.006</td>
<td>2.5e-11</td>
<td>-24.497</td>
<td>-24.3829</td>
<td>-24.2102</td>
</tr>
<tr>
<td>2</td>
<td>588.859</td>
<td>24.901*</td>
<td>9</td>
<td>0.003</td>
<td>2.3e-11*</td>
<td>-24.5942*</td>
<td>-24.3661</td>
<td>-24.0205</td>
</tr>
<tr>
<td>3</td>
<td>591.237</td>
<td>4.7566</td>
<td>9</td>
<td>0.855</td>
<td>2.7e-11</td>
<td>-24.4076</td>
<td>-24.0655</td>
<td>-23.5472</td>
</tr>
<tr>
<td>4</td>
<td>598.457</td>
<td>14.438</td>
<td>9</td>
<td>0.108</td>
<td>2.9e-11</td>
<td>-23.3575</td>
<td>-23.9012</td>
<td>-23.2102</td>
</tr>
</tbody>
</table>
```

Endogenous: dln_inv dln_inc dln_consump
Exogenous: _cons

The sample used begins in 1961q2 because all the VARs are fit to the sample defined by any if or in conditions and the available data for the maximum lag specified. The default maximum number of lags is four. Because we specified the `lutstats` option, the table contains the Lütkepohl (2005) versions of the information criteria, which differ from the standard definitions in that they drop the constant term from the log likelihood. In this example, the likelihood-ratio tests selected a model with two lags. AIC and FPE have also both chosen a model with two lags, whereas SBIC and HQIC have both selected a model with zero lags.

Example 2: Postestimation

`varsoc` works as a postestimation command when no dependent variables are specified.

```stata
.var dln_inc dln_consump if qtr<=tq(1978q4), lutstats exog(l.dln_inv)
(output omitted)
.varsoc
Selection-order criteria (lutstats)
Sample: 1960q4 - 1978q4 Number of obs = 73

<table>
<thead>
<tr>
<th>lag</th>
<th>LL</th>
<th>LR</th>
<th>df</th>
<th>p</th>
<th>FPE</th>
<th>AIC</th>
<th>HQIC</th>
<th>SBIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>460.646</td>
<td>1.3e-08</td>
<td>-18.2962</td>
<td>-18.2962*</td>
<td>-18.2962*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>467.606</td>
<td>13.919</td>
<td>4</td>
<td>0.008</td>
<td>1.2e-08</td>
<td>-18.3773</td>
<td>-18.3273</td>
<td>-18.2518</td>
</tr>
<tr>
<td>2</td>
<td>477.087</td>
<td>18.962*</td>
<td>4</td>
<td>0.001</td>
<td>1.0e-08*</td>
<td>-18.5275*</td>
<td>-18.4274*</td>
<td>-18.2764</td>
</tr>
</tbody>
</table>
```

Endogenous: dln_inc dln_consump
Exogenous: L.dln_inv _cons

Because we included one lag of dln_inv in our original model, `varsoc` did likewise with each model it fit.

Based on the work of Tsay (1984), Paulsen (1984), and Nielsen (2001), these lag-order selection criteria can be used to determine the lag length of the VAR underlying a VECM. See [TS] vec intro for an example in which we use `varsoc` to choose the lag order for a VECM.
varsoc — Obtain lag-order selection statistics for VARs and VECMs

Stored results

varsoc stores the following in r():

Scalars
- \( r(N) \) number of observations
- \( r(t_{\text{max}}) \) last time period in sample
- \( r(t_{\text{min}}) \) first time period in sample
- \( r(m_{\text{lag}}) \) maximum lag order
- \( r(N_{\text{gaps}}) \) the number of gaps in the sample

Macros
- \( r(\text{endog}) \) names of endogenous variables
- \( r(\text{luts}t_{\text{stats}}) \) lutstats, if specified
- \( r(\text{cns}#) \) the \#th constraint
- \( r(\text{exog}) \) names of exogenous variables
- \( r(\text{rmluts}t_{\text{stats}}) \) rmlutstats, if specified

Matrices
- \( r(\text{stats}) \) LL, LR, FPE, AIC, HQIC, SBIC, and \( p \)-values

Methods and formulas

Methods and formulas are presented under the following headings:

- Likelihood-ratio statistic
- Model-order statistics
- Lutstats

Likelihood-ratio statistic

As shown by Hamilton (1994, 295–296), the log likelihood for a VAR\((p)\) is

\[
LL = \left( \frac{T}{2} \right) \left\{ \ln(|\hat{\Sigma}^{-1}|) - K \ln(2\pi) - K \right\}
\]

where \( T \) is the number of observations, \( K \) is the number of equations, and \( \hat{\Sigma} \) is the maximum likelihood estimate of \( E[u_t u_t'] \), where \( u_t \) is the \( K \times 1 \) vector of disturbances. Because

\[
\ln(|\hat{\Sigma}^{-1}|) = -\ln(|\hat{\Sigma}|)
\]

the log likelihood can be rewritten as

\[
LL = -\left( \frac{T}{2} \right) \left\{ \ln(|\hat{\Sigma}|) + K \ln(2\pi) + K \right\}
\]

Letting \( LL(j) \) be the value of the log likelihood with \( j \) lags yields the LR statistic for lag order \( j \) as

\[
LR(j) = 2\{LL(j) - LL(j - 1)\}
\]
Model-order statistics

The formula for the FPE given in Lütkepohl (2005, 147) is

\[ \text{FPE} = |\Sigma_u| \left( \frac{T + Kp + 1}{T - Kp - 1} \right)^K \]

This formula, however, assumes that there is a constant in the model and that none of the variables are dropped because of collinearity. To deal with these problems, the FPE is implemented as

\[ \text{FPE} = |\Sigma_u| \left( \frac{T + \bar{m}}{T - \bar{m}} \right)^K \]

where \( \bar{m} \) is the average number of parameters over the \( K \) equations. This implementation accounts for variables dropped because of collinearity.

By default, the AIC, SBIC, and HQIC are computed according to their standard definitions, which include the constant term from the log likelihood. That is,

\[
\text{AIC} = -2 \left( \frac{\text{LL}}{T} \right) + \frac{2t_p}{T} \\
\text{SBIC} = -2 \left( \frac{\text{LL}}{T} \right) + \ln \left( \frac{T}{T} \right) t_p \\
\text{HQIC} = -2 \left( \frac{\text{LL}}{T} \right) + \frac{2\ln \{\ln (T)\}}{T} t_p
\]

where \( t_p \) is the total number of parameters in the model and \( \text{LL} \) is the log likelihood.

Lutstats

Lütkepohl (2005) advocates dropping the constant term from the log likelihood because it does not affect inference. The Lütkepohl versions of the information criteria are

\[
\text{AIC} = \ln (|\Sigma_u|) + \frac{2pK^2}{T} \\
\text{SBIC} = \ln (|\Sigma_u|) + \frac{\ln (T)}{T} pK^2 \\
\text{HQIC} = \ln (|\Sigma_u|) + \frac{2\ln \{\ln (T)\}}{T} pK^2
\]

References


Also see

[Ts] **var** — Vector autoregressive models

[Ts] **var intro** — Introduction to vector autoregressive models

[Ts] **var svar** — Structural vector autoregressive models

[Ts] **varbasic** — Fit a simple VAR and graph IRFs or FEVDs

[Ts] **vec** — Vector error-correction models

[Ts] **vec intro** — Introduction to vector error-correction models
Title

**varstable — Check the stability condition of VAR or SVAR estimates**

<table>
<thead>
<tr>
<th>Description</th>
<th>Quick start</th>
<th>Menu</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Options

- Remarks and examples
- Also see

References

- Also see

---

**Description**

`varstable` checks the eigenvalue stability condition after estimating the parameters of a vector autoregression using `var` or `svar`.

**Quick start**

Check eigenvalue stability condition after `var` or `svar`

`varstable`

As above, and graph the eigenvalues of the companion matrix

`varstable, graph`

As above, and label each eigenvalue with its distance from the unit circle

`varstable, graph dlabel`

As above, but label the eigenvalues with their moduli

`varstable, graph modlabel`

**Menu**

Statistics > Multivariate time series > VAR diagnostics and tests > Check stability condition of VAR estimates
Syntax

```
varstable [, options]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td></td>
</tr>
<tr>
<td>estimates(estname)</td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
<tr>
<td>amat(matrix_name)</td>
<td>save the companion matrix as matrix_name</td>
</tr>
<tr>
<td>graph</td>
<td>graph eigenvalues of the companion matrix</td>
</tr>
<tr>
<td>dlabel</td>
<td>label eigenvalues with the distance from the unit circle</td>
</tr>
<tr>
<td>modlabel</td>
<td>label eigenvalues with the modulus</td>
</tr>
<tr>
<td>marker_options</td>
<td>change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td>rlopts(cline_options)</td>
<td>affect rendition of reference unit circle</td>
</tr>
<tr>
<td>nogrid</td>
<td>suppress polar grid circles</td>
</tr>
<tr>
<td>pgrid([...])</td>
<td>specify radii and appearance of polar grid circles; see Options for details</td>
</tr>
<tr>
<td><strong>Add plots</strong></td>
<td></td>
</tr>
<tr>
<td>addplot(plot)</td>
<td>add other plots to the generated graph</td>
</tr>
</tbody>
</table>

Y axis, X axis, Titles, Legend, Overall

twoway_options any options other than by() documented in [G-3] twoway_options

varstable can be used only after var or svar; see [TS] var and [TS] var svar.

Options

**Main**

*estimates(estname)* requests that *varstable* use the previously obtained set of var estimates stored as estname. By default, *varstable* uses the active estimation results. See [R] estimates for information on manipulating estimation results.

*amat(matrix_name)* specifies a valid Stata matrix name by which the companion matrix A can be saved (see Methods and formulas for the definition of the matrix A). The default is not to save the A matrix.

*graph* causes *varstable* to draw a graph of the eigenvalues of the companion matrix.

*dlabel* labels each eigenvalue with its distance from the unit circle. *dlabel* cannot be specified with *modlabel*.

*modlabel* labels the eigenvalues with their moduli. *modlabel* cannot be specified with *dlabel*.

*marker_options* specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] marker_options.

*rlopts(cline_options)* affect the rendition of the reference unit circle; see [G-3] cline_options.

*nogrid* suppresses the polar grid circles.

*pgrid([numlist] [ , line_options ])* determines the radii and appearance of the polar grid circles. By default, the graph includes nine polar grid circles with radii 0.1, 0.2, ..., 0.9 that have the grid line style. The *numlist* specifies the radii for the polar grid circles. The *line_options* determine the appearance of the polar grid circles; see [G-3] line_options. Because the *pgrid()* option can be repeated, circles with different radii can have distinct appearances.
Add plots

\texttt{addplot(plot)} adds specified plots to the generated graph. See [G-3] \texttt{addplot_option}.

\texttt{Y axis, X axis, Titles, Legend, Overall}

twoway_options are any of the options documented in [G-3] \texttt{twoway_options}, except by(). These include options for titling the graph (see [G-3] \texttt{title_options}) and for saving the graph to disk (see [G-3] \texttt{saving_option}).

Remarks and examples

Inference after \texttt{var} and \texttt{svar} requires that variables be covariance stationary. The variables in $y_t$ are covariance stationary if their first two moments exist and are independent of time. More explicitly, a variable $y_t$ is covariance stationary if

1. $E[y_t]$ is finite and independent of $t$.
2. $\text{Var}[y_t]$ is finite and independent of $t$
3. $\text{Cov}[y_t, y_s]$ is a finite function of $|t - s|$ but not of $t$ or $s$ alone.

Interpretation of VAR models, however, requires that an even stricter stability condition be met. If a VAR is stable, it is invertible and has an infinite-order vector moving-average representation. If the VAR is stable, impulse–response functions and forecast-error variance decompositions have known interpretations.

Lütkepohl (2005) and Hamilton (1994) both show that if the modulus of each eigenvalue of the matrix $A$ is strictly less than one, the estimated VAR is stable (see \texttt{Methods and formulas} for the definition of the matrix $A$).

\textbf{Example 1}

After fitting a VAR with \texttt{var}, we can use \texttt{varstable} to check the stability condition. Using the same VAR model that was used in \texttt{TS var}, we demonstrate the use of \texttt{varstable}.

```
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4)
(output omitted)
. varstable, graph
```

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Eigenvalue & Modulus \\
\hline
.5456253 & .545625 \\
-.3785754 + .3853982i & .540232 \\
-.3785754 - .3853982i & .540232 \\
-.0643276 + .4595944i & .464074 \\
-.0643276 - .4595944i & .464074 \\
-.3698058 & .369806 \\
\hline
\end{tabular}
\end{table}

All the eigenvalues lie inside the unit circle. 
VAR satisfies stability condition.

Because the modulus of each eigenvalue is strictly less than 1, the estimates satisfy the eigenvalue stability condition.
Specifying the `graph` option produced a graph of the eigenvalues with the real components on the \( x \) axis and the complex components on the \( y \) axis. The graph below indicates visually that these eigenvalues are well inside the unit circle.

![Roots of the companion matrix](image)

### Example 2

This example illustrates two other features of the `varstable` command. First, `varstable` can check the stability of the estimates of the VAR underlying an SVAR fit by `var svar`. Second, `varstable` can check the stability of any previously stored `var` or `var svar` estimates.

We begin by refitting the previous VAR and storing the results as `var1`. Because this is the same VAR that was fit in the previous example, the stability results should be identical.

```stata
. var dln_inv dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4)
(output omitted)
. estimates store var1
```

Now we use `svar` to fit an SVAR with a different underlying VAR and check the estimates of that underlying VAR for stability.

```stata
. matrix A = (.,0
. ,. )
. matrix B = I(2)
. svar d.ln_inc d.ln_consump, aeq(A) beq(B)
(output omitted)
. varstable
```

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.548711</td>
<td>.548711</td>
</tr>
<tr>
<td>-.2979493 + .4328013i</td>
<td>.525443</td>
</tr>
<tr>
<td>-.2979493 - .4328013i</td>
<td>.525443</td>
</tr>
<tr>
<td>-.3570825</td>
<td>.357082</td>
</tr>
</tbody>
</table>

All the eigenvalues lie inside the unit circle. VAR satisfies stability condition.
The `estimates()` option allows us to check the stability of the `var` results stored as `var1`.

```
.varstable, est(var1)
```

**Eigenvalue stability condition**

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.545625</td>
<td>.545625</td>
</tr>
<tr>
<td>-.3785754 + .3853982i</td>
<td>.540232</td>
</tr>
<tr>
<td>-.3785754 - .3853982i</td>
<td>.540232</td>
</tr>
<tr>
<td>-.0643276 + .4595944i</td>
<td>.464074</td>
</tr>
<tr>
<td>-.0643276 - .4595944i</td>
<td>.464074</td>
</tr>
<tr>
<td>-.3698058</td>
<td>.369806</td>
</tr>
</tbody>
</table>

All the eigenvalues lie inside the unit circle. VAR satisfies stability condition.

The results are identical to those obtained in the previous example, confirming that we were checking the results in `var1`.

### Stored results

`varstable` stores the following in `r()`:

- `r(Re)` real part of the eigenvalues of $A$
- `r(Im)` imaginary part of the eigenvalues of $A$
- `r(Modulus)` modulus of the eigenvalues of $A$

### Methods and formulas

`varstable` forms the companion matrix

$$
A = \begin{pmatrix}
A_1 & A_2 & \cdots & A_{p-1} & A_p \\
I & 0 & \cdots & 0 & 0 \\
0 & I & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & I & 0 
\end{pmatrix}
$$

and obtains its eigenvalues by using `matrix eigenvalues`. The modulus of the complex eigenvalue $r + ci$ is $\sqrt{r^2 + c^2}$. As shown by Lütkepohl (2005) and Hamilton (1994), the VAR is stable if the modulus of each eigenvalue of $A$ is strictly less than 1.

### References


Also see

[TS] var — Vector autoregressive models
[TS] var intro — Introduction to vector autoregressive models
[TS] var svar — Structural vector autoregressive models
[TS] varbasic — Fit a simple VAR and graph IRFs or FEVDs
varwle — Obtain Wald lag-exclusion statistics after var or svar

Description

varwle reports Wald tests the hypothesis that the endogenous variables at a given lag are jointly zero for each equation and for all equations jointly.

Quick start

Wald lag-exclusion statistics after var or svar

varwle

As above, but use VAR or SVAR estimation results stored in myest

varwle, estimates(myest)

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > Wald lag-exclusion statistics
Syntax

\texttt{varwle [ , estimates(estname) separator(#)]}

\texttt{varwle} can be used only after \texttt{var} or \texttt{svar}; see \texttt{[TS] var} and \texttt{[TS] var svar}.

Options

\texttt{estimates(estname)} requests that \texttt{varwle} use the previously obtained set of \texttt{var} or \texttt{svar} estimates stored as \texttt{estname}. By default, \texttt{varwle} uses the active estimation results. See \texttt{[R] estimates} for information on manipulating estimation results.

\texttt{separator(\#)} specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, \texttt{separator(1)} would draw a line between each row, \texttt{separator(2)} between every other row, and so on.

Remarks and examples

After fitting a \texttt{VAR}, one hypothesis of interest is that all the endogenous variables at a given lag are jointly zero. \texttt{varwle} reports Wald tests of this hypothesis for each equation and for all equations jointly. \texttt{varwle} uses the estimation results from a previously fitted \texttt{var} or \texttt{svar}. By default, \texttt{varwle} uses the active estimation results, but you may also use a stored set of estimates by specifying the \texttt{estimates()} option.

If the \texttt{VAR} was fit with the \texttt{small} option, \texttt{varwle} also presents small-sample $F$ statistics; otherwise, \texttt{varwle} presents large-sample chi-squared statistics.
Example 1: After var

We analyze the model with the German data described in [TS] var using varwle.

```
. use https://www.stata-press.com/data/r16/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk small
(output omitted)
. varwle
```

Equation: dln_inv

<table>
<thead>
<tr>
<th>lag</th>
<th>F</th>
<th>df</th>
<th>df_r</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.64902</td>
<td>3</td>
<td>66</td>
<td>0.0560</td>
</tr>
<tr>
<td>2</td>
<td>1.25799</td>
<td>3</td>
<td>66</td>
<td>0.2960</td>
</tr>
</tbody>
</table>

Equation: dln_inc

<table>
<thead>
<tr>
<th>lag</th>
<th>F</th>
<th>df</th>
<th>df_r</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.19276</td>
<td>3</td>
<td>66</td>
<td>0.0971</td>
</tr>
<tr>
<td>2</td>
<td>0.907499</td>
<td>3</td>
<td>66</td>
<td>0.4423</td>
</tr>
</tbody>
</table>

Equation: dln_consump

<table>
<thead>
<tr>
<th>lag</th>
<th>F</th>
<th>df</th>
<th>df_r</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.80804</td>
<td>3</td>
<td>66</td>
<td>0.1543</td>
</tr>
<tr>
<td>2</td>
<td>5.57645</td>
<td>3</td>
<td>66</td>
<td>0.0018</td>
</tr>
</tbody>
</table>

Equation: All

<table>
<thead>
<tr>
<th>lag</th>
<th>F</th>
<th>df</th>
<th>df_r</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.78884</td>
<td>9</td>
<td>66</td>
<td>0.0007</td>
</tr>
<tr>
<td>2</td>
<td>2.96811</td>
<td>9</td>
<td>66</td>
<td>0.0050</td>
</tr>
</tbody>
</table>

Because the VAR was fit with the dfk and small options, varwle used the small-sample estimator of $\hat{\Sigma}$ in constructing the VCE, producing an $F$ statistic. The first two equations appear to have a different lag structure from that of the third. In the first two equations, we cannot reject the null hypothesis that all three endogenous variables have zero coefficients at the second lag. The hypothesis that all three endogenous variables have zero coefficients at the first lag can be rejected at the 10% level for both of the first two equations. In contrast, in the third equation, the coefficients on the second lag of the endogenous variables are jointly significant, but not those on the first lag. However, we strongly reject the hypothesis that the coefficients on the first lag of the endogenous variables are zero in all three equations jointly. Similarly, we can also strongly reject the hypothesis that the coefficients on the second lag of the endogenous variables are zero in all three equations jointly.

If we believe these results strongly enough, we might want to refit the original VAR, placing some constraints on the coefficients. See [TS] var for details on how to fit VAR models with constraints.
Example 2: After svar

Here we fit a simple SVAR and then run varwle:

```
.matrix a = (.,0,)
.matrix b = I(2)
svar dln_inc dln_consump, aeq(a) beq(b)
```

Estimating short-run parameters

```
Iteration 0: log likelihood = -159.21683
Iteration 1: log likelihood = 490.92264
Iteration 2: log likelihood = 528.66126
Iteration 3: log likelihood = 573.96363
Iteration 4: log likelihood = 578.05136
Iteration 5: log likelihood = 578.27633
Iteration 6: log likelihood = 578.27699
Iteration 7: log likelihood = 578.27699
```

Structural vector autoregression

```
( 1) [A]1_2 = 0
( 2) [B]1_1 = 1
( 3) [B]1_2 = 0
( 4) [B]2_1 = 0
( 5) [B]2_2 = 1
```

Sample: 1960q4 - 1982q4
Number of obs = 89
Exactly identified model
Log likelihood = 578.277

|    | Coef.  | Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|----|--------|-----------|------|------|----------------------|
| A  |        |           |      |      |                      |
| 1_1| 89.72411| 6.725107  | 13.34| 0.000| 76.54315 – 102.9051  |
| 2_1| -64.73622| 10.67698 | -6.06| 0.000| -85.66271 – -43.80973|
| 1_2| 0 (constrained)|  |
| 2_2| 126.2964 | 9.466318 | 13.34| 0.000| 107.7428 - 144.8501  |

|    | Coef.  | Std. Err. | z    | P>|z|  | [95% Conf. Interval] |
|----|--------|-----------|------|------|----------------------|
| B  |        |           |      |      |                      |
| 1_1| 1 (constrained)|  |
| 2_1| 0 (constrained)|  |
| 1_2| 0 (constrained)|  |
| 2_2| 1 (constrained)|  |

The output table from var svar gives information about the estimates of the parameters in the A and B matrices in the structural VAR. But, as discussed in [TS] var svar, an SVAR model builds on an underlying VAR. When varwle uses the estimation results produced by svar, it performs Wald lag-exclusion tests on the underlying VAR model. Next we run varwle on these svar results.
. varwle

Equation: dln_inc

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.88775</td>
<td>2</td>
<td>0.032</td>
</tr>
<tr>
<td>2</td>
<td>1.873546</td>
<td>2</td>
<td>0.392</td>
</tr>
</tbody>
</table>

Equation: dln_consump

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.938547</td>
<td>2</td>
<td>0.007</td>
</tr>
<tr>
<td>2</td>
<td>13.89996</td>
<td>2</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Equation: All

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34.54276</td>
<td>4</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>19.44093</td>
<td>4</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Now we fit the underlying VAR with two lags and apply varwle to these results.

. var dln_inc dln_consump

(output omitted)

. varwle

Equation: dln_inc

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.88775</td>
<td>2</td>
<td>0.032</td>
</tr>
<tr>
<td>2</td>
<td>1.873546</td>
<td>2</td>
<td>0.392</td>
</tr>
</tbody>
</table>

Equation: dln_consump

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.938547</td>
<td>2</td>
<td>0.007</td>
</tr>
<tr>
<td>2</td>
<td>13.89996</td>
<td>2</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Equation: All

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34.54276</td>
<td>4</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>19.44093</td>
<td>4</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Because varwle produces the same results in these two cases, we can conclude that when varwle is applied to svar results, it performs Wald lag-exclusion tests on the underlying VAR.
Stored results

`varwle` stores the following in `r()`:

Matrices
if `e(small)==""`
  `r(chi2)` \( \chi^2 \) test statistics
  `r(df)` degrees of freedom
  `r(p)` \( p \)-values
if `e(small)!=""
  `r(F)` \( F \) test statistics
  `r(df_r)` numerator degrees of freedom
  `r(df)` denominator degree of freedom
  `r(p)` \( p \)-values

Methods and formulas

`varwle` uses `test` to obtain Wald statistics of the hypotheses that all the endogenous variables at a given lag are jointly zero for each equation and for all equations jointly. Like the `test` command, `varwle` uses estimation results stored by `var` or `var svar` to determine whether to calculate and report small-sample \( F \) statistics or large-sample \( \chi^2 \)-squared statistics.

Abraham Wald (1902–1950) was born in Cluj, in what is now Romania. He studied mathematics at the University of Vienna, publishing at first on geometry, but then became interested in economics and econometrics. He moved to the United States in 1938 and later joined the faculty at Columbia. His major contributions to statistics include work in decision theory, optimal sequential sampling, large-sample distributions of likelihood-ratio tests, and nonparametric inference. Wald died in a plane crash in India.

References


Also see

[TS] `var` — Vector autoregressive models

[TS] `var intro` — Introduction to vector autoregressive models

[TS] `var svar` — Structural vector autoregressive models

[TS] `varbasic` — Fit a simple VAR and graph IRFs or FEVDs
Introduction to vector error-correction models

**Description**

Stata has a suite of commands for fitting, forecasting, interpreting, and performing inference on vector error-correction models (VECMs) with cointegrating variables. After fitting a VECM, the `irf` commands can be used to obtain impulse–response functions (IRFs) and forecast-error variance decompositions (FEVDs). The table below describes the available commands.

**Fitting a VECM**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vec</code></td>
<td>Fit vector error-correction models</td>
</tr>
</tbody>
</table>

**Model diagnostics and inference**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vecrank</code></td>
<td>Estimate the cointegrating rank of a VECM</td>
</tr>
<tr>
<td><code>veclmar</code></td>
<td>Perform LM test for residual autocorrelation</td>
</tr>
<tr>
<td><code>vecnorm</code></td>
<td>Test for normally distributed disturbances after vec</td>
</tr>
<tr>
<td><code>vecstable</code></td>
<td>Check the stability condition of VECM estimates</td>
</tr>
<tr>
<td><code>varsoc</code></td>
<td>Obtain lag-order selection statistics for VARs and VECMs</td>
</tr>
</tbody>
</table>

**Forecasting from a VECM**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fcast compute</code></td>
<td>Compute dynamic forecasts after var, svar, or vec</td>
</tr>
<tr>
<td><code>fcast graph</code></td>
<td>Graph forecasts after <code>fcast compute</code></td>
</tr>
</tbody>
</table>

**Working with IRFs and FEVDs**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>irf</code></td>
<td>Create and analyze IRFs and FEVDs</td>
</tr>
</tbody>
</table>

This manual entry provides an overview of the commands for VECMs; provides an introduction to integration, cointegration, estimation, inference, and interpretation of VECM models; and gives an example of how to use Stata’s `vec` commands.

**Remarks and examples**

`vec` estimates the parameters of cointegrating VECMs. You may specify any of the five trend specifications in Johansen (1995, sec. 5.7). By default, identification is obtained via the Johansen normalization, but `vec` allows you to obtain identification by placing your own constraints on the parameters of the cointegrating vectors. You may also put more restrictions on the adjustment coefficients.

`vecrank` is the command for determining the number of cointegrating equations. `vecrank` implements Johansen’s multiple trace test procedure, the maximum eigenvalue test, and a method based on minimizing either of two different information criteria.
Because Nielsen (2001) has shown that the methods implemented in \texttt{varsoc} can be used to choose the order of the autoregressive process, no separate \texttt{vec} command is needed; you can simply use \texttt{varsoc}. \texttt{vecmar} tests that the residuals have no serial correlation, and \texttt{vecnorm} tests that they are normally distributed.

All the \texttt{irf} routines described in [TS] \texttt{irf} are available for estimating, interpreting, and managing estimated IRFs and FEVDs for VECMs.

Remarks are presented under the following headings:

- Introduction to cointegrating VECMs
- What is cointegration?
- The multivariate VECM specification
- Trends in the Johansen VECM framework
- VECM estimation in Stata
  - Selecting the number of lags
  - Testing for cointegration
  - Fitting a VECM
  - Fitting VECMs with Johansen’s normalization
  - Postestimation specification testing
  - Impulse–response functions for VECMs
  - Forecasting with VECMs

### Introduction to cointegrating VECMs

This section provides a brief introduction to integration, cointegration, and cointegrated vector error-correction models. For more details about these topics, see Hamilton (1994), Johansen (1995), Lütkepohl (2005), Watson (1994), and Becketti (2013).

### What is cointegration?

Standard regression techniques, such as ordinary least squares (OLS), require that the variables be covariance stationary. A variable is covariance stationary if its mean and all its autocovariances are finite and do not change over time. Cointegration analysis provides a framework for estimation, inference, and interpretation when the variables are not covariance stationary.

Instead of being covariance stationary, many economic time series appear to be “first-difference stationary”. This means that the level of a time series is not stationary but its first difference is. First-difference stationary processes are also known as integrated processes of order 1, or I(1) processes. Covariance-stationary processes are I(0). In general, a process whose $d$th difference is stationary is an integrated process of order $d$, or I($d$).

The canonical example of a first-difference stationary process is the random walk. This is a variable $x_t$ that can be written as

\[
x_t = x_{t-1} + \epsilon_t
\]

where the $\epsilon_t$ are independent and identically distributed with mean zero and a finite variance $\sigma^2$. Although $E[x_t] = 0$ for all $t$, $\text{Var}[x_t] = T\sigma^2$ is not time invariant, so $x_t$ is not covariance stationary. Because $\Delta x_t = x_t - x_{t-1} = \epsilon_t$ and $\epsilon_t$ is covariance stationary, $x_t$ is first-difference stationary.

These concepts are important because, although conventional estimators are well behaved when applied to covariance-stationary data, they have nonstandard asymptotic distributions and different rates of convergence when applied to I(1) processes. To illustrate, consider several variants of the model

\[
y_t = a + bx_t + \epsilon_t
\]

Throughout the discussion, we maintain the assumption that $E[\epsilon_t] = 0$. 
If both \( y_t \) and \( x_t \) are covariance-stationary processes, \( e_t \) must also be covariance stationary. As long as \( E[x_t e_t] = 0 \), we can consistently estimate the parameters \( a \) and \( b \) by using OLS. Furthermore, the distribution of the OLS estimator converges to a normal distribution centered at the true value as the sample size grows.

If \( y_t \) and \( x_t \) are independent random walks and \( b = 0 \), there is no relationship between \( y_t \) and \( x_t \), and (2) is called a spurious regression. Granger and Newbold (1974) performed Monte Carlo experiments and showed that the usual t statistics from OLS regression provide spurious results: given a large enough dataset, we can almost always reject the null hypothesis of the test that \( b = 0 \) even though \( b \) is in fact zero. Here the OLS estimator does not converge to any well-defined population parameter.

Phillips (1986) later provided the asymptotic theory that explained the Granger and Newbold (1974) results. He showed that the random walks \( y_t \) and \( x_t \) are first-difference stationary processes and that the OLS estimator does not have its usual asymptotic properties when the variables are first-difference stationary.

Because \( \Delta y_t \) and \( \Delta x_t \) are covariance stationary, a simple regression of \( \Delta y_t \) on \( \Delta x_t \) appears to be a viable alternative. However, if \( y_t \) and \( x_t \) cointegrate, as defined below, the simple regression of \( \Delta y_t \) on \( \Delta x_t \) is misspecified.

If \( y_t \) and \( x_t \) are I(1) and \( b \neq 0 \), \( e_t \) could be either I(0) or I(1). Phillips and Durlauf (1986) have derived the asymptotic theory for the OLS estimator when \( e_t \) is I(1), though it has not been widely used in applied work. More interesting is the case in which \( e_t = y_t - a - bx_t \) is I(0). \( y_t \) and \( x_t \) are then said to be cointegrated. Two variables are cointegrated if each is an I(1) process but a linear combination of them is an I(0) process.

It is not possible for \( y_t \) to be a random walk and \( x_t \) and \( e_t \) to be covariance stationary. As Granger (1981) pointed out, because a random walk cannot be equal to a covariance-stationary process, the equation does not “balance”. An equation balances when the processes on each side of the equal sign are of the same order of integration. Before attacking any applied problem with integrated variables, make sure that the equation balances before proceeding.

An example from Engle and Granger (1987) provides more intuition. Redefine \( y_t \) and \( x_t \) to be

\[
y_t + \beta x_t = \epsilon_t, \quad \epsilon_t = \epsilon_{t-1} + \xi_t
\]

\[
y_t + \alpha x_t = \nu_t, \quad \nu_t = \rho \nu_{t-1} + \zeta_t, \quad |\rho| < 1
\]

(3) (4)

where \( \xi_t \) and \( \zeta_t \) are i.i.d. disturbances over time that are correlated with each other. Because \( \epsilon_t \) is I(1), (3) and (4) imply that both \( x_t \) and \( y_t \) are I(1). The condition that \( |\rho| < 1 \) implies that \( \nu_t \) and \( y_t + \alpha x_t \) are I(0). Thus \( y_t \) and \( x_t \) cointegrate, and \( (1, \alpha) \) is the cointegrating vector.

Using a bit of algebra, we can rewrite (3) and (4) as

\[
\Delta y_t = \beta \delta z_{t-1} + \eta_{1t}
\]

\[
\Delta x_t = -\delta z_{t-1} + \eta_{2t}
\]

(5) (6)

where \( \delta = (1 - \rho)/(\alpha - \beta) \), \( z_t = y_t + \alpha x_t \), and \( \eta_{1t} \) and \( \eta_{2t} \) are distinct, stationary, linear combinations of \( \xi_t \) and \( \zeta_t \). This representation is known as the vector error-correction model (VECM). One can think of \( z_t = 0 \) as being the point at which \( y_t \) and \( x_t \) are in equilibrium. The coefficients on \( z_{t-1} \) describe how \( y_t \) and \( x_t \) adjust to \( z_{t-1} \) being nonzero, or out of equilibrium. \( z_t \) is the “error” in the system, and (5) and (6) describe how system adjusts or corrects back to the equilibrium. As \( \rho \to 1 \), the system degenerates into a pair of correlated random walks. The VECM parameterization highlights this point, because \( \delta \to 0 \) as \( \rho \to 1 \).
If we knew \( \alpha \), we would know \( z_t \), and we could work with the stationary system of (5) and (6). Although knowing \( \alpha \) seems silly, we can conduct much of the analysis as if we knew \( \alpha \) because there is an estimator for the cointegrating parameter \( \alpha \) that converges to its true value at a faster rate than the estimator for the adjustment parameters \( \beta \) and \( \delta \).

The definition of a bivariate cointegrating relation requires simply that there exist a linear combination of the I(1) variables that is I(0). If \( y_t \) and \( x_t \) are I(1) and there are two finite real numbers \( a \neq 0 \) and \( b \neq 0 \), such that \( ay_t + bx_t \) is I(0), then \( y_t \) and \( x_t \) are cointegrated. Although there are two parameters, \( a \) and \( b \), only one will be identifiable because if \( ay_t + bx_t \) is I(0), so is \( cay_t + cbx_t \) for any finite, nonzero, real number \( c \). Obtaining identification in the bivariate case is relatively simple. The coefficient on \( y_t \) in (4) is unity. This natural construction of the model placed the necessary identification restriction on the cointegrating vector. As we discuss below, identification in the multivariate case is more involved.

If \( y_t \) is a \( K \times 1 \) vector of I(1) variables and there exists a vector \( \beta \), such that \( \beta y_t \) is a vector of I(0) variables, then \( y_t \) is said to be cointegrating of order \((1,0)\) with cointegrating vector \( \beta \). We say that the parameters in \( \beta \) are the parameters in the cointegrating equation. For a vector of length \( K \), there may be at most \( K - 1 \) distinct cointegrating vectors. Engle and Granger (1987) provide a more general definition of cointegration, but this one is sufficient for our purposes.

**The multivariate VECM specification**

In practice, most empirical applications analyze multivariate systems, so the rest of our discussion focuses on that case. Consider a VAR with \( p \) lags

\[
y_t = \mathbf{v} + \mathbf{A}_1 y_{t-1} + \mathbf{A}_2 y_{t-2} + \cdots + \mathbf{A}_p y_{t-p} + \epsilon_t
\]

(7)

where \( y_t \) is a \( K \times 1 \) vector of variables, \( \mathbf{v} \) is a \( K \times 1 \) vector of parameters, \( \mathbf{A}_1 - \mathbf{A}_p \) are \( K \times K \) matrices of parameters, and \( \epsilon_t \) is a \( K \times 1 \) vector of disturbances. \( \epsilon_t \) has mean 0, has covariance matrix \( \Sigma \), and is i.i.d. normal over time. Any VAR(\( p \)) can be rewritten as a VECM. Using some algebra, we can rewrite (7) in VECM form as

\[
\Delta y_t = \mathbf{v} + \Pi y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \epsilon_t
\]

(8)

where \( \Pi = \sum_{j=1}^{p} \mathbf{A}_j - \mathbf{I}_K \) and \( \Gamma_i = -\sum_{j=i+1}^{p} \mathbf{A}_j \). The \( \mathbf{v} \) and \( \epsilon_t \) in (7) and (8) are identical.

Engle and Granger (1987) show that if the variables \( y_t \) are I(1) the matrix \( \Pi \) in (8) has rank \( 0 \leq r < K \), where \( r \) is the number of linearly independent cointegrating vectors. If the variables cointegrate, \( 0 < r < K \) and (8) shows that a VAR in first differences is misspecified because it omits the lagged level term \( \Pi y_{t-1} \).

Assume that \( \Pi \) has reduced rank \( 0 < r < K \) so that it can be expressed as \( \Pi = \alpha \beta' \), where \( \alpha \) and \( \beta \) are both \( r \times K \) matrices of rank \( r \). Without further restrictions, the cointegrating vectors are not identified: the parameters \((\alpha, \beta)\) are indistinguishable from the parameters \((\alpha \mathbf{Q}, \beta \mathbf{Q}^{-1'})\) for any \( r \times r \) nonsingular matrix \( \mathbf{Q} \). Because only the rank of \( \Pi \) is identified, the VECM is said to identify the rank of the cointegrating space, or equivalently, the number of cointegrating vectors. In practice, the estimation of the parameters of a VECM requires at least \( r^2 \) identification restrictions. Stata’s \texttt{vec} command can apply the conventional Johansen restrictions discussed below or use constraints that the user supplies.

The VECM in (8) also nests two important special cases. If the variables in \( y_t \) are I(1) but not cointegrated, \( \Pi \) is a matrix of zeros and thus has rank 0. If all the variables are I(0), \( \Pi \) has full rank \( K \).
There are several different frameworks for estimation and inference in cointegrating systems. Although the methods in Stata are based on the maximum likelihood (ML) methods developed by Johansen (1988, 1991, 1995), other useful frameworks have been developed by Park and Phillips (1988, 1989); Sims, Stock, and Watson (1990); Stock (1987); and Stock and Watson (1988); among others. The ML framework developed by Johansen was independently developed by Ahn and Reinsel (1990). Maddala and Kim (1998) and Watson (1994) survey all of these methods. The cointegration methods in Stata are based on Johansen’s maximum likelihood framework because it has been found to be particularly useful in several comparative studies, including Gonzalo (1994) and Hubrich, Lütkepohl, and Saikkonen (2001).

Trends in the Johansen VECM framework

Deterministic trends in a cointegrating VECM can stem from two distinct sources; the mean of the cointegrating relationship and the mean of the differenced series. Allowing for a constant and a linear trend and assuming that there are $r$ cointegrating relations, we can rewrite the VECM in (8) as

$$
\Delta y_t = \alpha \beta' y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \nu + \delta t + \epsilon_t
$$

where $\delta$ is a $K \times 1$ vector of parameters. Because (9) models the differences of the data, the constant implies a linear time trend in the levels, and the time trend $\delta t$ implies a quadratic time trend in the levels of the data. Often we may want to include a constant or a linear time trend for the differences without allowing for the higher-order trend that is implied for the levels of the data. VECMs exploit the properties of the matrix $\alpha$ to achieve this flexibility.

Because $\alpha$ is a $K \times r$ rank matrix, we can rewrite the deterministic components in (9) as

$$
\nu = \alpha \mu + \gamma \quad (10a)
$$

$$
\delta t = \alpha \rho t + \tau \quad (10b)
$$

where $\mu$ and $\rho$ are $r \times 1$ vectors of parameters and $\gamma$ and $\tau$ are $K \times 1$ vectors of parameters. $\gamma$ is orthogonal to $\alpha \mu$, and $\tau$ is orthogonal to $\alpha \rho$; that is, $\gamma' \alpha \mu = 0$ and $\tau' \alpha \rho = 0$, allowing us to rewrite (9) as

$$
\Delta y_t = \alpha (\beta' y_{t-1} + \mu + \rho t) + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \gamma + \tau t + \epsilon_t
$$

(11)

Placing restrictions on the trend terms in (11) yields five cases.

CASE 1: Unrestricted trend

If no restrictions are placed on the trend parameters, (11) implies that there are quadratic trends in the levels of the variables and that the cointegrating equations are stationary around time trends (trend stationary).

CASE 2: Restricted trend, $\tau = 0$

By setting $\tau = 0$, we assume that the trends in the levels of the data are linear but not quadratic. This specification allows the cointegrating equations to be trend stationary.

CASE 3: Unrestricted constant, $\tau = 0$ and $\rho = 0$

By setting $\tau = 0$ and $\rho = 0$, we exclude the possibility that the levels of the data have quadratic trends, and we restrict the cointegrating equations to be stationary around constant means. Because $\gamma$ is not restricted to zero, this specification still puts a linear time trend in the levels of the data.
CASE 4: Restricted constant, $\tau = 0$, $\rho = 0$, and $\gamma = 0$

By adding the restriction that $\gamma = 0$, we assume there are no linear time trends in the levels of the data. This specification allows the cointegrating equations to be stationary around a constant mean, but it allows no other trends or constant terms.

CASE 5: No trend, $\tau = 0$, $\rho = 0$, $\gamma = 0$, and $\mu = 0$

This specification assumes that there are no nonzero means or trends. It also assumes that the cointegrating equations are stationary with means of zero and that the differences and the levels of the data have means of zero.

This flexibility does come at a price. Below we discuss testing procedures for determining the number of cointegrating equations. The asymptotic distribution of the LR for hypotheses about $r$ changes with the trend specification, so we must first specify a trend specification. A combination of theory and graphical analysis will aid in specifying the trend before proceeding with the analysis.

**VECM estimation in Stata**

We provide an overview of the `vec` commands in Stata through an extended example. We have monthly data on the average selling prices of houses in four cities in Texas: Austin, Dallas, Houston, and San Antonio. In the dataset, these average housing prices are contained in the variables `austin`, `dallas`, `houston`, and `sa`. The series begin in January of 1990 and go through December 2003, for a total of 168 observations. The following graph depicts our data.

The plots on the graph indicate that all the series are trending and potential I(1) processes. In a competitive market, the current and past prices contain all the information available, so tomorrow’s price will be a random walk from today’s price. Some researchers may opt to use [TS] `dfgls` to investigate the presence of a unit root in each series, but the test for cointegration we use includes the case in which all the variables are stationary, so we defer formal testing until we test for cointegration. The time trends in the data appear to be approximately linear, so we will specify `trend(constant)` when modeling these series, which is the default with `vec`.

The next graph shows just Dallas’s and Houston’s data, so we can more carefully examine their relationship.
Except for the crash at the end of 1991, housing prices in Dallas and Houston appear closely related. Although average prices in the two cities will differ because of resource variations and other factors, if the housing markets become too dissimilar, people and businesses will migrate, bringing the average housing prices back toward each other. We therefore expect the series of average housing prices in Houston to be cointegrated with the series of average housing prices in Dallas.

### Selecting the number of lags

To test for cointegration or fit cointegrating VECMs, we must specify how many lags to include. Building on the work of Tsay (1984) and Paulsen (1984), Nielsen (2001) has shown that the methods implemented in `varsoc` can be used to determine the lag order for a VAR model with I(1) variables. As can be seen from (9), the order of the corresponding VECM is always one less than the VAR. `vec` makes this adjustment automatically, so we will always refer to the order of the underlying VAR. The output below uses `varsoc` to determine the lag order of the VAR of the average housing prices in Dallas and Houston.

```
. use https://www.stata-press.com/data/r16/txhprice
. varsoc dallas houston
```

```
Selection-order criteria
Sample: 1990m5 - 2003m12 Number of obs = 164

<table>
<thead>
<tr>
<th>lag</th>
<th>LL</th>
<th>LR</th>
<th>df</th>
<th>p</th>
<th>FPE</th>
<th>AIC</th>
<th>HQIC</th>
<th>SBIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>299.525</td>
<td>0.000091</td>
<td>.3.62835</td>
<td>.3.61301</td>
<td>.3.59055</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>577.483</td>
<td>555.92</td>
<td>4</td>
<td>0.000</td>
<td>3.2e-06</td>
<td>.6.9693</td>
<td>.6.92326</td>
<td>.6.85589</td>
</tr>
<tr>
<td>2</td>
<td>590.978</td>
<td>26.3991*</td>
<td>4</td>
<td>0.000</td>
<td>-7.0851*</td>
<td>.7.00837*</td>
<td>.6.89608*</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>593.437</td>
<td>4.918</td>
<td>4</td>
<td>0.296</td>
<td>2.9e-06</td>
<td>.7.06631</td>
<td>.6.95888</td>
<td>.6.80168</td>
</tr>
<tr>
<td>4</td>
<td>596.364</td>
<td>5.8532</td>
<td>4</td>
<td>0.210</td>
<td>3.0e-06</td>
<td>.7.05322</td>
<td>.6.9151</td>
<td>.6.71299</td>
</tr>
</tbody>
</table>

Endogenous: dallas houston
Exogenous: _cons
```

We will use two lags for this bivariate model because the Hannan–Quinn information criterion (HQIC) method, Schwarz Bayesian information criterion (SBIC) method, and sequential likelihood-ratio (LR) test all chose two lags, as indicated by the “*” in the output.

The reader can verify that when all four cities’ data are used, the LR test selects three lags, the HQIC method selects two lags, and the SBIC method selects one lag. We will use three lags in our four-variable model.
Testing for cointegration

The tests for cointegration implemented in vecrank are based on Johansen’s method. If the log likelihood of the unconstrained model that includes the cointegrating equations is significantly different from the log likelihood of the constrained model that does not include the cointegrating equations, we reject the null hypothesis of no cointegration.

Here we use vecrank to determine the number of cointegrating equations:

```
. vecrank dallas houston
Johansen tests for cointegration
Trend: constant Number of obs = 166
Sample: 1990m3 - 2003m12 Lags = 2

<table>
<thead>
<tr>
<th>rank</th>
<th>parms</th>
<th>LL</th>
<th>eigenvalue</th>
<th>statistic</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>576.26444</td>
<td>.</td>
<td>46.8252</td>
<td>15.41</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>599.58781</td>
<td>0.24498</td>
<td>0.1785*</td>
<td>3.76</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>599.67706</td>
<td>0.00107</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Besides presenting information about the sample size and time span, the header indicates that test statistics are based on a model with two lags and a constant trend. The body of the table presents test statistics and their critical values of the null hypotheses of no cointegration (line 1) and one or fewer cointegrating equations (line 2). The eigenvalue shown on the last line is used to compute the trace statistic in the line above it. Johansen’s testing procedure starts with the test for zero cointegrating equations (a maximum rank of zero) and then accepts the first null hypothesis that is not rejected.

In the output above, we strongly reject the null hypothesis of no cointegration and fail to reject the null hypothesis of at most one cointegrating equation. Thus we accept the null hypothesis that there is one cointegrating equation in the bivariate model.

Using all four series and a model with three lags, we find that there are two cointegrating relationships.

```
. vecrank austin dallas houston sa, lag(3)
Johansen tests for cointegration
Trend: constant Number of obs = 165
Sample: 1990m4 - 2003m12 Lags = 3

<table>
<thead>
<tr>
<th>rank</th>
<th>parms</th>
<th>LL</th>
<th>eigenvalue</th>
<th>statistic</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>36</td>
<td>1107.7833</td>
<td>.</td>
<td>101.6070</td>
<td>47.21</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>1137.7484</td>
<td>0.30456</td>
<td>41.6768</td>
<td>29.68</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>1153.6435</td>
<td>0.17524</td>
<td>9.8865*</td>
<td>15.41</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>1158.4191</td>
<td>0.05624</td>
<td>0.3354</td>
<td>3.76</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>1158.5868</td>
<td>0.00203</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Fitting a VECM

vec estimates the parameters of cointegrating VECMs. There are four types of parameters of interest:

1. The parameters in the cointegrating equations $\beta$
2. The adjustment coefficients $\alpha$
3. The short-run coefficients
4. Some standard functions of $\beta$ and $\alpha$ that have useful interpretations
Although all four types are discussed in [TS] vec, here we discuss only types 1–3 and how they appear in the output of vec.

Having determined that there is a cointegrating equation between the Dallas and Houston series, we now want to estimate the parameters of a bivariate cointegrating VECM for these two series by using vec.

```
.vec dallas houston
Vector error-correction model
Sample: 1990m3 - 2003m12 Number of obs = 166
AIC = -7.115516
Log likelihood = 599.5878 HQIC = -7.04703
Det(Sigma_ml) = 2.50e-06 SBIC = -6.946794
Equation Parms RMSE R-sq chi2 P>chi2
D_dallas 4 .038546 0.1692 32.98959 0.0000
D_houston 4 .045348 0.3737 96.66399 0.0000

Coef. Std. Err. z P>|z| [95% Conf. Interval]
D_dallas
  _ce1 L1. -.3038799 .0908504 -3.34 0.001 -.4819434 -.1258165
dallas LD. -.1647304 .0879356 -1.87 0.061 -.337081 .0076202
houston LD. -.0998368 .0650838 -1.53 0.125 -.2273988 .0277251
  _cons .0056128 .0030341 1.85 0.064 -.0003339 .0115595
D_houston
  _ce1 L1. .5027143 .1068838 4.70 0.000 .2932258 .7122028
dallas LD. -.0619653 .1034547 -0.60 0.549 -.2647327 .1408022
houston LD. -.3328437 .07657 -4.35 0.000 -.4829181 -.1827693
  _cons .0033928 .0035695 0.95 0.342 -.0036034 .010389

Cointegrating equations
Equation Parms chi2 P>chi2
  _ce1 1 1640.088 0.0000

Identification: beta is exactly identified
Johansen normalization restriction imposed

beta
  Coef. Std. Err. z P>|z| [95% Conf. Interval]
  _ce1
dallas 1 . . . .
houston -.8675936 .0214231 -40.50 0.000 -.9095821 -.825605
  _cons -1.688897 . . . .
```
The header contains information about the sample, the fit of each equation, and overall model fit statistics. The first estimation table contains the estimates of the short-run parameters, along with their standard errors, $z$ statistics, and confidence intervals. The two coefficients on $L._{ce1}$ are the parameters in the adjustment matrix $\alpha$ for this model. The second estimation table contains the estimated parameters of the cointegrating vector for this model, along with their standard errors, $z$ statistics, and confidence intervals.

Using our previous notation, we have estimated

$$\hat{\alpha} = (-0.304, 0.503) \quad \hat{\beta} = (1, -0.868) \quad \hat{\nu} = (0.0056, 0.0034)$$

and

$$\hat{\Gamma} = \begin{pmatrix} -0.165 & -0.0998 \\ -0.062 & -0.333 \end{pmatrix}$$

Overall, the output indicates that the model fits well. The coefficient on $\text{houston}$ in the cointegrating equation is statistically significant, as are the adjustment parameters. The adjustment parameters in this bivariate example are easy to interpret, and we can see that the estimates have the correct signs and imply rapid adjustment toward equilibrium. When the predictions from the cointegrating equation are positive, $\text{dallas}$ is above its equilibrium value because the coefficient on $\text{dallas}$ in the cointegrating equation is positive. The estimate of the coefficient $[D_{dallas}]L._{ce1}$ is $-0.3$. Thus when the average housing price in Dallas is too high, it quickly falls back toward the Houston level. The estimated coefficient $[D_{houston}]L._{ce1}$ of 0.5 implies that when the average housing price in Dallas is too high, the average price in Houston quickly adjusts toward the Dallas level at the same time that the Dallas prices are adjusting.

**Fitting VECMs with Johansen’s normalization**

As discussed by Johansen (1995), if there are $r$ cointegrating equations, then at least $r^2$ restrictions are required to identify the free parameters in $\beta$. Johansen proposed a default identification scheme that has become the conventional method of identifying models in the absence of theoretically justified restrictions. Johansen’s identification scheme is

$$\beta' = (I_r, \tilde{\beta}')$$

where $I_r$ is the $r \times r$ identity matrix and $\tilde{\beta}$ is an $(K - r) \times r$ matrix of identified parameters. vec applies Johansen’s normalization by default.

To illustrate, we fit a VECM with two cointegrating equations and three lags on all four series. We are interested only in the estimates of the parameters in the cointegrating equations, so we can specify the noetable option to suppress the estimation table for the adjustment and short-run parameters.
Vector error-correction model

Sample: 1990m4 - 2003m12  
Number of obs = 165

Log likelihood = 1153.644

AIC = -13.40174

Log likelihood = 1153.644

HQIC = -13.03496

Det(Sigma_ml) = 9.93e-12

SBIC = -12.49819

Cointegrating equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ce1</td>
<td>2</td>
<td>586.3044</td>
<td>0.0000</td>
</tr>
<tr>
<td>_ce2</td>
<td>2</td>
<td>2169.826</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Identification: beta is exactly identified

Johansen normalization restrictions imposed

| beta     | Coef.   | Std. Err. | z   | P>|z|      | [95% Conf. Interval] |
|----------|---------|-----------|-----|----------|---------------------|
| _ce1     |         |           |     |          |                     |
| austin   | 1       |           |     |          |                     |
| dallas   | 0       | (omitted)|     |          |                     |
| houston  | -0.2623782 | 0.1893625 | -1.39 | 0.166 | -0.6335219 - 0.1087655 |
| sa       | -1.241805 | 0.229643  | -5.41 | 0.000 | -1.691897 - 0.7917128 |
| _cons    | 5.577099 |           |     |          |                     |
| _ce2     |         |           |     |          |                     |
| austin   | 0       | (omitted)|     |          |                     |
| dallas   | 1       |           |     |          |                     |
| houston  | -1.095652 | 0.0669898 | -16.36 | 0.000 | -1.22695 - 0.9643545 |
| sa       | 0.2883986 | 0.0812396 | 3.55  | 0.000 | 0.1291718 - 0.4476253 |
| _cons    | -2.351372|           |     |          |                     |

The Johansen identification scheme has placed four constraints on the parameters in $\beta$: $[_ce1]austin = 1$, $[_ce1]dallas = 0$, $[_ce2]austin = 0$, and $[_ce2]dallas = 1$. We interpret the results of the first equation as indicating the existence of an equilibrium relationship between the average housing price in Austin and the average prices of houses in Houston and San Antonio.

The Johansen normalization restricted the coefficient on dallas to be unity in the second cointegrating equation, but we could instead constrain the coefficient on houston. Both sets of restrictions define just-identified models, so fitting the model with the latter set of restrictions will yield the same maximized log likelihood. To impose the alternative set of constraints, we use the constraint command.

```
. constraint define 1 [_ce1]austin = 1
. constraint define 2 [_ce1]dallas = 0
. constraint define 3 [_ce2]austin = 0
. constraint define 4 [_ce2]houston = 1
```
. vec austin dallas houston sa, lags(3) rank(2) noetable bconstraints(1/4)

Iteration 1:  log likelihood = 1148.8745
(output omitted)

Iteration 25:  log likelihood = 1153.6435

Vector error-correction model

Sample: 1990m4 - 2003m12  Number of obs = 165

AIC = -13.40174

Log likelihood = 1153.644

SBIC = -12.49819

Det(Sigma_ml) = 9.93e-12

Cointegrating equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ce1</td>
<td>2</td>
<td>586.3392</td>
<td>0.0000</td>
</tr>
<tr>
<td>_ce2</td>
<td>2</td>
<td>3455.469</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Identification: beta is exactly identified

(1) [ _ce1 ] austin = 1
(2) [ _ce1 ] dallas = 0
(3) [ _ce2 ] austin = 0
(4) [ _ce2 ] houston = 1

| beta      | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|-----------|-------|-----------|------|------|----------------------|
| _ce1      |       |           |      |      |                      |
| austin    | 1     | .         | .    |      |                      |
| dallas    | 0     | (omitted) | .    |      |                      |
| houston   | -.2623784 | .1876727 | -1.40 | .162 | -.6302102 - .1054534 |
| sa        | -1.241805 | .2277537 | -5.45 | 0.000 | -1.688194 - .7954157 |
| _cons     | 5.577099 | .         |      |      |                      |
| _ce2      |       |           |      |      |                      |
| austin    | 0     | (ommitted)| .    |      |                      |
| dallas    | -.9126985 | .0595804 | -15.32 | 0.000 | -1.029474 - .7959231 |
| houston   | 1     | .         | .    |      |                      |
| sa        | -.2632209 | .0628791 | -4.19 | 0.000 | -.3864617 - .1399802 |
| _cons     | 2.146094 | .         |      |      |                      |

Only the estimates of the parameters in the second cointegrating equation have changed, and the new estimates are simply the old estimates divided by $-1.095652$ because the new constraints are just an alternative normalization of the same just-identified model. With the new normalization, we can interpret the estimates of the parameters in the second cointegrating equation as indicating an equilibrium relationship between the average house price in Houston and the average prices of houses in Dallas and San Antonio.

**Postestimation specification testing**

Inference on the parameters in $\alpha$ depends crucially on the stationarity of the cointegrating equations, so we should check the specification of the model. As a first check, we can predict the cointegrating equations and graph them over time.

. predict ce1, ce equ(#1)
. predict ce2, ce equ(#2)
Although the large shocks apparent in the graph of the levels have clear effects on the predictions from the cointegrating equations, our only concern is the negative trend in the first cointegrating equation since the end of 2000. The graph of the levels shows that something put a significant brake on the growth of housing prices after 2000 and that the growth of housing prices in San Antonio slowed during 2000 but then recuperated while Austin maintained slower growth. We suspect that this indicates that the end of the high-tech boom affected Austin more severely than San Antonio. This difference is what causes the trend in the first cointegrating equation. Although we could try to account for this effect with a more formal analysis, we will proceed as if the cointegrating equations are stationary.

We can use vecstable to check whether we have correctly specified the number of cointegrating equations. As discussed in [TS] vecstable, the companion matrix of a VECM with \( K \) endogenous variables and \( r \) cointegrating equations has \( K - r \) unit eigenvalues. If the process is stable, the moduli of the remaining \( r \) eigenvalues are strictly less than one. Because there is no general distribution
theory for the moduli of the eigenvalues, ascertaining whether the moduli are too close to one can be difficult.

```
. vecstable, graph

Eigenvalue stability condition

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-0.6698661</td>
<td>0.669866</td>
</tr>
<tr>
<td>0.3740191 + 0.4475996i</td>
<td>0.583297</td>
</tr>
<tr>
<td>0.3740191 - 0.4475996i</td>
<td>0.583297</td>
</tr>
<tr>
<td>-0.386377 + 0.395972i</td>
<td>0.553246</td>
</tr>
<tr>
<td>-0.386377 - 0.395972i</td>
<td>0.553246</td>
</tr>
<tr>
<td>0.540117</td>
<td>0.540117</td>
</tr>
<tr>
<td>-0.0749239 + 0.5274203i</td>
<td>0.532715</td>
</tr>
<tr>
<td>-0.0749239 - 0.5274203i</td>
<td>0.532715</td>
</tr>
<tr>
<td>-0.2023955</td>
<td>0.202395</td>
</tr>
<tr>
<td>0.09923966</td>
<td>0.09924</td>
</tr>
</tbody>
</table>
```

The VECM specification imposes 2 unit moduli.

Because we specified the graph option, vecstable plotted the eigenvalues of the companion matrix. The graph of the eigenvalues shows that none of the remaining eigenvalues appears close to the unit circle. The stability check does not indicate that our model is misspecified.

Here we use veclmar to test for serial correlation in the residuals.

```
. veclmar, mlag(4)

Lagrange-multiplier test

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56.8757</td>
<td>16</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>31.1970</td>
<td>16</td>
<td>0.01270</td>
</tr>
<tr>
<td>3</td>
<td>30.6818</td>
<td>16</td>
<td>0.01477</td>
</tr>
<tr>
<td>4</td>
<td>14.6493</td>
<td>16</td>
<td>0.55046</td>
</tr>
</tbody>
</table>

H0: no autocorrelation at lag order
```
The results clearly indicate serial correlation in the residuals. The results in Gonzalo (1994) indicate that underspecifying the number of lags in a VECM can significantly increase the finite-sample bias in the parameter estimates and lead to serial correlation. For this reason, we refit the model with five lags instead of three.

```
.vec austin dallas houston sa, lags(5) rank(2) noetable bconstraints(1/4)
```

```
Iteration 1:   log likelihood = 1200.5402
(output omitted)
Iteration 20:  log likelihood = 1203.9465
```

Vector error-correction model

Sample: 1990m6 - 2003m12   Number of obs = 163
Log likelihood = 1203.946   AIC = -13.79075
Det(Sigma_ml) = 4.51e-12    HQIC = -13.1743
SBIC = -12.27235

Cointegrating equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ce1</td>
<td>2</td>
<td>498.4682</td>
<td>0.0000</td>
</tr>
<tr>
<td>_ce2</td>
<td>2</td>
<td>4125.926</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Identification: beta is exactly identified

( 1) [._ce1]austin = 1
( 2) [._ce1]dallas = 0
( 3) [._ce2]austin = 0
( 4) [._ce2]houston = 1

| beta    | Coef. | Std. Err. | z  | P>|z| | [95% Conf. Interval]|
|---------|-------|-----------|----|------|----------------|
|_ce1     |       |           |    |      |                |
| austin  | 1     | .         | .  | .    | .              |
| dallas  | 0 (omitted) |       |    |      | .              |
| houston | -.6525574 | .2047061| -3.19| 0.001| -1.053774 -.2513407 |
| sa      | -.6960166 | .2494167| -2.79| 0.005| -1.184864 -.2071688 |
|_cons   | 3.846275 | .         | .  | .    | .              |

|_ce2     |       |           |    |      |                |
| austin  | 0 (omitted) |       |    |      | .              |
| dallas  | -.932048 | .0564332| -16.52| 0.000| -1.042655 -.8214409 |
| houston | 1      | .         | .  | .    | .              |
| sa      | -.2363915 | .0599348| -3.94| 0.000| -.3538615 -.1189215 |
|_cons   | 2.065719 | .         | .  | .    | .              |

Comparing these results with those from the previous model reveals that
1. there is now evidence that the coefficient [._ce1]houston is not equal to zero,
2. the two sets of estimated coefficients for the first cointegrating equation are different, and
3. the two sets of estimated coefficients for the second cointegrating equation are similar.

The assumption that the errors are independent and are identically and normally distributed with zero mean and finite variance allows us to derive the likelihood function. If the errors do not come from a normal distribution but are just independent and identically distributed with zero mean and finite variance, the parameter estimates are still consistent, but they are not efficient.
We use `vecnorm` to test the null hypothesis that the errors are normally distributed.

```stata
. quietly vec austin dallas houston sa, lags(5) rank(2) bconstraints(1/4)
. vecnorm
```

<table>
<thead>
<tr>
<th>Equation</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_austin</td>
<td>74.324</td>
<td>2</td>
<td>0.00000</td>
</tr>
<tr>
<td>D_dallas</td>
<td>3.501</td>
<td>2</td>
<td>0.17370</td>
</tr>
<tr>
<td>D_houston</td>
<td>245.032</td>
<td>2</td>
<td>0.00000</td>
</tr>
<tr>
<td>D_sa</td>
<td>8.426</td>
<td>2</td>
<td>0.01481</td>
</tr>
<tr>
<td>ALL</td>
<td>331.283</td>
<td>8</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

**Jarque-Bera test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Skewness</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_austin</td>
<td>.60265</td>
<td>9.867</td>
<td>1</td>
<td>0.00168</td>
</tr>
<tr>
<td>D_dallas</td>
<td>.09996</td>
<td>0.271</td>
<td>1</td>
<td>0.60236</td>
</tr>
<tr>
<td>D_houston</td>
<td>-1.0444</td>
<td>29.635</td>
<td>1</td>
<td>0.00000</td>
</tr>
<tr>
<td>D_sa</td>
<td>.38019</td>
<td>3.927</td>
<td>1</td>
<td>0.04752</td>
</tr>
<tr>
<td>ALL</td>
<td>43.699</td>
<td>4</td>
<td></td>
<td>0.00000</td>
</tr>
</tbody>
</table>

**Skewness test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Kurtosis</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_austin</td>
<td>6.0807</td>
<td>64.458</td>
<td>1</td>
<td>0.00000</td>
</tr>
<tr>
<td>D_dallas</td>
<td>3.6896</td>
<td>3.229</td>
<td>1</td>
<td>0.07232</td>
</tr>
<tr>
<td>D_houston</td>
<td>8.6316</td>
<td>215.397</td>
<td>1</td>
<td>0.00000</td>
</tr>
<tr>
<td>D_sa</td>
<td>3.8139</td>
<td>4.499</td>
<td>1</td>
<td>0.03392</td>
</tr>
<tr>
<td>ALL</td>
<td>287.583</td>
<td>4</td>
<td></td>
<td>0.00000</td>
</tr>
</tbody>
</table>

**Kurtosis test**

The results indicate that we can strongly reject the null hypothesis of normally distributed errors. Most of the errors are both skewed and kurtotic.

**Impulse–response functions for VECMs**

With a model that we now consider acceptably well specified, we can use the `irf` commands to estimate and interpret the IRFs. Whereas IRFs from a stationary VAR die out over time, IRFs from a cointegrating VECM do not always die out. Because each variable in a stationary VAR has a time-invariant mean and finite, time-invariant variance, the effect of a shock to any one of these variables must die out so that the variable can revert to its mean. In contrast, the I(1) variables modeled in a cointegrating VECM are not mean reverting, and the unit moduli in the companion matrix imply that the effects of some shocks will not die out over time.

These two possibilities gave rise to new terms. When the effect of a shock dies out over time, the shock is said to be transitory. When the effect of a shock does not die out over time, the shock is said to be permanent.

Below we use `irf create` to estimate the IRFs and `irf graph` to graph two of the orthogonalized IRFs.
The graphs indicate that an orthogonalized shock to the average housing price in Austin has a permanent effect on the average housing price in San Antonio but that an orthogonalized shock to the average price of housing in Dallas has a transitory effect. According to this model, unexpected shocks that are local to the Austin housing market will have a permanent effect on the housing market in San Antonio, but unexpected shocks that are local to the Dallas housing market will have only a transitory effect on the housing market in San Antonio.

**Forecasting with VECMs**

Cointegrating VECMs are also used to produce forecasts of both the first-differenced variables and the levels of the variables. Comparing the variances of the forecast errors of stationary VARs with those from a cointegrating VECM reveals a fundamental difference between the two models. Whereas the variances of the forecast errors for a stationary VAR converge to a constant as the prediction horizon grows, the variances of the forecast errors for the levels of a cointegrating VECM diverge with the forecast horizon. (See sec. 6.5 of Lütkepohl [2005] for more about this result.) Because all the variables in the model for the first differences are stationary, the forecast errors for the dynamic forecasts of the first differences remain finite. In contrast, the forecast errors for the dynamic forecasts of the levels diverge to infinity.

We use `fcast compute` to obtain dynamic forecasts of the levels and `fcast graph` to graph these dynamic forecasts, along with their asymptotic confidence intervals.
As expected, the widths of the confidence intervals grow with the forecast horizon.

References


Beckett, S. 2013. *Introduction to Time Series Using Stata*. College Station, TX: Stata Press.


**Also see**

[TS] *irf* — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] *vec* — Vector error-correction models
vec fits a type of vector autoregression in which some of the variables are cointegrated by using Johansen’s (1995) maximum likelihood method. Constraints may be placed on the parameters in the cointegrating equations or on the adjustment terms. See [TS] vec intro for a list of commands that are used in conjunction with vec.

Quick start

Vector error-correction model for \(y_1\), \(y_2\), and \(y_3\) using \texttt{tsset} data

\begin{verbatim}
vec y1 y2 y3
\end{verbatim}

Use 4 lags for the underlying \textsc{VAR} model

\begin{verbatim}
vec y1 y2 y3, lags(4)
\end{verbatim}

Use 2 cointegrating equations

\begin{verbatim}
vec y1 y2 y3, lags(4) rank(2)
\end{verbatim}

Add a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data

\begin{verbatim}
vec y1 y2 y3, lags(4) rank(2) trend(trend)
\end{verbatim}

As above, but without a trend or a constant

\begin{verbatim}
vec y1 y2 y3, lags(4) rank(2) trend(none)
\end{verbatim}

Menu

Statistics \> Multivariate time series \> Vector error-correction model (VECM)
## Syntax

```
vec depvarlist [if] [in] [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rank(#)</code></td>
<td>use <code>#</code> cointegrating equations; default is <code>rank(1)</code></td>
</tr>
<tr>
<td><code>lags(#)</code></td>
<td>use <code>#</code> for the maximum lag in underlying VAR model</td>
</tr>
<tr>
<td><code>trend(constant)</code></td>
<td>include an unrestricted constant in model; the default</td>
</tr>
<tr>
<td><code>trend(rconstant)</code></td>
<td>include a restricted constant in model</td>
</tr>
<tr>
<td><code>trend(trend)</code></td>
<td>include a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data</td>
</tr>
<tr>
<td><code>trend(rtrend)</code></td>
<td>include a restricted trend in model</td>
</tr>
<tr>
<td><code>trend(none)</code></td>
<td>do not include a trend or a constant</td>
</tr>
<tr>
<td><code>bconstraints(constraints_{bc})</code></td>
<td>place <code>constraints_{bc}</code> on cointegrating vectors</td>
</tr>
<tr>
<td><code>acconstraints(constraints_{ac})</code></td>
<td>place <code>constraints_{ac}</code> on adjustment parameters</td>
</tr>
<tr>
<td><code>sindicators(varlist_{si})</code></td>
<td>include normalized seasonal indicator variables <code>varlist_{si}</code></td>
</tr>
<tr>
<td><code>noreduce</code></td>
<td>do not perform checks and corrections for collinearity among lags of dependent variables</td>
</tr>
<tr>
<td><code>level(#)</code></td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td><code>nobtable</code></td>
<td>do not report parameters in the cointegrating equations</td>
</tr>
<tr>
<td><code>noidtest</code></td>
<td>do not report the likelihood-ratio test of overidentifying restrictions</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>report adjustment parameters in separate table</td>
</tr>
<tr>
<td><code>pi</code></td>
<td>report parameters in ( \Pi = \alpha\beta' )</td>
</tr>
<tr>
<td><code>noptable</code></td>
<td>do not report elements of ( \Pi ) matrix</td>
</tr>
<tr>
<td><code>mai</code></td>
<td>report parameters in the moving-average impact matrix</td>
</tr>
<tr>
<td><code>noetable</code></td>
<td>do not report adjustment and short-run parameters</td>
</tr>
<tr>
<td><code>dforce</code></td>
<td>force reporting of short-run, beta, and alpha parameters when the parameters in beta are not identified; advanced option</td>
</tr>
<tr>
<td><code>nocnsreport</code></td>
<td>do not display constraints</td>
</tr>
<tr>
<td><code>display_options</code></td>
<td>control columns and column formats, row spacing, and line width</td>
</tr>
</tbody>
</table>

**Maximization**

```
maximize_options
```

control the maximization process; seldom used

```
coeflegend
```

display legend instead of statistics

---

vec does not allow gaps in the data.
You must tsset your data before using vec; see [TS] tsset.

varlist must contain at least two variables and may contain time-series operators; see [U] 11.4.4 Time-series varlists.
by, fp, rolling, statsby, and xi are allowed; see [U] 11.1.10 Prefix commands.

coeflegend does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.
Options

**Model**

- `rank(#)`: specifies the number of cointegrating equations; `rank(1)` is the default.
- `lags(#)` specifies the maximum lag to be included in the underlying VAR model. The maximum lag in a VECM is one smaller than the maximum lag in the corresponding VAR in levels; the number of lags must be greater than zero but small enough so that the degrees of freedom used up by the model are fewer than the number of observations. The default is `lags(2)`.
- `trend(trend_spec)` specifies which of Johansen’s five trend specifications to include in the model. These specifications are discussed in Specification of constants and trends below. The default is `trend(constant)`.
- `bconstraints(constraintsbc)` specifies the constraints to be placed on the parameters of the cointegrating equations. When no constraints are placed on the adjustment parameters—that is, when the `aconstraints()` option is not specified—the default is to place the constraints defined by Johansen’s normalization on the parameters of the cointegrating equations. When constraints are placed on the adjustment parameters, the default is not to place constraints on the parameters in the cointegrating equations.
- `aconstraints(constraintsac)` specifies the constraints to be placed on the adjustment parameters. By default, no constraints are placed on the adjustment parameters.

**Adv. model**

- `sindicators(varlistsi)` specifies the normalized seasonal indicator variables to include in the model. The indicator variables specified in this option must be normalized as discussed in Johansen (1995). If the indicators are not properly normalized, the estimator of the cointegrating vector does not converge to the asymptotic distribution derived by Johansen (1995). More details about how these variables are handled are provided in Methods and formulas. `sindicators()` cannot be specified with `trend(none)` or with `trend(rconstant)`.
- `noreduce` causes vec to skip the checks and corrections for collinearity among the lags of the dependent variables. By default, vec checks to see whether the current lag specification causes some of the regressions performed by vec to contain perfectly collinear variables; if so, it reduces the maximum lag until the perfect collinearity is removed.

**Reporting**

- `level(#)`; see [R] Estimation options.
- `nobtable` suppresses the estimation table for the parameters in the cointegrating equations. By default, vec displays the estimation table for the parameters in the cointegrating equations.
- `noidtest` suppresses the likelihood-ratio test of the overidentifying restrictions, which is reported by default when the model is overidentified.
- `alpha` displays a separate estimation table for the adjustment parameters, which is not displayed by default.
- `pi` displays a separate estimation table for the parameters in $\Pi = \alpha \beta'$, which is not displayed by default.
- `noptable` suppresses the estimation table for the elements of the $\Pi$ matrix, which is displayed by default when the parameters in the cointegrating equations are not identified.
- `mai` displays a separate estimation table for the parameters in the moving-average impact matrix, which is not displayed by default.
noetable suppresses the main estimation table that contains information about the estimated adjustment parameters and the short-run parameters, which is displayed by default.

dforce displays the estimation tables for the short-run parameters and $\alpha$ and $\beta$—if the last two are requested—when the parameters in $\beta$ are not identified. By default, when the specified constraints do not identify the parameters in the cointegrating equations, estimation tables are displayed only for $\Pi$ and the MAI.

nocnsreport; see [R] Estimation options.

display_options: noci, nopvvalues, vsquish, cformat(%fmt), pformat(%fmt), sformat(%fmt), and nolstretch; see [R] Estimation options.

Maximization

maximize_options: iterate(#), [no]log, trace, toltrace, tolerance(#), ltolerance(#), afrom(matrix_a), and bfrom(matrix_b); see [R] Maximize.

toltrace displays the relative differences for the log likelihood and the coefficient vector at every iteration. This option cannot be specified if no constraints are defined or if nolog is specified.

afrom(matrix_a) specifies a $1 \times (K \times r)$ row vector with starting values for the adjustment parameters, where $K$ is the number of endogenous variables and $r$ is the number of cointegrating equations specified in the rank() option. The starting values should be ordered as they are reported in e(alpha). This option cannot be specified if no constraints are defined.

bfrom(matrix_b) specifies a $1 \times (m_1 \times r)$ row vector with starting values for the parameters of the cointegrating equations, where $m_1$ is the number of variables in the trend-augmented system and $r$ is the number of cointegrating equations specified in the rank() option. (See Methods and formulas for more details about $m_1$.) The starting values should be ordered as they are reported in e(betavec). As discussed in Methods and formulas, for some trend specifications, e(beta) contains parameter estimates that are not obtained directly from the optimization algorithm. bfrom() should specify only starting values for the parameters reported in e(betavec). This option cannot be specified if no constraints are defined.

The following option is available with vec but is not shown in the dialog box:

coeflegend; see [R] Estimation options.

Remarks and examples

Remarks are presented under the following headings:

  Introduction
  Specification of constants and trends
  Collinearity

Introduction

VECMs are used to model the stationary relationships between multiple time series that contain unit roots. vec implements Johansen’s approach for estimating the parameters of a VECM.

[TS] vec intro reviews the basics of integration and cointegration and highlights why we need special methods for modeling the relationships between processes that contain unit roots. This manual entry assumes familiarity with the material in [TS] vec intro and provides examples illustrating how to use the vec command. See Johansen (1995), Hamilton (1994), and Becketti (2013) for more in-depth introductions to cointegration analysis.
Example 1

This example uses annual data on the average per-capita disposable personal income in the eight U.S. Bureau of Economic Analysis (BEA) regions of the United States. We use data from 1948–2002 in logarithms. Unit-root tests on these series fail to reject the null hypothesis that per-capita disposable income in each region contains a unit root. Because capital and labor can move easily between the different regions of the United States, we would expect that no one series will diverge from all the remaining series and that cointegrating relationships exist.

Below we graph the natural logs of average disposal income in the New England and the Southeast regions.

```
. use https://www.stata-press.com/data/r16/rdinc
. line ln_ne ln_se year
```

![Graph showing the natural logs of average disposal income in New England and Southeast regions](image)

The graph indicates a differential between the two series that shrinks between 1960 and about 1980 and then grows until it stabilizes around 1990. We next estimate the parameters of a bivariate VECM with one cointegrating relationship.
Vector error-correction model

Sample: 1950 - 2002
Number of obs = 53

AIC = -11.00462
Log likelihood = 300.6224
HQIC = -10.87595
Det(Sigma_ml) = 4.06e-08
SBIC = -10.67004

Equation Parms RMSE R-sq chi2 P>chi2
D_ln_ne 4 .017896 0.9313 664.4668 0.0000
D_ln_se 4 .018723 0.9292 642.7179 0.0000

|            | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|------------|-------|-----------|------|------|----------------------|
| D_ln_ne    |       |           |      |      |                      |
| _ce1       |       |           |      |      |                      |
| L1.        | -.4337524 | .0721365  | -6.01 | 0.000 | -.5751373 to -.2923675 |
| ln_ne      | .7168658  | .1889085  | 3.79  | 0.000 | .3466119 to 1.08712   |
| L1.        | -.6748754 | .2117975  | -3.19 | 0.001 | -1.089991 to -.2597599 |
| ln_se      | -.0019846 | .0080291  | -0.25 | 0.805 | -.0177214 to .0137521 |
| _cons      | .002429  | .0084004  | 0.29  | 0.772 | -.0140355 to .0188936 |

D_ln_se

|            | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|------------|-------|-----------|------|------|----------------------|
| _ce1       |       |           |      |      |                      |
| L1.        | -.3543935 | .0754725  | -4.70 | 0.000 | -.5023168 to -.2064701 |
| ln_ne      | .3366786  | .1976448  | 1.70  | 0.088 | -.050698 to .7240553  |
| L1.        | -.1605811 | .2215922  | -0.72 | 0.469 | -.5948939 to .2737317 |
| ln_se      | .002429  | .0084004  | 0.29  | 0.772 | -.0140355 to .0188936 |
| _cons      | .002429  | .0084004  | 0.29  | 0.772 | -.0140355 to .0188936 |

Cointegrating equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ce1</td>
<td>1</td>
<td>29805.02</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Identification: beta is exactly identified
Johansen normalization restriction imposed

| beta | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|------|-------|-----------|------|------|----------------------|
| _ce1 |       |           |      |      |                      |
| ln_ne| .9433708 | .0054643  | -172.64 | 0.000 | -.9540807 to -.9326609 |
| ln_se| -.8964065 | . .       | .     | .   | .                    |
| _cons| .       | .         | .     | .   | .                    |
The results indicate strong support for a cointegrating equation such that

$$\ln ne - 0.943 \ln se - 0.896$$

should be a stationary series. Identification of the parameters in the cointegrating equation is achieved by constraining some of them to be fixed, and fixed parameters do not have standard errors. In this example, the coefficient on $\ln ne$ has been normalized to 1, so its standard error is missing. As discussed in *Methods and formulas*, the constant term in the cointegrating equation is not directly estimated in this trend specification but rather is backed out from other estimates. Not all the elements of the VCE that correspond to this parameter are readily available, so the standard error for the _cons parameter is missing.

To get a better idea of how our model fits, we predict the cointegrating equation and graph it over time:

```
predict ce, ce
line ce year
```

![Predicted cointegrated equation](image)

Although the predicted cointegrating equation has the right appearance for the time before the mid-1960s, afterward the predicted cointegrating equation does not look like a stationary series. A better model would account for the trends in the size of the differential.

As discussed in [*vec intro*], simply normalizing one of the coefficients to be one is sufficient to identify the parameters of the single cointegrating vector. When there is more than one cointegrating equation, more restrictions are required.

**Example 2**

We have data on monthly unemployment rates in Indiana, Illinois, Kentucky, and Missouri from January 1978 through December 2003. We suspect that factor mobility will keep the unemployment rates in equilibrium. The following graph plots the data.
The graph shows that although the series do appear to move together, the relationship is not as clear as in the previous example. There are periods when Indiana has the highest rate and others when Indiana has the lowest rate. Although the Kentucky rate moves closely with the other series for most of the sample, there is a period in the mid-1980s when the unemployment rate in Kentucky does not fall at the same rate as the other series.

We will model the series with two cointegrating equations and no linear or quadratic time trends in the original series. Because we are focusing on the cointegrating vectors, we use the `noetable` option to suppress displaying the short-run estimation table.
. vec missouri indiana kentucky illinois, trend(rconstant) rank(2) lags(4)
> noetable

Vector error-correction model

Sample: 1978m5 - 2003m12  Number of obs = 308
AIC = -2.306048
Log likelihood = 417.1314  HQIC = -2.005818
Det(Sigma_ml) = 7.83e-07  SBIC = -1.555184

Cointegrating equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ce1</td>
<td>2</td>
<td>133.3885</td>
<td>0.0000</td>
</tr>
<tr>
<td>_ce2</td>
<td>2</td>
<td>195.6324</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Identification: beta is exactly identified

Johansen normalization restrictions imposed

| beta     | Coef.  | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|----------|--------|-----------|------|------|---------------------|
| _ce1     |        |           |      |      |                     |
| missouri | 1      |           |      | 0.78 | -1.36302 - 0.5868787|
| indiana  | 0      |           |      |      |                     |
| kentucky | .3493902 | .2005537 | 1.74 | 0.081| -.0436879 - .7424683|
| illinois | -1.135152 | .2069063 | -5.49| 0.000| -1.540681 - .7296235|
| _cons    | -.3880707 | .4974323 | -0.78 | 0.435| -1.36302 - .5868787|

| _ce2     |        |           |      |      |                     |
| missouri | -1.11e-16 |           |      |      |                     |
| indiana  | 1      |           |      |      |                     |
| kentucky | .2059473 | .2718678 | 0.76 | 0.449| -.3269038 - .7387985|
| illinois | -1.51962 | .2804792 | -5.42 | 0.000| -2.069349 - .9698907|
| _cons    | 2.92857 | .6743122 | 4.34 | 0.000| 1.606942 - 4.250197 |

Except for the coefficients on kentucky in the two cointegrating equations and the constant term in the first, all the parameters are significant at the 5% level. We can refit the model with the Johansen normalization and the overidentifying constraint that the coefficient on kentucky in the second cointegrating equation is zero.

. constraint define 1 [_ce1]missouri = 1
. constraint define 2 [_ce1]indiana = 0
. constraint define 3 [_ce2]missouri = 0
. constraint define 4 [_ce2]indiana = 1
. constraint define 5 [_ce2]kentucky = 0
. vec missouri indiana kentucky illinois, trend(rconstant) rank(2)
> lags(4) noetable bconstraints(1/5)
Iteration 1: log likelihood =  416.97177
(output omitted)
Iteration 20: log likelihood =  416.9744
Vector error-correction model

Sample:  1978m5 - 2003m12 Number of obs = 308
AIC = -2.311522
Log likelihood = 416.9744 HQIC = -2.016134
Det(Sigma_ml) = 7.84e-07 SBIC = -1.572769

Cointegrating equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ce1</td>
<td>2</td>
<td>145.233</td>
<td>0.0000</td>
</tr>
<tr>
<td>_ce2</td>
<td>1</td>
<td>209.9344</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Identification: beta is overidentified

(1) [_ce1]missouri = 1
(2) [_ce1]indiana = 0
(3) [_ce2]missouri = 0
(4) [_ce2]indiana = 1
(5) [_ce2]kentucky = 0

| beta        | Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|-------------|--------|-----------|-------|------|---------------------|
| _ce1        |        |           |       |      |                     |
| missouri    | 1      | .         | .     | .    |                     |
| indiana     | 0      | (omitted) |       | .    |                     |
| kentucky    | .2521685 | .1649653 | 1.53  | 0.126 | -.0711576           | .5754946 |
| illinois    | -1.037453 | .1734165 | -5.98 | 0.000 | -1.377343           | -.6975626 |
| _cons       | -.3891102 | .4726968 | -0.82 | 0.410 | -1.315579           | .5373586 |
| _ce2        |        |           |       |      |                     |
| missouri    | 0      | (omitted) |       | .    |                     |
| indiana     | 1      | .         | .     | .    |                     |
| kentucky    | 0      | (omitted) |       | .    |                     |
| illinois    | -1.314265 | .0907071 | -14.49 | 0.000 | -1.492048           | -1.136483 |
| _cons       | 2.937016 | .6448924 | 4.55  | 0.000 | 1.67305             | 4.200982 |

LR test of identifying restrictions: chi2(1) = .3139  Prob > chi2 = 0.575

The test of the overidentifying restriction does not reject the null hypothesis that the restriction is valid, and the $p$-value on the coefficient on kentucky in the first cointegrating equation indicates that it is not significant. We will leave the variable in the model and attribute the lack of significance to whatever caused the kentucky series to temporarily rise above the others from 1985 until 1990, though we could instead consider removing kentucky from the model.

Next, we look at the estimates of the adjustment parameters. In the output below, we replay the previous results. We specify the alpha option so that vec will display an estimation table for the estimates of the adjustment parameters, and we specify noetable to suppress the table for the parameters of the cointegrating equations because we have already looked at those.
. vec, alpha notable notable

Vector error-correction model

Sample: 1978m5 - 2003m12  Number of obs = 308
Log likelihood = 416.9744
AIC = -2.311522
Log likelihood = 416.9744  HQIC = -2.016134
Det(Sigma_ml) = 7.84e-07  SBIC = -1.572769

Adjustment parameters

<table>
<thead>
<tr>
<th>Equation</th>
<th>Parms</th>
<th>chi2</th>
<th>P&gt;chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_missouri</td>
<td>2</td>
<td>19.39607</td>
<td>0.0001</td>
</tr>
<tr>
<td>D_indiana</td>
<td>2</td>
<td>6.426086</td>
<td>0.0402</td>
</tr>
<tr>
<td>D_kentucky</td>
<td>2</td>
<td>8.524901</td>
<td>0.0141</td>
</tr>
<tr>
<td>D_illinois</td>
<td>2</td>
<td>22.32893</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

alpha

| D_missouri   | Coef.   | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|--------------|---------|-----------|-----|------|----------------------|
| _ce1 L1.     | -.0683152 | .0185763 | -3.68 | 0.000 | -.1047242 -.0319063 |
| _ce2 L1.     | .0405613  | .0112417 | 3.61  | 0.000 | .018528 .0625946    |

| D_indiana    | Coef.   | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|--------------|---------|-----------|-----|------|----------------------|
| _ce1 L1.     | -.0342096 | .0220955 | -1.55 | 0.122 | -.0775159 .0090967  |
| _ce2 L1.     | .0325804  | .0133713 | 2.44  | 0.015 | .0063732 .0587877   |

| D_kentucky   | Coef.   | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|--------------|---------|-----------|-----|------|----------------------|
| _ce1 L1.     | -.0482012 | .0231633 | -2.08 | 0.037 | -.0936004 -.0028021 |
| _ce2 L1.     | .0374395  | .0140175 | 2.67  | 0.008 | .0099657 .0649133   |

| D_illinois   | Coef.   | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|--------------|---------|-----------|-----|------|----------------------|
| _ce1 L1.     | .0138224  | .0227041 | 0.61  | 0.543 | -.0306768 .0583215  |
| _ce2 L1.     | .0567664  | .0137396 | 4.13  | 0.000 | .0298373 .0836955   |

LR test of identifying restrictions: chi2(1) = .3139  Prob > chi2 = 0.575

All the coefficients are significant at the 5% level, except those on Indiana and Illinois in the first cointegrating equation. From an economic perspective, the issue is whether the unemployment rates in Indiana and Illinois adjust when the first cointegrating equation is out of equilibrium. We could impose restrictions on one or both of those parameters and refit the model, or we could just decide to use the current results.
Technical note

vec can be used to fit models in which the parameters in $\beta$ are not identified, in which case only the parameters in $\Xi$ and the moving-average impact matrix $C$ are identified. When the parameters in $\beta$ are not identified, the values of $\hat{\beta}$ and $\hat{\alpha}$ can vary depending on the starting values. However, the estimates of $\Xi$ and $C$ are identified and have known asymptotic distributions. This method is valid because these additional normalization restrictions impose no restriction on $\Xi$ or $C$.

Specification of constants and trends

As discussed in [TS] vec intro, allowing for a constant term and linear time trend allow us to write the VECM as

$$\Delta y_t = \alpha (\beta y_{t-1} + \mu + \rho t) + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \gamma + \tau t + \epsilon_t$$

Five different trend specifications are available:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>trend</td>
<td>none</td>
<td>$H(r)$</td>
</tr>
<tr>
<td>r trend</td>
<td>$\tau = 0$</td>
<td>$H^*(r)$</td>
</tr>
<tr>
<td>constant</td>
<td>$\rho = 0$, and $\tau = 0$</td>
<td>$H_1(r)$</td>
</tr>
<tr>
<td>r constant</td>
<td>$\rho = 0$, $\gamma = 0$ and $\tau = 0$</td>
<td>$H^*_1(r)$</td>
</tr>
<tr>
<td>none</td>
<td>$\mu = 0$, $\rho = 0$, $\gamma = 0$, and $\tau = 0$</td>
<td>$H_2(r)$</td>
</tr>
</tbody>
</table>

trend(trend) allows for a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data. A linear trend in the cointegrating equations implies that the cointegrating equations are assumed to be trend stationary.

trend(r trend) defines a restricted trend model that excludes linear trends in the differenced data but allows for linear trends in the cointegrating equations. As in the previous case, a linear trend in a cointegrating equation implies that the cointegrating equation is trend stationary.

trend(constant) defines a model with an unrestricted constant. This allows for a linear trend in the undifferenced data and cointegrating equations that are stationary around a nonzero mean. This is the default.

trend(r constant) defines a model with a restricted constant in which there is no linear or quadratic trend in the undifferenced data. A nonzero $\mu$ allows for the cointegrating equations to be stationary around nonzero means, which provide the only intercepts for differenced data. Seasonal indicators are not allowed with this specification.

trend(none) defines a model that does not include a trend or a constant. When there is no trend or constant, the cointegrating equations are restricted to being stationary with zero means. Also, after adjusting for the effects of lagged endogenous variables, the differenced data are modeled as having mean zero. Seasonal indicators are not allowed with this specification.
Technical note

vec uses a switching algorithm developed by Boswijk (1995) to maximize the log-likelihood function when constraints are placed on the parameters. The starting values affect both the ability of the algorithm to find a maximum and its speed in finding that maximum. By default, vec uses the parameter estimates that correspond to Johansen’s normalization. Sometimes, other starting values will cause the algorithm to find a maximum faster.

To specify starting values for the parameters in $\alpha$, we specify a $1 \times (K \times r)$ matrix in the `afrom()` option. Specifying starting values for the parameters in $\beta$ is slightly more complicated. As explained in Methods and formulas, specifying `trend(constant), trend(rtrend), or trend(trend)` causes some of the estimates of the trend parameters appearing in $\hat{\beta}$ to be “backed out”. The switching algorithm estimates only the parameters of the cointegrating equations whose estimates are stored in `e(betavec)`. For this reason, only the parameters stored in `e(betavec)` can have their initial values set via `bfrom()`.

The table below describes which trend parameters in the cointegrating equations are estimated by the switching algorithm for each of the five specifications.

<table>
<thead>
<tr>
<th>Trend specification</th>
<th>Trend parameters in cointegrating equations</th>
<th>Trend parameter estimated via switching algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>rconstant</td>
<td>_cons</td>
<td>_cons</td>
</tr>
<tr>
<td>constant</td>
<td>_cons</td>
<td>none</td>
</tr>
<tr>
<td>rtrend</td>
<td>_cons, _trend</td>
<td>_trend</td>
</tr>
<tr>
<td>trend</td>
<td>_cons, _trend</td>
<td>none</td>
</tr>
</tbody>
</table>

Collinearity

As expected, collinearity among variables causes some parameters to be unidentified numerically. If vec encounters perfect collinearity among the dependent variables, it exits with an error.

In contrast, if vec encounters perfect collinearity that appears to be due to too many lags in the model, vec displays a warning message and reduces the maximum lag included in the model in an effort to find a model with fewer lags in which all the parameters are identified by the data. Specifying the `noreduce` option causes vec to skip over these additional checks and corrections for collinearity. Thus the `noreduce` option can be used to force the estimation to proceed when not all the parameters are identified by the data. When some parameters are not identified because of collinearity, the results cannot be interpreted but can be used to find the source of the collinearity.
Stored results

vec stores the following in e():

Scalars
- e(N) number of observations
- e(k_rank) number of unconstrained parameters
- e(k_eq) number of equations in e(b)
- e(k_dv) number of dependent variables
- e(k_ce) number of cointegrating equations
- e(n_lags) number of lags
- e(df_m) model degrees of freedom
- e(ll) log likelihood
- e(chi2_res) value of test of overidentifying restrictions
- e(df_lr) degrees of freedom of the test of overidentifying restrictions
- e(beta_iden) 1 if the parameters in \( \beta \) are identified and 0 otherwise
- e(beta_icnt) number of independent restrictions placed on \( \beta \)
- e(k_eq) number of variables in equation #
- e(df_m#) model degrees of freedom in equation #
- e(r2#) \( R^2 \) of equation #
- e(chi2#) \( \chi^2 \) statistic for equation #
- e(rmse#) RMSE of equation #
- e(aic) value of AIC
- e(hqic) value of HQIC
- e(sbic) value of SBIC
- e(tmin) minimum time
- e(tmax) maximum time
- e(detsig_ml) determinant of the estimated covariance matrix
- e(rank) rank of \( e(V) \)
- e(converge) 1 if the switching algorithm converged, 0 if it did not converge

Macros
- e(cmd) vec
- e(cmdline) command as typed
- e(trend) trend specified
- e(tsfmt) format of the time variable
- e(tvar) variable denoting time within groups
- e(endog) endogenous variables
- e(covariates) list of covariates
- e(eqnames) equation names
- e(cenames) names of cointegrating equations
- e(reduce_opt) noreduce, if noreduce is specified
- e(reduce_lags) list of maximum lags to which the model has been reduced
- e(title) title in estimation output
- e(aconstraints) constraints placed on \( \alpha \)
- e(bconstraints) constraints placed on \( \beta \)
- e(sindicators) seasonal indicator variables
- e(properties) b V
- e(predict) program used to implement predict
- e(marginsok) predictions allowed by margins
- e(marginsnotok) predictions disallowed by margins
- e(marginsdefault) default predict() specification for margins
Methods and formulas

Methods and formulas are presented under the following headings:

General specification of the VECM
The log-likelihood function
   Unrestricted trend
   Restricted trend
   Unrestricted constant
   Restricted constant
   No trend
Estimation with Johansen identification
Estimation with constraints: $\beta$ identified
Estimation with constraints: $\beta$ not identified
Formulas for the information criteria
Formulas for predict

General specification of the VECM

vec estimates the parameters of a VECM that can be written as

$$\Delta y_t = \alpha \beta' y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + v + \delta t + w_1 s_1 + \cdots + w_m s_m + \epsilon_t \quad (1)$$

where

$y_t$ is a $K \times 1$ vector of endogenous variables,
$\alpha$ is a $K \times r$ matrix of parameters,
$\beta$ is a $K \times r$ matrix of parameters,
$\Gamma_1, \ldots, \Gamma_{p-1}$ are $K \times K$ matrices of parameters,
$v$ is a $K \times 1$ vector of parameters,
$\delta$ is a $K \times 1$ vector of trend coefficients,
t is a linear time trend,
s_1, \ldots, s_m$ are orthogonalized seasonal indicators specified in the sindicators() option, and
w_1, \ldots, w_m are $K \times 1$ vectors of coefficients on the orthogonalized seasonal indicators.
There are two types of deterministic elements in (1): the trend, $v + \delta t$, and the orthogonalized seasonal terms, $w_1 s_1 + \cdots + w_m s_m$. Johansen (1995, chap. 11) shows that inference about the number of cointegrating equations is based on nonstandard distributions and that the addition of any term that generalizes the deterministic specification in (1) changes the asymptotic distributions of the statistics used for inference on the number of cointegrating equations and the asymptotic distribution of the ML estimator of the cointegrating equations. In fact, Johansen (1995, 84) notes that including event indicators causes the statistics used for inference on the number of cointegrating equations to have asymptotic distributions that must be computed case by case. For this reason, event indicators may not be specified in the present version of vec.

If seasonal indicators are included in the model, they cannot be collinear with a constant term. If they are collinear with a constant term, one of the indicator variables is omitted.

As discussed in Specification of constants and trends, we can reparameterize the model as

$$
\Delta y_t = \alpha (\beta y_{t-1} + \mu + \rho t) + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \gamma + \tau t + \epsilon_t
$$

(2)

The log-likelihood function

We can maximize the log-likelihood function much more easily by writing it in concentrated form. In fact, as discussed below, in the simple case with the Johansen normalization on $\beta$ and no constraints on $\alpha$, concentrating the log-likelihood function produces an analytical solution for the parameter estimates.

To concentrate the log likelihood, rewrite (2) as

$$
Z_{0t} = \alpha \tilde{\beta}' Z_{1t} + \Psi Z_{2t} + \epsilon_t
$$

(3)

where $Z_{0t}$ is a $K \times 1$ vector of variables $\Delta y_t$, $\alpha$ is the $K \times r$ matrix of adjustment coefficients, and $\epsilon_t$ is a $K \times 1$ vector of independent and identically distributed normal vectors with mean 0 and contemporaneous covariance matrix $\Omega$. $Z_{1t}$, $Z_{2t}$, $\tilde{\beta}$, and $\Psi$ depend on the trend specification and are defined below.

The log-likelihood function for the model in (3) is

$$
L = -\frac{1}{2} \left\{ TK \ln(2\pi) + T \ln(|\Omega|) \\
+ \sum_{t=1}^{T} (Z_{0t} - \alpha \tilde{\beta}' Z_{1t} - \Psi Z_{2t})' \Omega^{-1} (Z_{0t} - \alpha \tilde{\beta}' Z_{1t} - \Psi Z_{2t}) \right\}
$$

(4)

with the constraints that $\alpha$ and $\tilde{\beta}$ have rank $r$.

Johansen (1995, chap. 6), building on Anderson (1951), shows how the $\Psi$ parameters can be expressed as analytic functions of $\alpha$, $\tilde{\beta}$, and the data, yielding the concentrated log-likelihood function

$$
L_c = -\frac{1}{2} \left\{ TK \ln(2\pi) + T \ln(|\Omega|) \\
+ \sum_{t=1}^{T} (R_{0t} - \alpha \tilde{\beta}' R_{1t})' \Omega^{-1} (R_{0t} - \alpha \tilde{\beta}' R_{1t}) \right\}
$$

(5)
where
\[ M_{ij} = T^{-1} \sum_{t=1}^{T} Z_{it} Z'_{jt}, \quad i, j \in \{0, 1, 2\}; \]
\[ R_{0t} = Z_{0t} - M_{02} M^{-1}_{22} Z_{2t}; \text{ and} \]
\[ R_{1t} = Z_{1t} - M_{12} M^{-1}_{22} Z_{2t}. \]

The definitions of \( Z_{1t}, Z_{2t}, \tilde{\beta}, \text{ and } \Psi \) change with the trend specifications, although some of their components stay the same.

**Unrestricted trend**

When the trend in the VECM is unrestricted, we can define the variables in (3) directly in terms of the variables in (1):

\[ Z_{1t} = y_{t-1} \text{ is } K \times 1 \]
\[ Z_{2t} = (\Delta y'_{t-1}, \ldots, \Delta y'_{t-p+1}, 1, t, s_1, \ldots, s_m)' \text{ is } \{K(p - 1) + 2 + m\} \times 1; \]
\[ \Psi = (\Gamma_1, \ldots, \Gamma_{p-1}, v, \delta, w_1, \ldots, w_m) \text{ is } K \times \{K(p - 1) + 2 + m\} \]
\[ \tilde{\beta} = \beta \text{ is the } K \times r \text{ matrix composed of the } r \text{ cointegrating vectors.} \]

In the unrestricted trend specification, \( m_1 = K, m_2 = K(p - 1) + 2 + m, \) and there are \( n_{\text{parms}} = K(r + 1) + K\{K(p - 1) + 2 + m\} \) parameters in (3).

**Restricted trend**

When there is a restricted trend in the VECM in (2), \( \tau = 0, \) but the intercept \( v = \alpha \mu + \gamma \) is unrestricted. The VECM with the restricted trend can be written as

\[ \Delta y_t = \alpha(\beta', \rho) \left( y_{t-1}'/t \right) + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + v + w_1 s_1 + \cdots + w_m s_m + \epsilon_t \]

This equation can be written in the form of (3) by defining

\[ Z_{1t} = (y'_{t-1}, t)' \text{ is } (K + 1) \times 1 \]
\[ Z_{2t} = (\Delta y'_{t-1}, \ldots, \Delta y'_{t-p+1}, 1, s_1, \ldots, s_m)' \text{ is } \{K(p - 1) + 1 + m\} \times 1 \]
\[ \Psi = (\Gamma_1, \ldots, \Gamma_{p-1}, v, w_1, \ldots, w_m) \text{ is } K \times \{K(p - 1) + 1 + m\} \]
\[ \tilde{\beta} = (\beta', \rho)' \text{ is the } (K + 1) \times r \text{ matrix composed of the } r \text{ cointegrating vectors and the } r \text{ trend coefficients } \rho \]

In the restricted trend specification, \( m_1 = K + 1, m_2 = \{K(p - 1) + 1 + m\}, \) and there are \( n_{\text{parms}} = K(r + 1) + K\{K(p - 1) + 1 + m\} \) parameters in (3).

**Unrestricted constant**

An unrestricted constant in the VECM in (2) is equivalent to setting \( \delta = 0 \) in (1), which can be written in the form of (3) by defining

\[ Z_{1t} = y_{t-1} \text{ is } (K \times 1) \]
\[ Z_{2t} = (\Delta y'_{t-1}, \ldots, \Delta y'_{t-p+1}, 1, s_1, \ldots, s_m)' \text{ is } \{K(p - 1) + 1 + m\} \times 1; \]
\[ \Psi = (\Gamma_1, \ldots, \Gamma_{p-1}, v, w_1, \ldots, w_m) \text{ is } K \times \{K(p - 1) + 1 + m\} \]
\[ \tilde{\beta} = \beta \text{ is the } K \times r \text{ matrix composed of the } r \text{ cointegrating vectors} \]
In the unrestricted constant specification, \( m_1 = K, m_2 = \{K(p - 1) + 1 + m\} \), and there are \( n_{\text{parms}} = Kr + K r + K \{K(p - 1) + 1 + m\} \) parameters in (3).

**Restricted constant**

When there is a restricted constant in the VECM in (2), it can be written in the form of (3) by defining

\[
Z_{1t} = (y'_{t-1}, 1)' \text{ is } (K + 1) \times 1 \\
Z_{2t} = (\Delta y'_{t-1}, \ldots, \Delta y'_{t-p+1})' \text{ is } K(p - 1) \times m \\
\Psi = (\Gamma_1, \ldots, \Gamma_{p-1}) \text{ is } K \times K(p - 1) \\
\tilde{\beta} = (\beta', \mu)' \text{ is the } (K + 1) \times r \text{ matrix composed of the } r \text{ cointegrating vectors and the } r \text{ constants in the cointegrating relations.}
\]

In the restricted trend specification, \( m_1 = K + 1, m_2 = K(p - 1) \), and there are \( n_{\text{parms}} = Kr + (K + 1)r + K \{K(p - 1)\} \) parameters in (3).

**No trend**

When there is no trend in the VECM in (2), it can be written in the form of (3) by defining

\[
Z_{1t} = y_{t-1} \text{ is } K \times 1 \\
Z_{2t} = (\Delta y'_{t-1}, \ldots, \Delta y'_{t-p+1})' \text{ is } K(p - 1) + m \times 1 \\
\Psi = (\Gamma_1, \ldots, \Gamma_{p-1}) \text{ is } K \times K(p - 1) \\
\tilde{\beta} = \beta \text{ is } K \times r \text{ matrix of } r \text{ cointegrating vectors}
\]

In the no-trend specification, \( m_1 = K, m_2 = K(p - 1) \), and there are \( n_{\text{parms}} = Kr + K r + K \{K(p - 1)\} \) parameters in (3).

**Estimation with Johansen identification**

Not all the parameters in \( \alpha \) and \( \tilde{\beta} \) are identified. Consider the simple case in which \( \tilde{\beta} \) is \( K \times r \) and let \( Q \) be a nonsingular \( r \times r \) matrix. Then

\[
\alpha \tilde{\beta}' = \alpha QQ^{-1}\tilde{\beta}' = \alpha Q(\tilde{\beta}Q'^{-1})' = \hat{\alpha}\hat{\beta}'
\]

Substituting \( \hat{\alpha}\hat{\beta}' \) into the log likelihood in (5) for \( \alpha \tilde{\beta}' \) would not change the value of the log likelihood, so some a priori identification restrictions must be found to identify \( \alpha \) and \( \tilde{\beta} \). As discussed in Johansen (1995, chap. 5 and 6) and Boswijk (1995), if the restrictions exactly identify or overidentify \( \tilde{\beta} \), the estimates of the unconstrained parameters in \( \tilde{\beta} \) will be superconsistent, meaning that the estimates of the free parameters in \( \tilde{\beta} \) will converge at a faster rate than estimates of the short-run parameters in \( \alpha \) and \( \Gamma_i \). This allows the distribution of the estimator of the short-run parameters to be derived conditional on the estimated \( \tilde{\beta} \).

Johansen (1995, chap. 6) has proposed a normalization method for use when theory does not provide sufficient a priori restrictions to identify the cointegrating vector. This method has become widely adopted by researchers. Johansen’s identification scheme is

\[
\tilde{\beta}' = (I_r, \beta')
\]

where \( I_r \) is the \( r \times r \) identity matrix and \( \tilde{\beta} \) is a \( (m_1 - r) \times r \) matrix of identified parameters.
Johansen’s identification method places \( r^2 \) linearly independent constraints on the parameters in \( \tilde{\beta} \), thereby defining an exactly identified model. The total number of freely estimated parameters is 
\[
\text{parms} - r^2 = \{K + m_2 + (K + m_1 - r) r\},
\]
and the degrees of freedom \( d \) is calculated as the integer part of \((\text{parms} - r^2)/K\).

When only the rank and the Johansen identification restrictions are placed on the model, we can further manipulate the log likelihood in (5) to obtain analytic formulas for the parameters in \( \tilde{\beta}, \alpha \), and \( \Omega \). For a given value of \( \tilde{\beta} \), \( \alpha \) and \( \Omega \) can be found by regressing \( R_{0t} \) on \( \tilde{\beta}'R_{1t} \). This allows a further simplification of the problem in which
\[
\alpha(\tilde{\beta}) = S_{01}\tilde{\beta}(\tilde{\beta}'S_{11}\tilde{\beta})^{-1}
\]
\[
\Omega(\tilde{\beta}) = S_{00} - S_{01}\tilde{\beta}(\tilde{\beta}'S_{11}\tilde{\beta})^{-1}\tilde{\beta}'S_{10}
\]
\[
S_{ij} = \left(\frac{1}{T}\right)\sum_{t=1}^{T} R_{it}R_{jt}' \quad i, j \in \{0, 1\}
\]

Johansen (1995) shows that by inserting these solutions into equation (5), \( \hat{\beta} \) is given by the \( r \) eigenvectors \( v_1, \ldots, v_r \) corresponding to the \( r \) largest eigenvalues \( \lambda_1, \ldots, \lambda_r \) that solve the generalized eigenvalue problem
\[
|\lambda_i S_{11} - S_{10}S_{00}^{-1}S_{01}| = 0
\]

The eigenvectors corresponding to \( \lambda_1, \ldots, \lambda_r \) that solve (7) are the unidentified parameter estimates. To impose the identification restrictions in (6), we normalize the eigenvectors such that
\[
\lambda_i S_{11}v_i = S_{01}S_{00}^{-1}S_{01}v_i
\]
and
\[
v_i'S_{11}v_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}
\]

At the optimum the log-likelihood function with the Johansen identification restrictions can be expressed in terms of \( T, K, S_{00} \), and the \( r \) largest eigenvalues
\[
L_e = -\frac{1}{2}T\left\{K \ln(2\pi) + K + \ln(|S_{00}|) + \sum_{i=1}^{r} \ln(1 - \hat{\lambda}_i) \right\}
\]
where the \( \hat{\lambda}_i \) are the eigenvalues that solve (7), (8), and (9).

Using the normalized \( \tilde{\beta} \), we can then obtain the estimates
\[
\hat{\alpha} = S_{01}\tilde{\beta}(\tilde{\beta}'S_{11}\tilde{\beta})^{-1}
\]
and
\[
\hat{\Omega} = S_{00} - \hat{\alpha}\tilde{\beta}'S_{10}
\]

Let \( \hat{\beta}_y \) be a \( K \times r \) matrix that contains the estimates of the parameters in \( \beta \) in (1), \( \hat{\beta}_y \) differs from \( \tilde{\beta} \) in that any trend parameter estimates are omitted from \( \tilde{\beta} \). We can then use \( \hat{\beta}_y \) to obtain predicted values for the \( r \) nondemeaned cointegrating equations
\[
\hat{E}_t = \hat{\beta}_y y_t
\]
The $r$ series in $\widehat{E}_t$ are called the predicted, nondemeaned cointegrating equations because they still contain the terms $\mu$ and $\rho$. We want to work with the predicted, demeaned cointegrating equations. Thus we need estimates of $\mu$ and $\rho$. In the trend(rconstant) specification, the algorithm directly produces the estimator $\widehat{\mu}$. Similarly, in the trend(rtrend) specification, the algorithm directly produces the estimator $\widehat{\rho}$. In the remaining cases, to back out estimates of $\mu$ and $\rho$, we need estimates of $v$ and $\delta$, which we can obtain by estimating the parameters of the following VAR:

$$\Delta y_t = \alpha \widehat{E}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + v + \delta t + w_1 s_1 + \cdots + w_m s_m + \epsilon_t$$

Depending on the trend specification, we use $\widehat{\alpha}$ to back out the estimates of

$$\widehat{\mu} = (\widehat{\alpha}' \widehat{\alpha})^{-1} \widehat{\alpha}' \widehat{v}$$

$$\widehat{\rho} = (\widehat{\alpha}' \widehat{\alpha})^{-1} \widehat{\alpha}' \widehat{\delta}$$

if they are not already in $\widehat{\beta}$ and are included in the trend specification.

We then augment $\widehat{\beta}_y$ to

$$\widehat{\beta}_f' = (\widehat{\beta}_y', \widehat{\mu}, \widehat{\rho})$$

where the estimates of $\widehat{\mu}$ and $\widehat{\rho}$ are either obtained from $\widehat{\beta}$ or backed out using (12) and (13). We next use $\widehat{\beta}_f$ to obtain the $r$ predicted, demeaned cointegrating equations, $\widehat{E}_t$, via

$$\widehat{E}_t = \widehat{\beta}_f' (y_t', 1, t)'$$

We last obtain estimates of all the short-run parameters from the VAR:

$$\Delta y_t = \alpha \widehat{E}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \gamma + \tau t + w_1 s_1 + \cdots + w_m s_m + \epsilon_t$$

Because the estimator $\widehat{\beta}_f$ converges in probability to its true value at a rate faster than $T^{-\frac{1}{2}}$, we can take our estimated $\widehat{E}_{t-1}$ as given data in (14). This allows us to estimate the variance–covariance (VCE) matrix of the estimates of the parameters in (14) by using the standard VAR VCE estimator. Equation (11) can be used to obtain consistent estimates of all the parameters and of the VCE of all the parameters, except $v$ and $\delta$. The standard VAR VCE of $\widehat{v}$ and $\widehat{\delta}$ is incorrect because these estimates converge at a faster rate. This is why it is important to use the predicted, demeaned cointegrating equations, $\widehat{E}_{t-1}$, when estimating the short-run parameters and trend terms. In keeping with the cointegration literature, vec makes a small-sample adjustment to the VCE estimator so that the divisor is $(T - d)$ instead of $T$, where $d$ represents the degrees of freedom of the model. $d$ is calculated as the integer part of $n_{\text{parms}}/K$, where $n_{\text{parms}}$ is the total number of freely estimated parameters in the model.

In the trend(rconstant) specification, the estimation procedure directly estimates $\mu$. For trend(constant), trend(rtrend), and trend(trend), the estimates of $\mu$ are backed out using (12). In the trend(rtrend) specification, the estimation procedure directly estimates $\rho$. In the trend(trend) specification, the estimates of $\rho$ are backed out using (13). Because the elements of the estimated VCE are readily available only when the estimates are obtained directly, when the trend parameter estimates are backed out, their elements in the VCE for $\widehat{\beta}_f$ are missing.
Under the Johansen identification restrictions, vec obtains \( \hat{\beta} \), the estimates of the parameters in the \( r \times m_1 \) matrix \( \tilde{\beta}' \) in (5). The VCE of vec(\( \hat{\beta} \)) is \( rm_1 \times rm_1 \). Per Johansen (1995), the asymptotic distribution of \( \hat{\beta} \) is mixed Gaussian, and its VCE is consistently estimated by

\[
\left( \frac{1}{T-d} \right) \left( I_r \otimes H_J \right) \left\{ (\hat{\alpha}' \Omega^{-1} \hat{\alpha}) \otimes (H'_J S_{11} H_J) \right\}^{-1} (I_r \otimes H_J)' \tag{15}
\]

where \( H_J \) is the \( m_1 \times (m_1 - r) \) matrix given by \( H_J = (0'_{r \times (m_1 - r)}, I_{m_1 - r})' \). The VCE reported in \( e(V_{\beta}) \) is the estimated VCE in (15) augmented with missing values to account for any backed-out estimates of \( \mu \) or \( \rho \).

The parameter estimates \( \hat{\alpha} \) can be found either as a function of \( \hat{\beta} \), using (10) or from the VAR in (14). The estimated VCE of \( \hat{\alpha} \) reported in \( e(V_{\alpha}) \) is given by

\[
\frac{1}{(T-d)} \hat{\Omega} \otimes \hat{\Sigma}_B
\]

where \( \hat{\Sigma}_B = (\hat{\beta}' S_{11} \hat{\beta})^{-1} \).

As we would expect, the estimator of \( \Pi = \alpha \beta' \) is

\[
\hat{\Pi} = \hat{\alpha} \hat{\beta}'
\]

and its estimated VCE is given by

\[
\frac{1}{(T-d)} \hat{\Omega} \otimes (\hat{\beta} \hat{\Sigma}_B \hat{\beta}')
\]

The moving-average impact matrix \( C \) is estimated by

\[
\hat{C} = \hat{\beta}_\bot (\hat{\alpha}_\bot \hat{\Gamma}_\bot)^{-1} \hat{\alpha}_\bot'
\]

where \( \hat{\beta}_\bot \) is the orthogonal complement of \( \hat{\beta}_y \), \( \hat{\alpha}_\bot \) is the orthogonal complement of \( \hat{\alpha} \), and \( \hat{\Gamma} = \hat{I}_K - \sum_{i=1}^{p-1} \hat{\Gamma}_i \). The orthogonal complement of a \( K \times r \) matrix \( Q \) that has rank \( r \) is a matrix \( Q_\bot \) of rank \( K - r \), such that \( Q' Q_\bot = 0 \). Although this operation is not uniquely defined, the results used by vec do not depend on the method of obtaining the orthogonal complement. vec uses the following method: the orthogonal complement of \( Q \) is given by the \( r \) eigenvectors with the highest eigenvalues from the matrix \( Q'(Q'Q)^{-1}Q' \).

Per Johansen (1995, chap. 13) and Drukker (2004), the VCE of \( \hat{C} \) is estimated by

\[
\frac{T-d}{T} \hat{S}_q \hat{V}_\hat{\nu} \hat{S}_q'
\]

where

\[
\hat{S}_q = \hat{C} \otimes \hat{\xi}
\]

\[
\hat{\xi} = \begin{cases} 
(\hat{\xi}_1, \hat{\xi}_2) & \text{if } p > 1 \\
\hat{\xi}_1 & \text{if } p = 1
\end{cases}
\]

\[
\hat{\xi}_1 = (\hat{C}' \hat{\Gamma}' - \hat{I}_K) \hat{\alpha}
\]

\[
\hat{\alpha} = \hat{\alpha} (\hat{\alpha}' \hat{\alpha})^{-1}
\]

\[
\hat{\xi}_2 = \iota_{p-1} \otimes \hat{C}
\]

\( \iota_{p-1} \) is a \((p-1) \times 1\) vector of ones

\( \hat{V}_\hat{\nu} \) is the estimated VCE of \( \hat{\nu} = (\hat{\alpha}, \hat{\Gamma}_1, \ldots, \hat{\Gamma}_{p-1}) \)
Estimation with constraints: $\beta$ identified

vec can also fit models in which the adjustment parameters are subject to homogeneous linear constraints and the cointegrating vectors are subject to general linear restrictions. Mathematically, vec allows for constraints of the form

$$R^\prime_\alpha \text{vec}(\alpha) = 0 \quad (17)$$

where $R_\alpha$ is a known $Kr \times n_\alpha$ constraint matrix, and

$$R^\prime_{\tilde{\beta}} \text{vec}(\tilde{\beta}) = b \quad (18)$$

where $R_{\tilde{\beta}}$ is a known $m_1r \times n_\beta$ constraint matrix and $b$ is a known $n_\beta \times 1$ vector of constants. Although (17) and (18) are intuitive, they can be rewritten in a form to facilitate computation. Specifically, (17) can be written as

$$\text{vec}(\alpha^\prime) = Ga \quad (19)$$

where $G$ is $Kr \times n_\alpha$ and $a$ is $n_\alpha \times 1$. Equation (18) can be rewritten as

$$\text{vec}(\tilde{\beta}) = Hb + h_0 \quad (20)$$

where $H$ is a known $n_1r \times n_\beta$ matrix, $b$ is an $n_\beta \times 1$ matrix of parameters, and $h_0$ is a known $n_1r \times 1$ matrix. See [P] makecns for a discussion of the different ways of specifying the constraints.

When constraints are specified via the aconstraints() and bconstraints() options, the Boswijk (1995) rank method determines whether the parameters in $\tilde{\beta}$ are underidentified, exactly identified, or overidentified.

Boswijk (1995) uses the Rothenberg (1971) method to determine whether the parameters in $\tilde{\beta}$ are identified. Thus the parameters in $\tilde{\beta}$ are exactly identified if $\rho_\beta = r^2$, and the parameters in $\tilde{\beta}$ are overidentified if $\rho_\beta > r^2$, where

$$\rho_\beta = \text{rank}\left\{R_{\tilde{\beta}}(I_r \otimes \tilde{\beta})\right\}$$

and $\tilde{\beta}$ is a full-rank matrix with the same dimensions as $\tilde{\beta}$. The computed $\rho_\beta$ is stored in e(beta_icnt).

Similarly, the number of freely estimated parameters in $\alpha$ and $\tilde{\beta}$ is given by $\rho_{\text{jacob}}$, where

$$\rho_{\text{jacob}} = \text{rank}\left\{(\hat{\alpha} \otimes I_{m_1})H, (I_K \otimes \hat{\beta})G\right\}$$

Using $\rho_{\text{jacob}}$, we can calculate several other parameter counts of interest. In particular, the degrees of freedom of the overidentifying test are given by $(K + m_1 - r)r - \rho_{\text{jacob}}$, and the number of freely estimated parameters in the model is $n_{\text{parms}} = Km_2 + \rho_{\text{jacob}}$.

Although the problem of maximizing the log-likelihood function in (4), subject to the constraints in (17) and (18), could be handled by the algorithms in [R] ml, the switching algorithm of Boswijk (1995) has proven to be more convergent. For this reason, vec uses the Boswijk (1995) switching algorithm to perform the optimization.
Given starting values \((\hat{b}_0, \hat{a}_0, \hat{\Omega}_0)\), the algorithm iteratively updates the estimates until convergence is achieved, as follows:

\[
\hat{\alpha}_j \text{ is constructed from (19) and } \hat{\beta}_j \text{ is constructed from (20) and } \hat{b}_j
\]

\[
\hat{b}_{j+1} = \{H'(\hat{\alpha}_j^{-1} \hat{\Omega}_j \otimes S_{11})H\}^{-1}H'(\hat{\alpha}_j^{-1} \hat{\Omega}_j \otimes S_{11})\{\text{vec}(\hat{P}) - (\hat{\alpha}_j \otimes I_{nz})h_0\}
\]

\[
\hat{a}_{j+1} = \{G(\hat{\Omega}_j^{-1} \otimes \hat{\beta}_j S_{11})G\}^{-1}G'(\hat{\Omega}_j^{-1} \otimes \hat{\beta}_j S_{11})\text{vec}(\hat{P})
\]

\[
\hat{\Omega}_{j+1} = S_{00} - S_{01} \hat{\alpha}_j' \hat{\beta}_j - \hat{\alpha}_j \hat{\beta}_j' S_{10} + \hat{\alpha}_j \hat{\beta}_j S_{11} \hat{\alpha}_j
\]

The estimated VCE of \(\hat{\beta}\) is given by

\[
\frac{1}{(T-d)} H\{H'(W \otimes S_{11})H\}^{-1}H'
\]

where \(W\) is \(\hat{\alpha}' \hat{\Omega}^{-1} \hat{\alpha}\). As in the case without constraints, the estimated VCE of \(\hat{\alpha}\) can be obtained either from the VCE of the short-run parameters, as described below, or via the formula

\[
\hat{V}_\hat{\alpha} = \frac{1}{(T-d)} G \left[ G'(\hat{\Omega}^{-1} \otimes (\hat{\beta}' S_{11}) \hat{\beta})^{-1} \right] G'
\]

Boswijk (1995) notes that, as long as the parameters of the cointegrating equations are exactly identified or overidentified, the constrained ML estimator produces superconsistent estimates of \(\hat{\beta}\). This implies that the method of estimating the short-run parameters described above applies in the presence of constraints, as well, albeit with a caveat: when there are constraints placed on \(\alpha\), the VARs must be estimated subject to these constraints.

With these estimates and the estimated VCE of the short-run parameter matrix \(\hat{V}_\hat{\nu}\), Drukker (2004) shows that the estimated VCE for \(\hat{\Pi}\) is given by

\[
(\hat{\beta} \otimes I_K) \hat{V}_\hat{\alpha}(\hat{\beta} \otimes I_K)'
\]

Drukker (2004) also shows that the estimated VCE of \(\hat{C}\) can be obtained from (16) with the extension that \(\hat{V}_\hat{\nu}\) is the estimated VCE of \(\hat{\nu}\) that takes into account any constraints on \(\hat{\alpha}\).

**Estimation with constraints: \(\beta\) not identified**

When the parameters in \(\beta\) are not identified, only the parameters in \(\Pi = \alpha \beta\) and \(C\) are identified. The estimates of \(\Pi\) and \(C\) would not change if more identification restrictions were imposed to achieve exact identification. Thus the VCE matrices for \(\hat{\Pi}\) and \(\hat{C}\) can be derived as if the model exactly identified \(\beta\).
Formulas for the information criteria

The AIC, SBIC, and HQIC are calculated according to their standard definitions, which include the constant term from the log likelihood; that is,

\[
\text{AIC} = -2 \left( \frac{L}{T} \right) + \frac{2n_{\text{parms}}}{T}
\]

\[
\text{SBIC} = -2 \left( \frac{L}{T} \right) + \frac{\ln(T)}{T} n_{\text{parms}}
\]

\[
\text{HQIC} = -2 \left( \frac{L}{T} \right) + \frac{2\ln\{\ln(T)\}}{T} n_{\text{parms}}
\]

where \(n_{\text{parms}}\) is the total number of parameters in the model and \(L\) is the value of the log likelihood at the optimum.

Formulas for predict

 xb, residuals and stdp are standard and are documented in [R] predict. ce causes predict to compute \(\hat{E}_t = \hat{\beta} f y_t\) for the requested cointegrating equation.

levels causes predict to compute the predictions for the levels of the data. Let \(\hat{y}^d_t\) be the predicted value of \(\Delta y_t\). Because the computations are performed for a given equation, \(y_t\) is a scalar. Using \(\hat{y}^d_t\), we can predict the level by \(\hat{y}_t = \hat{y}^d_t + y_{t-1}\).

Because the residuals from the VECM for the differences and the residuals from the corresponding VAR in levels are identical, there is no need for an option for predicting the residuals in levels.

References


Becketti, S. 2013. Introduction to Time Series Using Stata. College Station, TX: Stata Press.


**Also see**

[TS] vec postestimation — Postestimation tools for vec

[TS] tsset — Declare data to be time-series data

[TS] var — Vector autoregressive models

[TS] var svar — Structural vector autoregressive models

[TS] vec intro — Introduction to vector error-correction models

[U] 20 Estimation and postestimation commands
Postestimation commands

The following postestimation commands are of special interest after vec:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fcast compute</code></td>
<td>obtain dynamic forecasts</td>
</tr>
<tr>
<td><code>fcast graph</code></td>
<td>graph dynamic forecasts obtained from <code>fcast compute</code></td>
</tr>
<tr>
<td><code>irf</code></td>
<td>create and analyze IRFs and FEVDs</td>
</tr>
<tr>
<td><code>veclmar</code></td>
<td>LM test for autocorrelation in residuals</td>
</tr>
<tr>
<td><code>vecnorm</code></td>
<td>test for normally distributed residuals</td>
</tr>
<tr>
<td><code>vecstable</code></td>
<td>check stability condition of estimates</td>
</tr>
</tbody>
</table>

The following standard postestimation commands are also available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estat ic</code></td>
<td>Akaike’s and Schwarz’s Bayesian information criteria (AIC and BIC)</td>
</tr>
<tr>
<td><code>estat summarize</code></td>
<td>summary statistics for the estimation sample</td>
</tr>
<tr>
<td><code>estat vce</code></td>
<td>variance–covariance matrix of the estimators (VCE)</td>
</tr>
<tr>
<td><code>estimates</code></td>
<td>cataloging estimation results</td>
</tr>
<tr>
<td><code>forecast</code></td>
<td>dynamic forecasts and simulations</td>
</tr>
<tr>
<td><code>lincom</code></td>
<td>point estimates, standard errors, testing, and inference for linear combinations of coefficients</td>
</tr>
<tr>
<td><code>lrtest</code></td>
<td>likelihood-ratio test</td>
</tr>
<tr>
<td><code>margins</code></td>
<td>marginal means, predictive margins, marginal effects, and average marginal effects</td>
</tr>
<tr>
<td><code>marginsplot</code></td>
<td>graph the results from margins (profile plots, interaction plots, etc.)</td>
</tr>
<tr>
<td><code>nlcom</code></td>
<td>point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients</td>
</tr>
<tr>
<td><code>predict</code></td>
<td>predictions, residuals, influence statistics, and other diagnostic measures</td>
</tr>
<tr>
<td><code>predictnl</code></td>
<td>point estimates, standard errors, testing, and inference for generalized predictions</td>
</tr>
<tr>
<td><code>test</code></td>
<td>Wald tests of simple and composite linear hypotheses</td>
</tr>
<tr>
<td><code>testnl</code></td>
<td>Wald tests of nonlinear hypotheses</td>
</tr>
</tbody>
</table>

888
predict

Description for predict

predict creates a new variable containing predictions such as expected values, residuals, and
co-integrating equations.

Menu for predict

Statistics > Postestimation

Syntax for predict

    predict [type] newvar [if] [in] [ , statistic equation(eqno|eqname)]

statistic     Description

Main
    xb          fitted value for the specified equation; the default
    stdp        standard error of the linear prediction
    residuals   residuals
    ce          the predicted value of specified co-integrating equation
    levels      one-step prediction of the level of the endogenous variable
    usece(varlist_ce) compute the predictions using previously predicted co-integrating equations

These statistics are available both in and out of sample; type predict ... if e(sample) ... if wanted only for
the estimation sample.

Options for predict

    xb, the default, calculates the fitted values for the specified equation. The form of the VECM implies
    that these fitted values are the one-step predictions for the first-differenced variables.
    stdp calculates the standard error of the linear prediction for the specified equation.
    residuals calculates the residuals from the specified equation of the VECM.
    ce calculates the predicted value of the specified co-integrating equation.
    levels calculates the one-step prediction of the level of the endogenous variable in the requested
    equation.
    usece(varlist_ce) specifies that previously predicted co-integrating equations saved under the names in
    varlist_ce be used to compute the predictions. The number of variables in the varlist_ce must equal
    the number of co-integrating equations specified in the model.
    equation(eqno|eqname) specifies to which equation you are referring.

    equation() is filled in with one eqno or eqname for xb, residuals, stdp, ce, and levels
    options. equation(#1) would mean that the calculation is to be made for the first equation,
    equation(#2) would mean the second, and so on. You could also refer to the equation by its
    name. equation(D_income) would refer to the equation named D_income and equation(_ce1),
    to the first co-integrating equation, which is named _ce1 by vec.

    If you do not specify equation(), the results are as if you specified equation(#1).

For more information on using predict after multiple-equation estimation commands, see [R] predict.
margins

Description for margins

margins estimates margins of response for linear predictions.

Menu for margins

Statistics > Postestimation

Syntax for margins

margins [marginlist] [ , options ]
margins [marginlist] , predict(statistic ...) [ predict(statistic ...) ... ] [options]

<table>
<thead>
<tr>
<th>statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>linear predictions for each equation</td>
</tr>
<tr>
<td>xb</td>
<td>linear prediction for a specified equation</td>
</tr>
<tr>
<td>stdp</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>residuals</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>ce</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>levels</td>
<td>not allowed with margins</td>
</tr>
<tr>
<td>usece(varlistce)</td>
<td>not allowed with margins</td>
</tr>
</tbody>
</table>

xb defaults to the first equation.

Statistics not allowed with margins are functions of stochastic quantities other than e(b).

For the full syntax, see [R] margins.

Remarks and examples

Remarks are presented under the following headings:

Model selection and inference
Forecasting

Model selection and inference

See the following sections for information on model selection and inference after vec.

[TS] irf — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
[TS] varsoc — Obtain lag-order selection statistics for VARs and VECMs
[TS] veclmar — LM test for residual autocorrelation after vec
[TS] vecnorm — Test for normally distributed disturbances after vec
[TS] vecrank — Estimate the cointegrating rank of a VECM
[TS] vecstable — Check the stability condition of VECM estimates
Forecasting

See the following sections for information on obtaining forecasts after vec:

[Ts] fcast compute — Compute dynamic forecasts after var, svar, or vec
[Ts] fcast graph — Graph forecasts after fcast compute

Also see

[Ts] vec — Vector error-correction models
[Ts] vec intro — Introduction to vector error-correction models
[U] 20 Estimation and postestimation commands
Description

`veclmar` implements a Lagrange multiplier (LM) test for autocorrelation in the residuals of vector error-correction models (VECMs).

Quick start

Test of residual autocorrelation for the first two lags of the residuals after `vec`

`veclmar`

As above, but test the first 5 lags

`veclmar, mlag(5)`

As above, but perform test using stored estimates `myest` from a VECM

`veclmar, mlag(5) estimates(myest)`

Menu

Statistics > Multivariate time series > VEC diagnostics and tests > LM test for residual autocorrelation
Syntax

veclmar [, options]

options Description

mlag(#) use # for the maximum order of autocorrelation; default is mlag(2)
estimates(estname) use previously stored results estname; default is to use active results
separator(#) draw separator line after every # rows

veclmar can be used only after vec; see [TS] vec.
You must tsset your data before using veclmar; see [TS] tsset.

Options

mlag(#) specifies the maximum order of autocorrelation to be tested. The integer specified in mlag() must be greater than 0; the default is 2.
estimates(estname) requests that veclmar use the previously obtained set of vec estimates stored as estname. By default, veclmar uses the active results. See [R] estimates for information on manipulating estimation results.
separator(#) specifies how many rows should appear in the table between separator lines. By default, separator lines do not appear. For example, separator(1) would draw a line between each row, separator(2) between every other row, and so on.

Remarks and examples

Estimation, inference, and postestimation analysis of VECMs is predicated on the errors’ not being autocorrelated. veclmar implements the LM test for autocorrelation in the residuals of a VECM discussed in Johansen (1995, 21–22). The test is performed at lags \( j = 1, \ldots, \text{mlag}() \). For each \( j \), the null hypothesis of the test is that there is no autocorrelation at lag \( j \).

Example 1

We fit a VECM using the regional income data described in [TS] vec and then call veclmar to test for autocorrelation.

```
. use https://www.stata-press.com/data/r16/rdinc
. vec ln_ne ln_se
    (output omitted)
. veclmar, mlag(4)

Lagrange-multiplier test

<table>
<thead>
<tr>
<th>lag</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.9586</td>
<td>4</td>
<td>0.06214</td>
</tr>
<tr>
<td>2</td>
<td>4.9809</td>
<td>4</td>
<td>0.28926</td>
</tr>
<tr>
<td>3</td>
<td>4.8519</td>
<td>4</td>
<td>0.30284</td>
</tr>
<tr>
<td>4</td>
<td>0.3270</td>
<td>4</td>
<td>0.98801</td>
</tr>
</tbody>
</table>
```

H0: no autocorrelation at lag order
At the 5% level, we cannot reject the null hypothesis that there is no autocorrelation in the residuals for any of the orders tested. Thus this test finds no evidence of model misspecification.

Stored results

veclmar stores the following in r():

Matrices

r(lm) \( \chi^2 \), df, and \( p \)-values

Methods and formulas

Consider a VECM without any trend:

\[
\Delta y_t = \alpha \beta y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \epsilon_t
\]

As discussed in [TS] vec, as long as the parameters in the cointegrating vectors, \( \beta \), are exactly identified or overidentified, the estimates of these parameters are superconsistent. This implies that the \( r \times 1 \) vector of estimated cointegrating relations

\[
\hat{E}_t = \hat{\beta} y_t
\]

(1)

can be used as data with standard estimation and inference methods. When the parameters of the cointegrating equations are not identified, (1) does not provide consistent estimates of \( \hat{E}_t \); in these cases, veclmar exits with an error message.

The VECM above can be rewritten as

\[
\Delta y_t = \alpha \hat{E}_t + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \epsilon_t
\]

which is just a VAR with \( p - 1 \) lags where the endogenous variables have been first-differenced and is augmented with the exogenous variables \( \hat{E} \). veclmar fits this VAR and then calls varlmar to compute the LM test for autocorrelation.

The above discussion assumes no trend and implicitly ignores constraints on the parameters in \( \alpha \). As discussed in vec, the other four trend specifications considered by Johansen (1995, sec. 5.7) complicate the estimation of the free parameters in \( \beta \) but do not alter the basic result that the \( \hat{E}_t \) can be used as data in the subsequent VAR. Similarly, constraints on the parameters in \( \alpha \) imply that the subsequent VAR must be estimated with these constraints applied, but \( \hat{E}_t \) can still be used as data in the VAR.

See [TS] varlmar for more information on the Johansen LM test.
Reference

Also see

[TS] **varmar** — LM test for residual autocorrelation after var or svar
[TS] **vec** — Vector error-correction models
[TS] **vec intro** — Introduction to vector error-correction models
Description

vecnorm computes and reports a series of statistics against the null hypothesis that the disturbances in a VECM are normally distributed.

Quick start

Compute Jarque–Bera, skewness, and kurtosis statistics after vec to test the null hypothesis that the residuals are normally distributed

```plaintext
vecnorm
```

As above, but only report the Jarque–Bera statistic

```plaintext
vecnorm, jbera
```

As above, but only report kurtosis

```plaintext
vecnorm, kurtosis
```

Menu

Statistics > Multivariate time series > VEC diagnostics and tests > Test for normally distributed disturbances
vecnorm — Test for normally distributed disturbances after vec

Syntax

\[ \text{vecnorm}[\text{, options}] \]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{jbera}</td>
<td>report Jarque–Bera statistic; default is to report all three statistics</td>
</tr>
<tr>
<td>\text{skewness}</td>
<td>report skewness statistic; default is to report all three statistics</td>
</tr>
<tr>
<td>\text{kurtosis}</td>
<td>report kurtosis statistic; default is to report all three statistics</td>
</tr>
<tr>
<td>\text{estimates}(\text{estname})</td>
<td>use previously stored results \text{estname}; default is to use active results</td>
</tr>
<tr>
<td>\text{dfk}</td>
<td>make small-sample adjustment when computing the estimated variance–covariance matrix of the disturbances</td>
</tr>
<tr>
<td>\text{separator}(#)</td>
<td>draw separator line after every # rows</td>
</tr>
</tbody>
</table>

vecnorm can be used only after vec; see [TS] vec.

Options

\text{jbera} requests that the Jarque–Bera statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.

\text{skewness} requests that the skewness statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.

\text{kurtosis} requests that the kurtosis statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.

\text{estimates}(\text{estname}) requests that \text{vecnorm} use the previously obtained set of vec estimates stored as \text{estname}. By default, \text{vecnorm} uses the active results. See [R] \text{estimates} for information on manipulating estimation results.

\text{dfk} requests that a small-sample adjustment be made when computing the estimated variance–covariance matrix of the disturbances.

\text{separator}(\#) specifies how many rows should appear in the table between separator lines. By default, separator lines do not appear. For example, \text{separator}(1) would draw a line between each row, \text{separator}(2) between every other row, and so on.

Remarks and examples

\text{vecnorm} computes a series of test statistics of the null hypothesis that the disturbances in a VECM are normally distributed. For each equation and all equations jointly, up to three statistics may be computed: a skewness statistic, a kurtosis statistic, and the Jarque–Bera statistic. By default, all three statistics are reported; if you specify only one statistic, the others are not reported. The Jarque–Bera statistic tests skewness and kurtosis jointly. The single-equation results are against the null hypothesis that the disturbance for that particular equation is normally distributed. The results for all the equations are against the null that all \(K\) disturbances have a \(K\)-dimensional multivariate normal distribution. Failure to reject the null hypothesis indicates lack of model misspecification.

As noted by Johansen (1995, 141), the log likelihood for the VECM is derived assuming the errors are independent and identically distributed normal, though many of the asymptotic properties can be derived under the weaker assumption that the errors are merely independent and identically distributed. Many researchers still prefer to test for normality. \text{vecnorm} uses the results from vec to produce a series of statistics against the null hypothesis that the \(K\) disturbances in the VECM are normally distributed.
Example 1

This example uses `vecnorm` to test for normality after estimating the parameters of a VECM using the regional income data.

```
. use https://www.stata-press.com/data/r16/rdinc
. vec ln_ne ln_se
    (output omitted)
. vecnorm
```

<table>
<thead>
<tr>
<th>Equation</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_ln_ne</td>
<td>0.094</td>
<td>2</td>
<td>0.95417</td>
</tr>
<tr>
<td>D_ln_se</td>
<td>0.586</td>
<td>2</td>
<td>0.74608</td>
</tr>
<tr>
<td>ALL</td>
<td>0.680</td>
<td>4</td>
<td>0.95381</td>
</tr>
</tbody>
</table>

**Skewness test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Skewness</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_ln_ne</td>
<td>0.05982</td>
<td>0.032</td>
<td>1</td>
<td>0.85890</td>
</tr>
<tr>
<td>D_ln_se</td>
<td>0.243</td>
<td>0.522</td>
<td>1</td>
<td>0.47016</td>
</tr>
<tr>
<td>ALL</td>
<td>0.553</td>
<td>2</td>
<td>0.75835</td>
<td></td>
</tr>
</tbody>
</table>

**Kurtosis test**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Kurtosis</th>
<th>chi2</th>
<th>df</th>
<th>Prob &gt; chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_ln_ne</td>
<td>3.1679</td>
<td>0.062</td>
<td>1</td>
<td>0.80302</td>
</tr>
<tr>
<td>D_ln_se</td>
<td>2.8294</td>
<td>0.064</td>
<td>1</td>
<td>0.79992</td>
</tr>
<tr>
<td>ALL</td>
<td>0.126</td>
<td>2</td>
<td>0.93873</td>
<td></td>
</tr>
</tbody>
</table>

The Jarque–Bera results present test statistics for each equation and for all equations jointly against the null hypothesis of normality. For the individual equations, the null hypothesis is that the disturbance term in that equation has a univariate normal distribution. For all equations jointly, the null hypothesis is that the $K$ disturbances come from a $K$-dimensional normal distribution. In this example, the single-equation and overall Jarque–Bera statistics do not reject the null of normality.

The single-equation skewness test statistics are of the null hypotheses that the disturbance term in each equation has zero skewness, which is the skewness of a normally distributed variable. The row marked ALL shows the results for a test that the disturbances in all equations jointly have zero skewness. The skewness results shown above do not suggest nonnormality.

The kurtosis of a normally distributed variable is three, and the kurtosis statistics presented in the table test the null hypothesis that the disturbance terms have kurtosis consistent with normality. The results in this example do not reject the null hypothesis.

The statistics computed by `vecnorm` are based on the estimated variance–covariance matrix of the disturbances. `vec` saves the ML estimate of this matrix, which `vecnorm` uses by default. Specifying the `dfk` option instructs `vecnorm` to make a small-sample adjustment to the variance–covariance matrix before computing the test statistics.
Stored results

`vecnorm` stores the following in `r()`:

Macros
- `r(dfk)` dfk, if specified

Matrices
- `r(jb)` Jarque–Bera $\chi^2$, df, and $p$-values
- `r(skewness)` skewness $\chi^2$, df, and $p$-values
- `r(kurtosis)` kurtosis $\chi^2$, df, and $p$-values

Methods and formulas

As discussed in *Methods and formulas* of [TS] `vec`, a cointegrating VECM can be rewritten as a VAR in first differences that includes the predicted cointegrating equations as exogenous variables. `vecnorm` computes the tests discussed in [TS] `varnorm` for the corresponding augmented VAR in first differences. See *Methods and formulas* of [TS] `veclmar` for more information on this approach.

When the parameters of the cointegrating equations are not identified, the consistent estimates of the cointegrating equations are not available, and, in these cases, `vecnorm` exits with an error message.

References


Also see

[TS] `varnorm` — Test for normally distributed disturbances after var or svar
[TS] `vec` — Vector error-correction models
[TS] `vec intro` — Introduction to vector error-correction models
vecrank — Estimate the cointegrating rank of a VECM

Description

vecrank produces statistics used to determine the number of cointegrating equations in a vector error-correction model (VECM).

Quick start

Estimate the cointegrating rank for a VECM of y1, y2, and y3 using tsset data
vecrank y1 y2 y3

As above, but specify that the underlying VAR model has 6 lags
vecrank y1 y2 y3, lags(6)

As above, but specify that the model includes a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data
vecrank y1 y2 y3, lags(6) trend(trend)

As above, and report information criteria
vecrank y1 y2 y3, lags(6) trend(trend) ic

Menu

Statistics > Multivariate time series > Cointegrating rank of a VECM
## Syntax

```
vecrank `depvarlist' [ `if' ] [ `in' ] [ , `options' ]
```

<table>
<thead>
<tr>
<th><code>options</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td></td>
</tr>
<tr>
<td><code>lags(#)</code></td>
<td>use # for the maximum lag in underlying VAR model</td>
</tr>
<tr>
<td><code>trend(constant)</code></td>
<td>include an unrestricted constant in model; the default</td>
</tr>
<tr>
<td><code>trend(rconstant)</code></td>
<td>include a restricted constant in model</td>
</tr>
<tr>
<td><code>trend(trend)</code></td>
<td>include a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data</td>
</tr>
<tr>
<td><code>trend(rtrend)</code></td>
<td>include a restricted trend in model</td>
</tr>
<tr>
<td><code>trend(none)</code></td>
<td>do not include a trend or a constant</td>
</tr>
<tr>
<td><strong>Adv. model</strong></td>
<td></td>
</tr>
<tr>
<td><code>sindicators(varlist_si)</code></td>
<td>include normalized seasonal indicator variables <code>varlist_si</code></td>
</tr>
<tr>
<td><code>noreduce</code></td>
<td>do not perform checks and corrections for collinearity among lags of dependent variables</td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td></td>
</tr>
<tr>
<td><code>notrace</code></td>
<td>do not report the trace statistic</td>
</tr>
<tr>
<td><code>max</code></td>
<td>report maximum-eigenvalue statistic</td>
</tr>
<tr>
<td><code>ic</code></td>
<td>report information criteria</td>
</tr>
<tr>
<td><code>level99</code></td>
<td>report 1% critical values instead of 5% critical values</td>
</tr>
<tr>
<td><code>levela</code></td>
<td>report both 1% and 5% critical values</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `vecrank`; see `[TS] tsset.
`depvar` may contain time-series operators; see `[U] 11.4.4 Time-series varlists.
`by`, `rolling`, and `statsby` are allowed; see `[U] 11.1.10 Prefix commands.
`vecrank` does not allow gaps in the data.

## Options

- **Model**
  - `lags(#)` specifies the number of lags in the VAR representation of the model. The VECM will include one fewer lag of the first differences. The number of lags must be greater than zero but small enough so that the degrees of freedom used by the model are less than the number of observations.

- **trend(trend_spec)** specifies one of five trend specifications to include in the model. See `[TS] vec intro` and `[TS] vec` for descriptions. The default is `trend(constant)`.

- **Adv. model**
  - `sindicators(varlist_si)` specifies normalized seasonal indicator variables to be included in the model. The indicator variables specified in this option must be normalized as discussed in Johansen (1995, 84). If the indicators are not properly normalized, the likelihood-ratio–based tests for the number of cointegrating equations do not converge to the asymptotic distributions derived by Johansen. For details, see *Methods and formulas* of `[TS] vec`. `sindicators()` cannot be specified with `trend(none)` or `trend(rconstant)`.
noreduce causes vecrank to skip the checks and corrections for collinearity among the lags of
the dependent variables. By default, vecrank checks whether the current lag specification causes
some of the regressions performed by vecrank to contain perfectly collinear variables and reduces
the maximum lag until the perfect collinearity is removed. See Collinearity in [TS] vec for more
information.

Reporting

notrace requests that the output for the trace statistic not be displayed. The default is to display the
trace statistic.

max requests that the output for the maximum-eigenvalue statistic be displayed. The default is to not
display this output.

ic causes the output for the information criteria to be displayed. The default is to not display this
output.

level99 causes the 1% critical values to be displayed instead of the default 5% critical values.

levela causes both the 1% and the 5% critical values to be displayed.

Remarks and examples

Remarks are presented under the following headings:

Introduction
The trace statistic
The maximum-eigenvalue statistic
Minimizing an information criterion

Introduction

Before estimating the parameters of a VECM models, you must choose the number of lags in the
underlying VAR, the trend specification, and the number of cointegrating equations. vecrank offers
several ways of determining the number of cointegrating vectors conditional on a trend specification
and lag order.

vecrank implements three types of methods for determining $r$, the number of cointegrating
equations in a VECM. The first is Johansen’s “trace” statistic method. The second is his “maximum
eigenvalue” statistic method. The third method chooses $r$ to minimize an information criterion.

All three methods are based on Johansen’s maximum likelihood (ML) estimator of the parameters
of a cointegrating VECM. The basic VECM is

$$
\Delta y_t = \alpha \beta' y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \epsilon_t
$$

where $y$ is a $(K \times 1)$ vector of I(1) variables, $\alpha$ and $\beta$ are $(K \times r)$ parameter matrices with rank
$r < K$, $\Gamma_1, \ldots, \Gamma_{p-1}$ are $(K \times K)$ matrices of parameters, and $\epsilon_t$ is a $(K \times 1)$ vector of normally
distributed errors that is serially uncorrelated but has contemporaneous covariance matrix $\Omega$.

Building on the work of Anderson (1951), Johansen (1995) derives an ML estimator for the
parameters and two likelihood-ratio (LR) tests for inference on $r$. These LR tests are known as the
trace statistic and the maximum-eigenvalue statistic because the log likelihood can be written as the
log of the determinant of a matrix plus a simple function of the eigenvalues of another matrix.
Let \( \lambda_1, \ldots, \lambda_K \) be the \( K \) eigenvalues used in computing the log likelihood at the optimum. Furthermore, assume that these eigenvalues are sorted from the largest \( \lambda_1 \) to the smallest \( \lambda_K \). If there are \( r < K \) cointegrating equations, \( \alpha \) and \( \beta \) have rank \( r \) and the eigenvalues \( \lambda_{r+1}, \ldots, \lambda_K \) are zero.

The trace statistic

The null hypothesis of the trace statistic is that there are no more than \( r \) cointegrating relations. Restricting the number of cointegrating equations to be \( r \) or less implies that the remaining \( K - r \) eigenvalues are zero. Johansen (1995, chap. 11 and 12) derives the distribution of the trace statistic

\[
-T \sum_{i=r+1}^{K} \ln(1 - \hat{\lambda}_i)
\]

where \( T \) is the number of observations and the \( \hat{\lambda}_i \) are the estimated eigenvalues. For any given value of \( r \), large values of the trace statistic are evidence against the null hypothesis that there are \( r \) or fewer cointegrating relations in the VECM.

One of the problems in determining the number of cointegrating equations is that the process involves more than one statistical test. Johansen (1995, chap. 6, 11, and 12) derives a method based on the trace statistic that has nominal coverage despite evaluating multiple tests. This method can be interpreted as being an estimator \( \hat{r} \) of the true number of cointegrating equations \( r_0 \). The method starts testing at \( r = 0 \) and accepts as \( \hat{r} \) the first value of \( r \) for which the trace statistic fails to reject the null.

Example 1

We have quarterly data on the natural logs of aggregate consumption, investment, and GDP in the United States from the first quarter of 1959 through the fourth quarter of 1982. As discussed in King et al. (1991), the balanced-growth hypothesis in economics implies that we would expect to find two cointegrating equations among these three variables. In the output below, we use \texttt{vecrank} to determine the number of cointegrating equations using Johansen’s multiple-trace test method.
Estimate the cointegrating rank of a VECM

Estimate the cointegrating rank of a VECM

```
. use https://www.stata-press.com/data/r16/balance2
   (macro data for VECM/balance study)
. vecrank y i c, lags(5)
```

Johansen tests for cointegration

Trend: constant
Sample: 1960q2 - 1982q4

<table>
<thead>
<tr>
<th>maximum</th>
<th>rank</th>
<th>parms</th>
<th>LL</th>
<th>eigenvalue</th>
<th>trace statistic</th>
<th>value</th>
<th>5% critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>39</td>
<td>1231.1041</td>
<td>.</td>
<td>.</td>
<td>46.1492</td>
<td>29.68</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>1245.3882</td>
<td>0.26943</td>
<td>17.5810</td>
<td>15.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>1252.5055</td>
<td>0.14480</td>
<td>3.3465*</td>
<td>3.76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>1254.1787</td>
<td>0.03611</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The header produces information about the sample, the trend specification, and the number of lags included in the model. The main table contains a separate row for each possible value of r, the number of cointegrating equations. When r = 3, all three variables in this model are stationary.

In this example, because the trace statistic at r = 0 of 46.1492 exceeds its critical value of 29.68, we reject the null hypothesis of no cointegrating equations. Similarly, because the trace statistic at r = 1 of 17.581 exceeds its critical value of 15.41, we reject the null hypothesis that there is one or fewer cointegrating equation. In contrast, because the trace statistic at r = 2 of 3.3465 is less than its critical value of 3.76, we cannot reject the null hypothesis that there are two or fewer cointegrating equations. Because Johansen’s method for estimating r is to accept as the first r for which the null hypothesis is not rejected, we accept r = 2 as our estimate of the number of cointegrating equations between these three variables. The “*” by the trace statistic at r = 2 indicates that this is the value of r selected by Johansen’s multiple-trace test procedure. The eigenvalue shown in the last line of output computes the trace statistic in the preceding line.

Example 2

In the previous example, we used the default 5% critical values. We can estimate r with 1% critical values instead by specifying the level99 option.

```
. vecrank y i c, lags(5) level99
```

Johansen tests for cointegration

Trend: constant
Sample: 1960q2 - 1982q4

<table>
<thead>
<tr>
<th>maximum</th>
<th>rank</th>
<th>parms</th>
<th>LL</th>
<th>eigenvalue</th>
<th>trace statistic</th>
<th>value</th>
<th>1% critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>39</td>
<td>1231.1041</td>
<td>.</td>
<td>.</td>
<td>46.1492</td>
<td>35.65</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>1245.3882</td>
<td>0.26943</td>
<td>17.5810*</td>
<td>20.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>1252.5055</td>
<td>0.14480</td>
<td>3.3465</td>
<td>6.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>1254.1787</td>
<td>0.03611</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The output indicates that switching from the 5% to the 1% level changes the resulting estimate from r = 2 to r = 1.
The maximum-eigenvalue statistic

The alternative hypothesis of the trace statistic is that the number of cointegrating equations is strictly larger than the number \( r \) assumed under the null hypothesis. Instead, we could assume a given \( r \) under the null hypothesis and test this against the alternative that there are \( r + 1 \) cointegrating equations. Johansen (1995, chap. 6, 11, and 12) derives an LR test of the null of \( r \) cointegrating relations against the alternative of \( r + 1 \) cointegrating relations. Because the part of the log likelihood that changes with \( r \) is a simple function of the eigenvalues of a \((K \times K)\) matrix, this test is known as the maximum-eigenvalue statistic. This method is used less often than the trace statistic method because no solution to the multiple-testing problem has yet been found.

Example 3

In the output below, we reexamine the balanced-growth hypothesis. We use the `levela` option to obtain both the 5% and 1% critical values, and we use the `notrace` option to suppress the table of trace statistics.

```
.vecrank y i c, lags(5) max levela notrace

Johansen tests for cointegration
Trend: constant Number of obs = 91
Sample: 1960q2 - 1982q4 Lags = 5

<table>
<thead>
<tr>
<th>maximum rank</th>
<th>params</th>
<th>LL</th>
<th>eigenvalue</th>
<th>max statistic</th>
<th>5% critical value</th>
<th>1% critical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>39</td>
<td>1231.1041</td>
<td>28.5682</td>
<td>20.97</td>
<td>25.52</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>1245.3882</td>
<td>0.26943</td>
<td>14.2346</td>
<td>14.07</td>
<td>18.63</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>1252.5055</td>
<td>0.14480</td>
<td>3.3465</td>
<td>3.76</td>
<td>6.65</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>1254.1787</td>
<td>0.03611</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

We can reject \( r = 1 \) in favor of \( r = 2 \) at the 5% level but not at the 1% level. As with the trace statistic method, whether we choose to specify one or two cointegrating equations in our VECM will depend on the significance level we use here.

Minimizing an information criterion

Many multiple-testing problems in the time-series literature have been solved by defining an estimator that minimizes an information criterion with known asymptotic properties. Selecting the lag length in an autoregressive model is probably the best-known example. Gonzalo and Pitarakis (1998) and Aznar and Salvador (2002) have shown that this approach can be applied to determining the number of cointegrating equations in a VECM. As in the lag-length selection problem, choosing the number of cointegrating equations that minimizes either the Schwarz Bayesian information criterion (SBIC) or the Hannan and Quinn information criterion (HQIC) provides a consistent estimator of the number of cointegrating equations.

Example 4

We use these information-criteria methods to estimate the number of cointegrating equations in our balanced-growth data.
. vecrank y i c, lags(5) ic notrace

Johansen tests for cointegration

<table>
<thead>
<tr>
<th>maximum rank</th>
<th>parms</th>
<th>LL</th>
<th>eigenvalue</th>
<th>SBIC</th>
<th>HQIC</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>39</td>
<td>1231.1041</td>
<td>0.26943</td>
<td>-25.12401</td>
<td>-25.76596</td>
<td>-26.20009</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>1245.3882</td>
<td>0.14480</td>
<td>-25.19781*</td>
<td>-25.97144*</td>
<td>-26.49463</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>1252.5055</td>
<td>0.03611</td>
<td>-25.18501</td>
<td>-25.97511</td>
<td>-26.50942</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>1254.1787</td>
<td>0.03611</td>
<td>-25.18501</td>
<td>-25.97511</td>
<td>-26.50942</td>
</tr>
</tbody>
</table>

Both the SBIC and the HQIC estimators suggest that there are two cointegrating equations in the balanced-growth data.

Stored results

vecrank stores the following in e():

Scalars
- e(N) number of observations
- e(k_eq) number of equations in e(b)
- e(k_dv) number of dependent variables
- e(tmin) minimum time
- e(tmax) maximum time
- e(n_lags) number of lags
- e(k_ce95) number of cointegrating equations chosen by multiple trace tests with level(95)
- e(k_ce99) number of cointegrating equations chosen by multiple trace tests with level(99)
- e(k_cesbic) number of cointegrating equations chosen by minimizing SBIC
- e(k_cehqic) number of cointegrating equations chosen by minimizing HQIC

Macros
- e(cmd) vecrank
- e(cmdline) command as typed
- e(trend) trend specified
- e(reduced_lags) list of maximum lags to which the model has been reduced
- e(reduce_opt) noreduce, if noreduce is specified
- e(tsfmt) format for current time variable

Matrices
- e(max) vector of maximum-eigenvalue statistics
- e(trace) vector of trace statistics
- e(ll) vector of model log likelihoods
- e(lambda) vector of eigenvalues
- e(k_rank) vector of numbers of unconstrained parameters
- e(hqic) vector of HQIC values
- e(sbic) vector of SBIC values
- e(aic) vector of AIC values
Methods and formulas

As shown in Methods and formulas of [TS] vec, given a lag, trend, and seasonal specification when there are \(0 \leq r \leq K\) cointegrating equations, the log likelihood with the Johansen identification restrictions can be written as

\[
L = -\frac{1}{2}T \left[ K \{ \ln(2\pi) + 1 \} + \ln(|S_{00}|) + \sum_{i=1}^{r} \ln \left(1 - \hat{\lambda}_i\right) \right] \quad (1)
\]

where the \((K \times K)\) matrix \(S_{00}\) and the eigenvalues \(\hat{\lambda}_i\) are defined in Methods and formulas of [TS] vec.

The trace statistic compares the null hypothesis that there are \(r\) or fewer cointegrating relations with the alternative hypothesis that there are more than \(r\) cointegrating equations. Under the alternative hypothesis, the log likelihood is

\[
L_A = -\frac{1}{2}T \left[ K \{ \ln(2\pi) + 1 \} + \ln(|S_{00}|) + \sum_{i=1}^{K} \ln \left(1 - \hat{\lambda}_i\right) \right] \quad (2)
\]

Thus the LR test that compares the unrestricted model in (2) with the restricted model in (1) is given by

\[
LR_{\text{trace}} = -T \sum_{i=r+1}^{K} \ln \left(1 - \hat{\lambda}_i\right)
\]

As discussed by Johansen (1995), the trace statistic has a nonstandard distribution under the null hypothesis because the null hypothesis places restrictions on the coefficients on \(y_{t-1}\), which is assumed to have \(K - r\) random-walk components. vecrank reports the Osterwald-Lenum (1992) critical values.

The maximum-eigenvalue statistic compares the null model containing \(r\) cointegrating relations with the alternative model that has \(r+1\) cointegrating relations. Thus using these two values for \(r\) in (1) and a few lines of algebra implies that the LR test of this hypothesis is

\[
LR_{\text{max}} = -T \ln \left(1 - \hat{\lambda}_{r+1}\right)
\]

As for the trace statistic, because this test involves restrictions on the coefficients on a vector of I(1) variables, the test statistic’s distribution will be nonstandard. vecrank reports the Osterwald-Lenum (1992) critical values.

The formulas for the AIC, SBIC, and HQIC are given in Methods and formulas of [TS] vec.

Søren Johansen (1939–) earned degrees in mathematical statistics at the University of Copenhagen, where he is now based. In addition to making contributions to mathematical statistics, probability theory, and medical statistics, he has worked mostly in econometrics—in particular, on the theory of cointegration.
References


Also see

*TS* `tsset` — Declare data to be time-series data

*TS* `vec` — Vector error-correction models

*TS* `vec intro` — Introduction to vector error-correction models
**vecstable** — Check the stability condition of VECM estimates

### Description

`vecstable` checks the eigenvalue stability condition in a vector error-correction model (VECM) fit using `vec`.

### Quick start

Check eigenvalue stability condition after `vec` `vecstable`

As above, and graph the eigenvalues of the companion matrix `vecstable, graph`

As above, and label each eigenvalue with its distance from the unit circle `vecstable, graph dlabel`

As above, but label the eigenvalues with their moduli `vecstable, graph modlabel`

### Menu

Statistics > Multivariate time series > VEC diagnostics and tests > Check stability condition of VEC estimates
vecstable — Check the stability condition of VECM estimates

Syntax

    vecstable [ , options ]

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Main</td>
</tr>
<tr>
<td>estimates(estname)</td>
<td>use previously stored results estname; default is to use active results</td>
</tr>
<tr>
<td>amat(matrix_name)</td>
<td>save the companion matrix as matrix_name</td>
</tr>
<tr>
<td>graph</td>
<td>graph eigenvalues of the companion matrix</td>
</tr>
<tr>
<td>dlabel</td>
<td>label eigenvalues with the distance from the unit circle</td>
</tr>
<tr>
<td>modlabel</td>
<td>label eigenvalues with the modulus</td>
</tr>
<tr>
<td>marker_options</td>
<td>change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td>rlopts(line_options)</td>
<td>affect rendition of reference unit circle</td>
</tr>
<tr>
<td>nogrid</td>
<td>suppress polar grid circles</td>
</tr>
<tr>
<td>pgrid([...])</td>
<td>specify radii and appearance of polar grid circles; see Options for details</td>
</tr>
<tr>
<td></td>
<td>Add plots</td>
</tr>
<tr>
<td>addplot(plot)</td>
<td>add other plots to the generated graph</td>
</tr>
<tr>
<td></td>
<td>Y axis, X axis, Titles, Legend, Overall</td>
</tr>
<tr>
<td>twoway_options</td>
<td>any options other than by() documented in [G-3] twoway_options</td>
</tr>
</tbody>
</table>

vecstable can be used only after vec; see [TS] vec.

Options

Main

estimates(estname) requests that vecstable use the previously obtained set of vec estimates stored as estname. By default, vecstable uses the active results. See [R] estimates for information on manipulating estimation results.

amat(matrix_name) specifies a valid Stata matrix name by which the companion matrix can be saved. The companion matrix is referred to as the A matrix in Lütkepohl (2005) and [TS] varstable. The default is not to save the companion matrix.

graph causes vecstable to draw a graph of the eigenvalues of the companion matrix.

dlabel labels the eigenvalues with their distances from the unit circle. dlabel cannot be specified with modlabel.

modlabel labels the eigenvalues with their moduli. modlabel cannot be specified with dlabel.

marker_options specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] marker_options.

rlopts(line_options) affects the rendition of the reference unit circle; see [G-3] line_options.

nogrid suppresses the polar grid circles.

pgrid([numlist], line_options) [pgrid([numlist], line_options)] ...

pgrid([numlist], line_options) determines the radii and appearance of the polar grid circles. By default, the graph includes nine polar grid circles with radii 0.1, 0.2, ..., 0.9 that have the grid linestyle. The numlist specifies the radii for the polar grid circles. The line_options determine the appearance of the polar grid circles; see [G-3] line_options. Because the pgrid() option can be repeated, circles with different radii can have distinct appearances.
Add plots

addplot(plot) adds specified plots to the generated graph; see [G-3] addplot_option.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option).

Remarks and examples

Inference after vec requires that the cointegrating equations be stationary and that the number of cointegrating equations be correctly specified. Although the methods implemented in vecrank identify the number of stationary cointegrating equations, they assume that the individual variables are I(1). vecstable provides indicators of whether the number of cointegrating equations is misspecified or whether the cointegrating equations, which are assumed to be stationary, are not stationary.

vecstable is analogous to varstable. vecstable uses the coefficient estimates from the previously fitted VECM to back out estimates of the coefficients of the corresponding VAR and then compute the eigenvalues of the companion matrix. See [TS] varstable for details about how the companion matrix is formed and about how to interpret the resulting eigenvalues for covariance-stationary VAR models.

If a VECM has \( K \) endogenous variables and \( r \) cointegrating vectors, there will be \( K - r \) unit moduli in the companion matrix. If any of the remaining moduli computed by vecrank are too close to one, either the cointegrating equations are not stationary or there is another common trend and the rank() specified in the vec command is too high. Unfortunately, there is no general distribution theory that allows you to determine whether an estimated root is too close to one for all the cases that commonly arise in practice.

Example 1

In example 1 of [TS] vec, we estimated the parameters of a bivariate VECM of the natural logs of the average disposable incomes in two of the economic regions created by the U.S. Bureau of Economic Analysis. In that example, we concluded that the predicted cointegrating equation was probably not stationary. Here we continue that example by refitting that model and using vecstable to analyze the eigenvalues of the companion matrix of the corresponding VAR.

```
use https://www.stata-press.com/data/r16/rdinc
.vec ln_ne ln_se
(output omitted)
.vecstable
```

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>.9477854</td>
<td>.947785</td>
</tr>
<tr>
<td>.2545357 + .2312756i</td>
<td>.343914</td>
</tr>
<tr>
<td>.2545357 - .2312756i</td>
<td>.343914</td>
</tr>
</tbody>
</table>

The VECM specification imposes a unit modulus.
The output contains a table showing the eigenvalues of the companion matrix and their associated moduli. The table shows that one of the roots is 1. The table footer reminds us that the specified VECM imposes one unit modulus on the companion matrix.

The output indicates that there is a real root at about 0.95. Although there is no distribution theory to measure how close this root is to one, per other discussions in the literature (for example, Johansen [1995, 137–138]), we conclude that the root of 0.95 supports our earlier analysis, in which we concluded that the predicted cointegrating equation is probably not stationary.

If we had included the graph option with vecstable, the following graph would have been displayed:

![Graph showing roots of the companion matrix](image)

The graph plots the eigenvalues of the companion matrix with the real component on the x axis and the imaginary component on the y axis. Although the information is the same as in the table, the graph shows visually how close the root with modulus 0.95 is to the unit circle.

### Stored results

vecstable stores the following in r():

Scalars

- r(unitmod) number of unit moduli imposed on the companion matrix

Matrices

- r(Re) real part of the eigenvalues of A
- r(Im) imaginary part of the eigenvalues of A
- r(Modulus) moduli of the eigenvalues of A

where A is the companion matrix of the VAR that corresponds to the VECM.

### Methods and formulas

vecstable uses the formulas given in Methods and formulas of [TS] irf create to obtain estimates of the parameters in the corresponding VAR from the vec estimates. With these estimates, the calculations are identical to those discussed in [TS] varstable. In particular, the derivation of the companion matrix, A, from the VAR point estimates is given in [TS] varstable.
References


Also see

[TS] vec — Vector error-correction models

[TS] vec intro — Introduction to vector error-correction models
Bartlett’s periodogram-based test for white noise (\texttt{wntestb})

\texttt{wntestb} performs Bartlett’s periodogram-based test for white noise. The result is presented graphically by default but optionally may be presented as text in a table.

\textbf{Quick start}

Bartlett’s periodogram-based test for white noise on series \textit{y} using \texttt{tsset} data

\begin{verbatim}
     wntestb y
\end{verbatim}

As above, but report table instead of graph

\begin{verbatim}
     wntestb y, table
\end{verbatim}

As above, but specify 90\%, instead of 95\%, confidence interval

\begin{verbatim}
     wntestb y, table level(90)
\end{verbatim}

\textbf{Menu}

Statistics > Time series > Tests > Bartlett's periodogram-based white-noise test
Syntax

```
wnetestb varname [if] [in] [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong></td>
<td></td>
</tr>
<tr>
<td><code>table</code></td>
<td>display a table instead of graphical output</td>
</tr>
<tr>
<td><code>level(#)</code></td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td><strong>Plot</strong></td>
<td></td>
</tr>
<tr>
<td><code>marker_options</code></td>
<td>change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td><code>marker_label_options</code></td>
<td>add marker labels; change look or position</td>
</tr>
<tr>
<td><code>cline_options</code></td>
<td>add connecting lines; change look</td>
</tr>
<tr>
<td><strong>Add plots</strong></td>
<td></td>
</tr>
<tr>
<td><code>addplot(plot)</code></td>
<td>add other plots to the generated graph</td>
</tr>
<tr>
<td><strong>Y axis, X axis, Titles, Legend, Overall</strong></td>
<td></td>
</tr>
<tr>
<td><code>twoway_options</code></td>
<td>any options other than <code>by()</code> documented in [G-3] <code>twoway_options</code></td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `wnetestb`; see [TS] `tsset`. In addition, the time series must be dense (nonmissing with no gaps in the time variable) in the specified sample.

`varname` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

**Main**

- `table` displays the test results as a table instead of as the default graph.
- `level(#)` specifies the confidence level, as a percentage, for the confidence bands included on the graph. The default is `level(95)` or as set by `set level`; see [U] 20.8 Specifying the width of confidence intervals.

**Plot**

- `marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] `marker_options`.
- `marker_label_options` specify if and how the markers are to be labeled; see [G-3] `marker_label_options`.
- `cline_options` specify if the points are to be connected with lines and the rendition of those lines; see [G-3] `cline_options`.

**Add plots**

- `addplot(plot)` adds specified plots to the generated graph; see [G-3] `addplot_option`.

**Y axis, X axis, Titles, Legend, Overall**

- `twoway_options` are any of the options documented in [G-3] `twoway_options`, excluding `by()`. These include options for titling the graph (see [G-3] `title_options`) and for saving the graph to disk (see [G-3] `saving_option`).
Remarks and examples

Bartlett’s test is a test of the null hypothesis that the data come from a white-noise process of uncorrelated random variables having a constant mean and a constant variance.

For a discussion of this test, see Bartlett (1955, 92–94), Newton (1988, 172), or Newton (1996).

Example 1

In this example, we generate two time series and show the graphical and statistical tests that can be obtained from this command. The first time series is a white-noise process, and the second is a white-noise process with an embedded deterministic cosine curve.

```
. drop _all
. set seed 12393
. set obs 100
   number of observations (_N) was 0, now 100
. generate x1 = rnormal()
. generate x2 = rnormal() + cos(2*_pi*(_n-1)/10)
. generate time = _n
. tsset time
   time variable: time, 1 to 100
   delta: 1 unit
```

We can then submit the white-noise data to the `wntestb` command by typing

```
. wntestb x1
```

We can see in the graph that the values never appear outside the confidence bands. The test statistic has a $p$-value of 0.91, so we conclude that the process is not different from white noise. If we had wanted only the statistic without the plot, we could have used the `table` option.
Turning our attention to the other series \((x_2)\), we type

```
   . wntestb x2
```

Here the process does appear outside of the bands. In fact, it steps out of the bands at a frequency of 0.1 (exactly as we synthesized this process). We also have confirmation from the test statistic, at a \(p\)-value of 0.001, that the process is significantly different from white noise.

### Stored results

`wntestb` stores the following in `r()`:

- Scalars
  - `r(stat)` Bartlett's statistic
  - `r(p)` probability value

### Methods and formulas

If \(x(1), \ldots, x(T)\) is a realization from a white-noise process with variance \(\sigma^2\), the spectral distribution would be given by \(F(\omega) = \sigma^2 \omega\) for \(\omega \in [0, 1]\), and we would expect the cumulative periodogram (see [TS] `cumsq`) of the data to be close to the points \(S_k = k/q\) for \(q = \lfloor n/2 \rfloor + 1, k = 1, \ldots, q\). \([n/2]\) is the greatest integer less than or equal to \(n/2\).

Except for \(\omega = 0\) and \(\omega = .5\), the random variables \(2\hat{f}(\omega_k)/\sigma^2\) are asymptotically independent and identically distributed as \(\chi^2_2\). Because \(\chi^2_2\) is the same as twice a random variable distributed exponentially with mean 1, the cumulative periodogram has approximately the same distribution as the ordered values from a uniform (on the unit interval) distribution. Feller (1948) shows that this results in

\[
\lim_{q \to \infty} \Pr \left( \max_{1 \leq k \leq q} \sqrt{q} \left| U_k - \frac{k}{q} \right| \leq a \right) = \sum_{j=-\infty}^{\infty} (-1)^j e^{-2a^2j^2} = G(a)
\]

where \(U_k\) is the ordered uniform quantile. The Bartlett statistic is computed as

\[
B = \max_{1 \leq k \leq q} \sqrt{\frac{n}{2}} \left| \hat{F}_k - \frac{k}{q} \right|
\]
where $\hat{F}_k$ is the cumulative periodogram defined in terms of the sample spectral density $\hat{f}$ (see [TS] pergram) as

$$\hat{F}_k = \frac{\sum_{j=1}^{k} \hat{f}(\omega_j)}{\sum_{j=1}^{q} \hat{f}(\omega_j)}$$

The associated $p$-value for the Bartlett statistic and the confidence bands on the graph are computed as $1 - G(B)$ using Feller’s result.

Maurice Stevenson Bartlett (1910–2002) was a British statistician. Apart from a short period in industry, he spent his career teaching and researching at the universities of Cambridge, Manchester, London (University College), and Oxford. His many contributions include work on the statistical analysis of multivariate data (especially factor analysis) and time series and on stochastic models of population growth, epidemics, and spatial processes.

Acknowledgment

`wntestb` is based on the `wntestf` command by H. Joseph Newton (1996) of the Department of Statistics at Texas A&M University and coeditor of the `Stata Journal`.

References


Also see

[TS] `corrgram` — Tabulate and graph autocorrelations

[TS] `cump` — Graph cumulative spectral distribution

[TS] `pergram` — Periodogram

[TS] `tsset` — Declare data to be time-series data

[TS] `wntestq` — Portmanteau (Q) test for white noise
Description

wntestq performs the portmanteau (or Q) test for white noise.

Quick start

Perform Portmanteau’s test for white noise on series y using tsset data

```
   wntestq y
```

As above, but calculate 10 autocorrelations

```
   wntestq y, lags(10)
```

Menu

Statistics > Time series > Tests > Portmanteau white-noise test

Syntax

```
   wntestq varname [if] [in] [, lags(#) ]
```

You must tsset your data before using wntestq; see [TS] tsset. Also the time series must be dense (nonmissing with no gaps in the time variable) in the specified sample.

varname may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Option

lags(#) specifies the number of autocorrelations to calculate. The default is to use min([n/2] − 2, 40), where [n/2] is the greatest integer less than or equal to n/2.

Remarks and examples

Box and Pierce (1970) developed a portmanteau test of white noise that was refined by Ljung and Box (1978). See also Diggle (1990, sec. 2.5).
Example 1

In the example shown in [TS] `wntestb`, we generated two time series. One (x1) was a white-noise process, and the other (x2) was a white-noise process with an embedded cosine curve. Here we compare the output of the two tests.

```
. drop _all
. set seed 12393
. set obs 100
  number of observations (_N) was 0, now 100
. generate x1 = rnormal()
. generate x2 = rnormal() + cos(2*_pi*(_n-1)/10)
. generate time = _n
. tsset time
   time variable: time, 1 to 100
   delta: 1 unit
. wntestb x1, table
Cumulative periodogram white-noise test
  Bartlett's (B) statistic = 0.8038
  Prob > B = 0.5380
. wntestq x1
  Portmanteau test for white noise
  Portmanteau (Q) statistic = 31.0396
  Prob > chi2(40) = 0.8443
. wntestb x2, table
Cumulative periodogram white-noise test
  Bartlett's (B) statistic = 2.1653
  Prob > B = 0.0002
. wntestq x2
  Portmanteau test for white noise
  Portmanteau (Q) statistic = 225.6211
  Prob > chi2(40) = 0.0000
```

This example shows that both tests agree. For the first process, the Bartlett and portmanteau tests result in nonsignificant test statistics: a p-value of 0.5380 for `wntestb` and one of 0.8443 for `wntestq`.

For the second process, each test has a significant result to less than 0.0005.

### Stored results

`wntestq` stores the following in `r()`:

Scalars

- `r(stat)` — Q statistic
- `r(df)` — degrees of freedom
- `r(p)` — probability value
Methods and formulas

The portmanteau test relies on the fact that if \( x(1), \ldots, x(n) \) is a realization from a white-noise process. Then

\[
Q = n(n + 2) \sum_{j=1}^{m} \frac{1}{n - j} \hat{\rho}^2(j) \rightarrow \chi^2_m
\]

where \( m \) is the number of autocorrelations calculated (equal to the number of lags specified) and \( \rightarrow \) indicates convergence in distribution to a \( \chi^2 \) distribution with \( m \) degrees of freedom. \( \hat{\rho}_j \) is the estimated autocorrelation for lag \( j \); see [TS] corrgram for details.

References


Also see

[TS] corrgram — Tabulate and graph autocorrelations
[TS] cumsp — Graph cumulative spectral distribution
[TS] pergram — Periodogram
[TS] tsset — Declare data to be time-series data
[TS] wntestb — Bartlett’s periodogram-based test for white noise
Description

xcorr plots the sample cross-correlation function.

Quick start

Plot sample cross-correlation function for \( y_1 \) and \( y_2 \) using \texttt{tsset} data

\texttt{xcorr y1 y2}

As above, but display a table instead of a graph

\texttt{xcorr y1 y2, table}

As above, but only include 10 lags and leads of the cross-correlation function

\texttt{xcorr y1 y2, table lags(10)}

As above, and generate \texttt{xcorrvar} containing the cross-correlation values

\texttt{xcorr y1 y2, table lags(10) generate(xcorrvar)}

Menu

Statistics \> Time series \> Graphs \> Cross-correlogram for bivariate time series
Syntax

```
xcorr varname1 varname2 [if] [in] [, options]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generate(newvar)</code></td>
<td>create <code>newvar</code> containing cross-correlation values</td>
</tr>
<tr>
<td><code>table</code></td>
<td>display a table instead of graphical output</td>
</tr>
<tr>
<td><code>noplott</code></td>
<td>do not include the character-based plot in tabular output</td>
</tr>
<tr>
<td><code>lags(#)</code></td>
<td>include # lags and leads in graph</td>
</tr>
<tr>
<td><code>base(#)</code></td>
<td>value to drop to; default is 0</td>
</tr>
<tr>
<td><code>marker_options</code></td>
<td>change look of markers (color, size, etc.)</td>
</tr>
<tr>
<td><code>marker_label_options</code></td>
<td>add marker labels; change look or position</td>
</tr>
<tr>
<td><code>line_options</code></td>
<td>change look of dropped lines</td>
</tr>
<tr>
<td><code>addplot(plot)</code></td>
<td>add other plots to the generated graph</td>
</tr>
</tbody>
</table>

You must `tsset` your data before using `xcorr`; see [TS] `tsset`.

`varname1` and `varname2` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Options

`generate(newvar)` specifies a new variable to contain the cross-correlation values.
`table` requests that the results be presented as a table rather than the default graph.
`noplott` requests that the table not include the character-based plot of the cross-correlations.
`lags(#)` indicates the number of lags and leads to include in the graph. The default is to use \( \min(\lfloor n/2 \rfloor - 20) \).

`base(#)` specifies the value from which the lines should extend. The default is `base(0)`.
`marker_options`, `marker_label_options`, and `line_options` affect the rendition of the plotted cross-correlations.

`marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G-3] `marker_options`.

`marker_label_options` specify if and how the markers are to be labeled; see [G-3] `marker_label_options`.

`line_options` specify the look of the dropped lines, including pattern, width, and color; see [G-3] `line_options`.
Add plots

addplot(plot) provides a way to add other plots to the generated graph; see [G-3] addplot_option.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] twoway_options, excluding by(). These include options for titling the graph (see [G-3] title_options) and for saving the graph to disk (see [G-3] saving_option).

Remarks and examples

Example 1

We have a bivariate time series (Box, Jenkins, and Reinsel 2008, Series J) on the input and output of a gas furnace, where 296 paired observations on the input (gas rate) and output (% CO₂) were recorded every 9 seconds. The cross-correlation function is given by

```
use https://www.stata-press.com/data/r16/furnace
(TIMESLAB: Gas furnace)
xcorr input output, xline(5) lags(40)
```

Remarks and examples
We included a vertical line at lag 5, because there is a well-defined peak at this value. This peak indicates that the output lags the input by five periods. Further, the fact that the correlations are negative indicates that as input (coded gas rate) is increased, output (% CO$_2$) decreases.

We may obtain the table of autocorrelations and the character-based plot of the cross-correlations (analogous to the univariate time-series command `corrgram`) by specifying the `table` option.

```
.xcorr input output, table
```

<table>
<thead>
<tr>
<th>LAG</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>-0.1033</td>
</tr>
<tr>
<td>-19</td>
<td>-0.1027</td>
</tr>
<tr>
<td>-18</td>
<td>-0.0998</td>
</tr>
<tr>
<td>-17</td>
<td>-0.0932</td>
</tr>
<tr>
<td>-16</td>
<td>-0.0832</td>
</tr>
<tr>
<td>-15</td>
<td>-0.0727</td>
</tr>
<tr>
<td>-14</td>
<td>-0.0660</td>
</tr>
<tr>
<td>-13</td>
<td>-0.0662</td>
</tr>
<tr>
<td>-12</td>
<td>-0.0751</td>
</tr>
<tr>
<td>-11</td>
<td>-0.0927</td>
</tr>
<tr>
<td>-10</td>
<td>-0.1180</td>
</tr>
<tr>
<td>-9</td>
<td>-0.1484</td>
</tr>
<tr>
<td>-8</td>
<td>-0.1793</td>
</tr>
<tr>
<td>-7</td>
<td>-0.2059</td>
</tr>
<tr>
<td>-6</td>
<td>-0.2266</td>
</tr>
<tr>
<td>-5</td>
<td>-0.2429</td>
</tr>
<tr>
<td>-4</td>
<td>-0.2604</td>
</tr>
<tr>
<td>-3</td>
<td>-0.2865</td>
</tr>
<tr>
<td>-2</td>
<td>-0.3287</td>
</tr>
<tr>
<td>-1</td>
<td>-0.3936</td>
</tr>
<tr>
<td>0</td>
<td>-0.4845</td>
</tr>
<tr>
<td>1</td>
<td>-0.5985</td>
</tr>
<tr>
<td>2</td>
<td>-0.7251</td>
</tr>
<tr>
<td>3</td>
<td>-0.8429</td>
</tr>
<tr>
<td>4</td>
<td>-0.9246</td>
</tr>
<tr>
<td>5</td>
<td>-0.9503</td>
</tr>
<tr>
<td>6</td>
<td>-0.9146</td>
</tr>
<tr>
<td>7</td>
<td>-0.8294</td>
</tr>
<tr>
<td>8</td>
<td>-0.7166</td>
</tr>
<tr>
<td>9</td>
<td>-0.5998</td>
</tr>
<tr>
<td>10</td>
<td>-0.4952</td>
</tr>
<tr>
<td>11</td>
<td>-0.4107</td>
</tr>
<tr>
<td>12</td>
<td>-0.3479</td>
</tr>
<tr>
<td>13</td>
<td>-0.3049</td>
</tr>
<tr>
<td>14</td>
<td>-0.2779</td>
</tr>
<tr>
<td>15</td>
<td>-0.2632</td>
</tr>
<tr>
<td>16</td>
<td>-0.2548</td>
</tr>
<tr>
<td>17</td>
<td>-0.2463</td>
</tr>
<tr>
<td>18</td>
<td>-0.2332</td>
</tr>
<tr>
<td>19</td>
<td>-0.2135</td>
</tr>
<tr>
<td>20</td>
<td>-0.1869</td>
</tr>
</tbody>
</table>

Once again, the well-defined peak is apparent in the plot.
Methods and formulas

The cross-covariance function of lag \( k \) for time series \( x_1 \) and \( x_2 \) is given by

\[
\text{Cov}\{x_1(t), x_2(t + k)\} = R_{12}(k)
\]

This function is not symmetric about lag zero; that is,

\[
R_{12}(k) \neq R_{12}(-k)
\]

We define the cross-correlation function as

\[
\rho_{ij}(k) = \text{Corr}\{x_i(t), x_j(t + k)\} = \frac{R_{ij}(k)}{\sqrt{R_{ii}(0)R_{jj}(0)}}
\]

where \( \rho_{11} \) and \( \rho_{22} \) are the autocorrelation functions for \( x_1 \) and \( x_2 \), respectively. The sequence \( \rho_{12}(k) \) is the cross-correlation function and is drawn for lags \( k \in (-Q, -Q + 1, \ldots, -1, 0, 1, \ldots, Q - 1, Q) \).

If \( \rho_{12}(k) = 0 \) for all lags, \( x_1 \) and \( x_2 \) are not cross-correlated.

References


Also see

[TS] corrgam — Tabulate and graph autocorrelations
[TS] pergram — Periodogram
[TS] tsset — Declare data to be time-series data
**add factor.** An add factor is a quantity added to an endogenous variable in a forecast model. Add factors can be used to incorporate outside information into a model, and they can be used to produce forecasts under alternative scenarios.

**ARCH model.** An autoregressive conditional heteroskedasticity (ARCH) model is a regression model in which the conditional variance is modeled as an autoregressive (AR) process. The ARCH($m$) model is

$$y_t = x_t \beta + \epsilon_t$$

$$E(\epsilon_t^2 | \epsilon_{t-1}^2, \epsilon_{t-2}^2, \ldots) = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \cdots + \alpha_m \epsilon_{t-m}^2$$

where $\epsilon_t$ is a white-noise error term. The equation for $y_t$ represents the conditional mean of the process, and the equation for $E(\epsilon_t^2 | \epsilon_{t-1}^2, \epsilon_{t-2}^2, \ldots)$ specifies the conditional variance as an autoregressive function of its past realizations. Although the conditional variance changes over time, the unconditional variance is time invariant because $y_t$ is a stationary process. Modeling the conditional variance as an AR process raises the implied unconditional variance, making this model particularly appealing to researchers modeling fat-tailed data, such as financial data.

**ARFIMA model.** An autoregressive fractionally integrated moving-average (ARFIMA) model is a time-series model suitable for use with long-memory processes. ARFIMA models generalize autoregressive integrated moving-average (ARIMA) models by allowing the differencing parameter to be a real number in $(-0.5, 0.5)$ instead of requiring it to be an integer.

**ARIMA model.** An autoregressive integrated moving-average (ARIMA) model is a time-series model suitable for use with integrated processes. In an ARIMA($p,d,q$) model, the data is differenced $d$ times to obtain a stationary series, and then an ARMA($p,q$) model is fit to this differenced data. ARIMA models that include exogenous explanatory variables are known as ARMAX models.

**ARMA model.** An autoregressive moving-average (ARMA) model is a time series in which the current value of the variable is a linear function of its own past values and a white-noise error term. An autoregressive process, denoted as an AR($p$) process, is

$$y_t = \rho y_{t-1} + \epsilon_t$$

where $\epsilon_t$ is a white-noise error term. The equation for $y_t$ represents the conditional mean of the process, and the equation for $E(\epsilon_t^2 | \epsilon_{t-1}^2, \epsilon_{t-2}^2, \ldots)$ specifies the conditional variance as an autoregressive function of its past realizations. Although the conditional variance changes over time, the unconditional variance is time invariant because $y_t$ is a stationary process. Modeling the conditional variance as an AR process raises the implied unconditional variance, making this model particularly appealing to researchers modeling fat-tailed data, such as financial data.

**ARMAX model.** An ARMAX model is a time-series model in which the current period’s realization is the sum of an autoregressive (AR) process and a moving-average (MA) process. An ARMA($p,q$) model includes $p$ AR terms and $q$ MA terms. ARMA models with just a few lags are often able to fit data as well as pure AR or MA models with many more lags.

**ARMA process.** An autoregressive moving average (ARMA) process is a time series in which the current value of the variable is a linear function of its own past values and a weighted average of current and past realizations of a white-noise process. It consists of an autoregressive component and a moving-average component; see autoregressive (AR) process and moving-average (MA) process.

**ARMAX model.** An ARMAX model is a time-series model in which the current period’s realization is an ARMA process plus a linear function of a set of exogenous variables. Equivalently, an ARMAX model is a linear regression model in which the error term is specified to follow an ARMA process.

**autocorrelation function.** The autocorrelation function (ACF) expresses the correlation between periods $t$ and $t-k$ of a time series as function of the time $t$ and the lag $k$. For a stationary time series, the ACF does not depend on $t$ and is symmetric about $k = 0$, meaning that the correlation between periods $t$ and $t-k$ is equal to the correlation between periods $t$ and $t+k$.

**autoregressive (AR) process.** An autoregressive (AR) process is a time series in which the current value of a variable is a linear function of its own past values and a white-noise error term. A first-order AR process, denoted as an AR(1) process, is $y_t = \rho y_{t-1} + \epsilon_t$. An AR($p$) model contains $p$ lagged values of the dependent variable.
band-pass filter. Time-series filters are designed to pass or block stochastic cycles at specified frequencies. Band-pass filters, such as those implemented in tsfilter bk and tsfilter cf, pass through stochastic cycles in the specified range of frequencies and block all other stochastic cycles.

Cholesky ordering. Cholesky ordering is a method used to orthogonalize the error term in a VAR or VECM to impose a recursive structure on the dynamic model, so that the resulting impulse–response functions can be given a causal interpretation. The method is so named because it uses the Cholesky decomposition of the error-covariance matrix.

Cochrane–Orcutt estimator. This estimation is a linear regression estimator that can be used when the error term exhibits first-order autocorrelation. An initial estimate of the autocorrelation parameter $\rho$ is obtained from OLS residuals, and then OLS is performed on the transformed data $\hat{y}_t = y_t - \rho y_{t-1}$ and $\hat{x}_t = x_t - \rho x_{t-1}$.

cointegrating vector. A cointegrating vector specifies a stationary linear combination of nonstationary variables. Specifically, if each of the variables $x_1, x_2, \ldots, x_k$ is integrated of order one and there exists a set of parameters $\beta_1, \beta_2, \ldots, \beta_k$ such that $z_t = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$ is a stationary process, the variables $x_1, x_2, \ldots, x_k$ are said to be cointegrated, and the vector $\beta$ is known as a cointegrating vector.

conditional variance. Although the conditional variance is simply the variance of a conditional distribution, in time-series analysis the conditional variance is often modeled as an autoregressive (AR) process, giving rise to ARCH models.

correlogram. A correlogram is a table or graph showing the sample autocorrelations or partial autocorrelations of a time series.

covariance stationary process. A process is covariance stationary if the mean of the process is finite and independent of $t$, the unconditional variance of the process is finite and independent of $t$, and the covariance between periods $t$ and $t - s$ is finite and depends on $t - s$ but not on $t$ or $s$ themselves. Covariance stationary processes are also known as weakly stationary processes. See also stationary process.

cross-correlation function. The cross-correlation function expresses the correlation between one series at time $t$ and another series at time $t - k$ as a function of the time $t$ and lag $k$. If both series are stationary, the function does not depend on $t$. The function is not symmetric about $k = 0$: $\rho_{12}(k) \neq \rho_{12}(-k)$.

cyclical component. A cyclical component is a part of a time series that is a periodic function of time. Deterministic functions of time are deterministic cyclical components, and random functions of time are stochastic cyclical components. For example, fixed seasonal effects are deterministic cyclical components and random seasonal effects are stochastic seasonal components. Random coefficients on time inside of periodic functions form an especially useful class of stochastic cyclical components; see [TS] ucm.

deterministic trend. A deterministic trend is a deterministic function of time that specifies the long-run tendency of a time series.

difference operator. The difference operator $\Delta$ denotes the change in the value of a variable from period $t - 1$ to period $t$. Formally, $\Delta y_t = y_t - y_{t-1}$, and $\Delta^2 y_t = \Delta (y_t - y_{t-1}) = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2}$.

drift. Drift is the constant term in a unit-root process. In
$$y_t = \alpha + y_{t-1} + \epsilon_t$$
$\alpha$ is the drift when $\epsilon_t$ is a stationary, zero-mean process.
**dynamic forecast.** A dynamic forecast uses forecast values wherever lagged values of the endogenous variables appear in the model, allowing one to forecast multiple periods into the future. See also *static forecast.*

**dynamic-multiplier function.** A dynamic-multiplier function measures the effect of a shock to an exogenous variable on an endogenous variable. The $k$th dynamic-multiplier function of variable $i$ on variable $j$ measures the effect on variable $j$ in period $t + k$ in response to a one-unit shock to variable $i$ in period $t$, holding everything else constant.

**endogenous variable.** An endogenous variable is a regressor that is correlated with the unobservable error term. Equivalently, an endogenous variable is one whose values are determined by the equilibrium or outcome of a structural model.

**exogenous variable.** An exogenous variable is a regressor that is not correlated with any of the unobservable error terms in the model. Equivalently, an exogenous variable is one whose values change independently of the other variables in a structural model.

**exponential smoothing.** Exponential smoothing is a method of smoothing a time series in which the smoothed value at period $t$ is equal to a fraction $\alpha$ of the series value at time $t$ plus a fraction $1 - \alpha$ of the previous period’s smoothed value. The fraction $\alpha$ is known as the smoothing parameter.

**forecast-error variance decomposition.** Forecast-error variance decompositions measure the fraction of the error in forecasting variable $i$ after $h$ periods that is attributable to the orthogonalized shocks to variable $j$.

**forward operator.** The forward operator $F$ denotes the value of a variable at time $t + 1$. Formally, $Fy_t = y_{t+1}$, and $F^2y_t = Fy_{t+1} = y_{t+2}$. A forward operator is also known as a lead operator.

**frequency-domain analysis.** Frequency-domain analysis is analysis of time-series data by considering its frequency properties. The spectral density function and the spectral distribution function are key components of frequency-domain analysis, so it is often called spectral analysis. In Stata, the *cumsp* and *pergram* commands are used to analyze the sample spectral distribution and density functions, respectively. *psdensity* estimates the spectral density or the spectral distribution function after estimating the parameters of a parametric model using *arfima*, *arima*, or *ucm*.

**gain (of a linear filter).** The gain of a linear filter scales the spectral density of the unfiltered series into the spectral density of the filtered series for each frequency. Specifically, at each frequency, multiplying the spectral density of the unfiltered series by the square of the gain of a linear filter yields the spectral density of the filtered series. If the gain at a particular frequency is 1, the filtered and unfiltered spectral densities are the same at that frequency and the corresponding stochastic cycles are passed through perfectly. If the gain at a particular frequency is 0, the filter removes all the corresponding stochastic cycles from the unfiltered series.

**GARCH model.** A generalized autoregressive conditional heteroskedasticity (GARCH) model is a regression model in which the conditional variance is modeled as an ARMA process. The GARCH($m$, $k$) model is

$$y_t = x_t \beta + \epsilon_t$$

$$\sigma_t^2 = \gamma_0 + \gamma_1 \epsilon_{t-1}^2 + \cdots + \gamma_m \epsilon_{t-m}^2 + \delta_1 \sigma_{t-1}^2 + \cdots + \delta_k \sigma_{t-k}^2$$

where the equation for $y_t$ represents the conditional mean of the process and $\sigma_t$ represents the conditional variance. See [TS] *arch* or Hamilton (1994, chap. 21) for details on how the conditional variance equation can be viewed as an ARMA process. GARCH models are often used because the ARMA specification often allows the conditional variance to be modeled with fewer parameters than are required by a pure ARCH model. Many extensions to the basic GARCH model exist; see [TS] *arch* for those that are implemented in Stata. See also *ARCH model.*
generalized least-squares estimator. A generalized least-squares (GLS) estimator is used to estimate the parameters of a regression function when the error term is heteroskedastic or autocorrelated. In the linear case, GLS is sometimes described as “OLS on transformed data” because the GLS estimator can be implemented by applying an appropriate transformation to the dataset and then using OLS.

Granger causality. The variable \( x \) is said to Granger-cause variable \( y \) if, given the past values of \( y \), past values of \( x \) are useful for predicting \( y \).

high-pass filter. Time-series filters are designed to pass or block stochastic cycles at specified frequencies. High-pass filters, such as those implemented in \texttt{tsfilter bw} and \texttt{tsfilter hp}, pass through stochastic cycles above the cutoff frequency and block all other stochastic cycles.

Holt–Winters smoothing. A set of methods for smoothing time-series data that assume that the value of a time series at time \( t \) can be approximated as the sum of a mean term that drifts over time, as well as a time trend whose strength also drifts over time. Variations of the basic method allow for seasonal patterns in data, as well.

impulse–response function. An impulse–response function (IRF) measures the effect of a shock to an endogenous variable on itself or another endogenous variable. The \( k \)th impulse–response function of variable \( i \) on variable \( j \) measures the effect on variable \( j \) in period \( t + k \) in response to a one-unit shock to variable \( i \) in period \( t \), holding everything else constant.

independent and identically distributed. A series of observations is independent and identically distributed (i.i.d.) if each observation is an independent realization from the same underlying distribution. In some contexts, the definition is relaxed to mean only that the observations are independent and have identical means and variances; see Davidson and MacKinnon (1993, 42).

integrated process. A nonstationary process is integrated of order \( d \), written \( I(d) \), if the process must be differenced \( d \) times to produce a stationary series. An \( I(1) \) process \( y_t \) is one in which \( \Delta y_t \) is stationary.

Kalman filter. The Kalman filter is a recursive procedure for predicting the state vector in a state-space model.

lag operator. The lag operator \( L \) denotes the value of a variable at time \( t - 1 \). Formally, \( Ly_t = y_{t-1} \), and \( L^2 y_t = Ly_{t-1} = y_{t-2} \).

lead operator. See forward operator.

linear filter. A linear filter is a sequence of weights used to compute a weighted average of a time series at each time period. More formally, a linear filter \( \alpha(L) \) is

\[
\alpha(L) = \alpha_0 + \alpha_1 L + \alpha_2 L^2 + \cdots = \sum_{\tau=0}^{\infty} \alpha_\tau L^\tau
\]

where \( L \) is the lag operator. Applying the linear filter \( \alpha(L) \) to the time series \( x_t \) yields a sequence of weighted averages of \( x_t \):

\[
\alpha(L)x_t = \sum_{\tau=0}^{\infty} \alpha_\tau L^\tau x_{t-\tau}
\]

long-memory process. A long-memory process is a stationary process whose autocorrelations decay at a slower rate than a short-memory process. ARFIMA models are typically used to represent long-memory processes, and ARMA models are typically used to represent short-memory processes.
moving-average (MA) process. A moving-average (MA) process is a time-series process in which the current value of a variable is modeled as a weighted average of current and past realizations of a white-noise process and, optionally, a time-invariant constant. By convention, the weight on the current realization of the white-noise process is equal to one, and the weights on the past realizations are known as the MA coefficients. A first-order MA process, denoted as an MA(1) process, is $y_t = \theta \epsilon_{t-1} + \epsilon_t$.

multivariate GARCH models. Multivariate GARCH models are multivariate time-series models in which the conditional covariance matrix of the errors depends on its own past and its past shocks. The acute trade-off between parsimony and flexibility has given rise to a plethora of models; see [TS] mgarch.

Newey–West covariance matrix. The Newey–West covariance matrix is a member of the class of heteroskedasticity- and autocorrelation-consistent (HAC) covariance matrix estimators used with time-series data that produces covariance estimates that are robust to both arbitrary heteroskedasticity and autocorrelation up to a prespecified lag.

one-step-ahead forecast. See static forecast.

orthogonalized impulse–response function. An orthogonalized impulse–response function (OIRF) measures the effect of an orthogonalized shock to an endogenous variable on itself or another endogenous variable. An orthogonalized shock is one that affects one variable at time $t$ but no other variables. See [TS] irf create for a discussion of the difference between IRFs and OIRFs.

output gap. The output gap, sometimes called the GDP gap, is the difference between the actual output of an economy and its potential output.

partial autocorrelation function. The partial autocorrelation function (PACF) expresses the correlation between periods $t$ and $t-k$ of a time series as a function of the time $t$ and lag $k$, after controlling for the effects of intervening lags. For a stationary time series, the PACF does not depend on $t$. The PACF is not symmetric about $k = 0$: the partial autocorrelation between $y_t$ and $y_{t-k}$ is not equal to the partial autocorrelation between $y_t$ and $y_{t+k}$.

periodogram. A periodogram is a graph of the spectral density function of a time series as a function of frequency. The pergram command first standardizes the amplitude of the density by the sample variance of the time series, and then plots the logarithm of that standardized density. Peaks in the periodogram represent cyclical behavior in the data.

phase function. The phase function of a linear filter specifies how the filter changes the relative importance of the random components at different frequencies in the frequency domain.

Phillips curve. The Phillips curve is a macroeconomic relationship between inflation and economic activity, usually expressed as an equation involving inflation and the output gap. Historically, the Phillips curve describes an inverse relationship between the unemployment rate and the rate of rises in wages.

portmanteau statistic. The portmanteau, or $Q$, statistic is used to test for white noise and is calculated using the first $m$ autocorrelations of the series, where $m$ is chosen by the user. Under the null hypothesis that the series is a white-noise process, the portmanteau statistic has a $\chi^2$ distribution with $m$ degrees of freedom.

Prais–Winsten estimator. A Prais–Winsten estimator is a linear regression estimator that is used when the error term exhibits first-order autocorrelation; see also Cochrane–Orcutt estimator. Here the first observation in the dataset is transformed as $\tilde{y}_1 = \sqrt{1 - \rho^2} \ y_1$ and $\tilde{x}_1 = \sqrt{1 - \rho^2} \ x_1$, so that the first observation is not lost. The Prais–Winsten estimator is a generalized least-squares estimator.

priming values. Priming values are the initial, preestimation values used to begin a recursive process.
random walk. A random walk is a time-series process in which the current period’s realization is equal to the previous period’s realization plus a white-noise error term: \( y_t = y_{t-1} + \epsilon_t \). A random walk with drift also contains a nonzero time-invariant constant: \( y_t = \delta + y_{t-1} + \epsilon_t \). The constant term \( \delta \) is known as the drift parameter. An important property of random-walk processes is that the best predictor of the value at time \( t + 1 \) is the value at time \( t \) plus the value of the drift parameter.

recursive regression analysis. A recursive regression analysis involves performing a regression at time \( t \) by using all available observations from some starting time \( t_0 \) through time \( t \), performing another regression at time \( t + 1 \) by using all observations from time \( t_0 \) through time \( t + 1 \), and so on. Unlike a rolling regression analysis, the first period used for all regressions is held fixed.

regressand. The regressand is the variable that is being explained or predicted in a regression model. Synonyms include dependent variable, left-hand-side variable, and endogenous variable.

regressor. Regressors are variables in a regression model used to predict the regressand. Synonyms include independent variable, right-hand-side variable, explanatory variable, predictor variable, and exogenous variable.

rolling regression analysis. A rolling, or moving window, regression analysis involves performing regressions for each period by using the most recent \( m \) periods’ data, where \( m \) is known as the window size. At time \( t \) the regression is fit using observations for times \( t - 19 \) through time \( t \); at time \( t + 1 \) the regression is fit using the observations for time \( t - 18 \) through \( t + 1 \); and so on.

seasonal difference operator. The period-\( s \) seasonal difference operator \( \Delta_s \) denotes the difference in the value of a variable at time \( t \) and time \( t - s \). Formally, \( \Delta_s y_t = y_t - y_{t-s} \), and \( \Delta^2_s y_t = \Delta_s(y_t - y_{t-s}) = (y_t - y_{t-s}) - (y_{t-s} - y_{t-2s}) = y_t - 2y_{t-s} + y_{t-2s} \).

serial correlation. Serial correlation refers to regression errors that are correlated over time. If a regression model does not contain lagged dependent variables as regressors, the OLS estimates are consistent in the presence of mild serial correlation, but the covariance matrix is incorrect. When the model includes lagged dependent variables and the residuals are serially correlated, the OLS estimates are biased and inconsistent. See, for example, Davidson and MacKinnon (1993, chap. 10) for more information.

serial correlation tests. Because OLS estimates are at least inefficient and potentially biased in the presence of serial correlation, econometricians have developed many tests to detect it. Popular ones include the Durbin–Watson (1950, 1951, 1971) test, the Breusch–Pagan (1980) test, and Durbin’s (1970) alternative test. See [R] regress postestimation time series.

smoothing. Smoothing a time series refers to the process of extracting an overall trend in the data. The motivation behind smoothing is the belief that a time series exhibits a trend component as well as an irregular component and that the analyst is interested only in the trend component. Some smoothers also account for seasonal or other cyclical patterns.

spectral analysis. See frequency-domain analysis.

spectral density function. The spectral density function is the derivative of the spectral distribution function. Intuitively, the spectral density function \( f(\omega) \) indicates the amount of variance in a time series that is attributable to sinusoidal components with frequency \( \omega \). See also spectral distribution function. The spectral density function is sometimes called the spectrum.

spectral distribution function. The (normalized) spectral distribution function \( F(\omega) \) of a process describes the proportion of variance that can be explained by sinusoids with frequencies in the range \( (0, \omega) \), where \( 0 \leq \omega \leq \pi \). The spectral distribution and density functions used in frequency-domain analysis are closely related to the autocorrelation function used in time-domain analysis; see Chatfield (2004, chap. 6) and Wei (2006, chap. 12).
spectrum. See *spectral density function*.

**state-space model.** A state-space model describes the relationship between an observed time series and an unobservable state vector that represents the “state” of the world. The measurement equation expresses the observed series as a function of the state vector, and the transition equation describes how the unobserved state vector evolves over time. By defining the parameters of the measurement and transition equations appropriately, one can write a wide variety of time-series models in the state-space form.

**static forecast.** A static forecast uses actual values wherever lagged values of the endogenous variables appear in the model. As a result, static forecasts perform at least as well as dynamic forecasts, but static forecasts cannot produce forecasts into the future if lags of the endogenous variables appear in the model. Because actual values will be missing beyond the last historical time period in the dataset, static forecasts can forecast only one period into the future (assuming only first lags appear in the model); thus they are often called one-step-ahead forecasts.

**stationary process.** A process is stationary if the joint distribution of \( y_1, \ldots, y_k \) is the same as the joint distribution of \( y_{1+\tau}, \ldots, y_{k+\tau} \) for all \( k \) and \( \tau \). Intuitively, shifting the origin of the series by \( \tau \) units has no effect on the joint distributions; the marginal distribution of the series does not change over time. A stationary process is also known as a strictly stationary process or a strongly stationary process. See also *covariance stationary process*.

**steady-state equilibrium.** The steady-state equilibrium is the predicted value of a variable in a dynamic model, ignoring the effects of past shocks, or, equivalently, the value of a variable, assuming that the effects of past shocks have fully died out and no longer affect the variable of interest.

**stochastic cycle.** A stochastic cycle is a cycle characterized by an amplitude, phase, or frequency that can be random functions of time. See *cyclical component*.

**stochastic equation.** A stochastic equation, in contrast to an identity, is an equation in a forecast model that includes a random component, most often in the form of an additive error term. Stochastic equations include parameters that must be estimated from historical data.

**stochastic trend.** A stochastic trend is a nonstationary random process. Unit-root process and random coefficients on time are two common stochastic trends. See [TS] ucm for examples and discussions of more commonly applied stochastic trends.

**strictly stationary process.** See *stationary process*.

**strongly stationary process.** See *stationary process*.

**structural model.** In time-series analysis, a structural model is one that describes the relationship among a set of variables, based on underlying theoretical considerations. Structural models may contain both endogenous and exogenous variables.

**SV AR.** A structural vector autoregressive (SVAR) model is a type of VAR in which short- or long-run constraints are placed on the resulting *impulse–response functions*. The constraints are usually motivated by economic theory and therefore allow causal interpretations of the IRFs to be made.

**time-domain analysis.** Time-domain analysis is analysis of data viewed as a sequence of observations observed over time. The autocorrelation function, linear regression, ARCH models, and ARIMA models are common tools used in time-domain analysis.
**trend.** The trend specifies the long-run behavior in a time series. The trend can be deterministic or stochastic. Many economic, biological, health, and social time series have long-run tendencies to increase or decrease. Before the 1980s, most time-series analysis specified the long-run tendencies as deterministic functions of time. Since the 1980s, the stochastic trends implied by unit-root processes have become a standard part of the toolkit.

**unit-root process.** A unit-root process is one that is integrated of order one, meaning that the process is nonstationary but that first-differencing the process produces a stationary series. The simplest example of a unit-root process is the random walk. See Hamilton (1994, chap. 15) for a discussion of when general ARMA processes may contain a unit root.

**unit-root tests.** Whether a process has a unit root has both important statistical and economic ramifications, so a variety of tests have been developed to test for them. Among the earliest tests proposed is the one by Dickey and Fuller (1979), though most researchers now use an improved variant called the augmented Dickey–Fuller test instead of the original version. Other common unit-root tests implemented in Stata include the DF–GLS test of Elliott, Rothenberg, and Stock (1996) and the Phillips–Perron (1988) test. See [TS] dfuller, [TS] dfgls, and [TS] pperron. Variants of unit-root tests suitable for panel data have also been developed; see [XT] xtunitroot.

**VAR.** A vector autoregressive (VAR) model is a multivariate regression technique in which each dependent variable is regressed on lags of itself and on lags of all the other dependent variables in the model. Occasionally, exogenous variables are also included in the model.

**VECM.** A vector error-correction model (VECM) is a type of VAR that is used with variables that are cointegrated. Although first-differencing variables that are integrated of order one makes them stationary, fitting a VAR to such first-differenced variables results in misspecification error if the variables are cointegrated. See The multivariate VECM specification in [TS] vec intro for more on this point.

**weak stationary process.** See covariance stationary process.

**white-noise process.** A variable $u_t$ represents a white-noise process if the mean of $u_t$ is zero, the variance of $u_t$ is $\sigma^2$, and the covariance between $u_t$ and $u_s$ is zero for all $s \neq t$.

**Yule–Walker equations.** The Yule–Walker equations are a set of difference equations that describe the relationship among the autocovariances and autocorrelations of an autoregressive moving-average (ARMA) process.

**References**


Subject and author index

See the combined subject index and the combined author index in the Glossary and Index.