Title

stata.com

forecast solve — Obtain static and dynamic forecasts

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
```

Same as above, but with forecasts starting at 1990q1 and ending at 1995q3

```
forecast solve, begin(q(1990q1)) end(q(1995q3))
```

Same as above, and change prefix of predicted endogenous variables to `hat`

```
forecast solve, begin(q(1990q1)) end(q(1995q3)) prefix(hat)
```

Same 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_)))
```

Menu

Statistics > Time series > Forecasting

Syntax

```
forecast solve [ , { prefix(stub) | suffix(stub) } options ]
```

options	Description
Model	
* `prefix(`*string*`)`	specify prefix for forecast variables
* `suffix(`*string*`)`	specify suffix for forecast variables
`begin(`*time_constant*`)`	specify period to begin forecasting
† `end(`*time_constant*`)`	specify period to end forecasting
† `periods(`*#*`)`	specify number of periods to forecast
`double`	store forecast variables as `doubles` instead of as `floats`
`static`	produce static forecasts instead of dynamic forecasts
`actuals`	use actual values if available instead of forecasts
Simulation	
`simulate(`*sim_technique*`,` *sim_statistic sim_options*`)`	specify simulation technique and options
Reporting	
`log(`*log_level*`)`	specify level of logging display; *log_level* may be `detail`, `on`, `brief`, or `off`
Solver	
`vtolerance(`*#*`)`	specify tolerance for forecast values
`ztolerance(`*#*`)`	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.

`collect` is allowed; see [U] **11.1.10 Prefix commands**.

sim_technique	Description
`betas`	draw multivariate-normal parameter vectors
`errors`	draw additive errors from multivariate normal distribution
`residuals`	draw additive residuals based on static forecast errors

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	Description
mean	record the mean of the simulation forecasts
variance	record the variance of the simulation forecasts
stddev	record the standard deviation of the simulation forecasts

sim_options	Description
saving(*filename*, ...)	save results to file; save statistics in double precision; save results to filename every # replications
nodots	suppress replication dots
reps(#)	perform # replications; default is reps(50)

Options

Model

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.

end(*time_constant*) requests that forecast end forecasting at period *time_constant*. By default, forecast produces forecasts for all periods on or after begin() in the dataset.

periods(#) specifies the number of periods after begin() to forecast. By default, forecast produces forecasts for all periods on or after begin() in the dataset.

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.

Simulation

`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`. 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

`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_statistic*s, 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)`.

Reporting

`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.

Solver

`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()**.

`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 `technique(dampedgaussseidel 0.2)`, which works well in most situations. If you have convergence issues, first try continuing to use `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 [M-5] **solvenl()**.

Remarks and examples

stata.com

For an overview of the `forecast` commands, see [TS] **forecast**. This manual entry assumes you have already read that manual entry. The `forecast solve` command solves a forecast model in Stata. Before you can solve a model, you must first create a model using `forecast create`, and you must add at least one equation using `forecast estimates`, `forecast coefvector`, or `forecast identity`. We covered the most commonly used options of `forecast solve` in the examples in [TS] **forecast**.

Here we focus on two sets of options that are available with `forecast solve`. First, we discuss the `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:

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 `actuals` option of `forecast solve`.

▷ Example 1: Specifying alternative scenarios

`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 autoregressive (VAR) model to forecast the variables `gdp` and `oil`. Results obtained from the `varsoc` command indicate that the Hannan–Quinn information criterion is minimized when the VAR model includes two lags. First, we fit our VAR model and store the estimation results:

```
. use https://www.stata-press.com/data/r18/gdpoil
. var gdp oil, lags(1 2)
Vector autoregression
Sample: 1986q4 thru 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
Equation           Parms      RMSE     R-sq      chi2     P>chi2
------------------------------------------------------------------------------
gdp                    5     1.88516   0.1820   18.91318   0.0008
oil                    5     11.8776   0.1140   10.93614   0.0273
------------------------------------------------------------------------------
```

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
gdp						
gdp						
L1.	.1498285	.1015076	1.48	0.140	-.0491227	.3487797
L2.	.3465238	.1022446	3.39	0.001	.146128	.5469196
oil						
L1.	-.0374609	.0167968	-2.23	0.026	-.070382	-.0045399
L2.	.0119564	.0164599	0.73	0.468	-.0203043	.0442172
_cons	1.519983	.4288145	3.54	0.000	.6795226	2.360444
oil						
gdp						
L1.	.8102233	.6395579	1.27	0.205	-.4432871	2.063734
L2.	1.090244	.6442017	1.69	0.091	-.1723684	2.352856
oil						
L1.	.0995271	.1058295	0.94	0.347	-.1078949	.3069491
L2.	-.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 `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:

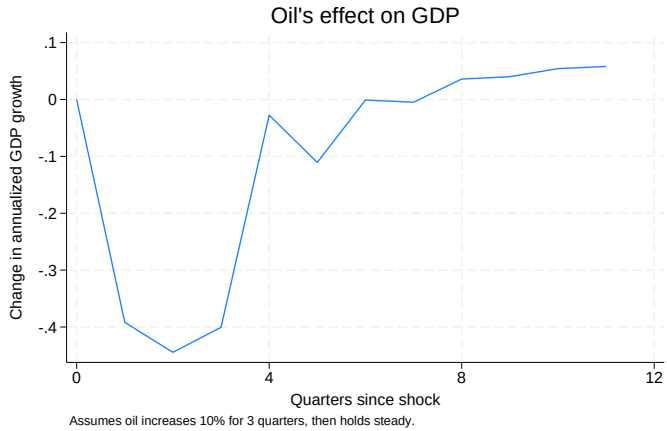
```
. replace oil = 10 if qdate == tq(2008q1)
(1 real change made)
. replace oil = 10 if qdate == tq(2008q2)
(1 real change made)
. replace oil = 10 if qdate == tq(2008q3)
(1 real change made)
. replace oil = 0 if qdate > tq(2008q3)
(9 real changes made)
```

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:

```
. 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:

$$y_t = \alpha + \beta x_t + \epsilon_t \tag{1}$$

where ϵ_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 $\widehat{\alpha}$ and $\widehat{\beta}$. Assuming we have data for exogenous variable x at time $T+1$, we could forecast y_{T+1} as

$$\widehat{y}_{T+1} = \widehat{\alpha} + \widehat{\beta} x_{T+1} \tag{2}$$

However, there are several factors that prevent us from guaranteeing ex ante that y_{T+1} will indeed equal $\widehat{y}_{T+1}$. 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 α and β by OLS to obtain $\widehat{\alpha}$ and $\widehat{\beta}$, 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 α and β mean that there is some level of uncertainty surrounding our predicted value $\widehat{y}_{T+1}$ as well.

Second, (1) states that y_t depends not just on α, β, and x_t but also on an unobserved error term ϵ_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 $(\widehat{\alpha}, \widehat{\beta})$ 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 ϵ_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 $\widehat{y}_{T+1}$ 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:

```
. use https://www.stata-press.com/data/r18/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 oilmodel 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.
```

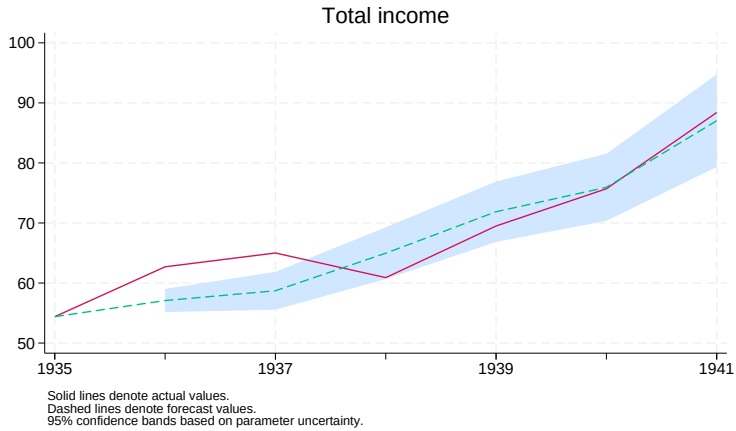
Now we are ready to solve our model. We are going to begin dynamic forecasts in 1936, and we are going to perform 100 replications. We will store the point forecasts in variables prefixed with `d_`, and we will store the standard deviations of our forecasts in variables prefixed with `sd_`. Because the 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

```
. set seed 1
. forecast solve, prefix(d_) begin(1936)
> simulate(betas, 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): ......................
> ........................    50
..................................................   100

Forecast 7 variables spanning 6 periods.
------------------------------------------------------------
```

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:

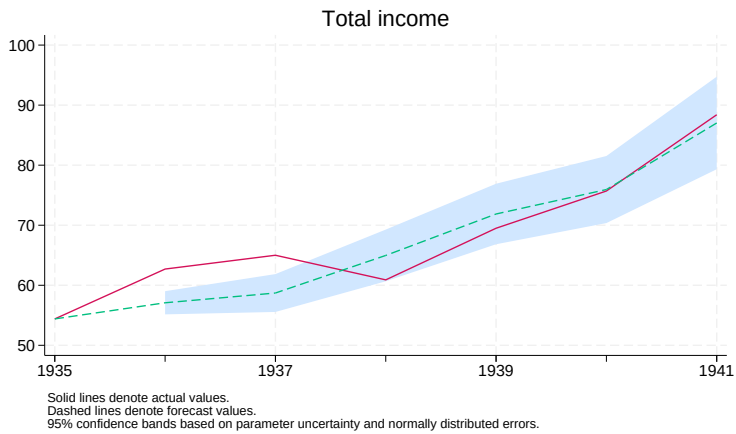
```
. drop d_y_dn d_y_up
```

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): ......................
> ..........................    50
..................................................    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.

◁

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 $\mathbf{y}$, and we represent the exogenous variables in the model as the $m \times 1$ vector $\mathbf{x}$. We refer to the contemporaneous values as $\mathbf{y}_t$ and $\mathbf{x}_t$; for notational simplicity, we refer to lagged values as $\mathbf{y}_{t-1}$ and $\mathbf{x}_{t-1}$ with the implication that further lags of the variables can also be included with no loss of generality. We use $\boldsymbol{\theta}$ to refer to the vector of all the estimated parameters in all the equations of the model. We use $\mathbf{u}_t$ and $\mathbf{u}_{t-1}$ to refer to contemporaneous and lagged error terms, respectively.

The `forecast` commands solve models of the form

$$y_{it} = f_i(\mathbf{y}_{-i,t}, \mathbf{y}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t-1}, \mathbf{u_t}, \mathbf{u}_{t-1}; \boldsymbol{\theta}) \tag{3}$$

where $i = 1, \ldots, k$ and $\mathbf{y}_{-i,t}$ 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 $f_i(\cdot)$ 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 coefvector` 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 $\mathbf{R}$. Then when `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 jth row from $\mathbf{R}$. That preserves the correlation structure among the error terms of the estimation result's equations.

If you specify a coefficient vector using `forecast coefvector` and specify either the `variance()` option or the `errorvariance()` option (or both), *sim_technique* `residuals` considers the equation represented by the coefficient vector to be stochastic and resamples residuals for that equation.

When you specify the `simulate()` option with *sim_technique* `errors`, `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 `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 `forecast` commands look in `e(rmse)`, `e(sigma)`, and `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 *sim_technique* `errors`, `forecast solve` exits with an error message. Multiple-equation commands typically set `e(Sigma)` so that the randomly drawn errors reflect the estimated error correlation structure.

If you specify a coefficient vector using `forecast coefvector` and specify the `errorvariance()` option, *sim_technique* `errors` simulates errors for that equation. Otherwise, the equation is treated like an identity and no errors are added.

`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; `forecast solve` does not restrict itself to the static-forecast residuals for a single panel when simulating that panel.

References

Broyden, C. G. 1970. Recent developments in solving nonlinear algebraic systems. In *Numerical Methods for Nonlinear Algebraic Equations*, ed. P. Rabinowitz, 61–73. London: Gordon and Breach Science Publishers.

Fair, R. C. 1984. *Specification, Estimation, and Analysis of Macroeconometric Models*. Cambridge, MA: Harvard University Press.

Hamilton, J. D. 2003. What is an oil shock? *Journal of Econometrics* 113: 363–398. https://doi.org/10.1016/S0304-4076(02)00207-5.

Kilian, L., and R. J. Vigfusson. 2013. Do oil prices help forecast U.S. real GDP? The role of nonlinearities and asymmetries. *Journal of Business and Economic Statistics* 31: 78–93. https://doi.org/10.1080/07350015.2012.740436.

Klein, L. R. 1950. *Economic Fluctuations in the United States 1921–1941*. New York: Wiley.

Powell, M. J. D. 1970. A hybrid method for nonlinear equations. In *Numerical Methods for Nonlinear Algebraic Equations*, ed. P. Rabinowitz, 87–114. London: Gordon and Breach Science Publishers.

Also see

[TS] **forecast** — Econometric model forecasting

[TS] **forecast adjust** — Adjust variables to produce alternative forecasts

[TS] **forecast drop** — Drop forecast variables

[R] **set seed** — Specify random-number seed and state