rolling — Rolling-window and recursive estimation

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)
```

Same 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
```

options	Description
Main	
* window(#)	number of consecutive data points in each sample
recursive	use recursive samples
rrecursive	use reverse recursive samples
Options	
clear	replace data in memory with results
saving(*filename*, ...)	save results to *filename*; save statistics in double precision; save results to *filename* every # replications
stepsize(#)	number of periods to advance window
start(*time_constant*)	period at which rolling is to start
end(*time_constant*)	period at which rolling is to end
keep(*varname*[, start])	save *varname* with results; optionally, use value at left edge of window
Reporting	
nodots	suppress replication dots
dots(#)	display dots every # replications
noisily	display any output from *command*
trace	trace *command*'s execution
Advanced	
reject(*exp*)	identify invalid results

* 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

Main

`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.

Options

`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 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**.

`stepsize(#)` specifies the number of periods the window is to be advanced each time *command* is executed.

`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.

Reporting

`nodots` and `dots(#)` specify whether to display replication dots. By default, one dot character is displayed for each window. An "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.

Advanced

`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_list, 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/r19/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)
(file betas.dta not found)
Rolling replications (295): .........10.........20.........30.........40.......
> ..50.........60.........70.........80.........90.........100.........110.....
> ....120.........130.........140.........150.........160.........170.........1
> 80.........190.........200.........210.........220.........230.........240...
> ......250.........260.........270.........280.........290..... done
file betas.dta saved
```

Our dataset has both a time variable `t` that runs consecutively and a date variable `date` that measures the calendar date and therefore has gaps at weekends and holidays. Had we used the `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 `keep(date)` option so that we could refer to the `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
```

	start	end	date	_b_spxadj	_b_cons	_se_spxadj	_se_cons
1.	1	200	16oct2003	1.043422	-.0181504	.0658531	.0748295
2.	2	201	17oct2003	1.039024	-.0126876	.0656893	.074609
3.	3	202	20oct2003	1.038371	-.0235616	.0654591	.0743851

The variables `start` and `end` indicate the first and last observations used each time that `rolling` called `regress`, and the `date` variable contains the calendar date corresponding the period represented by `end`. The remaining variables are the estimated coefficients and standard errors from the regression. In our example , `_b_spxadj` contains the estimated betas, and `_b_cons` contains the estimated alphas. The variables `_se_spxadj` and `_se_cons` have the corresponding standard errors.

Finally, we compute the confidence intervals for the betas and examine how they have changed over time:

```
. 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") xtitle("Date")
```

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

```
. 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

```
. use https://www.stata-press.com/data/r19/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)
```

	start	end	_b_spxadj	_b_cons	_se_spxadj	_se_cons
1.	1	200	1.043422	-.0181504	.0658531	.0748295
2.	1	201	1.039024	-.0126876	.0656893	.074609
3.	1	202	1.037687	-.016475	.0655896	.0743481

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 ϵ_t is a regression error term, and then compute

$$\widehat{i_{t+1}} = \widehat{\beta_0} + \widehat{\beta_1} i_t + \widehat{\beta_2} s_t$$

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/r19/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* and coauthor of the Stata Press book *Environmental Econometrics Using Stata* for an earlier rolling regression command.

References

Kmenta, J. 1997. *Elements of Econometrics*. 2nd ed. Ann Arbor: University of Michigan Press. https://doi.org/10.3998/mpub.15701.

Rajbhandari, A. 2016. Tests of forecast accuracy and forecast encompassing. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2016/06/01/tests-of-forecast-accuracy-and-forecast-encompassing/.

Stock, J. H., and M. W. Watson. 2019. *Introduction to Econometrics*. 4th ed. New York: Pearson.

Also see

[D] **statsby** — Collect statistics for a command across a by list

[R] **Stored results** — Stored results