simulate — Monte Carlo simulations

Description

simulate eases the programming task of performing Monte Carlo–type simulations. Typing

```
. simulate exp_list, reps(#): command
```

runs *command* for # replications and collects the results in *exp_list*.

command defines the command that performs one simulation. Most Stata commands and user-written programs can be used with simulate, as long as they follow standard Stata syntax; see **[U] 11 Language syntax**. The by prefix may not be part of *command*.

exp_list specifies the expression to be calculated from the execution of *command*. If no expressions are given, *exp_list* assumes a default, depending upon whether *command* changes results in e() or r(). If *command* changes results in e(), the default is _b. If *command* changes results in r() (but not e()), the default is all the scalars posted to r(). It is an error not to specify an expression in *exp_list* otherwise.

Quick start

Simple program for use with simulate

Define program myreg to generate data and fit a linear regression

```
program myreg, eclass
   drop _all
   set obs 25
   generate x = rnormal()
   generate y = 3*x + 1 + rnormal()
   regress y x
end
```

Perform simulation

Record coefficients and SEs from 1,000 simulated replications of program myreg

```
simulate _b _se, reps(1000): myreg
```

Same as above, and set random-number seed to 5,762 for reproducible results

```
simulate _b _se, reps(1000) seed(5762): myreg
```

Syntax

`simulate` [*exp_list*] , `reps(#)` [*options*] : *command*

options	Description
`nodots`	suppress replication dots
`dots(#)`	display dots every # replications
`noisily`	display any output from *command*
`trace`	trace *command*
`saving(`*filename*`, ...)`	save results to *filename*
`nolegend`	suppress table legend
`verbose`	display the full table legend
`seed(#)`	set random-number seed to #

All weight types supported by *command* are allowed; 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

`reps(#)` is required—it specifies the number of replications to be performed.

`nodots` and `dots(#)` specify whether to display replication dots. By default, one dot character is displayed for each successful replication. 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 displayed using `set dots`; see [R] **set**.

`nodots` suppresses display of the replication dots.

`dots(#)` displays dots every # replications. `dots(0)` is a synonym for `nodots`.

`noisily` requests that any output from *command* be displayed. This option implies the `nodots` option.

`trace` causes a trace of the execution of *command* to be displayed. This option implies the `noisily` option.

`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 only in conjunction with `saving()` when *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**.

`replace` specifies that *filename* be overwritten if it exists.

`nolegend` suppresses display of the table legend. The table legend identifies the rows of the table with the expressions they represent.

`verbose` requests that the full table legend be displayed. By default, coefficients and standard errors are not displayed.

`seed(#)` sets the random-number seed. Specifying this option is equivalent to typing the following command before calling `simulate`:

```
. set seed #
```

Remarks and examples

For an introduction to Monte Carlo methods, see Cameron and Trivedi (2022, chap. 5). White (2010) provides a command for analyzing results of simulation studies.

▷ Example 1: Simulating basic summary statistics

We have a dataset containing means and variances of 100-observation samples from a lognormal distribution (as a first step in evaluating, say, the coverage of a 95%, t-based confidence interval). Then we perform the experiment 1,000 times.

The following command definition will generate 100 independent observations from a lognormal distribution and compute the summary statistics for this sample.

```
program lnsim, rclass
        version 19.5      // (or version 19 if you do not have StataNow)
        drop _all
        set obs 100
        generate z = exp(rnormal())
        summarize z
        return scalar mean = r(mean)
        return scalar Var  = r(Var)
end
```

We can save 1,000 simulated means and variances from `lnsim` by typing

```
. set seed 1234
. simulate mean=r(mean) var=r(Var), reps(1000) nodots: lnsim
      Command: lnsim
         mean: r(mean)
          var: r(Var)
```

```
. describe *

Variable      Storage   Display    Value
    name         type    format    label      Variable label
--------------------------------------------------------------------------------
mean            float   %9.0g                 r(mean)
var             float   %9.0g                 r(Var)

. summarize

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
        mean |      1,000    1.630648    .2173062   1.106372   2.612052
         var |      1,000     4.60798    4.502166    .966087    70.5597
```

Technical note

Before executing our lnsim simulator, we can verify that it works by executing it interactively.

```
. set seed 1234

. lnsim
Number of observations (_N) was 0, now 100.

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
           z |        100    1.534256    1.584568   .0400387   9.818309

. return list

scalars:
             r(Var) =  2.510857086217961
            r(mean) =  1.53425569280982
```

Example 2: Simulating a regression model

Consider a more complicated problem. Let's experiment with fitting $y_j = a + bx_j + u_j$ when the true model has $a = 1$, $b = 2$, $u_j = z_j + cx_j$, and when z_j is $N(0, 1)$. We will save the parameter estimates and standard errors and experiment with varying c. x_j will be fixed across experiments but will originally be generated as $N(0, 1)$. We begin by interactively making the true data:

```
. drop _all

. set obs 100
Number of observations (_N) was 0, now 100.

. set seed 54321

. generate x = rnormal()

. generate true_y = 1+2*x

. save truth
file truth.dta saved
```

Our program is

```
program hetero1
        version 19.5      // (or version 19 if you do not have StataNow)
        args c
        use truth, clear
        generate y = true_y + (rnormal() + `c'*x)
        regress y x
end
```

Note the use of `c' in our statement for generating y. c is a local macro generated from args c and thus refers to the first argument supplied to hetero1. If we want $c = 3$ for our experiment, we type

```
. simulate _b _se, reps(10000): hetero1 3
  (output omitted)
```

Our program hetero1 could, however, be more efficient because it rereads the file truth once every replication. It would be better if we could read the data just once. In fact, if we read in the data right before running simulate, we really should not have to reread for each subsequent replication. A faster version reads

```
program hetero2
        version 19.5      // (or version 19 if you do not have StataNow)
        args c
        capture drop y
        generate y = true_y + (rnormal() + `c'*x)
        regress y x
end
```

Requiring that the current dataset has the variables true_y and x may become inconvenient. Another improvement would be to require that the user supply variable names, such as in

```
program hetero3
        version 19.5      // (or version 19 if you do not have StataNow)
        args truey x c
        capture drop y
        generate y = `truey' + (rnormal() + `c'*`x')
        regress y x
end
```

Thus, we can type

```
. simulate _b _se, reps(10000): hetero3 true_y x 3
  (output omitted)
```

◁

▷ Example 3: Simulating a ratio of statistics

Now, let's consider the problem of simulating the ratio of two medians. Suppose that each sample of size n_i comes from a normal population with a mean μ_i and standard deviation σ_i, where $i = 1, 2$. We write the program below and save it as a text file called myratio.ado (see [U] **17 Ado-files**). Our program is an rclass command that requires six arguments as input, identified by the local macros n1, mu1, sigma1, n2, mu2, and sigma2, which correspond to n_1, μ_1, σ_1, n_2, μ_2, and σ_2, respectively. With these arguments, myratio will generate the data for the two samples, use summarize to compute the two medians and store the ratio of the medians in r(ratio).

```
program myratio, rclass
        version 19.5      // (or version 19 if you do not have StataNow)
        args n1 mu1 sigma1 n2 mu2 sigma2
        // generate the data
        drop _all
        local N = `n1'+`n2'
        set obs `N'
        tempvar y
        generate `y' = rnormal()
        replace `y' = cond(_n<=`n1',`mu1'+`y'*`sigma1',`mu2'+`y'*`sigma2')
        // calculate the medians
        tempname m1
```

```
        summarize `y' if _n<=`n1', detail
        scalar `m1' = r(p50)
        summarize `y' if _n>`n1', detail
        //  store the results
        return scalar ratio = `m1' / r(p50)
end
```

The result of running our simulation is

```
. set seed 19192
. simulate ratio=r(ratio), reps(1000) nodots: myratio 5 3 1 10 3 2
      Command: myratio 5 3 1 10 3 2
        ratio: r(ratio)

. summarize
```

Variable	Obs	Mean	Std. dev.	Min	Max
ratio	1,000	1.10875	.5219166	.3606606	9.857285

◁

❑ Technical note

Stata lets us do simulations of simulations and simulations of bootstraps. Stata's `bootstrap` command (see [R] **bootstrap**) works much like `simulate`, except that it feeds the user-written program a bootstrap sample. Say that we want to evaluate the bootstrap estimator of the standard error of the median when applied to lognormally distributed data. We want to perform a simulation, resulting in a dataset of medians and bootstrap estimated standard errors.

As background, `summarize` (see [R] **summarize**) calculates summary statistics, leaving the mean in `r(mean)` and the standard deviation in `r(sd)`. `summarize` with the `detail` option also calculates summary statistics, but more of them, and leaves the median in `r(p50)`.

Thus, our plan is to perform simulations by randomly drawing a dataset: we calculate the median of our random sample, we use `bootstrap` to obtain a dataset of medians calculated from bootstrap samples of our random sample, the standard deviation of those medians is our estimate of the standard error, and the summary statistics are stored in the results of `summarize`.

Our simulator is

```
program define bsse, rclass
        version 19.5      // (or version 19 if you do not have StataNow)
        drop _all
        set obs 100
        generate x = rnormal()
        tempfile bsfile
        bootstrap midp=r(p50), rep(100) saving(`bsfile'): summarize x, detail
        use `bsfile', clear
        summarize midp
        return scalar mean = r(mean)
        return scalar sd   = r(sd)
end
```

We can obtain final results, running our simulation 1,000 times, by typing

```
. set seed 48901
. simulate med=r(mean) bs_se=r(sd), reps(1000): bsse
      Command: bsse
          med: r(mean)
        bs_se: r(sd)
Simulations (1,000): .........10.........20.........30.........40.........50....
> .....60.........70.........80.........90.........100.........110.........120..
> .......130.........140.........150.........160.........170.........180.......
> .190.........200.........210.........220.........230.........240.........250..
> .......260.........270.........280.........290.........300.........310.......
> .320.........330.........340.........350.........360.........370.........380..
> .......390.........400.........410.........420.........430.........440.......
> .450.........460.........470.........480.........490.........500.........510..
> .......520.........530.........540.........550.........560.........570.......
> .580.........590.........600.........610.........620.........630.........640..
> .......650.........660.........670.........680.........690.........700.......
> .710.........720.........730.........740.........750.........760.........770..
> .......780.........790.........800.........810.........820.........830.......
> .840.........850.........860.........870.........880.........890.........900..
> .......910.........920.........930.........940.........950.........960.......
> .970.........980.........990.........1,000 done
. summarize
```

Variable	Obs	Mean	Std. dev.	Min	Max
med	1,000	-.0013359	.1221602	-.3795549	.3656219
bs_se	1,000	.1278773	.0303109	.0614031	.2484805

This is a case where the simulation dots (drawn by default, unless the `nodots` option is specified) will give us an idea of how long this simulation will take to finish as it runs.

❏

References

Cameron, A. C., and P. K. Trivedi. 2022. *Microeconometrics Using Stata*. 2nd ed. College Station, TX: Stata Press.

Hilbe, J. M. 2010. Creating synthetic discrete-response regression models. *Stata Journal* 10: 104–124.

Taylor, M. A. 2018. Simulating the central limit theorem. *Stata Journal* 18: 345–356.

White, I. R. 2010. simsum: Analyses of simulation studies including Monte Carlo error. *Stata Journal* 10: 369–385.

Also see

[R] **bootstrap** — Bootstrap sampling and estimation

[R] **jackknife** — Jackknife estimation

[R] **permute** — Permutation tests

[R] **set rngstream** — Specify the stream for the stream random-number generator

Stata, Stata Press, Mata, NetCourse, and NetCourseNow are registered trademarks of StataCorp LLC. Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations. StataNow is a trademark of StataCorp LLC. Other brand and product names are registered trademarks or trademarks of their respective companies. Copyright © 1985–2025 StataCorp LLC, College Station, TX, USA. All rights reserved.

For suggested citations, see the FAQ on citing Stata documentation.