Title

stata.com

simulate — Monte Carlo simulations

Syntax

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

options	Description
`nodots`	suppress replication dots
`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.

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.

Options

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

`nodots` suppresses display of the replication dots. By default, one dot character is displayed for each successful replication. A red 'x' is displayed if *command* returns an error or if one of the values in *exp_list* is missing.

`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 simulated values.

`double` specifies that the results for each replication be saved as `double`s, meaning 8-byte reals. By default, they are saved as `float`s, 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

stata.com

For an introduction to Monte Carlo methods, see Cameron and Trivedi (2010, chap. 4). 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 13
        drop _all
        set obs 100
        gen 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 *

              storage   display    value
variable 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 |      1000    1.638466     .214371   1.095099   2.887392
         var |      1000     4.63856    6.428406      .8626   175.3746
```

◁

❑ Technical note

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

```
. set seed 1234

. lnsim
obs was 0, now 100

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
           z |       100    1.597757    1.734328   .0625807   12.71548

. return list

scalars:
                 r(Var) =  3.007893773683719
                r(mean) =  1.59775722913444
```

❑

▷ 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
obs was 0, now 100
. set seed 54321
. gen x = rnormal()
. gen true_y = 1+2*x
. save truth
file truth.dta saved
```

Our program is

```
program hetero1
        version 13
        args c
        use truth, clear
        gen 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 13
        args c
        capture drop y
        gen 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 13
        args truey x c
        capture drop y
        gen 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 13
        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	1000	1.08571	.4427828	.3834799	6.742217

◁

❑ 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 13
        drop _all
        set obs 100
        gen 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 (1000)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
..................................................   100
..................................................   150
..................................................   200
..................................................   250
..................................................   300
..................................................   350
..................................................   400
..................................................   450
..................................................   500
..................................................   550
..................................................   600
..................................................   650
..................................................   700
..................................................   750
..................................................   800
..................................................   850
..................................................   900
..................................................   950
..................................................  1000

. summarize

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         med |       1000   -.0008696    .1210451  -.3132536   .4058724
       bs_se |       1000     .126236     .029646   .0326791   .2596813
```

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. 2010. *Microeconometrics Using Stata*. Rev. ed. College Station, TX: Stata Press.

Gould, W. W. 1994. ssi6.1: Simplified Monte Carlo simulations. *Stata Technical Bulletin* 20: 22–24. Reprinted in *Stata Technical Bulletin Reprints*, vol. 4, pp. 207–210. College Station, TX: Stata Press.

Hamilton, L. C. 2013. *Statistics with Stata: Updated for Version 12*. 8th ed. Boston: Brooks/Cole.

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

Weesie, J. 1998. ip25: Parameterized Monte Carlo simulations: Enhancement to the simulation command. *Stata Technical Bulletin* 43: 13–15. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 75–77. College Station, TX: Stata Press.

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** — Monte Carlo permutation tests