Monte Carlo Simulation in Stata

Christopher F Baum

Faculty Micro Resource Center
Boston College

July 2007

We often want to evaluate the properties of estimators, or compare a proposed estimator to another, in a context where analytical derivation of those properties is not feasible. In that case, econometricians resort to **Monte Carlo studies**: simulation methods making use of (pseudo-)random draws from an error distribution and multiple replications over a set of known parameters. This methodology is particularly relevant in situations where the only analytical findings involve asymptotic, large-sample results.

Monte Carlo studies, although they do not generalize to cases beyond those performed in the experiment, also are useful in modelling quantities for which no analytical results have yet been derived: for instance, the critical values for many unit-root test statistics have been derived by simulation experiments, in the absence of closed-form expressions for the sampling distributions of the statistics.

We often want to evaluate the properties of estimators, or compare a proposed estimator to another, in a context where analytical derivation of those properties is not feasible. In that case, econometricians resort to **Monte Carlo studies**: simulation methods making use of (pseudo-)random draws from an error distribution and multiple replications over a set of known parameters. This methodology is particularly relevant in situations where the only analytical findings involve asymptotic, large-sample results.

Monte Carlo studies, although they do not generalize to cases beyond those performed in the experiment, also are useful in modelling quantities for which no analytical results have yet been derived: for instance, the critical values for many unit-root test statistics have been derived by simulation experiments, in the absence of closed-form expressions for the sampling distributions of the statistics.

Most econometric software provide some facilities for Monte Carlo experiments. Although one can write the code to generate an experiment in any programming language, it is most useful to do so in a context where one may readily save the results of each replication for further analysis.

The quality of the pseudo-random number generators available is also an important concern. State-of-the-art pseudo-random number generators do exist, and you should use a package that implements them: not all do.

You will also want a package with a full set of statistical functions, permitting random draws to be readily made from a specified distribution: not merely normal or t, but from a number of additional distributions, depending upon the experiment.

Stata version 10 provides a useful environment for Monte Carlo simulations. Setting up a simulation requires that you write a Stata program: not merely a "do-file" containing a set of Stata commands, but a sequence of commands beginning with the `program define` statement.

This program sets up the simulation experiment and specifies what is to be done in one replication; you then invoke it with the `simulate` command to execute a specified number of replications.

For instance, let us consider simulating the performance of the estimator of sample mean, $\bar{x}$, in a context of heteroskedasticity. As the sample mean is a least squares estimator, we know that its point estimate will remain unbiased, but interval estimates will be biased. We could derive the analytical results for this simple model, but in this case let us compute the degree of bias of the interval estimates by simulation.

Take the model to be $y_i = \mu + \epsilon_i$, with $\epsilon_i \sim N(0, \sigma^2)$. Let ϵ be a N(0,1) variable multiplied by a factor cz_i, where z_i varies over i.

We will vary parameter c between 0.1 and 1.0 and determine its effect on the point and interval estimates of μ; as a comparison, we will compute a second random variable which is homoskedastic, with the scale factor equalling $c\bar{z}$.

For instance, let us consider simulating the performance of the estimator of sample mean, $\bar{x}$, in a context of heteroskedasticity. As the sample mean is a least squares estimator, we know that its point estimate will remain unbiased, but interval estimates will be biased. We could derive the analytical results for this simple model, but in this case let us compute the degree of bias of the interval estimates by simulation.

Take the model to be $y_i = \mu + \epsilon_i$, with $\epsilon_i \sim N(0, \sigma^2)$. Let ϵ be a N(0,1) variable multiplied by a factor cz_i, where z_i varies over i.

We will vary parameter c between 0.1 and 1.0 and determine its effect on the point and interval estimates of μ; as a comparison, we will compute a second random variable which is homoskedastic, with the scale factor equalling $c\bar{z}$.

For instance, let us consider simulating the performance of the estimator of sample mean, $\bar{x}$, in a context of heteroskedasticity. As the sample mean is a least squares estimator, we know that its point estimate will remain unbiased, but interval estimates will be biased. We could derive the analytical results for this simple model, but in this case let us compute the degree of bias of the interval estimates by simulation.

Take the model to be $y_i = \mu + \epsilon_i$, with $\epsilon_i \sim N(0, \sigma^2)$. Let ϵ be a N(0,1) variable multiplied by a factor cz_i, where z_i varies over i.

We will vary parameter c between 0.1 and 1.0 and determine its effect on the point and interval estimates of μ; as a comparison, we will compute a second random variable which is homoskedastic, with the scale factor equalling $c\bar{z}$.

We first define the simulation program:

```
program define mcsimul1, rclass
      version 10.0
      syntax [, c(real 1)]
      tempvar e1 e2
      gen double `e1'=invnorm(uniform())*`c'*zmu
      gen double `e2'=invnorm(uniform())*`c'*z_factor
      replace y1 = true_y + `e1'
      replace y2 = true_y + `e2'
      summ y1
      return scalar mu1 = r(mean)
      return scalar se_mu1 = r(sd)/sqrt(r(N))
      summ y2
      return scalar mu2 = r(mean)
      return scalar se_mu2 = r(sd)/sqrt(r(N))
      return scalar c = `c'
end
```

In this program, we define two random variables: `y1`, which contains a homoskedastic error `e1`, and `y2`, which contains a heteroskedastic error `e2`. Those errors are generated as temporary variables in the program and added to the common variable `true_y`. In the example below, that variable is actual data.

We calculate the sample mean and its standard error for variables `y1` and `y2`, and return those four quantities as scalars as well as `c`, the degree of heteroskedasticity.

In this program, we define two random variables: `y1`, which contains a homoskedastic error `e1`, and `y2`, which contains a heteroskedastic error `e2`. Those errors are generated as temporary variables in the program and added to the common variable `true_y`. In the example below, that variable is actual data.

We calculate the sample mean and its standard error for variables `y1` and `y2`, and return those four quantities as scalars as well as `c`, the degree of heteroskedasticity.

We now set up the simulation environment:

```
local reps 1000
```

We will perform 1000 Monte Carlo replications for each level of `c`, which will be varied as 10, 20, ... 100.

We use the `census2` dataset to define 50 observations and their `region` variable, which identifies each state. The arbitrary coding of the `region` variable (as 1,2,3,4) is used as the `z_factor` in the simulation to create heteroskedasticity in the errors across regions. The mean of the `y1` and `y2` variables will be set to the actual value of `age` in each state.

The do-file for our simulation continues as:

```
forv i=1/10 {
qui webuse census2, clear
gen true_y = age
gen z_factor = region
sum z_factor, meanonly
scalar zmu = r(mean)
qui {
gen y1 = .
gen y2 = .
local c = `i'*10
simulate c=r(c)  mu1=r(mu1) se_mu1=r(se_mu1) ///
                 mu2=r(mu2) se_mu2=r(se_mu2), ///
saving(cc`i',replace) nodots reps(`reps'): ///
                 mcsimul1, c(`c')
}
}
```

This do-file first contains a loop over values 1..10. For each value of `i`, we reload the `census2` dataset and calculate the variable `z_factor` and the scalar `zmu`. We initialize the values of `y1` and `y2` to missing, define the local `c` for this level of heteroskedasticity, and invoke the `simulate` command.

The `simulate` command contains a list of objects to be created, followed by options, followed by a colon and the name of the program to be simulated: in our case `mcsimul1`. The program name is followed, optionally, by any arguments to be passed to our program. In our case we only pass the option `c` with the value of the local macro `c`.

The options to `simulate` define new variables created by the simulation as `c`, `mu1, se_mu1, mu2,se_mu2`, specify that `reps` repetitions are to be performed, and that the results of the simulation should be saved as a Stata data file named `cc`i'.dta`.

This do-file first contains a loop over values 1..10. For each value of `i`, we reload the `census2` dataset and calculate the variable `z_factor` and the scalar `zmu`. We initialize the values of `y1` and `y2` to missing, define the local `c` for this level of heteroskedasticity, and invoke the `simulate` command.

The `simulate` command contains a list of objects to be created, followed by options, followed by a colon and the name of the program to be simulated: in our case `mcsimul1`. The program name is followed, optionally, by any arguments to be passed to our program. In our case we only pass the option `c` with the value of the local macro `c`.

The options to `simulate` define new variables created by the simulation as `c`, `mu1, se_mu1, mu2,se_mu2`, specify that `reps` repetitions are to be performed, and that the results of the simulation should be saved as a Stata data file named `cc`i'.dta`.

This do-file first contains a loop over values 1..10. For each value of `i`, we reload the `census2` dataset and calculate the variable `z_factor` and the scalar `zmu`. We initialize the values of `y1` and `y2` to missing, define the local `c` for this level of heteroskedasticity, and invoke the `simulate` command.

The `simulate` command contains a list of objects to be created, followed by options, followed by a colon and the name of the program to be simulated: in our case `mcsimul1`. The program name is followed, optionally, by any arguments to be passed to our program. In our case we only pass the option `c` with the value of the local macro `c`.

The options to `simulate` define new variables created by the simulation as `c`, `mu1, se_mu1, mu2,se_mu2`, specify that `reps` repetitions are to be performed, and that the results of the simulation should be saved as a Stata data file named `cc`i'.dta`.

The first time through the `forvalues` loop, the program will create datafile `cc1.dta` with 1000 observations on `c, mu1, se_mu1, mu2, se_mu2`. We include `c` because we will want to combine these datafiles into one, and must identify those observations that were generated by a particular value of `c` (the degree of heteroskedasticity) in that combined file.

We now combine those datafiles into a single file:

```
use cc1
forv i=2/10 {
append using cc`i'
}
gen het_infl = se_mu2 / se_mu1
save cc_1_10,replace
```

The first time through the `forvalues` loop, the program will create datafile `cc1.dta` with 1000 observations on `c, mu1, se_mu1, mu2, se_mu2`. We include `c` because we will want to combine these datafiles into one, and must identify those observations that were generated by a particular value of `c` (the degree of heteroskedasticity) in that combined file.

We now combine those datafiles into a single file:

```
use cc1
forv i=2/10 {
append using cc`i'
}
gen het_infl = se_mu2 / se_mu1
save cc_1_10,replace
```

File `cc_1_10.dta` now contains 10,000 observations on `c, mu1, se_mu1, mu2, se_mu2`, as well as a new variable, `het_infl` which contains the ratio of the standard error of the heteroskedastic variable to that of the homoskedastic variable.

To evaluate the results of the simulation, we calculate descriptive statistics for the results file by values of `c`:

```
tabstat mu1 se_mu1 mu2 se_mu2 het_infl, ///
  stat(mean) by(c)
tabstat het_infl, stat(mean q iqr) by(c)
```

The first tabulation provides the average values of the variables stored for each value of `c`. The second tabulation focuses on the ratio `het_infl`, computing its mean and quartiles.

File `cc_1_10.dta` now contains 10,000 observations on `c, mu1, se_mu1, mu2, se_mu2`, as well as a new variable, `het_infl` which contains the ratio of the standard error of the heteroskedastic variable to that of the homoskedastic variable.

To evaluate the results of the simulation, we calculate descriptive statistics for the results file by values of `c`:

```
tabstat mu1 se_mu1 mu2 se_mu2 het_infl, ///
  stat(mean) by(c)
tabstat het_infl, stat(mean q iqr) by(c)
```

The first tabulation provides the average values of the variables stored for each value of `c`. The second tabulation focuses on the ratio `het_infl`, computing its mean and quartiles.

c	mean	p25	p50	p75
10	1.002349	.9801603	1.001544	1.023048
20	1.007029	.9631492	1.006886	1.047167
30	1.018764	.9556518	1.015823	1.070618
40	1.021542	.9427943	1.015947	1.092599
50	1.039481	.9546044	1.03039	1.114773
60	1.043277	.944645	1.03826	1.130782
70	1.04044	.9386177	1.035751	1.126928
80	1.057522	.9555817	1.050923	1.156035
90	1.047648	.9436705	1.038448	1.140402
100	1.063031	.9514042	1.048108	1.159396
Total	1.034108	.9575833	1.020554	1.098434

These results clearly indicate that as the degree of heteroskedasticity increases, the standard error of mean is biased upward by more than 6 per cent on average (almost 5 per cent in terms of the median, or `p50`) for the most serious case considered.

We consider now how a small variation on this program and do-file can be used to evaluate the power of a test, using the same underlying data generating process to compare two series that contain homoskedastic and heteroskedastic errors. In this case, we will not use actual data for these series, but treat them as being random variations around a constant value.

These results clearly indicate that as the degree of heteroskedasticity increases, the standard error of mean is biased upward by more than 6 per cent on average (almost 5 per cent in terms of the median, or `p50`) for the most serious case considered.

We consider now how a small variation on this program and do-file can be used to evaluate the power of a test, using the same underlying data generating process to compare two series that contain homoskedastic and heteroskedastic errors. In this case, we will not use actual data for these series, but treat them as being random variations around a constant value.

We first define the new simulation program:

```
program define mcsimul2, rclass
      version 10.0
      syntax [, c(real 1)]
      tempvar e1 e2
      gen `e1' = invnorm(uniform())*`c'*zmu
      gen `e2' = invnorm(uniform())*`c'*z_factor
      replace y1 = true_y + `e1'
      replace y2 = true_y + `e2'
      ttest y1 = 0
      return scalar p1 = r(p)
      ttest y2 = 0
      return scalar p2 = r(p)
      return scalar c = `c'
end
```

This program differs from `mcsimul1` in that it will calculate two hypothesis tests, with null hypotheses that the means of `y1` and `y2` are zero. The *p*-values for those tests are returned to the calling program, along with the value of `c`, denoting the degree of heteroskedasticity.

We set the `true_y` variable to a constant using a global macro:

```
global true_mu 50
```

The do-file for our simulation continues as:

```
forv i=1/10 {
qui webuse census2, clear
gen true_y = $true_mu
gen z_factor = region
sum z_factor, meanonly
scalar zmu = r(mean)
qui {
gen y1 = .
gen y2 = .
local c = `i'*10
simulate c=r(c)  p1=r(p1) p2=r(p2), ///
saving(ccc`i',replace) nodots reps(`reps'): ///
                  mcsimul2, c(`c')
}
}
```

In this program, the options to `simulate` define new variables created by the simulation as `c`, `p1`, `p2`, specify that `reps` repetitions are to be performed, and that the results of the simulation should be saved as a Stata data file named `ccc`i'.dta`.

After executing this do-file, we again combine the separate datafiles created in the loop into a single datafile, and generate several new variables to evaluate the power of the *t*-tests:

```
gen RfNull_1 = (1-p1)*100
gen RfNull_2 = (1-p2)*100
gen R5pc_1 = (p1<0.05)/10
gen R5pc_2 = (p2<0.05)/10
```

In this program, the options to `simulate` define new variables created by the simulation as `c`, `p1`, `p2`, specify that `reps` repetitions are to be performed, and that the results of the simulation should be saved as a Stata data file named `ccc`i'.dta`.

After executing this do-file, we again combine the separate datafiles created in the loop into a single datafile, and generate several new variables to evaluate the power of the *t*-tests:

```
gen RfNull_1 = (1-p1)*100
gen RfNull_2 = (1-p2)*100
gen R5pc_1 = (p1<0.05)/10
gen R5pc_2 = (p2<0.05)/10
```

The `RFNull` variables compute the coverage of the test statistic in percent. If on average `p1` is 0.05, the test is rejecting 95 per cent of the false null hypotheses that the mean of `y1` or `y2` is zero when it is actually 50.

The `R5pc` variables evaluate the logical condition that `p1` (`p2`), the *p*-value of the *t*-test, is smaller that 0.05. They are divided by 10 because we would like to express these measures of power in percentage terms, which means multiplying by 100 but dividing by the 1000 replications carried out, and taking the sum of these values.

We then tabulate the variables `RFNull` and `R5pc` to evaluate how the power of the *t*-test varies over the degree of heteroskedasticity, `c`:

```
tabstat p1 p2 RfNull_1 RfNull_2,stat(mean) by(c)
tabstat R5pc_1 R5pc_2,stat(sum) by(c) nototal
```

The `RFNull` variables compute the coverage of the test statistic in percent. If on average `p1` is 0.05, the test is rejecting 95 per cent of the false null hypotheses that the mean of `y1` or `y2` is zero when it is actually 50.

The `R5pc` variables evaluate the logical condition that `p1` (`p2`), the *p*-value of the *t*-test, is smaller that 0.05. They are divided by 10 because we would like to express these measures of power in percentage terms, which means multiplying by 100 but dividing by the 1000 replications carried out, and taking the sum of these values.

We then tabulate the variables `RFNull` and `R5pc` to evaluate how the power of the *t*-test varies over the degree of heteroskedasticity, `c`:

```
tabstat p1 p2 RfNull_1 RfNull_2,stat(mean) by(c)
tabstat R5pc_1 R5pc_2,stat(sum) by(c) nototal
```

The `RFNull` variables compute the coverage of the test statistic in percent. If on average `p1` is 0.05, the test is rejecting 95 per cent of the false null hypotheses that the mean of `y1` or `y2` is zero when it is actually 50.

The `R5pc` variables evaluate the logical condition that `p1` (`p2`), the *p*-value of the *t*-test, is smaller that 0.05. They are divided by 10 because we would like to express these measures of power in percentage terms, which means multiplying by 100 but dividing by the 1000 replications carried out, and taking the sum of these values.

We then tabulate the variables `RFNull` and `R5pc` to evaluate how the power of the *t*-test varies over the degree of heteroskedasticity, `c`:

```
tabstat p1 p2 RfNull_1 RfNull_2,stat(mean) by(c)
tabstat R5pc_1 R5pc_2,stat(sum) by(c) nototal
```

c	p1	p2	RfNull_1	RfNull_2
10	8.25e-15	6.25e-12	100	100
20	5.29e-06	.0000439	99.99947	99.99561
30	.0025351	.0065619	99.74649	99.34381
40	.0222498	.0275325	97.77502	97.24675
50	.0598702	.069537	94.01298	93.0463
60	.1026593	.1214913	89.73407	87.85087
70	.1594983	.1962834	84.05017	80.37166
80	.2173143	.2267421	78.26857	77.32579
90	.2594667	.274362	74.05333	72.5638
100	.2898857	.3163445	71.01143	68.36555
Total	.1113485	.1238899	88.86515	87.61101

This table shows that the mean *p*-value in the homoskedastic case (`p1`) is smaller than 0.05 until `c` exceeds 40. Even in the homoskedastic case, an increase in the error variance makes it difficult to distinguish the sample mean from zero, and with `c`=100, almost 30 per cent of estimates fail to reject the null. The rejection frequencies are given by the `RFNull` columns.

For the heteroskedastic case, the *p*-values are systematically larger and the rejection frequencies correspondingly smaller, with only 68 per cent of estimated sample means able to reject the null with `c`=100. We may also evaluate the measures of power, calculated as the `R5pc` variables:

This table shows that the mean *p*-value in the homoskedastic case (`p1`) is smaller than 0.05 until `c` exceeds 40. Even in the homoskedastic case, an increase in the error variance makes it difficult to distinguish the sample mean from zero, and with `c`=100, almost 30 per cent of estimates fail to reject the null. The rejection frequencies are given by the `RFNull` columns.

For the heteroskedastic case, the *p*-values are systematically larger and the rejection frequencies correspondingly smaller, with only 68 per cent of estimated sample means able to reject the null with `c`=100. We may also evaluate the measures of power, calculated as the `R5pc` variables:

c	R5pc_1	R5pc_2
10	100	100
20	100	100
30	99.1	97.4
40	90.6	85.3
50	74.8	72.1
60	59.5	55.1
70	45.4	42.4
80	39.8	34.9
90	29.5	28.4
100	27.7	22.8

For c=10, we see that the test properly rejects the false null 100 per cent of the time in both the homoskedastic and heteroskedastic cases. Power falls off for both cases with increasing values of c, but declines more quickly for the heteroskedastic case.

The results we have shown here will differ each time the do-file is executed, as a different sequence of pseudo-random numbers will be computed. To generate replicable Monte Carlo results, use Stata's `set seed` command to initialize the random number generator at the top of the do-file (not inside the program!) This will cause the same sequence of PRNs to be produced every time the do-file is run.

For `c`=10, we see that the test properly rejects the false null 100 per cent of the time in both the homoskedastic and heteroskedastic cases. Power falls off for both cases with increasing values of `c`, but declines more quickly for the heteroskedastic case.

The results we have shown here will differ each time the do-file is executed, as a different sequence of pseudo-random numbers will be computed. To generate replicable Monte Carlo results, use Stata's `set seed` command to initialize the random number generator at the top of the do-file (not inside the program!) This will cause the same sequence of PRNs to be produced every time the do-file is run.