Gauss for Econometrics: Simulation

R.G. Pierse

1. Introduction

Simulation is a very useful tool in econometric modelling. It allows the economist to examine the properties of models and estimators when the true process generating the model, the *Data Generation Process*, is known by construction. Often the properties of a model cannot be derived analytically, so that numerical simulation may be the only way to discover them. In econometrics, Monte Carlo analysis allows the finite sample bias of estimators, and the power of test statistics to be computed to any degree of precision.

Simulation has always been regarded as a prohibitively expensive computer exercise and this has limited the use of simulation techniques. While it is true that generating a large number of pseudo-random numbers on a computer is still relatively time-consuming, the random number generator used by *GAUSS* is very fast, and small simulations are quite feasible even on slow computers. Efficient coding and the use of parallelisation can help reduce the cost.

2. Random Numbers

Random numbers are generated on a computer using pseudo-random number generators. These start from an integer value (the *seed*) and generate a sequence of real numbers that have the properties of uniformly distributed random numbers although in fact they are not randomly generated. Random numbers with distributions other than the uniform, are generated by transforming the uniform pseudo-random numbers. There is a large literature on the properties of pseudo-random number generators. Important issues are the autocorrelation properties of the numbers (ideally the random numbers should be independent but in practice some autocorrelation is inevitable) and the recyling period (how long before the sequence of numbers repeats itself).

GAUSS has two procedures to generate a matrix of pseudo-random numbers: **rndu**(p,q) which creates a $p \times q$ matrix of numbers uniformly distributed in the interval $[0,1]$, and **rndn**(p,q) which creates a $p \times q$ matrix of numbers independently normally distributed with mean 0 and standard deviation 1. To generate uniform random numbers with a different range or normal random numbers with a different mean or standard deviation, simply transform the numbers. For example

$$20 * \mathbf{rndu}(p,q) - 10$$

defines a matrix of uniform random numbers in the interval $[-10,10]$, while

$$4 + 5 * \mathbf{rndn}(p,q)$$

defines a matrix of variables independently distributed as $N(4,25)$.

2.1. Correlated variables

Correlated random variables can be generated from the standard functions by transformation. For example, suppose that we want to generate a $T \times n$ matrix of normal variables $\mathbf{U}$ with covariance matrix given by

$$Var(\mathrm{vec}(\mathbf{U})) = \mathbf{S}_n \otimes \mathbf{I}_T \,.$$

Let $\mathbf{V}$ be a $T \times n$ matrix of independent normal variables with unit variance

$$Var(\mathrm{vec}(\mathbf{V})) = \mathbf{I}_n \otimes \mathbf{I}_T$$

such as will be generated by the *GAUSS* **rndn**(T,n) command. Define $\mathbf{H}$ as the upper triangular matrix of the Cholesky decomposition of $\mathbf{S}_n$ such that $\mathbf{S}_n = \mathbf{H}'\mathbf{H}$. Then it follows that

$$(\mathbf{H}' \otimes \mathbf{I}_T)\,\mathrm{vec}(\mathbf{V})$$

is normally distributed with variance matrix

$$\mathbf{H}'\mathbf{H} \otimes \mathbf{I}_T = \mathbf{S}_n \otimes \mathbf{I}_T \,.$$

Thus the transformation

$$\mathrm{vec}(\mathbf{U}) = (\mathbf{H}' \otimes \mathbf{I}_T)\,\mathrm{vec}(\mathbf{V})$$

or, unvectorising,

$$\mathbf{U} = \mathbf{V}\mathbf{H}$$

has the appropriate distribution. This can be accomplished with the GAUSS command

$$U = \mathbf{rndn}(T,n)\text{*}\mathbf{chol}(S)\ ;$$

2.2. Other distributions

Random numbers with distributions other than the normal can be generated from uniform random numbers by transformation using the inverse of the cumulated density function for the distribution in question. Suppose F is a continuous cumulative density function

$$F : R \to [0, 1]$$

with inverse F^{-1} and that x is a random variable with uniform distribution

$$x \sim U(0, 1) .$$

Then $F^{-1}(x)$ is a random variable following the distribution function F. For example, the *GAUSS* commands

$$x = \mathbf{rndu}(k,1) \; ; \; z = \mathbf{ln}(x \cdot /(1 \text{ - } x)) \; ;$$

generate a $k \times 1$ vector of random variables z following the logistic distribution

$$F(z) = \frac{1}{1 + e^{-z}} .$$

2.3. Replicating random number sequences

By default, the random number generator uses a seed based on the computer clock. This seed is automatically updated as each random number is generated. Thus, every time a *GAUSS* job is run, a new sequence of random numbers will be generated. Sometimes it is useful to be able to generate a sequence of random numbers that can be replicated at a later date. To do this, the generator seed should be set explicitly to some integer value using the **rndseed** command, as in

rndseed 560025;

at the start of the *GAUSS* job. Each choice of integer value for the seed generates a different sequence of random numbers, but re-using the same integer value will replicate the same sequence of numbers. If the chosen seed is printed out along with the simulation output, then if necessary the simulation can always be re-run to exactly replicate the results.

3. Monte Carlo Simulation

Monte Carlo simulation is a technique for investigating the statistical properties of estimators or hypothesis tests through numerical simulation. The investigator generates a set of data $\mathbf{x}_j$ from a set of pseudo-random variables, representing a drawing j from the population distribution of the data. This is the *data generating process* (*DGP*) which will be a function of a set of parameters $\boldsymbol{\lambda}$ and is assumed to be known. The estimator or test statistic can be computed as a function of this data set

$$\theta_j(\mathbf{x}_j)\,.$$

This is then repeated with a new drawing of random variables, from the same *DGP*. If enough replications are performed, then it is appropriate to make use of the central limit theorem and assert that

$$\frac{1}{m}\sum_{j=1}^{m}\theta_j \sim N(E(\theta(\mathbf{x})), \frac{1}{m}Var(\theta(\mathbf{x})))$$

where m is the number of replications. The larger the number of replications, the smaller the variance of the distribution and so the more precise is the estimate of the mean of the distribution of the estimator $\theta(\mathbf{x})$. Typically, Monte Carlo simulations reported in econometrics journals use between 1000 and 10000 replications.

The distribution of $E(\theta(\mathbf{x}))$ will depend on the parameters of the *DGP* and usually also on the number of observations in the data set. Thus simulations will need to be performed using different values for these population parameters and different sample sizes to investigate the sensitivity of the results to these factors. It is then possible to formally estimate a *response surface* which is a function relating the distribution of $E(\theta(\mathbf{x}))$ to $\boldsymbol{\lambda}$ and to the sample size n.

If there are k parameters in the *DGP* and p different values for each parameter, then the number of simulations to be performed is p^k. This quickly becomes unmanageable if k is at all large. It is thus important to keep the model as simple as possible.

3.1. Time and space considerations

In general, efficient *GAUSS* programs avoid the use of loops. In the area of simulation however, loops are almost unavoidable. This is because matrices of order $n \times m$ will generally be too large to be stored. For example, for $n = 100$ and

$m = 10000$, a single matrix consumes 8Mb of RAM. Thus a GAUSS simulation program will generally need to have a loop over the replications. In this case, it is not necessary for the program to keep all the results from every replication. Space can be saved by storing only the cumulated mean (or sum) of the replications so far. This has an additional advantage in that the program can be interrupted before the replication loop has finished without losing all the results. This is illustrated in the example below.

Nevertheless, it is sometimes possible to improve the execution speed by running more than one simulation in parallel using the same drawing of random numbers.

4. Example

The following example illustrates some of the principles discussed above. It examines the performance of the OLS estimator $\widehat{p}$ in the dynamic model

$$y_t = py_{t-1} + u_t$$

when the error process u_t is generated by the first order autoregression

$$u_t = ru_{t-1} + e_t$$

and $e_t \sim iid\, N(0, \sigma^2)$. The parameters of the *DGP* are { p, r, σ }. The number of observations is n and the number of replications is *nrep*. All these parameters are set in the first three lines of code. The fourth line initialises the seed of the random number generator. The output is written to a file with name given by the parameter *rn*, appended by the suffix ".out". Only the first four lines of the code should need to be changed.

The main replication loop of the program contains a line that tests for a key having been pressed. If so, then the program breaks out of the loop and reports the results on the basis of the number of replications that have been executed. This means that the replication loop can be terminated by the user at any point without losing all the results.

/* Monte Carlo simulation */
rn = "sim1" ; *nrep* = 500; n = 100;
sigma = 1; r = -0.5; p = 0.7;

```
kk = 0; b = 0;
rndseed 8504966;
/* Replication loop */
repl = 1; do while repl <= nrep;
   e = sigma*rndn(n,1) ; /* {e} is white noise */
   u = zeros(n,1) ;
   j = 2; do while j <= n ;
      u[j] = r*u[j-1] + e[j] ; /* {u} is AR(1) */
      j = j + 1;
   endo;
   /* {y} is AR(1) with AR errors */
   y=zeros(n,1);
   j = 2; do while j <= n;
      y[j] = p*y[j-1] + u[j];
      j = j + 1;
   endo;
   x = y[1:n-1] ; y = y[2:n];
   locate 1,1; format /rd 5,0;
   print "rep=" repl " n=" n ;
   phat = invpd(x'x)*x'y ; b = b + phat ;
   repl = repl+1;
   /* break out if key has been pressed */
   kk = key; if kk ne 0; break ; endif ;
endo; /* end of replication loop */
nrep = repl-1;
b = b / nrep;
fname = rn $+ ".out"; output file=^fname reset;
format /rd 5,0;
```

print “Simulation with ” *nrep* “ replications”;
format /rd 4,2;
print “r = ” *r* “ ; p = ” *p*;
print “phat = ” *b*;
output off;
end;