Paper 236-25

Pseudo-Random Numbers: Out of Uniform

Robert E. Johnson, Virginia Commonwealth University, Richmond, Virginia
Hui Liu, Westat Inc., Rockville, Maryland

ABSTRACT

Pseudo-random numbers – both uniform and non-uniform – are used in random sampling, simulation, and Monte-Carlo estimation. Base SAS software contains numerous random number, quantile, and probability functions allowing the user to generate a selection of nominal, discrete, and continuous random variates. Other variates may be generated with a data step, often making use of SAS functions. We will present an overview of pseudo-random numbers and the functions of Base SAS. Applications will include simple random sampling from a data set and estimation of probabilities. The audience should have intermediate knowledge of data step coding, SAS functions, and the fundamentals of probability and statistics.

INTRODUCTION

Pseudo-random numbers appear random, but are generated using a deterministic algorithm. Good generators have properties that allow the user to treat its stream of numbers as if it were a random stream drawn from the distribution associated with the generator. This facilitates probability sampling, allows simulation of a process, and provides means for estimating probabilities associated with complex functions of random variables or processes. In this paper we will review the concepts of uniform random number generators, in particular, the function RANUNI found in SAS® software. Other random number generators will be presented and discussed. These include the RAN*xxx*, quantile (inverse probability), and probability (cumulative distribution) functions.

Selected applications out of uniform random numbers are presented. These include the generation of non-uniform random variates, random sampling, and simulation.

UNIFORM GENERATORS

Several forms of random number generators have been proposed in the literature. These include linear, power $\{x_{n+1}=(x_n)^d \pmod N\}$, and discrete exponential $\{x_{n+1}=g\text{^}(x_n) \pmod N\}$ to name a few. All are deterministic functions that produce streams of nonnegative integers. Some generators are rather difficult to study. Lagarias (1993) notes this in stating the *Unpredictability Paradox*: "*If a deterministic function is unpredictable, then it is difficult to prove anything about it; in particular, it is difficult to prove that it is unpredictable.*" Major focus has been placed on linear congruential generators, mainly because much is known about their properties and they are relatively easy to implement on *p*-bit machines.

A linear congruential generator is a deterministic function that produces a stream of nonnegative integers:

$$\mathbf{Z} = \{Z_0,\ Z_i = AZ_{i-1} + C \pmod M;\ i = 1, 2, \ldots\},$$

given a seed Z_0, positive integers A and M, and nonnegative integer C. This stream of integers may then be scaled to the interval (0,1):

$$\mathbf{U} = \{U_i = Z_i/(M-1);\ i = 1, 2, \ldots\}$$

to produce a source of pseudo-random values that – it is hoped – mimics a standard uniform distribution.

Lehmer (1951) proposed setting C to zero thus forming a class of multiplicative generators. These are the most commonly used generators in modern software. It is desirable for such generators to run through the entire stream {1, 2,…, M} before repeating any values. This property, *full-period*, is achieved for prime values of M when A is a primitive root of M [Fishman and Moore (1982)], e.g., $A^{M-1} \pmod M \equiv 1$ and $A^p \pmod M \text{^}\equiv 1$ for all $1<p<M-1$.

SAS software – Version 6 – uses a multiplicative generator with prime modulus $M=2^{31}-1$ and multiplier A=397204094 [SAS Institute Inc., 1990]. Older versions also provided a generator with non-prime modulus 2^{31} and multiplier 16807 [SAS Institute Inc., 1982]. In order to *improve* the properties of the latter one, a 64-bit shuffle was employed [Westlake, 1967; Kennedy and Gentle, 1980]. Basically, two sequences of values are generated. Certain bits of the value in one sequence are used to determine the extent to which the bits of the corresponding value in the other sequence are shifted (lower bits are shifted to the left and the displaced higher bits are put in their place). This is a prime example of the unpredictability paradox. SAS Version 6 software no longer supports the latter generator (the functions RANUNI and UNIFORM now perform the same way).

The SAS Version 6 generator – referred to hereafter as SASMG – is one of the four from a selected group of generators studied by Fishman and Moore (1982) that passes all of their tests (independence, uniform on the unit interval, uniform on the unit square, uniform on the unit cube, and the simultaneous holding of all four tests). SASMG is listed first among the four and has the smallest multiplier, though it is not the clear-cut best of the four. They also evaluate the multiplier 16807, but with modulus $2^{31}-1$ rather than 2^{31}. In a later paper, Fishman and Moore (1986) looked at 414 multipliers meeting a stringent criterion (concerning the distance between adjacent parallel hyperplanes). Five generators stood out, SASMG is not among them. They conclude that "If one feels compelled to rank the multipliers, one might regard A=950706376 as first…"

Implementing SASMG on a computer requires integer computations, shift operations, and sign checks. One cannot simply use the code X=MOD(A*X,2**31-1) and get the same sequence as RANUNI. Instead, the *MG1 Procedure* proposed by Payne, Rabung, and Bogyo (1969) is used. This procedure, summarized in Fishman and Moore (1982), takes advantage of binary computations by first using the modulus 2^{31} then, with a series of *corrections*, converts the value to that resulting from the modulus $2^{31}-1$.

The SAS function RANUNI is an implementation of the MG1 procedure and allows the specification of an explicit seed (ex.: RANUNI(443243) yields 0.36433564562552) or a seed generated by the computer's clock. There is a third option – implemented by inserting a negative value as the function parameter – that queries the clock for a new seed each time RANUNI is encountered, then returns the next value in the stream. This option is not recommended, as it is yet another candidate for the unpredictability paradox.

NON-UNIFORM GENERATORS

Uniform generators may be used to create streams of pseudo-random numbers that have other distributions. The basic techniques include the (1) inverse probability transform and other transformations, (2) composition method, and (3) acceptance-rejection methods. These are nicely summarized in Rubinstein (1981). We will focus on technique (1).

Inverse Probability Transform

This technique is simple in concept and often is simple to implement as well. Suppose X is a continuous random variable and F(x) is its corresponding *cdf* (cumulative distribution function). That is, F(x) describes the probability that X is less than or equal to x. Then it is well known that U=F(X) has a uniform distribution on (0,1). Since F(x) is a monotonically increasing function, it will have a unique inverse function $F^{-1}(u)$. Thus if U is uniform on (0,1), $X= F^{-1}(U)$ will be a random variable with the desired function.

A similar technique is used for finite discrete or even nominal variables. In this case, F(x) is a step function and $F^{-1}(u)$ maps successive, disjoint segments of the unit interval to the discrete or nominal values. This method is essentially a table look-up

method, which is exactly the concept behind the SAS function RANTBL.

Several SAS functions (Table 1) use the inverse transform method. Basically, each generates a stream of uniform values using SASMG, then converts these using an algorithm that implements the method.

TABLE 1: SAS FUNCTIONS OF THE FORM RANxxx

SAS Function	Technique	Distribution
RANBIN[a]	IT[b], T[c]	Binomial (user: # trials and success probability)
RANCAU	AR[d]	Cauchy (location=0, scale=1)
RANEXP	IT	Exponential (λ=1)
RANGAM	AR	Gamma (user: shape parameter, scale=1)
RANNOR[e]	T	Standard Normal
RANPOI[f]	IT, T	Poisson (user: mean)
RANTBL	IT	Finite Discrete or Nominal (user specifies all probabilities)
RANTRI	IT	Triangular on (0,1) (user: mode)

a. The normal approximation is used when the number of trials is large.
b. IT = Inverse (probability) transform
c. T = Other transformation
d. AR = Acceptance-rejection method
e. Box-Muller transformation [Kennedy and Gentle, 1980]
f. The normal approximation is used when the mean is large.

If the desired distribution is not covered by the RANxxx functions, and either the cumulative or the inverse cumulative distribution function is known, then, with a little programming, the inverse transform method may still be used. It is simple if the inverse *cdf* is known and is easily programmed. The SAS quantile functions, inverse *cdf*, are given in Table 2.

TABLE 2: SAS QUANTILE (INVERSE CDF) FUNCTIONS

Inverse CDF	Distribution
BETAINV	Beta (user: two shape parameters)
CINV	Chi-squared (user: *df*[a] and *ncp*[b])
FINV	F (user: numerator & denominator *df* and *ncp*)
GAMINV	Gamma distribution (user: shape parameter, scale=1)
PROBIT	Standard normal
TINV	Student's t (user: *df* and *ncp*)

a. Degrees of freedom
b. Noncentrality parameter

A chi-squared random variable may be generated using RANGAM from Table 1. A gamma variable with shape parameter equal to k/2 and scale parameter equal to 0.5 is distributed as a chi-squared random variable with k degrees of freedom. The following code will generate a stream of 10 chi-square pseudo-random numbers with 3 degrees of freedom.

```
*** Generating Chi-Squared Values ***;
data ChiSqV;
  do i=1 to 100;
    ChiSqV = RanGam(200025,3/2)/(0.5);
    Output;
  end;
```

Alternatively, CINV may be used to general similar – though a different stream of – values as follows.

```
*** Generating Chi-Squared Values ***;
data ChiSqV;
  do i=1 to 100;
    ChiSqV = CInv(RanUni(200025),3);
    Output;
  end;
```

When the inverse *cdf* is not available and it is computationally infeasible to compute it directly, numerical methods may be used. If the *cdf* and the probability density function (*pdf*) are available, the method of false position can be used. The *pdf* is the derivative of the *cdf* and may be approximated from the *cdf* if not available. Alternatively, the *cdf* may be approximated from the *pdf* using a numerical method to find integrals under the curve (such as quadrature or the trapezoidal method [Kennedy and Gentle, 1980]).

The method of false position involves applying Newton's method [Kennedy and Gentle, 1980] to the function h(x) = F(x)–u, where u is a value between 0 and 1. Here we are assuming F(x) is continuous. The object is to find the value of x such that h(x) is zero (it will be unique). This is equivalent to finding x such that F(x) = u. The method is thus: generate a value of U using SASMG and find the value X such that F(X)=U. Then X has the distribution associated with F(X).

The example below generates chi-squared (3 degrees of freedom) pseudo-random values using the method of false position and the functions PROBCHI (*cdf*), PDF (*pdf* for the named distribution).

```
*** Generating Chi-Squared Values ***;
data _null_;
/* Specify the degrees of freedom    */
  df=3 ;
do i=1 to 100;
/* Generate a Uniform random value   */
  u=ranuni(200025);
/* Null value to compare to  x.      */
  x0=0;
/* Initialize the value of x.        */
  x1=1;
          /* Stopping rule */
  do while(abs(x1-x0)>.000005);
/* Count the number of iterations.   */
     count+1;
/* Save the current value of x.      */
     x0=x1;
/* Newton's Formula */
     x1=x1 -
        (ProbChi(x1,df)-u)
        /PDF('CHISQUARED',x1,df);
     end;
/* Use SAS's inverse Chi-Sq function */
/* for comparison.                   */
  real=CInv(u,df);

  file print;
  put "estimated value= " x1 @32 count=
      @45 """real value""= " real;
end;
```

Other Transformations

Random variables may be generated by using functions of other random variables. For example, the method used to implement RANNOR is a function of two independent uniform variables, the Box-Muller method [Kennedy and Gentle, 1980]. If (U_1,U_2) is a pair of independent standard uniform random variables, then (X_1,X_2) is a pair of independent standard normal random variables where

$$X_1 = [-2 \ln(U_1)]^{1/2} \cos(2\pi U_2) \text{ and}$$
$$X_2 = [-2 \ln(U_1)]^{1/2} \sin(2\pi U_2).$$

[The proof of this is a nice exercise in a first semester mathematical statistics course.]

The sum of squares of k independent standard normal random variables is known to have a chi-squared distribution with k degrees of freedom. While this is not a very efficient method to generate a chi-squared variable, it serves as another example. (See SIMULATION OF PROBABILITIES below.)

SIMPLE RANDOM SAMPLING

An obvious application of uniform random numbers is simple random sampling from a data set. We learned in our introductory statistics class how to select a random sample using a random number table found in the back of our textbook. This cumbersome technique used tabulated numbers that were produced using a pseudo-random number generator. The same basic technique can be programmed with a simple data step.

SAS software comes with a sample library of programs. The example program shown below is a variation on one taken from the sample library. Suppose we wish to select 15 records from a data set with 500 (more or less). Once a record is selected, it is removed from the pool and cannot be selected again (sampling without replacement). One method is to pass through the data once and assign each record a uniform random value. Then sort the data on the assigned values and skim from the top the first 15 records. While this will work, it is not efficient, especially if the data set is very large.

Instead we will pass through the data once, deciding at each record if this one should be selected. If n is the total number of records and k is the desired sample size, then the probability the first record is in the final sample is k/n. We sample the record if a uniform random value is less than k/n. Else, we pass that record by. If the first record is selected, then the next record is selected with probability (k–1)/(n–1), else it is selected with probability k/(n–1). We proceed in this manner – decreasing n by 1 each time and decreasing k by one only if the current record is selected – until k reaches zero. The following code provides an example of this technique.

```
data select;
/* k= required sample size  */
   k=15;
/* n=population size,       */
/* k remains to be selected,*/
/* n_remain remaining pool  */
   do n_remain=n to 1 by -1 while (k>0);
/* k/n_select is condition probability  */
/* of selecting the next record         */
      if ranuni(0) < k/n_remain then do;
         /* Pick the next sample obs */
         set sdl.manpower(keep=ssn) nobs=n;
         output;
         /* One less to choose */
         k=k-1;
         end;
      end;
   stop;
```

SIMULATION OF PROBABILITIES

The *cdf* provides the means to find the probability that a random variable is less than or equal to a given value. For example, the probability that a chi-squared random variable with 3 degrees of freedom is less than 3 is given by PROBCHI(3,3). In cases where the *cdf* is not available, the probability may be simulated if a mechanism for producing random values of the desired variable is possible.

The example below shows how to simulate the above mentioned probability using the fact that the sum of three standard normal random variables has a chi-squared distribution with 3 degrees of freedom.

```
data empiric;
/* Estimation confidence level */
  Delta=.95;
/* Max. error in estimate,     */
/* with 100*delta% confidence  */
  Error=0.01;
/* Simulation Size             */
  SimSize=ceil((probit((1+delta)/2)
          /(2*Error))**2);
  do i=1 to SimSize;
     y=0;
     /* Sum of squares of three */
     /* standard normal values  */
     do j=1 to 3;
        y+rannor(88372)**2;
        end;
     /* How often did y<3 occur */
     event+(y<=3);
     end;
probevnt=event/Simsize;
realval=probchi(3,3);
file print;
put SimSize= probevnt= realval=;
```

CONCLUSION

Many useful applications arise out of uniform pseudo-random variables. Only a few are mentioned here. While SASMG provides a *good* stream of uniform values, other generators in the literature have been shown to be better (how much better is open to interpretation). However, this generator provides a valuable means for producing non-uniform variates, sampling from data sets, and simulation of processes and probabilities.

REFERENCES

Fishman, G.S. and L.R. Moore (1982), "A Statistical Evaluation of Multiplicative Congruential Random Number Generators With Modulus 2^{31}–1," *J. Amer. Stat. Assoc.*, 77(377), 129-136.

——— (1986), "An Exhaustive Analysis of Multiplicative Congruential Random Number Generators with Modulus 2^{31}–1," SIAM J. Sci. Stat. Comput., 7(1) 24-45.

Kennedy, W.J. and J.E. Gentle (1980), *Statistical Computing*, Marcel Dekker, Inc., New York.

Lehmer, D.H. (1951), "Mathematical Methods in Large Scale Computing Units," *Ann. Comp. Labs.*, Harvard Univ., 26, 141-146.

Lagarias, J.C. (1993), "Pseudorandom Numbers," Statistical Science, 8(1), 31-39.

Payne, W.J., Rabung, J.R. and T.P. Bogyo (1969), "Coding the Lehmer Pseudo-Random Number Generator," *Comm. Assoc. Comp. Mach.*, 12(2), 85-86.

Rubinstein, R.Y. (1981), *Simulation and the Monte Carlo Method*, John Wiley & Sons, Inc., New York.

SAS Institute Inc., (1982), *SAS User's Guide: Basics, 1982 Edition*, SAS Institute Inc., Cary, NC.

SAS Institute Inc., (1990), *SAS Language: Reference, Version 6, First Edition*, SAS Institute Inc., Cary, NC.

Westlake, W.J. (1967), "A Uniform Random Number Generator Based on the Combination of Two Congruential Generators," *J. Assoc. Comp. Mach.*, 14, 337-340.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Robert E. Johnson
Department of Mathematical Sciences
Division of Operations Research and Statistics
Virginia Commonwealth University
Richmond, VA 23284-2084
Phone: (804) 828-1301 x125
Fax: (804) 828-8785
Email: rjohnson@vcu.edu
WEB: http://saturn.vcu.edu/~rjohnson