NAG Fortran Library Chapter Introduction

G05 – Random Number Generators

Contents

1 Scope of the Chapter

This chapter is concerned with the generation of sequences of independent pseudo-random and quasi-random numbers from various distributions, and the generation of pseudo-random time series from specified time-series models.

2 Background to the Problems

A sequence of pseudo-random numbers is a sequence of numbers generated in some systematic way such that its statistical properties are as close as possible to those of true random numbers: for example, negligible correlation between consecutive numbers. The most common methods are based on the **multiplicative congruential** algorithm, see Knuth (1981). The basic algorithm is defined as:

$$n_i = (a \times n_{i-1}) \bmod m \tag{1}$$

The integers n_i are then divided by m to give uniformly distributed pseudo-random numbers lying in the interval (0,1).

Alternatively there is a variant known as the Wichmann–Hill algorithm, see Maclaren (1989), defined as:

$$\begin{aligned} n_{1,i} &= (a_1 \times n_{1,i-1}) \bmod m_1 \\ n_{2,i} &= (a_2 \times n_{2,i-1}) \bmod m_2 \\ n_{3,i} &= (a_3 \times n_{3,i-1}) \bmod m_3 \\ n_{4,i} &= (a_4 \times n_{4,i-1}) \bmod m_4 \\ U_i &= \left(\frac{n_{1,i}}{m_1} + \frac{n_{2,i}}{m_2} + \frac{n_{3,i}}{m_3} + \frac{n_{4,i}}{m_4}\right) \bmod 1.0 \end{aligned} \tag{2}$$

This generates pseudo-random numbers U_i, uniformly distributed in the interval (0,1).

Either of these algorithms can be selected to generate uniformly distributed pseudo-random numbers. If the basic algorithm (1) is selected then the NAG generator uses the values $a = 13^{13}$ and $m = 2^{59}$ in (1). This generator gives a **cycle length** (i.e., the number of random numbers before the sequence starts repeating itself) of 2^{57}. A good rule of thumb is never to use more numbers than the square root of the cycle length in any one experiment as the statistical properties are impaired. For closely related reasons, breaking numbers down into their bit patterns and using individual bits may cause trouble.

If the Wichmann–Hill algorithm is selected then one or more of 273 independent generators are available. Each of these is defined by the set of constants a_j and m_j for $j = 1, \ldots, 4$. The constants a_j are in the range 112 to 127 and the constants m_j are prime numbers in the range 16718909 to 16776971, which are close to $2^{24} = 16777216$. These constants have been chosen so that they give good results with the spectral test, see Knuth (1981) and Maclaren (1989). The period of each Wichmann–Hill generator would be at least 2^{92} if it were not for common factors between $(m_1 - 1)$, $(m_2 - 1)$, $(m_3 - 1)$ and $(m_4 - 1)$. However, each generator should still have a period of at least 2^{80}. Further discussion of the properties of these generators is given in Maclaren (1989) where it was shown that the generated pseudo-random sequences are essentially independent of one another according to the spectral test.

The sequence given in (1) needs an initial value n_0, known as the **seed**, while the sequence given in (2) needs four such seeds. The use of the same seed will lead to the same sequence of numbers when these are computed serially. One method of obtaining a seed is to use the real-time clock; this will give a non-repeatable sequence. It is important to note that the statistical properties of the random numbers are only guaranteed within sequences and not between sequences. Repeated initialization will thus render the numbers obtained less rather than more independent. Similarly the statistical properties of the random numbers are not guaranteed between two sequences generated using the two algorithms.

Random numbers from other distributions may be obtained from the uniform random numbers by the use of transformations and rejection techniques, and for discrete distributions, by table based methods.

(a) Transformation Methods

For a continuous random variable, if the cumulative distribution function (CDF) is $F(x)$ then for a uniform (0,1) random variate u, $y = F^{-1}(u)$ will have CDF $F(x)$. This method is only efficient in a few simple cases such as the exponential distribution with mean μ, in which case $F^{-1}(u) = -\mu \log u$. Other transformations are based on the joint distribution of several random variables. In the bivariate case, if v and w are random variates there may be a function g such that $y = g(v, w)$ has the required distribution; for example, the Student's t-distribution with n degrees of freedom in which v has a Normal distribution, w has a gamma distribution and $g(v, w) = v\sqrt{n/w}$.

(b) Rejection Methods

Rejection techniques are based on the ability to easily generate random numbers from a distribution (called the envelope) similar to the distribution required. The value from the envelope distribution is then accepted as a random number from the required distribution with a certain probability; otherwise, it is rejected and a new number is generated from the envelope distribution.

(c) Table Search Methods

For discrete distributions, if the cumulative probabilities, $P_i = \text{Prob}(x \leq i)$, are stored in a table then, given u from a uniform (0,1) distribution, the table is searched for i such that $P_{i-1} < u \leq P_i$. The returned value i will have the required distribution. The table searching can be made faster by means of an index, see Ripley (1987). The effort required to set up the table and its index may be considerable, but the methods are very efficient when many values are needed from the same distribution.

In addition to random numbers from various distributions, random compound structures can be generated. These include random time series, random matrices and random samples.

The efficiency of a simulation exercise may often be increased by the use of variance reduction methods (see Morgan (1984)). It is also worth considering whether a simulation is the best approach to solving the problem. For example, low-dimensional integrals are usually more efficiently calculated by routines in Chapter D01 rather than by Monte Carlo integration.

Quasi-random numbers are intended primarily for use in Monte Carlo integration. Like pseudo random numbers they are uniformly distributed but they are not statistically independent, rather they are designed to give a more even distribution in multidimensional space (uniformity). Therefore, they are often more efficient than pseudo random numbers in multidimensional Monte Carlo methods. There are several quasi-random generators, three of which are available in this chapter, they are the Sobol, Faure and Neiderreiter generators.

3 Recommendations on Choice and Use of Available Routines

Note: refer to the Users' Note for your implementation to check that a routine is available.

3.1 Design of the Chapter

All the generation routines call – directly or indirectly – an internal generator (selected to be either the basic generator (1) or a Wichmann–Hill generator (2)), which generates random numbers from a uniform distribution over (0,1). However, there are two distinct sets of generation routines representing two distinct methods for communicating the data representing the current state of a given generator. In the first set the data is stored and passed internally, while in the second set the data is held in parameters that are passed through the routine interfaces. It is recommended that the second set of routines are used.

3.1.1 Routines using internal communication

The first distinct set of generation routines are G05CAF through to G05HDF. These routines store and pass information relating to generator states internally via COMMON blocks and saved variables. These have the advantage of having a simpler interface since generator data does not have to be passed through it. However, this advantage is achieved through the use of potentially thread-unsafe constructs that could result in incorrect results when used within some multi-threaded applications. Note that, for this set of routines a call to any generation routine will affect all subsequent random numbers produced by any other routine in the set. Despite this effect, the values will remain as independent as if the different sequences were produced separately.

A utility routine is provided to select the internal generator for the set of routines G05CAF to G05HDF:

G05ZAF allows you to select either the basic generator (1) or the Wichman–Hill generators (2). It is recommended that the Wichman–Hill generators are selected since these have a longer cycle-length. The basic generator should be used when you wish to reproduce results obtained from code calling Chapter G05 routines from previous releases of the Library.

Two utility routines are provided to initialize the generator:

G05CBF initializes it to a repeatable (when executed serially) state, dependent on an integer parameter: two calls of G05CBF with the same parameter-value will result in the same subsequent sequences of random numbers (when both are generated serially). If G05ZAF is used to select the Wichmann–Hill generators, then G05CBF selects one of the 273 possible generators as the base generator depending on the seed used; the base generator number is computed as $\mathrm{mod}(\mathrm{seed}-1,273)+1$.

G05CCF initializes it to a non-repeatable state, in such a way that different calls of G05CCF, either in the same run or different runs of the program, will almost certainly result in different subsequent sequences of random numbers.

Two other utility routines, G05CFF and G05CGF, are provided to save or restore the state of the internal generator (including the seed(s) of the multiplicative congruential method used by the generator). G05CFF and G05CGF can be used to produce two or more sequences of numbers, where some are repeatable and some are not; for example, this can be used to simulate signal and noise. As their overheads are not negligible, numbers should be produced in batches when this technique is used. While they can be used to save the state of the internal generator between jobs, the two arrays must be restored accurately. The corresponding process between machines, while sometimes possible, is not advised. It also makes no sense to save the state from one generator and restore it for the alternative generator.

3.1.2 Routines communicating through the interface

It is recommended that this set of routines are used for the generation of sequences of independent pseudo-random numbers from various distributions, and the generation of pseudo-random time series from specified time-series models.

The second distinct set of generator routines are G05KAF through to G05QDF. These routines pass information relating to the generator and its current state through their interfaces using the parameters IGEN and ISEED. These routines do **not** contain any thread-unsafe constructs, but they do have at least two extra arguments in their interface.

Two utility routines are provided to initialize the generators:

G05KBF selects and initializes a generator to a repeatable (when executed serially) state: two calls of G05KBF with the same parameter-values will result in the same subsequent sequences of random numbers (when both are generated serially). The basic generator or one of 273 Wichmann–Hill generators can be selected as the base generator.

G05KCF selects and initializes a generator to a non-repeatable state, in such a way that different calls of G05KCF, either in the same run or different runs of the program, will almost certainly result in different subsequent sequences of random numbers.

Routines to save and restore generator states are not required for this set of routines since the ISEED parameter can have its values copied and stored at any time.

3.1.3 Repeated intialisation

As mentioned in Section 2, it is important to note that the statistical properties of pseudo-random numbers are only guaranteed within sequences and not between sequences produced by the same generator. Repeated initialization will thus render the numbers obtained less rather than more independent. In a simple case there should be only one call, for the first set of generator routines, to G05CBF or G05CCF, or for the second set of generator routines, one call to G05KBF or G05KCF; this call should be before any call to an actual generation routine.

3.2 Selection of Routine

The following distributions are available.

(a) Functions Returning a Single Random Variate

Real number from the continuous uniform distribution	`G05KAF`
Logical value .TRUE. or .FALSE.	`G05KEF`

(b) Vectors of Random Variates from Continuous Distributions

Normal distribution	`G05LAF`
Student's t-distribution	`G05LBF`
Chi-square distribution	`G05LCF`
F-distribution	`G05LDF`
Beta distribution	`G05LEF`
Gamma distribution	`G05LFF`
Uniform distribution	`G05LGF`
Triangular distribution	`G05LHF`
Negative exponential distribution	`G05LJF`
Lognormal distribution	`G05LKF`
Cauchy distribution	`G05LLF`
Weibull distribution	`G05LMF`
Logistic distribution	`G05LNF`
von Mises distribution	`G05LPF`
Exponential mix distribution	`G05LQF`

(c) Single Multivariate from Multivariate Distributions

Multivariate Normal distribution	`G05LZF`

(d) Vector of variates from Discrete Distributions

Uniform distribution	`G05MAF`
Geometric distribution	`G05MBF`
Negative binomial distribution	`G05MCF`
Logarithmic distribution	`G05MDF`
Binomial distribution	`G05MJF`
Poisson distribution	`G05MKF`
Hypergeometric distribution	`G05MLF`
Multinomial distribution	`G05MRF`
User-supplied distribution	`G05MZF`

Most of the above routines set up (or have the option to set up) a search table and index in a reference array; on subsequent calls the routine can generate the random variate from the information in the reference array.

(e) Variate array from Discrete Distributions with array of parameters

Poisson distribution with varying mean	`G05MEF`

(f) Generation of Time Series

Univariate ARMA model, Normal errors	`G05PAF`
Vector ARMA model, Normal errors	`G05PCF`
Symmetric GARCH or asymmetric GARCH Type I	`G05HKF`
Asymmetric GARCH Type II	`G05HLF`
Asymmetric GJR GARCH	`G05HMF`
EGARCH	`G05HNF`

(g) Sampling and Permutation

Random permutation of an integer vector	`G05NAF`
Random sample from an integer vector	`G05NBF`
Symmetric GARCH or asymmetric GARCH Type I	`G05HKF`
Asymmetric GARCH Type II	`G05HLF`
Asymmetric GJR GARCH	`G05HMF`

EGARCH	G05HNF

(h) Random Matrices

Random orthogonal matrix	G05QAF
Random correlation matrix	G05QBF

(i) Random Table

Random table	G05QDF

3.3 Programming Advice

Take care when programming calls to those routines in this chapter which are functions. The reason is that different calls with the same parameters are intended to give different results.

For example, if you wish to assign to Z the difference between two successive random numbers generated by G05KAF, beware of writing

```
      Z = G05KAF(IGEN,ISEED) - G05KAF(IGEN,ISEED)
```

It is quite legitimate for a Fortran compiler to compile zero, one or two calls to G05KAF; if two calls, they may be in either order (if zero or one calls are compiled, Z would be set to zero). A safe method to program this would be

```
      X = G05KAF(IGEN,ISEED)
      Y = G05KAF(IGEN,ISEED)
      Z = X-Y
```

4 Routines Withdrawn or Scheduled for Withdrawal

The following routines have either been withdrawn or superseded. Those routines indicated by * are still present at Mark 20 but will be omitted at Mark 21; those indicated by ** will be retained in the Library until at least Mark 22. Advice on replacing calls to those withdrawn since Mark 13 is given in the document ‘Advice on Replacement Calls for Withdrawn/Superseded Routines’.

Withdrawn Routine	**Mark of Withdrawal**	**Replacement Routine(s)**
G05AAF	7	G05KAF
G05ABF	7	G05LGF
G05ACF	7	G05LJF
G05ADF	7	G05LAF
G05AEF	7	G05LAF
G05AFF	7	G05LKF
G05AGF	7	G05LLF
G05AHF	7	G05LFF
G05AJF	7	G05LFF
G05AKF	7	G05LFF
G05ALF	7	G05LEF
G05AMF	7	G05LEF
G05ANF	7	G05LCF
G05APF	7	G05LBF
G05AQF	7	G05LDF
G05ARF	7	G05MZF
G05ASF	7	G05MJF
G05ATF	7	G05MAF
G05AUF	7	G05MLF
G05AVF	7	G05MKF
G05AWF	7	G05MZF
G05AZF	7	G05MZF
G05BAF	7	G05KBF
G05BBF	7	G05KCF
G05CAF**	22	G05KAF

G05CBF**	22	G05KBF
G05CCF**	22	G05KCF
G05CFF**	22	F06DFF
G05CGF**	22	F06DFF
G05DAF**	22	G05LGF
G05DBF**	22	G05LJF
G05DCF**	22	G05LNF
G05DDF**	22	G05LAF
G05DEF**	22	G05LKF
G05DFF**	22	G05LLF
G05DGF	16	G05LFF
G05DHF**	22	G05LCF
G05DJF**	22	G05LBF
G05DKF**	22	G05LDF
G05DLF	16	G05LEF
G05DMF	16	G05LEF
G05DPF**	22	G05LMF
G05DRF**	22	G05MEF
G05DYF**	22	G05MAF
G05DZF**	22	G05KEF
G05EAF**	22	G05LZF
G05EBF**	22	G05MAF
G05ECF**	22	G05MKF
G05EDF**	22	G05MJF
G05EEF**	22	G05MCF
G05EFF**	22	G05MLF
G05EGF**	22	G05PAF
G05EHF**	22	G05NAF
G05EJF**	22	G05NBF
G05EWF**	22	G05PAF
G05EXF**	22	G05MZF
G05EYF**	22	G05MZF
G05EZF**	22	G05LZF
G05FAF**	22	G05LGF
G05FBF**	22	G05LJF
G05FDF**	22	G05LAF
G05FEF**	22	G05LEF
G05FFF**	22	G05LFF
G05FSF**	22	G05LPF
G05GAF**	22	G05QAF
G05GBF**	22	G05QBF
G05HDF**	22	G05PCF

5 References

Knuth D E (1981) *The Art of Computer Programming (Volume 2)* (2nd Edition) Addison-Wesley

Maclaren N M (1989) The generation of multiple independent sequences of pseudorandom numbers *Appl. Statist.* **38** 351–359

Morgan B J T (1984) *Elements of Simulation* Chapman and Hall

Ripley B D (1987) *Stochastic Simulation* Wiley