NAG C 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 functions 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 Functions

3.1 Pseudo-random Generators

The functions allow the selection of either the basic generator or one of the suite of Wichmann–Hill generators. Information on the generator is passed between functions using the parameters *igen* and *iseed.* Two utility functions are provided to initialise the generator.

nag_rngs_init_repeatable (g05kbc)
selects a generator and initialises it to give a repeatable stream of random numbers.

nag_rngs_init_nonrepeatable (g05kcc)
selects a generator and initialises it to a non-repeatable state using the system clock.

These functions should only be used once for each stream of random numbers created. Each stream generated by a different generator will be independent. To generate high-dimensional variables one approach would be to use a different independent generator for each dimension.

3.2 Quasi-random Generators

The functions available for quasi-random number each provide the user the option of selecting a Solbol, Faure or Neiderreiter sequence. The maxium number of dimensions is 40. If higher dimensions are required these generators can be combined with the pseudo-random generators described above.

nag_quasi_random_uniform (g05yac)
Uniform distribution

nag_quasi_random_normal (g05ybc)
Normal distribution

4 Index

- Pseudo-random integer(s),
  - reference vector,
    - binomial distribution .... nag_ref_vec_binomial (g05edc)
    - cumulative distribution .... nag_ref_vec_discrete_pdf_cdf (g05exc)
    - Poisson distribution .... nag_ref_vec_poisson (g05ecc)
    - Poisson distribution or binomial distribution .... nag_return_discrete (g05eyc)
    - probability distribution .... nag_ref_vec_discrete_pdf_cdf (g05exc)
  - uniform distribution .... nag_random_discrete_uniform (g05dyc)
- Pseudo-random real number(s),
  - beta distribution .... nag_random_beta (g05fec)
  - gamma distribution .... nag_random_gamma (g05ffc)
  - multivariate Normal vector
    - reference vector .... nag_ref_vec_multi_normal (g05eac)
  - return multivariate Normal vector .... nag_return_multi_normal (g05ezc)
  - uniform distribution over (0,1) .... nag_random_continuous_uniform (g05cac)
  - uniform distribution over (a, b) .... nag_random_continuous_uniform_ab (g05dac)
- Sampling and permutation,
  - pseudo-random permutation .... nag_ran_permut_vec (g05ehc)
  - pseudo-random sample .... nag_ran_sample_vec (g05ejc)
- State functions,
  - initialise generator,
    - nonrepeatable sequence .... nag_random_init_nonrepeatable (g05ccc)
    - repeatable sequence .... nag_random_init_repeatable (g05cbc)
  - restore generator state .... nag_restore_random_state (g05cgc)
  - save generator state .... nag_save_random_state (g05cfc)
- Time series,
  - ARMA .... nag_arma_time_series (g05hac)

5 Functions Withdrawn or Scheduled for Withdrawal

None.

6 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