

NAG Numerical Functions for GPUs Chapter Introduction**gpu_g05 – Random Number Generators****Contents**

1	Scope of the Chapter	2
2	Background to the Problems	2
2.1	Pseudorandom Numbers	2
2.1.1	MRG32k3a I Generator	2
2.2	Quasi-random Numbers	2
2.3	Brownian Bridge	2
3	Recommendations on Choice and Use of Available Functions	3
4	Index	4
5	References	4

1 Scope of the Chapter

This chapter contains functions for the generation of sequences of pseudo-random numbers from various distributions.

2 Background to the Problems

2.1 Pseudorandom Numbers

A sequence of pseudorandom numbers is a sequence of numbers generated in some systematic way such that they are independent and statistically indistinguishable from a truly random sequence. A pseudorandom number generator (PRNG) is a mathematical algorithm that, given an initial state, produces a sequence of pseudorandom numbers. A PRNG has several advantages over a true random number generator in that the generated sequence is repeatable, has known mathematical properties and can be implemented without needing any specialist hardware. Many books on statistics and computer science have good introductions to PRNGs, for example Knuth (1981) or Banks (1998).

2.1.1 MRG32k3a I Generator

The Multiple Recursive Generator, MRG32k3a, is due to L'Ecuyer (1999), with a later paper L'Ecuyer *et al.* (2002) mentioning its parallel implementation. L'Ecuyer gives a very efficient serial implementation in his first paper. The parallel implementation relies on the fact that the two recurrences defining this generator are of the same form, and can be represented as simple matrices leading to an efficient means of computing skip ahead points.

The GPU implementation is fairly straightforward. The matrices defining the skip ahead can be pre-computed on the host, copied to device memory, and then read into shared memory by each thread block. Each thread uses the skip-ahead algorithm to compute the initial point and then uses the standard algorithm to compute a contiguous block of pseudorandom variates.

2.2 Quasi-random Numbers

Quasi-random numbers are intended primarily for use in Monte Carlo integration. Like pseudorandom numbers they are evenly distributed, but whereas pseudorandom sequences aim for statistical independence, quasi-random sequences aim for low discrepancy. Discrepancy is a measure of how evenly a sequence fills an area of multidimensional space. In a low discrepancy sequence, each point will tend to lie an equal distance from all its neighbouring points. This is typically not true for pseudorandom sequences, where one usually observes some form of clustering. For this reason quasi-random sequences are often more efficient in multidimensional Monte Carlo methods.

The low discrepancy sequence due to Sobol is provided here, based on the extension to higher dimensions described in Joe and Kuo (2003) and Joe and Kuo (2008).

2.3 Brownian Bridge

The term ‘Brownian bridge’ can mean one of two things: either it refers to a particular stochastic process which resembles a standard Brownian motion, but is forced to be zero at some time $T > 0$; or it refers to a particular algorithm used to construct Brownian sample paths. The bridge algorithm constructs a Brownian sample path by first simulating its final value and then recursively filling in intermediate values through an interpolation formula. When compared with standard path construction methods, the bridge algorithm often displays advantages when solving stochastic differential equations or when pricing path dependent options. It is also possible to construct sample paths of the Brownian bridge process via the Brownian bridge algorithm, and this could be useful in certain stochastic models.

The Brownian bridge can be constructed in many different orderings. A depth-ordered algorithm is employed here, where the discretized Brownian motion X_k , for $k = 0, 1, \dots, N$, on the time interval $[0, T]$ is generated in the order, $X_0, X_T, X_{T/2}, X_{T/4}, X_{T/8}, \dots, X_1, X_{3T/4}, X_{3T/8}, \dots, X_3, \dots, X_{N-1}$ where for simplicity the case of equal time increments with N a power of 2 is shown. The algorithm provided allows for unequal time increments and does not restrict N to be a power of 2. However, for the purpose of introducing the samples from a Normal distribution, the Brownian bridge is structured internally by levels in proceeding from the coarsest subdivisions of the time interval $[0, T]$ to the finest. This ensures

that largest part of the variance of the Brownian path is concentrated in the coarser levels which can be matched to the lowest dimensions of a low discrepancy sequence such as that of Sobol when these appear as the earliest entries in the input array.

3 Recommendations on Choice and Use of Available Functions

Prior to generating any pseudorandom variates the base generator being used must be initialized by calling the function `nag_gpu_mrg32k3a_init`. Once initialized, a distributional generator can be called to obtain the variates required.

The random numbers computed can be chosen to be either double or single precision, however, double precision is recommended since the algorithm is designed for double precision storage and in single precision the sequence cannot be guaranteed to pass all of the statistical tests. If a sequence of random variates from a uniform distribution on the half-closed interval $(0, 1]$, is required, then the uniform distribution function `nag_gpu_mrg32k3a_uniform` should be called. Functions for Normal (`nag_gpu_mrg32k3a_normal`) and exponential (`nag_gpu_mrg32k3a_exp`) distributions are also included.

`nag_gpu_depthbbinc_init` initializes a binary tree which defines the Brownian bridge to be constructed. `nag_gpu_depthbbinc` generates the Brownian bridge taking as input a previously computed array of pseudo or quasi-random numbers sampled from the standard Normal distribution.

`nag_gpu_depthbb_cleanup` must be called to release system resources following generation of the Brownian bridge.

The Brownian bridge generated by `nag_gpu_depthbbinc` can be used to solve a stochastic differential equation (SDE) driven by a Wiener process

$$dy(t) = f(t, y)dt + g(t, y)dW(t)$$

for

$$0 \leq t \leq T \quad \text{with} \quad y(0) = y_0,$$

where $f(t, y)$ is the drift coefficient, $g(t, y)$ is the diffusion coefficient and $W(t)$ is the standard Wiener process whose increment is

$$\Delta W(t) = W(t + \Delta t) - W(t) = \sqrt{\Delta t}Z$$

and $Z \sim \mathcal{N}(0, 1)$.

For numerical solution, the given SDE can be discretized using the Euler-Maruyama method to give

$$y_{n+1} = y_n + f(t_n, y_n)\Delta t + g(t_n, y_n)\Delta W_n,$$

with $t_n = n\Delta t$.

The output values $d_W[i]$, for $i = 0, 1, \dots, \text{dim} \times \text{npath} \times \text{nTimes}$ (see `nag_gpu_depthbbinc` and `nag_gpu_depthbbinc_init`), where `dim` is the dimension of the SDE, `npath` is the number of simulation paths and `nTimes` is the number of time steps, are of the form

$$\hat{W}_n = \frac{\Delta W_n}{\Delta t} = \frac{Z_n}{\sqrt{\Delta t}}$$

so that the discretized SDE is

$$y_{n+1} = y_n + \Delta t(f(t_n, y_n) + g(t_n, y, n)\hat{W}_n).$$

For other applications, `nag_gpu_depthbb` is provided. This returns the values of a Brownian path at the specified interpolation points rather than the scaled increments supplied by `nag_gpu_depthbbinc`. A call to the initialisation function, `nag_gpu_depthbb_init` must be made before calling `nag_gpu_depthbb` to generate the Brownian bridge and it should be followed by a call to `nag_gpu_depthbb_cleanup` to free resources.

The Brownian bridge construction is often used with a low discrepancy sequence, such as the Sobol sequence, to supply the Normal variates. The Sobol sequence generator must be initialised by a call to `nag_gpu_sobol_init`. The sequence of Normal variates can then be generated by `nag_gpu_sobol_normal` and resources freed by `nag_gpu_sobol_cleanup`.

Uniform and exponential distributions for the Sobol sequence can be computed by `nag_gpu_sobol_uniform` and `nag_gpu_sobol_exp` respectively.

4 Index

Brownian bridge, depth-ordered construction,

free resources following Brownian bridge `nag_gpu_depthbb_cleanup`
 generate Brownian bridge `nag_gpu_depthbb`
 generate incremental Brownian bridge `nag_gpu_depthbbinc`
 initialize Brownian bridge `nag_gpu_depthbb_init`
 initialize incremental Brownian bridge `nag_gpu_depthbbinc_init`

Pseudorandom numbers,

initialize generator `nag_gpu_mrg32k3a_init`
 variates from exponential distribution `nag_gpu_mrg32k3a_exp`
 variates from normal distribution `nag_gpu_mrg32k3a_normal`
 variates from uniform distribution `nag_gpu_mrg32k3a_uniform`

Quasi-random numbers,

Sobol sequence,

exponential distribution `nag_gpu_sobol_exp`
 free resources following Sobol generation `nag_gpu_sobol_cleanup`
 initialize generator `nag_gpu_sobol_init`
 Normal distribution `nag_gpu_sobol_normal`
 uniform distribution `nag_gpu_sobol_uniform`

5 References

Banks J (1998) *Handbook on Simulation* Wiley

Joe S and Kuo F Y (2003) Remark on Algorithm 659: Implementing Sobol's quasirandom sequence generator *ACM Trans. Math. Software (TOMS)* **29** 49–57

Joe S and Kuo F Y (2008) Constructing Sobol sequences with better two-dimensional projections *SIAM J. Sci. Comput.* **30** 2635–2654

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

L'Ecuyer P (1999) Good parameter sets for combined multiple recursive random number generators *Operations Research* **47:1** 159–164

L'Ecuyer P, Simar R, Chen E J and Kelton W D (2002) An object-oriented random-number package with many long streams and substreams *Operations Research* **50:6** 1073–1075

NVIDIA CUDA (2008) *Programming Guide version 2.0*<http://www.nvidia.com/cuda>