NAG Toolbox

nag_rand_bb_init (g05xa)

1 Purpose

nag_rand_bb_init (g05xa) initializes the Brownian bridge generator nag_rand_bb (g05xb). It must be called before any calls to nag_rand_bb (g05xb).

2 Syntax

```
[rcomm, ifail] = nag_rand_bb_init(t0, tend, times, 'ntimes', ntimes)
[rcomm, ifail] = g05xa(t0, tend, times, 'ntimes', ntimes)
```

3 Description

3.1 Brownian Bridge Algorithm

Details on the Brownian bridge algorithm and the Brownian bridge process (sometimes also called a non-free Wiener process) can be found in Section 2.6 in the G05 Chapter Introduction. We briefly recall some notation and definitions.

Fix two times $t_0 < T$ and let $(t_i)_{1\leq i\leq N}$ be any set of time points satisfying $t_0 < t_1 < t_2 < \cdots < t_N < T$. Let $(X_{t_i})_{1\leq i\leq N}$ denote a d-dimensional Wiener sample path at these time points, and let C be any d by d matrix such that CC^{T} is the desired covariance structure for the Wiener process. Each point X_{t_i} of the sample path is constructed according to the Brownian bridge interpolation algorithm (see Glasserman (2004) or Section 2.6 in the G05 Chapter Introduction). We always start at some fixed point $X_{t_0} = x \in \mathbb{R}^d$. If we set $X_T = x + C\sqrt{T - t_0}Z$ where Z is any d-dimensional standard Normal random variable, then X will behave like a normal (free) Wiener process. However if we fix the terminal value $X_T = w \in \mathbb{R}^d$, then X will behave like a non-free Wiener process.

3.2 Implementation

Given the start and end points of the process, the order in which successive interpolation times t_j are chosen is called the *bridge construction order*. The construction order is given by the array **times**. Further information on construction orders is given in Section 2.6.2 in the G05 Chapter Introduction. For clarity we consider here the common scenario where the Brownian bridge algorithm is used with quasi-random points. If pseudorandom numbers are used instead, these details can be ignored.

Suppose we require P Wiener sample paths each of dimension d. The main input to the Brownian bridge algorithm is then an array of quasi-random points $Z^1, Z^2, \ldots, Z^P$ where each point $Z^p = \left(Z_1^p, Z_2^p, \ldots, Z_D^p\right)$ has dimension $D = d(N + 1)$ or $D = dN$ respectively, depending on whether a free or non-free Wiener process is required. When nag_rand_bb (g05xb) is called, the pth sample path for $1 \leq p \leq P$ is constructed as follows: if a non-free Wiener process is required set X_T equal to the terminal value w, otherwise construct X_T as

$$X_T = X_{t_0} + C\sqrt{T - t_0}\begin{bmatrix} Z_1^p \\ \vdots \\ Z_d^p \end{bmatrix}$$

where C is the matrix described in Section 3.1. The array **times** holds the remaining time points $t_1, t_2, \ldots t_N$ in the order in which the bridge is to be constructed. For each $j = 1, \ldots, N$ set $r = \mathbf{times}(j)$, find

$$q = \max\{t_0, \mathbf{times}(i) : 1 \leq i < j, \mathbf{times}(i) < r\}$$

and

$$s = \min\{T, \mathbf{times}(i) : 1 \leq i < j, \mathbf{times}(i) > r\}$$

and construct the point X_r as

$$X_r = \frac{X_q(s-r) + X_s(r-q)}{s-q} + C\sqrt{\frac{(s-r)(r-q)}{(s-q)}} \begin{bmatrix} Z^p_{jd-ad+1} \\ \vdots \\ Z^p_{jd-ad+d} \end{bmatrix}$$

where $a = 0$ or $a = 1$ respectively depending on whether a free or non-free Wiener process is required. Note that in our discussion j is indexed from 1, and so X_r is interpolated between the nearest (in time) Wiener points which have already been constructed. The function nag_rand_bb_make_bridge_order (g05xe) can be used to initialize the **times** array for several predefined bridge construction orders.

4 References

Glasserman P (2004) *Monte Carlo Methods in Financial Engineering* Springer

5 Parameters

5.1 Compulsory Input Parameters

1: **t0** – REAL (KIND=nag_wp)

The starting value t_0 of the time interval.

2: **tend** – REAL (KIND=nag_wp)

The end value T of the time interval.

Constraint: **tend** $>$ **t0**.

3: **times**(**ntimes**) – REAL (KIND=nag_wp) array

The points in the time interval (t_0, T) at which the Wiener process is to be constructed. The order in which points are listed in **times** determines the bridge construction order. The function nag_rand_bb_make_bridge_order (g05xe) can be used to create predefined bridge construction orders from a set of input times.

Constraints:

t0 $<$ **times**(i) $<$ **tend**, for $i = 1, 2, \ldots,$ **ntimes**;
times$(i) \neq$ **times**(j), for $i, j = 1, 2, \ldots$ **ntimes** and $i \neq j$.

5.2 Optional Input Parameters

1: **ntimes** – INTEGER

Default: the dimension of the array **times**.

The length of **times**, denoted by N in Section 3.1.

Constraint: **ntimes** ≥ 1.

5.3 Output Parameters

1: **rcomm**(**12** $\times$ (**ntimes** $+$ **1**)) – REAL (KIND=nag_wp) array

Communication array, used to store information between calls to nag_rand_bb (g05xb). This array **must not** be directly modified.

2: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

Constraint: **tend** $>$ **t0**.

ifail $= 2$

Constraint: **ntimes** ≥ 1.

ifail $= 3$

Constraint: **t0** $<$ **times**(i) $<$ **tend** for all i.

ifail $= 4$

Constraint: all elements of **times** must be unique.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

The efficient implementation of a Brownian bridge algorithm requires the use of a workspace array called the *working stack*. Since previously computed points will be used to interpolate new points, they should be kept close to the hardware processing units so that the data can be accessed quickly. Ideally the whole stack should be held in hardware cache. Different bridge construction orders may require different amounts of working stack. Indeed, a naive bridge algorithm may require a stack of size $\frac{N}{4}$ or even $\frac{N}{2}$, which could be very inefficient when N is large. nag_rand_bb_init (g05xa) performs a detailed analysis of the bridge construction order specified by **times**. Heuristics are used to find an execution strategy which requires a small working stack, while still constructing the bridge in the order required.

9 Example

This example calls nag_rand_bb_init (g05xa), nag_rand_bb (g05xb) and nag_rand_bb_make_bridge_order (g05xe) to generate two sample paths of a three-dimensional free Wiener process. Pseudorandom variates are used to construct the sample paths.

See Section 10 in nag_rand_bb (g05xb) and nag_rand_bb_make_bridge_order (g05xe) for additional examples.

9.1 Program Text

```
     function g05xa_example

fprintf('g05xa example results\n\n');

% Get information required to set up the bridge
[bgord,t0,tend,ntimes,intime,nmove,move] = get_bridge_init_data();

% Make the bridge construction bgord
[times, ifail] = g05xe( ...
                        t0, tend, intime, move, 'bgord', bgord);

% Initialize the Brownian bridge generator
[rcomm, ifail] = g05xa( ...
                        t0, tend, times);

% Get additional information required by the bridge generator
[npaths,d,start,a,term,c] = get_bridge_gen_data();

% Generate the Z values
[z] = get_z(npaths, d, a, ntimes);

% Call the Brownian bridge generator routine
[z, b, ifail] = g05xb( ...
                       npaths, start, term, z, c, rcomm, 'a', a);

% Display the results
for i = 1:npaths
  fprintf('Weiner Path %d, %d time steps, %d dimensions\n', i, ntimes+1, d);
  w = transpose(reshape(b(:,i), d, ntimes+1));

  ifail = x04ca('G', ' ', w, '');

  fprintf('\n');
end

function [bgord,t0,tend,ntimes,intime,nmove,move] = get_bridge_init_data()
  % Set the basic parameters for a Wiener process
  t0 = 0;
  n = 10;
  ntimes = nag_int(n);

  % We want to generate the Wiener process at these time points
  intime = 1:n + t0;
  tend   = t0 + n + 1;

  nmove = nag_int(0);
  move  = zeros(nmove, 1, nag_int_name);
  bgord = nag_int(3);

function [npaths,d,start,a,term,c] = get_bridge_gen_data();
  % Set the basic parameters for a free Wiener process
  npaths = nag_int(2);
  d = 3;
  a = nag_int(0);
  start = zeros(d, 1);
  term  = zeros(d, 1);

  % As a=0, term need not be initialized

  % We want the following covariance matrix
  c = [ 6,   1,  -0.2;
        1,   5,   0.3;
       -0.2, 0.3, 4  ];

  % Cholesky factorization of the covariance matrix c
  [c, info] = f07fd('l', c);

function [z] = get_z(npaths, d, a, ntimes)
  idim = d*(ntimes+1-a);
```

```
  % Generate the input pseudorandom points

  % First initialize the base pseudorandom number generator
  state = initialize_prng(nag_int(6), nag_int(0), [nag_int(1023401)]);

  % Generate the pseudorandom points from N(0,1)
  [state, z, ifail] = g05sk( ...
                            nag_int(idim*npaths), 0, 1, state);

  z = reshape(z, idim, npaths);

function [state] = initialize_prng(genid, subid, seed)
  % Initialize the generator to a repeatable sequence
  [state, ifail] = g05kf( ...
                          genid, subid, seed);
```

9.2 Program Results

```
     g05xa example results

Weiner Path 1, 11 time steps, 3 dimensions
             1          2          3
  1      1.6020     0.5611     1.6975
  2      1.2767     0.3972    -1.7199
  3     -0.1895    -0.8812    -5.1908
  4     -2.8083    -4.4484    -6.7697
  5     -5.6251    -6.0375    -3.2551
  6     -6.5404    -6.2009    -5.5638
  7     -4.6398    -4.9675    -7.4454
  8     -5.3501    -4.8563    -9.9002
  9     -7.1683    -7.2638    -9.7825
 10     -1.9440    -7.0725   -10.7577
 11     -4.9941    -3.5442   -10.1561

Weiner Path 2, 11 time steps, 3 dimensions
             1          2          3
  1      2.6097     6.2430     0.0316
  2      3.5442     4.2836     2.5742
  3      1.3068     6.1511     4.5362
  4      2.7487     8.6021     2.6880
  5      3.4584     6.1778    -0.6274
  6      0.5965     8.3014     0.5933
  7     -3.2701     5.4787     1.0727
  8     -4.7527     7.0988     0.9120
  9     -4.9375     7.9486     0.7657
 10     -7.1302     7.3180     0.2706
 11     -0.6289     9.8866    -2.2762
```