

NAG Library Routine Document

G05XAF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details.

1 Purpose

G05XAF initializes the Brownian bridge generator G05XBF. It must be called before any calls to G05XBF.

2 Specification

```
SUBROUTINE G05XAF (TO, TEND, TIMES, NTIMES, RCOMM, IFAIL)
  INTEGER          NTIMES, IFAIL
  REAL (KIND=nag_wp) TO, TEND, TIMES(NTIMES), RCOMM(12*(NTIMES+1))
```

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 $t_0 < t_1 < t_2 < \dots < t_N < T$. 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, \dots, Z^P where each point $Z^p = (Z_1^p, Z_2^p, \dots, Z_D^p)$ has dimension $D = d(N+1)$ or $D = dN$ respectively, depending on whether a free or non-free Wiener process is required. When G05XBF is called, the p th 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, \dots, t_N in the order in which the bridge is to be constructed. For each $j = 1, \dots, N$ set $r = \text{TIMES}(j)$, find

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

and

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

and construct the point X_r as

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

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 routine G05XEF 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 Arguments

- 1: T0 – REAL (KIND=nag_wp) *Input*
On entry: the starting value t_0 of the time interval.
- 2: TEND – REAL (KIND=nag_wp) *Input*
On entry: the end value T of the time interval.
Constraint: TEND > T0.
- 3: TIMES(NTIMES) – REAL (KIND=nag_wp) array *Input*
On entry: 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 routine G05XEF can be used to create predefined bridge construction orders from a set of input times.
Constraints:
 $T0 < \text{TIMES}(i) < \text{TEND}$, for $i = 1, 2, \dots, \text{NTIMES}$;
 $\text{TIMES}(i) \neq \text{TIMES}(j)$, for $i, j = 1, 2, \dots, \text{NTIMES}$ and $i \neq j$.
- 4: NTIMES – INTEGER *Input*
On entry: the length of TIMES, denoted by N in Section 3.1.
Constraint: NTIMES \geq 1.
- 5: RCOMM($12 \times (\text{NTIMES} + 1)$) – REAL (KIND=nag_wp) array *Communication Array*
On exit: communication array, used to store information between calls to G05XBF. This array **must not** be directly modified.
- 6: IFAIL – INTEGER *Input/Output*
On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.
For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then

the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL = 0 unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or -1 , explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL = 1

On entry, TEND = $\langle value \rangle$ and T0 = $\langle value \rangle$.
Constraint: TEND > T0.

IFAIL = 2

On entry, NTIMES = $\langle value \rangle$.
Constraint: NTIMES \geq 1.

IFAIL = 3

On entry, TIMES($\langle value \rangle$) = $\langle value \rangle$, T0 = $\langle value \rangle$ and TEND = $\langle value \rangle$.
Constraint: T0 < TIMES(i) < TEND for all i .

IFAIL = 4

On entry, TIMES($\langle value \rangle$) and TIMES($\langle value \rangle$) both equal $\langle value \rangle$.
Constraint: all elements of TIMES must be unique.

IFAIL = -99

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL = -399

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL = -999

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

G05XAF is not threaded in any implementation.

9 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. G05XAF 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.

10 Example

This example calls G05XAF, G05XBF and G05XEF 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 G05XBF and G05XEF for additional examples.

10.1 Program Text

```

Program g05xafe

!      G05XAF Example Program Text
!
!      Mark 26 Release. NAG Copyright 2016.
!
!      .. Use Statements ..
!      Use nag_library, Only: g05xaf, g05xbf, g05xef, nag_wp
!      .. Implicit None Statement ..
!      Implicit None
!      .. Parameters ..
!      Integer, Parameter          :: nout = 6
!      .. Local Scalars ..
!      Real (Kind=nag_wp)          :: t0, tend
!      Integer                     :: a, bgord, d, ifail, ldb, ldc, ldz, &
!                                     nmove, npaths, ntimes, rcord
!
!      .. Local Arrays ..
!      Real (Kind=nag_wp), Allocatable :: b(:,,:), c(:,,:), intime(:), rcomm(:), &
!                                     start(:), term(:), times(:), z(:,,:)
!      Integer, Allocatable           :: move(:)
!      .. Intrinsic Procedures ..
!      Intrinsic                     :: size
!      .. Executable Statements ..
!      Get information required to set up the bridge
!      Call get_bridge_init_data(bgord,t0,tend,ntimes,intime,nmove,move)
!
!      Make the bridge construction bgord
!      Allocate (times(ntimes))
!      ifail = 0
!      Call g05xef(bgord,t0,tend,ntimes,intime,nmove,move,times,ifail)
!
!      Initialize the Brownian bridge generator
!      Allocate (rcomm(12*(ntimes+1)))
!      ifail = 0
!      Call g05xaf(t0,tend,times,ntimes,rcomm,ifail)
!
!      Get additional information required by the bridge generator
!      Call get_bridge_gen_data(npaths,rcord,d,start,a,term,c)
!
!      Generate the Z values
!      Call get_z(rcord,npaths,d,a,ntimes,z,b)
!
!      Leading dimensions for the various input arrays
!      ldz = size(z,1)
!      ldc = size(c,1)
!      ldb = size(b,1)

```

```

!      Call the Brownian bridge generator routine
      ifail = 0
      Call g05xbf(npaths,rcord,d,start,a,term,z,ldz,c,ldc,b,ldb,rcomm,ifail)

!      Display the results
      Call display_results(rcord,ntimes,d,b)

Contains
  Subroutine get_bridge_init_data(bgord,t0,tend,ntimes,intime,nmove,move)

!      .. Scalar Arguments ..
      Real (Kind=nag_wp), Intent (Out) :: t0, tend
      Integer, Intent (Out)           :: bgord, nmove, ntimes
!      .. Array Arguments ..
      Real (Kind=nag_wp), Allocatable, Intent (Out) :: intime(:)
      Integer, Allocatable, Intent (Out) :: move(:)
!      .. Local Scalars ..
      Integer :: i
!      .. Intrinsic Procedures ..
      Intrinsic :: real
!      .. Executable Statements ..
!      Set the basic parameters for a Wiener process
      ntimes = 10
      t0 = 0.0_nag_wp
      Allocate (intime(ntimes))

!      We want to generate the Wiener process at these time points
      Do i = 1, ntimes
        intime(i) = t0 + real(i,kind=nag_wp)
      End Do
      tend = t0 + real(ntimes+1,kind=nag_wp)

      nmove = 0
      Allocate (move(nmove))
      bgord = 3
End Subroutine get_bridge_init_data

Subroutine get_bridge_gen_data(npaths,rcord,d,start,a,term,c)

!      .. Use Statements ..
      Use nag_library, Only: dpotrf
!      .. Scalar Arguments ..
      Integer, Intent (Out)           :: a, d, npaths, rcord
!      .. Array Arguments ..
      Real (Kind=nag_wp), Allocatable, Intent (Out) :: c(:,,:), start(:), &
                                                term(:)
!      .. Local Scalars ..
      Integer :: info
!      .. Executable Statements ..
!      Set the basic parameters for a free Wiener process
      npaths = 2
      rcord = 1
      d = 3
      a = 0

      Allocate (start(d),term(d),c(d,d))

      start(1:d) = 0.0_nag_wp
!      As A = 0, TERM need not be initialized

!      We want the following covariance matrix
      c(:,1) = (/6.0_nag_wp,1.0_nag_wp,-0.2_nag_wp/)
      c(:,2) = (/1.0_nag_wp,5.0_nag_wp,0.3_nag_wp/)
      c(:,3) = (/ -0.2_nag_wp,0.3_nag_wp,4.0_nag_wp/)

!      G05XBF works with the Cholesky factorization of the covariance matrix
!      C so perform the decomposition
      Call dpotrf('Lower',d,c,d,info)
      If (info/=0) Then
        Write (nout,*)
          'Specified covariance matrix is not positive definite: info=', &

```

```

        info
        Stop
    End If
End Subroutine get_bridge_gen_data

Subroutine get_z(rcord,npaths,d,a,ntimes,z,b)

!      .. Use Statements ..
      Use nag_library, Only: g05skf
!      .. Scalar Arguments ..
      Integer, Intent (In)          :: a, d, npaths, ntimes, rcord
!      .. Array Arguments ..
      Real (Kind=nag_wp), Allocatable, Intent (Out) :: b(:,,:), z(:,,:)
!      .. Local Scalars ..
      Integer                        :: idim, ifail
!      .. Local Arrays ..
      Integer                        :: seed(1)
      Integer, Allocatable           :: state(:)
!      .. Executable Statements ..
      idim = d*(ntimes+1-a)

!      Allocate Z
      If (rcord==1) Then
        Allocate (z(idim,npaths))
        Allocate (b(d*(ntimes+1),npaths))
      Else
        Allocate (z(npaths,idim))
        Allocate (b(npaths,d*(ntimes+1)))
      End If

!      We now need to generate the input pseudorandom points
!      First initialize the base pseudorandom number generator
      seed(1) = 1023401
      Call initialize_prng(6,0,seed,size(seed),state)

!      Generate the pseudorandom points from N(0,1)
      ifail = 0
      Call g05skf(idim*npaths,0.0_nag_wp,1.0_nag_wp,state,z,ifail)
End Subroutine get_z

Subroutine initialize_prng(genid,subid,seed,lseed,state)

!      .. Use Statements ..
      Use nag_library, Only: g05kff
!      .. Scalar Arguments ..
      Integer, Intent (In)          :: genid, lseed, subid
!      .. Array Arguments ..
      Integer, Intent (In)          :: seed(lseed)
      Integer, Allocatable, Intent (Out) :: state(:)
!      .. Local Scalars ..
      Integer                        :: ifail, lstate
!      .. Executable Statements ..

!      Initial call to initializer to get size of STATE array
      lstate = 0
      Allocate (state(lstate))
      ifail = 0
      Call g05kff(genid,subid,seed,lseed,state,lstate,ifail)

!      Reallocate STATE
      Deallocate (state)
      Allocate (state(lstate))

!      Initialize the generator to a repeatable sequence
      ifail = 0
      Call g05kff(genid,subid,seed,lseed,state,lstate,ifail)
End Subroutine initialize_prng

Subroutine display_results(rcord,ntimes,d,b)

!      .. Scalar Arguments ..

```

```

Integer, Intent (In)          :: d, ntimes, rcord
! .. Array Arguments ..
Real (Kind=nag_wp), Intent (In) :: b(:, :)
! .. Local Scalars ..
Integer          :: i, j, k
! .. Executable Statements ..
Write (nout,*) 'G05XAF Example Program Results'
Write (nout,*)

Do i = 1, npaths
  Write (nout,99999) 'Weiner Path ', i, ', ', ntimes + 1,      &
    ' time steps, ', d, ' dimensions'
  Write (nout,99997)(j,j=1,d)
  k = 1
  Do j = 1, ntimes + 1
    If (rcord==1) Then
      Write (nout,99998) j, b(k:k+d-1,i)
    Else
      Write (nout,99998) j, b(i,k:k+d-1)
    End If
    k = k + d
  End Do
  Write (nout,*)
End Do
99999 Format (1X,A,I0,A,I0,A,I0,A)
99998 Format (1X,I2,1X,20(1X,F10.4))
99997 Format (1X,3X,20(9X,I2))
End Subroutine display_results
End Program g05xafe

```

10.2 Program Data

None.

10.3 Program Results

G05XAF Example Program 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