NAG Library Routine Document

G05XBF

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

G05XBF uses a Brownian bridge algorithm to construct sample paths for a free or non-free Wiener process. The initialization routine G05XAF must be called prior to the first call to G05XBF.

2 Specification

```
SUBROUTINE G05XBF (NPATHS, RCORD, D, START, A, TERM, Z, LDZ, C, LDC, B,          &
                   LDB, RCOMM, IFAIL)

INTEGER            NPATHS, RCORD, D, A, LDZ, LDC, LDB, IFAIL
REAL (KIND=nag_wp) START(D), TERM(D), Z(LDZ,*), C(LDC,*), B(LDB,*),             &
                   RCOMM(*)
```

3 Description

For details on the Brownian bridge algorithm and the bridge construction order see Section 2.6 in the G05 Chapter Introduction and Section 3 in G05XAF. Recall that the terms Wiener process (or free Wiener process) and Brownian motion are often used interchangeably, while a non-free Wiener process (also known as a Brownian bridge process) refers to a process which is forced to terminate at a given point.

4 References

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

5 Parameters

Note: the following variable is used in the parameter descriptions: $N =$ NTIMES, the length of the array TIMES passed to the initialization routine G05XAF.

1: NPATHS – INTEGER *Input*

On entry: the number of Wiener sample paths to create.

Constraint: NPATHS ≥ 1.

2: RCORD – INTEGER *Input*

On entry: the order in which Normal random numbers are stored in Z and in which the generated values are returned in B.

Constraint: RCORD $= 1$ or 2.

3: D – INTEGER *Input*

On entry: the dimension of each Wiener sample path.

Constraint: D ≥ 1.

4: START(D) – REAL (KIND=nag_wp) array *Input*

On entry: the starting value of the Wiener process.

5: A – INTEGER *Input*

On entry: if A $= 0$, a free Wiener process is created beginning at START and TERM is ignored.

If A $= 1$, a non-free Wiener process is created beginning at START and ending at TERM.

Constraint: A $= 0$ or 1.

6: TERM(D) – REAL (KIND=nag_wp) array *Input*

On entry: the terminal value at which the non-free Wiener process should end. If A $= 0$, TERM is ignored.

7: Z(LDZ,$*$) – REAL (KIND=nag_wp) array *Input/Output*

Note: the second dimension of the array Z must be at least NPATHS if RCORD $= 1$ and at least D $\times$ $(N + 1 -$ A$)$ if RCORD $= 2$.

On entry: the Normal random numbers used to construct the sample paths.

If RCORD $= 1$ and quasi-random numbers are used, the D $\times$ $(N + 1 -$ A$)$, where $N =$ nint RCOMM(2)-dimensional quasi-random points should be stored in successive columns of Z.

If RCORD $= 2$ and quasi-random numbers are used, the D $\times$ $(N + 1 -$ A$)$, where $N =$ nint RCOMM(2)-dimensional quasi-random points should be stored in successive rows of Z.

On exit: the Normal random numbers premultiplied by C.

8: LDZ – INTEGER *Input*

On entry: the first dimension of the array Z as declared in the (sub)program from which G05XBF is called.

Constraints:

if RCORD $= 1$, LDZ $\geq$ D $\times$ $(N + 1 -$ A$)$;
if RCORD $= 2$, LDZ $\geq$ NPATHS.

9: C(LDC,$*$) – REAL (KIND=nag_wp) array *Input*

Note: the second dimension of the array C must be at least D.

On entry: the lower triangular Cholesky factorization C such that CC^{T} gives the covariance matrix of the Wiener process. Elements of C above the diagonal are not referenced.

10: LDC – INTEGER *Input*

On entry: the first dimension of the array C as declared in the (sub)program from which G05XBF is called.

Constraint: LDC $\geq$ D.

11: B(LDB,$*$) – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array B must be at least NPATHS if RCORD $= 1$ and at least D $\times$ $(N + 1)$ if RCORD $= 2$.

On exit: the values of the Wiener sample paths.

Let $X_{p,i}^{k}$ denote the kth dimension of the ith point of the pth sample path where $1 \leq k \leq$ D, $1 \leq i \leq N + 1$ and $1 \leq p \leq$ NPATHS.

If RCORD $= 1$, the point $X_{p,i}^{k}$ will be stored at B$(k + (i - 1) \times$ D$, p)$.

If RCORD $= 2$, the point $X_{p,i}^{k}$ will be stored at B$(p, k + (i - 1) \times$ D$)$.

The starting value START is never stored, whereas the terminal value is always stored.

12: LDB – INTEGER *Input*

On entry: the first dimension of the array B as declared in the (sub)program from which G05XBF is called.

Constraints:

if RCORD = 1, LDB $\geq$ D $\times$ $(N + 1)$;
if RCORD = 2, LDB $\geq$ NPATHS.

13: RCOMM$(*)$ – REAL (KIND=nag_wp) array *Communication Array*

On entry: communication array as returned by the last call to G05XAF or G05XBF. This array must not be directly modified.

14: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction 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 parameter, 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, RCOMM was not initialized or has been corrupted. On entry, RCOMM was not initialized or has been corrupted. On entry, RCOMM was not initialized or has been corrupted.

IFAIL = 2

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

IFAIL = 3

On entry, the value of RCORD is invalid.

IFAIL = 4

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

IFAIL = 5

On entry, A = $\langle value \rangle$.
Constraint: A = 0 or 1.

IFAIL = 6

On entry, LDZ = $\langle value \rangle$ and D $\times$ (NTIMES + 1 $-$ A) = $\langle value \rangle$.
Constraint: LDZ $\geq$ D $\times$ (NTIMES + 1 $-$ A).

On entry, LDZ = ⟨*value*⟩ and NPATHS = ⟨*value*⟩.
Constraint: LDZ ≥ NPATHS.

IFAIL = 7

On entry, LDC = ⟨*value*⟩.
Constraint: LDC ≥ ⟨*value*⟩.

IFAIL = 8

On entry, LDB = ⟨*value*⟩ and D × (NTIMES + 1) = ⟨*value*⟩.
Constraint: LDB ≥ D × (NTIMES + 1).

On entry, LDB = ⟨*value*⟩ and NPATHS = ⟨*value*⟩.
Constraint: LDB ≥ NPATHS.

IFAIL = −999

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

This example calls G05XBF, G05XAF and G05XEF to generate two sample paths of a three dimensional non-free Wiener process. The process starts at zero and each sample path terminates at the point $(1.0, 0.5, 0.0)$. Quasi-random numbers are used to construct the sample paths.

See Section 9 in G05XAF and G05XEF for additional examples.

9.1 Program Text

```
    Program g05xbfe

!     G05XBF Example Program Text

!     Mark 24 Release. NAG Copyright 2012.

!     .. 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 and allocate B
      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 non-free Wiener process
```

```
        npaths = 2
        rcord = 2
        d = 3
        a = 1

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

        start(1:d) = 0.0_nag_wp
        term(1:d) = (/1.0_nag_wp,0.5_nag_wp,0.0_nag_wp/)

!       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: g05yjf
!       .. 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 ..
        Real (Kind=nag_wp), Allocatable     :: std(:), tz(:,:), xmean(:)
        Integer, Allocatable                :: iref(:), state(:)
        Integer                             :: seed(1)
!       .. Intrinsic Procedures ..
        Intrinsic                           :: transpose
!       .. 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 quasi-random points
!       First initialize the base pseudorandom number generator
        seed(1) = 1023401
        Call initialize_prng(6,0,seed,state)

!       Scrambled quasi-random sequences preserve the good discrepancy
!       properties of quasi-random sequences while counteracting the bias
!       some applications experience when using quasi-random sequences.
!       Initialize the scrambled quasi-random generator.
        Call initialize_scrambled_qrng(1,2,idim,state,iref)

!       Generate the quasi-random points from N(0,1)
        Allocate (xmean(idim),std(idim))
        xmean(1:idim) = 0.0_nag_wp
        std(1:idim) = 1.0_nag_wp
        If (rcord==1) Then
          Allocate (tz(npaths,idim))
```

```
          ifail = 0
          Call g05yjf(xmean,std,npaths,tz,iref,ifail)
          z(:,:) = transpose(tz)
        Else
          ifail = 0
          Call g05yjf(xmean,std,npaths,z,iref,ifail)
        End If
      End Subroutine get_z

      Subroutine initialize_prng(genid,subid,seed,state)

!       .. Use Statements ..
        Use nag_library, Only: g05kff
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: genid, subid
!       .. Array Arguments ..
        Integer, Intent (In)                :: seed(:)
        Integer, Allocatable, Intent (Out)  :: state(:)
!       .. Local Scalars ..
        Integer                             :: ifail, lseed, lstate
!       .. Executable Statements ..
        lseed = size(seed,1)

!       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 initialize_scrambled_qrng(genid,stype,idim,state,iref)

!       .. Use Statements ..
        Use nag_library, Only: g05ynf
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: genid, idim, stype
!       .. Array Arguments ..
        Integer, Allocatable, Intent (Out)  :: iref(:)
        Integer, Intent (Inout)             :: state(:)
!       .. Local Scalars ..
        Integer                             :: ifail, iskip, liref, nsdigits
!       .. Executable Statements ..
        liref = 32*idim + 7
        iskip = 0
        nsdigits = 32
        Allocate (iref(liref))
        ifail = 0
        Call g05ynf(genid,stype,idim,iref,liref,iskip,nsdigits,state,ifail)
      End Subroutine initialize_scrambled_qrng

      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,*) 'G05XBF 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 g05xbfe
```

9.2 Program Data

None.

9.3 Program Results

```
 G05XBF Example Program Results

 Weiner Path 1, 11 time steps, 3 dimensions
              1          2          3
  1     -1.0602    -2.8701    -0.9415
  2     -3.0575    -1.9502     0.2596
  3     -6.8274    -2.4434     0.4597
  4     -5.2855    -3.4475     0.0795
  5     -8.1784    -5.2296    -0.0921
  6     -4.6874    -5.0220     1.4862
  7     -3.0959    -4.8623    -4.4076
  8     -2.9605    -1.8936    -3.9539
  9     -5.4685    -2.3856    -3.2031
 10      0.1205    -5.0520    -1.0385
 11      1.0000     0.5000     0.0000

 Weiner Path 2, 11 time steps, 3 dimensions
              1          2          3
  1      0.6564     3.5142     1.5911
  2     -2.3773     3.1618     3.0316
  3      0.3020     6.8815     2.0875
  4     -0.2169     4.6026     1.1982
  5     -2.0684     4.1503     2.4758
  6     -5.1075     3.7303     2.7563
  7     -3.8497     3.6682     2.4827
  8     -1.8292     4.4153     0.1916
  9     -2.0649     0.6952    -2.1201
 10      0.1962     1.7769    -5.7685
 11      1.0000     0.5000     0.0000
```