NAG Library Routine Document

G05XDF

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

G05XDF computes scaled increments of sample paths of a free or non-free Wiener process, where the sample paths are constructed by a Brownian bridge algorithm. The initialization routine G05XCF must be called prior to the first call to G05XDF.

2 Specification

```
SUBROUTINE G05XDF (NPATHS, RCORD, D, A, DIFF, Z, LDZ, C, LDC, B, LDB,      &
                   RCOMM, IFAIL)
INTEGER            NPATHS, RCORD, D, A, LDZ, LDC, LDB, IFAIL
REAL (KIND=nag_wp) DIFF(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 G05XCF. 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.

Fix two times $t_0 < T$, let $(t_i)_{1 \leq i \leq N}$ be any set of time points satisfying $t_0 < t_1 < t_2 < \cdots < t_N < T$, and let X_{t_0}, $\left(X_{t_i}\right)_{1 \leq i \leq N}$, X_T denote a d-dimensional Wiener sample path at these time points.

The Brownian bridge increments generator uses the Brownian bridge algorithm to construct sample paths for the (free or non-free) Wiener process X, and then uses this to compute the *scaled Wiener increments*

$$\frac{X_{t_1} - X_{t_0}}{t_1 - t_0}, \frac{X_{t_2} - X_{t_1}}{t_2 - t_1}, \ldots, \frac{X_{t_N} - X_{t_{N-1}}}{t_N - t_{N-1}}, \frac{X_T - X_{t_N}}{T - t_N}$$

The example program in Section 9 shows how these increments can be used to compute a numerical solution to a stochastic differential equation (SDE) driven by a (free or non-free) Wiener process.

4 References

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

5 Parameters

Note: the following variable is used in the parameter descriptions: $N = \text{NTIMES}$, the length of the array TIMES passed to the initialization routine G05XCF.

1: NPATHS – INTEGER *Input*

On entry: the number of Wiener sample paths.

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: $\text{D} \geq 1$.

4: A – INTEGER *Input*

On entry: if $\text{A} = 0$, a free Wiener process is created and DIFF is ignored.

If $\text{A} = 1$, a non-free Wiener process is created where DIFF is the difference between the terminal value and the starting value of the process.

Constraint: $\text{A} = 0$ or 1.

5: DIFF(D) – REAL (KIND=nag_wp) array *Input*

On entry: the difference between the terminal value and starting value of the Wiener process. If $\text{A} = 0$, DIFF is ignored.

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

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

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

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

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

On exit: the Normal random numbers premultiplied by C.

7: LDZ – INTEGER *Input*

On entry: the first dimension of the array Z as declared in the (sub)program from which G05XDF is called.

Constraints:

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

8: 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.

9: LDC – INTEGER *Input*

On entry: the first dimension of the array C as declared in the (sub)program from which G05XDF is called.

Constraint: $\text{LDC} \geq \text{D}$.

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

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

On exit: the scaled Wiener increments.

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

If RCORD $= 1$, the increment $\dfrac{\left(X_{p,i}^{k} - X_{p,i-1}^{k}\right)}{(t_i - t_{i-1})}$ will be stored at $\mathrm{B}(k + (i-1) \times \mathrm{D}, p)$.

If RCORD $= 2$, the increment $\dfrac{\left(X_{p,i}^{k} - X_{p,i-1}^{k}\right)}{(t_i - t_{i-1})}$ will be stored at $\mathrm{B}(p, k + (i-1) \times \mathrm{D})$.

11: LDB – INTEGER *Input*

On entry: the first dimension of the array B as declared in the (sub)program from which G05XDF is called.

Constraints:

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

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

On entry: communication array as returned by the last call to G05XCF or G05XDF. This array **must not** be directly modified.

13: 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 ≥ 1.

IFAIL $= 3$

On entry, the value of RCORD is invalid.

IFAIL $= 4$

On entry, D $= \langle$*value*$\rangle$.
Constraint: D ≥ 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 $= \langle value \rangle$ and NPATHS $= \langle value \rangle$.
Constraint: LDZ $\geq$ NPATHS.

IFAIL $= 7$

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

IFAIL $= 8$

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

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

IFAIL $= -999$

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

The scaled Wiener increments produced by this routine can be used to compute numerical solutions to stochastic differential equations (SDEs) driven by (free or non-free) Wiener processes. Consider an SDE of the form

$$dY_t = f(t, Y_t)\mathrm{dt} + \sigma(t, Y_t)dX_t$$

on the interval $[t_0, T]$ where $(X_t)_{t_0 \leq t \leq T}$ is a (free or non-free) Wiener process and f and σ are suitable functions. A numerical solution to this SDE can be obtained by the Euler–Maruyama method. For any discretization $t_0 < t_1 < t_2 < \cdots < t_{N+1} = T$ of $[t_0, T]$, set

$$Y_{t_{i+1}} = Y_{t_i} + f\big(t_i, Y_{t_i}\big)(t_{i+1} - t_i) + \sigma\big(t_i, Y_{t_i}\big)\big(X_{t_{i+1}} - X_{t_i}\big)$$

for $i = 1, \ldots, N$ so that $Y_{t_{N+1}}$ is an approximation to Y_T. The scaled Wiener increments produced by G05XDF can be used in the Euler–Maruyama scheme outlined above by writing

$$Y_{t_{i+1}} = Y_{t_i} + (t_{i+1} - t_i)\left(f\big(t_i, Y_{t_i}\big) + \sigma\big(t_i, Y_{t_i}\big)\left(\frac{X_{t_{i+1}} - X_{t_i}}{t_{i+1} - t_i}\right)\right).$$

The following example program uses this method to solve the SDE for geometric Brownian motion

$$dS_t = rS_t\mathrm{dt} + \sigma S_t dX_t$$

where X is a Wiener process, and compares the results against the analytic solution

$$S_T = S_0 \exp\big(\big(r - \sigma^2/2\big)T + \sigma X_T\big).$$

Quasi-random variates are used to construct the Wiener increments.

9.1 Program Text

```
    Program g05xdfe

!     G05XDF Example Program Text

!     Mark 24 Release. NAG Copyright 2012.

!     .. Use Statements ..
      Use nag_library, Only: g05xcf, g05xdf, g05xef, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                  :: a = 0, d = 1, nout = 6, rcord = 2
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: r, s0, sigma, t0, tend
      Integer                             :: bgord, i, ifail, ldb, ldz,       &
                                             nmove, npaths, ntimesteps, p
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: analytic(:), b(:,:), rcomm(:),   &
                                             st(:,:), t(:), times(:), z(:,:)
      Real (Kind=nag_wp)                  :: c(d) = (/1.0_nag_wp/)
      Real (Kind=nag_wp)                  :: diff(d) = (/0.0_nag_wp/)
      Integer, Allocatable                :: move(:)
!     .. Intrinsic Procedures ..
      Intrinsic                           :: exp, real, size
!     .. Executable Statements ..
      ifail = 0

!     We wish to solve the stochastic differential equation (SDE)
!       dSt = r*St*dt + sigma*St*dXt
!     where X is a one dimensional Wiener process.
!     This means we have
!         A = 0
!         D = 1
!         C = 1
!     We now set the other parameters of the SDE and the Euler-Maruyama scheme

!     Initial value of the process
      s0 = 1.0_nag_wp
      r = 0.05_nag_wp
      sigma = 0.12_nag_wp
!     Number of paths to simulate
      npaths = 3
!     The time interval [t0,T] on which to solve the SDE
      t0 = 0.0_nag_wp
      tend = 1.0_nag_wp
!     The time steps in the discretization of [t0,T]
      ntimesteps = 20
      Allocate (t(ntimesteps))
      Do i = 1, ntimesteps
        t(i) = t0 + i*(tend-t0)/real(ntimesteps+1,kind=nag_wp)
      End Do

!     Make the bridge construction order
      nmove = 0
      Allocate (times(ntimesteps),move(nmove))
      bgord = 3
      Call g05xef(bgord,t0,tend,ntimesteps,t,nmove,move,times,ifail)

!     Generate the input Z values and allocate memory for b
      Call get_z(rcord,npaths,d,a,ntimesteps,z,b)
!     Leading dimensions for the various input arrays
      ldz = size(z,1)
      ldb = size(b,1)

!     Initialize the generator
      Allocate (rcomm(12*(ntimesteps+1)))
      Call g05xcf(t0,tend,times,ntimesteps,rcomm,ifail)

!     Get the scaled increments of the Wiener process
```

```
      Call g05xdf(npaths,rcord,d,a,diff,z,ldz,c,d,b,ldb,rcomm,ifail)


!     Do the Euler-Maruyama time stepping
      Allocate (st(npaths,ntimesteps+1),analytic(npaths))

!     Do first time step
      st(:,1) = s0 + (t(1)-t0)*(r*s0+sigma*s0*b(:,1))
      Do i = 2, ntimesteps
        Do p = 1, npaths
          st(p,i) = st(p,i-1) + (t(i)-t(i-1))*(r*st(p,i-1)+sigma*st(p,i-1)*b(p &
            ,i))
        End Do
      End Do
!     Do last time step
      st(:,i) = st(:,i-1) + (tend-t(i-1))*(r*st(:,i-1)+sigma*st(:,i-1)*b(:,i))

!     Compute the analytic solution:
!        ST = S0*exp( (r-sigma**2/2)T + sigma WT )

      analytic(:) = s0*exp((r-0.5_nag_wp*sigma*sigma)*tend+sigma*(tend-t0)*z(: &
        ,1))

!     Display the results
      Call display_results(ntimesteps,npaths,st,analytic)

    Contains

      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(ntimesteps,npaths,st,analytic)

!       .. Scalar Arguments ..
        Integer, Intent (In)                  :: npaths, ntimesteps
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)       :: analytic(:), st(:,:)
!       .. Local Scalars ..
        Integer                               :: i, p
!       .. Executable Statements ..
        Write (nout,*) 'G05XDF Example Program Results'
        Write (nout,*)
```

```
        Write (nout,*) 'Euler-Maruyama solution for Geometric Brownian motion'

        Write (nout,99999)('Path',p,p=1,npaths)
        Do i = 1, ntimesteps + 1
          Write (nout,99998) i, st(:,i)
        End Do
        Write (nout,*)

        Write (nout,'(A)') 'Analytic solution at final time step'
        Write (nout,99999)('Path',p,p=1,npaths)
        Write (nout,'(4X,20(1X,F10.4))') analytic(:)

99999   Format (4X,20(5X,A,I2))
99998   Format (1X,I2,1X,20(1X,F10.4))
      End Subroutine display_results
    End Program g05xdfe
```

9.2 Program Data

None.

9.3 Program Results

```
 G05XDF Example Program Results

 Euler-Maruyama solution for Geometric Brownian motion
         Path 1     Path 2     Path 3
  1      0.9668     1.0367     0.9992
  2      0.9717     1.0254     1.0077
  3      0.9954     1.0333     1.0098
  4      0.9486     1.0226     0.9911
  5      0.9270     1.0113     1.0630
  6      0.8997     1.0127     1.0164
  7      0.8955     1.0138     1.0771
  8      0.8953     0.9953     1.0691
  9      0.8489     1.0462     1.0484
 10      0.8449     1.0592     1.0429
 11      0.8158     1.0233     1.0625
 12      0.7997     1.0384     1.0729
 13      0.8025     1.0138     1.0725
 14      0.8187     1.0499     1.0554
 15      0.8270     1.0459     1.0529
 16      0.7914     1.0294     1.0783
 17      0.8076     1.0224     1.0943
 18      0.8208     1.0359     1.0773
 19      0.8190     1.0326     1.0857
 20      0.8217     1.0326     1.1095
 21      0.8084     0.9695     1.1389

Analytic solution at final time step
         Path 1     Path 2     Path 3
         0.8079     0.9685     1.1389
```