NAG Library Routine Document

G05YNF

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

G05YNF initializes a scrambled quasi-random generator prior to calling G05YJF, G05YKF or G05YMF. It must be preceded by a call to one of the pseudorandom initialization routines G05KFF or G05KGF.

2 Specification

```
SUBROUTINE G05YNF (GENID, STYPE, IDIM, IREF, LIREF, ISKIP, NSDIGI, STATE,      &
                   IFAIL)

INTEGER GENID, STYPE, IDIM, IREF(LIREF), LIREF, ISKIP, NSDIGI, STATE(*),       &
        IFAIL
```

3 Description

G05YNF selects a quasi-random number generator through the input value of GENID, a method of scrambling through the input value of STYPE and initializes the IREF communication array for use in the routines G05YJF, G05YKF or G05YMF.

Scrambled quasi-random sequences are an extension of standard quasi-random sequences that attempt to eliminate the bias inherent in a quasi-random sequence whilst retaining the low-discrepancy properties. The use of a scrambled sequence allows error estimation of Monte–Carlo results by performing a number of iterates and computing the variance of the results.

This implementation of scrambled quasi-random sequences is based on TOMS Algorithm 823 and details can be found in the accompanying paper, Hong and Hickernell (2003). Three methods of scrambling are supplied; the first a restricted form of Owen's scrambling (Owen (1995)), the second based on the method of Faure and Tezuka (2000) and the last method combines the first two.

Scrambled versions of the Niederreiter sequence and two sets of Sobol sequences are provided. The first Sobol sequence is obtained using GENID $= 1$. The first 10000 direction numbers for this sequence are based on the work of Joe and Kuo (2008). For dimensions greater than 10000 the direction numbers are randomly generated using the pseudorandom generator specified in STATE (see Jäckel (2002) for details). The second Sobol sequence is obtained using GENID $= 2$ and referred to in the documentation as 'Sobol (A659)'. The first 1111 direction numbers for this sequence are based on Algorithm 659 of Bratley and Fox (1988) with the extension proposed by Joe and Kuo (2003). For dimensions greater than 1111 the direction numbers are once again randomly generated. The Niederreiter sequence is obtained by setting GENID $= 3$.

4 References

Bratley P and Fox B L (1988) Algorithm 659: implementing Sobol's quasirandom sequence generator *ACM Trans. Math. Software* **14(1)** 88–100

Faure H and Tezuka S (2000) Another random scrambling of digital (t,s)-sequences *Monte Carlo and Quasi-Monte Carlo Methods* Springer-Verlag, Berlin, Germany (eds K T Fang, F J Hickernell and H Niederreiter)

Hong H S and Hickernell F J (2003) Algorithm 823: implementing scrambled digital sequences *ACM Trans. Math. Software* **29:2** 95–109

Jäckel P (2002) *Monte Carlo Methods in Finance* Wiley Finance Series, John Wiley and Sons, England

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

Niederreiter H (1988) Low-discrepancy and low dispersion sequences *Journal of Number Theory* **30** 51–70

Owen A B (1995) Randomly permuted (t,m,s)-nets and (t,s)-sequences *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Lecture Notes in Statistics* **106** Springer-Verlag, New York, NY 299–317 (eds H Niederreiter and P J-S Shiue)

5 Parameters

1: GENID – INTEGER *Input*

On entry: must identify the quasi-random generator to use.

GENID = 1
Sobol generator.

GENID = 2
Sobol (A659) generator.

GENID = 3
Niederreiter generator.

Constraint: GENID = 1, 2 or 3.

2: STYPE – INTEGER *Input*

On entry: must identify the scrambling method to use.

STYPE = 0
No scrambling. This is equivalent to calling G05YLF.

STYPE = 1
Owen like scrambling.

STYPE = 2
Faure–Tezuka scrambling.

STYPE = 3
Owen and Faure–Tezuka scrambling.

Constraint: STYPE = 0, 1, 2 or 3.

3: IDIM – INTEGER *Input*

On entry: the number of dimensions required.

Constraints:

if GENID = 1, $1 \leq$ IDIM ≤ 50000;
if GENID = 2, $1 \leq$ IDIM ≤ 50000;
if GENID = 3, $1 \leq$ IDIM ≤ 318.

4: IREF(LIREF) – INTEGER array *Communication Array*

On exit: contains initialization information for use by the generator routines G05YJF, G05YKF and G05YMF. IREF must not be altered in any way between initialization and calls of the generator routines.

5: LIREF – INTEGER *Input*

On entry: the dimension of the array IREF as declared in the (sub)program from which G05YNF is called.

Constraint: LIREF $\geq 32 \times$ IDIM $+ 7$.

6: ISKIP – INTEGER *Input*

On entry: the number of terms of the sequence to skip on initialization for the Sobol and Niederreiter generators.

Constraint: $0 \leq$ ISKIP $\leq 2^{30}$.

7: NSDIGI – INTEGER *Input*

On entry: controls the number of digits (bits) to scramble when GENID $= 1$ or 2, otherwise NSDIGI is ignored. If NSDIGI < 1 or NSDIGI > 30 then all the digits are scrambled.

8: STATE$(*)$ – INTEGER array *Communication Array*

Note: the actual argument supplied must be the array STATE supplied to the initialization routines G05KFF or G05KGF.

On entry: contains information on the selected base generator and its current state.

On exit: contains updated information on the state of the generator.

9: 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

IFAIL $= 1$

On entry, GENID $\neq 1$, 2 or 3.

IFAIL $= 2$

STYPE $\neq 0$, 1, 2 or 3.

IFAIL $= 3$

On entry, IDIM < 1,
or IDIM is too large.

IFAIL $= 5$

On entry, LIREF is too small.

IFAIL $= 6$

The value of ISKIP < 0 or ISKIP is too large.

IFAIL = 8

On entry, STATE vector was not initialized or has been corrupted.

7 Accuracy

Not applicable.

8 Further Comments

The additional computational cost in using a scrambled quasi-random sequence over a non-scrambled one comes entirely during the initialization. Once G05YNF has been called the computational cost of generating a scrambled sequence and a non-scrambled one is identical.

9 Example

This example calls G05KFF, G05YMF and G05YNF to estimate the value of the integral

$$\int_0^1 \cdots \int_0^1 \prod_{i=1}^{s} |4x_i - 2| dx_1, dx_2, \ldots, dx_s = 1,$$

where s, the number of dimensions, is set to 8.

9.1 Program Text

```
!   G05YNF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module g05ynfe_mod

!     G05YNF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                  :: lseed = 1, nin = 5, nout = 6
    Contains
      Function ifun(x,lx)
!       Function being integrated, in this case
!       ABS(4.0 X - 2)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: ifun
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: lx
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: x(lx)
!       .. Local Scalars ..
        Integer                             :: d
!       .. Intrinsic Procedures ..
        Intrinsic                           :: abs
!       .. Executable Statements ..
        ifun = 1.0E0_nag_wp
        Do d = 1, lx
          ifun = ifun*abs(4.0E0_nag_wp*x(d)-2.0E0_nag_wp)
        End Do
      End Function ifun
    End Module g05ynfe_mod
    Program g05ynfe

!     G05YNF Example Main Program

!     .. Use Statements ..
```

```
      Use nag_library, Only: g05kff, g05ymf, g05ynf, nag_wp, x04caf
      Use g05ynfe_mod, Only: ifun, lseed, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: sum, vsbl
      Integer                             :: dn, genid, i, idim, ifail,       &
                                             iskip, ldquas, liref, lstate, n, &
                                             nsdigi, pgenid, psubid, rcord,   &
                                             stype
      Character (80)                      :: title
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: quas(:,:)
      Integer, Allocatable                :: iref(:), state(:)
      Integer                             :: seed(lseed)
!     .. Intrinsic Procedures ..
      Intrinsic                           :: real
!     .. Executable Statements ..
      Write (nout,*) 'G05YNF Example Program Results'
      Write (nout,*)

!     Skip heading in data file
      Read (nin,*)

!     Read in the base generator information and seed
      Read (nin,*) pgenid, psubid, seed(1)

!     Initial call to initialiser to get size of STATE array
      lstate = 0
      Allocate (state(lstate))
      ifail = 0
      Call g05kff(pgenid,psubid,seed,lseed,state,lstate,ifail)

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

!     Initialize the generator to a repeatable sequence
      ifail = 0
      Call g05kff(pgenid,psubid,seed,lseed,state,lstate,ifail)

!     Fix the RCORD = 1, so QUAS(IDIM,N). As we
!     are accessing each dimension in turn for a given variate
!     when evaluating the function, this is more efficient
      rcord = 1

!     Read in quasi-random generator and scrambling to use
      Read (nin,*) genid, stype, nsdigi

!     Read in problem size
      Read (nin,*) n, idim, iskip

      If (genid==4) Then
        liref = 407
      Else
        liref = 32*idim + 7
      End If
      ldquas = idim
      Allocate (quas(ldquas,n),iref(liref))

!     Call the initializer for the quasi-random sequence
      ifail = 0
      Call g05ynf(genid,stype,idim,iref,liref,iskip,nsdigi,state,ifail)

!     Generate N quasi-random variates
      ifail = 0
      Call g05ymf(n,rcord,quas,ldquas,iref,ifail)

!     Evaluate the function, and sum
      sum = 0.0E0_nag_wp
      Do i = 1, n
```

```
        sum = sum + ifun(quas(1:idim,i),idim)
      End Do

!     Convert sum to mean value
      vsbl = sum/real(n,kind=nag_wp)
      Write (nout,99999)
      Write (nout,99999) 'Value of integral = ', vsbl

!     Read in number of variates to display
      Read (nin,*) dn

!     Display the first DN variates
      Write (nout,*)
      Write (title,99998) 'First ', dn, ' variates for all ', idim, &
        ' dimensions'
      Flush (nout)
      ifail = 0
      Call x04caf('General',' ',idim,dn,quas,ldquas,title,ifail)

99999 Format (1X,A,F8.4)
99998 Format (A,I0,A,I0,A)
    End Program g05ynfe
```

9.2 Program Data

```
G05YNF Example Program Data
1  1  1762543     :: PGENID,PSUBID,SEED(1)
1  1  0           :: GENID,STYPE,NSDIGI
200  8  1000      :: N,IDIM,ISKIP
5                :: DN
```

9.3 Program Results

```
 G05YNF Example Program Results


 Value of integral =   1.0169

 First 5 variates for all 8 dimensions
          1       2       3       4       5
 1   0.8688  0.6287  0.1244  0.1353  0.6154
 2   0.9719  0.3611  0.5349  0.4013  0.6962
 3   0.5375  0.4963  0.8645  0.6656  0.0321
 4   0.0876  0.8648  0.2621  0.4691  0.9000
 5   0.4721  0.0753  0.7523  0.9096  0.2307
 6   0.3800  0.0174  0.7212  0.9272  0.3186
 7   0.2977  0.7011  0.0538  0.5481  0.1989
 8   0.1010  0.2532  0.6231  0.4164  0.7102
```