NAG Library Routine Document

G05KJF

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

G05KJF allows for the generation of multiple, independent, sequences of pseudorandom numbers using the skip-ahead method.

The base pseudorandom number sequence defined by STATE is advanced n places.

2 Specification

```
SUBROUTINE G05KJF (N, STATE, IFAIL)
INTEGER N, STATE(*), IFAIL
```

3 Description

G05KJF adjusts a base generator to allow multiple, independent, sequences of pseudorandom numbers to be generated via the skip-ahead method (see the G05 Chapter Introduction for details).

If, prior to calling G05KJF the base generator defined by STATE would produce random numbers $x_1, x_2, x_3, \ldots$, then after calling G05KJF the generator will produce random numbers $x_{n+1}, x_{n+2}, x_{n+3}, \ldots$.

One of the initialization routines G05KFF (for a repeatable sequence if computed sequentially) or G05KGF (for a non-repeatable sequence) must be called prior to the first call to G05KJF.

The skip-ahead algorithm can be used in conjunction with any of the six base generators discussed in Chapter G05.

4 References

Haramoto H, Matsumoto M, Nishimura T, Panneton F and L'Ecuyer P (2008) Efficient jump ahead for F2-linear random number generators *INFORMS J. on Computing* **20(3)** 385–390

Knuth D E (1981) *The Art of Computer Programming (Volume 2)* (2nd Edition) Addison–Wesley

5 Parameters

1: N – INTEGER *Input*

On entry: n, the number of places to skip ahead.

Constraint: N ≥ 0.

2: 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.

3: 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, N $= \langle value \rangle$.
Constraint: N ≥ 0.

IFAIL $= 2$

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

IFAIL $= 3$

On entry, cannot use skip-ahead with the base generator defined by STATE.

IFAIL $= 4$

On entry, the base generator is Mersenne Twister, but the STATE vector defined on initialization is not large enough to perform a skip ahead. See the initialization routine G05KFF or G05KGF.

IFAIL $= -99$

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

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

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

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

Calling G05KJF and then generating a series of uniform values using G05SAF is more efficient than, but equivalent to, calling G05SAF and discarding the first n values. This may not be the case for distributions other than the uniform, as some distributional generators require more than one uniform variate to generate a single draw from the required distribution.

To skip ahead $k \times m$ places you can either

(a) call G05KJF once with N $= k \times m$, or

(b) call G05KJF k times with N $= m$, using the STATE vector output by the previous call as input to the next call

both approaches would result in the same sequence of values. When working in a multithreaded environment, where you want to generate (at most) m values on each of K threads, this would translate into either

(a) spawning the K threads and calling G05KJF once on each thread with N $= (k-1) \times m$, where k is a thread ID, taking a value between 1 and K, or

(b) calling G05KJF on a single thread with N $= m$, spawning the K threads and then calling G05KJF a further $k-1$ times on each of the thread.

Due to the way skip ahead is implemented for the Mersenne Twister, approach (a) will tend to be more efficient if more than 30 threads are being used (i.e., $K > 30$), otherwise approach (b) should probably be used. For all other base generators, approach (a) should be used. See the G05 Chapter Introduction for more details.

10 Example

This example initializes a base generator using G05KFF and then uses G05KJF to advance the sequence 50 places before generating five variates from a uniform distribution using G05SAF.

10.1 Program Text

```
    Program g05kjfe

!     G05KJF Example Program Text

!     Mark 25 Release. NAG Copyright 2014.

!     .. Use Statements ..
      Use nag_library, Only: g05kff, g05kjf, g05saf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: lseed = 1, nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                          :: genid, ifail, lstate, n, nv, subid
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: x(:)
      Integer                          :: seed(lseed)
      Integer, Allocatable             :: state(:)
!     .. Executable Statements ..
      Write (nout,*) 'G05KJF Example Program Results'
      Write (nout,*)

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

!     Read in the base generator information and seed
      Read (nin,*) genid, subid, seed(1)

!     Query G05KFF to get the require length 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)

!     Read in the skip ahead and sample size
      Read (nin,*) n, nv

      Allocate (x(nv))

!     Advance the sequence N places
      ifail = 0
      Call g05kjf(n,state,ifail)

!     Generate a NV variates from a uniform distribution
      ifail = 0
      Call g05saf(nv,state,x,ifail)

!     Display the variates
      Write (nout,99999) x(1:nv)

99999 Format (1X,F10.4)
    End Program g05kjfe
```

10.2 Program Data

```
G05KJF Example Program Data
1  1  1762543     :: GENID,SUBID,SEED(1)
50  5             :: N,NV
```

10.3 Program Results

```
 G05KJF Example Program Results

     0.2071
     0.8413
     0.8817
     0.5494
     0.5248
```