G05BCFP

NAG Parallel Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check for implementation-dependent details. You are advised to enclose any calls to NAG Parallel Library routines between calls to Z01AAFP and Z01ABFP.

1 Description

G05BCFP generates a vector of pseudo-random numbers of length n from a uniform distribution in the semi-open interval $[a, b)$.

A total of 273 statistically independent generators are available; it is possible to select a particular generator and initialize the seeds for the generator by a preceding call to G05BBFP. If G05BBFP is not used, default values for the generator and the seeds are assumed.

The routine G05BCFP always generates exactly the same pseudo-random numbers as would n consecutive calls of G05ACFP.

2 Specification

```
SUBROUTINE G05BCFP(A, B, N, X)
INTEGER            N
DOUBLE PRECISION   A, B, X(*)
```

3 Usage

3.1 Definitions

None.

3.2 Global and Local Arguments

All arguments are local.

4 Arguments

1: A — DOUBLE PRECISION *Local Input*

2: B — DOUBLE PRECISION *Local Input*

On entry: the end points of the distribution. It is not necessary to have A < B.

3: N — INTEGER *Local Input/Local Output*

On entry: n, the number of pseudo-random numbers to be generated. If N < 1, no pseudo-random numbers are generated.

On exit: the actual number of pseudo-random numbers which were generated.

4: X($*$) — DOUBLE PRECISION array *Local Output*

On exit: the n pseudo-random numbers from the specified uniform distribution.

5 Errors and Warnings

None.

6 Further Comments

Repeatable sequences of random numbers can be generated by calling G05BBFP to set the seeds and generator number before calling G05BCFP.

G05BCFP may be called without a prior call to Z01AAFP.

6.1 Algorithmic Detail

Each basic generator uses a Wichmann–Hill type generator (Wichmann and Hill [3]), which is a variant of a multiplicative congruential algorithm to produce real pseudo-random numbers v_i in the semi-open interval $[a, b)$:

$$\begin{aligned}
k_{1,i} &= (c_1 \times k_{1,i-1}) \bmod m_1 \\
k_{2,i} &= (c_2 \times k_{2,i-1}) \bmod m_2 \\
k_{3,i} &= (c_3 \times k_{3,i-1}) \bmod m_3 \\
k_{4,i} &= (c_4 \times k_{4,i-1}) \bmod m_4 \\
u_i &= \left(\frac{k_{1,i}}{m_1} + \frac{k_{2,i}}{m_2} + \frac{k_{3,i}}{m_3} + \frac{k_{4,i}}{m_4}\right) \bmod 1.0 \\
v_i &= (a + (b-a) \times u_i) \bmod b \quad \text{if} \quad a \le b \\
v_i &= (b + (a-b) \times u_i) \bmod a \quad \text{if} \quad a > b
\end{aligned}$$

where c_j and m_j, $j = 1{,}4$ are constant integers for each generator and $k_{j,i}$ on the left and right hand of the equations are newly generated integer seeds and old seeds, respectively. The real values u_i give pseudo-random numbers in the semi-open interval $[0, 1)$. The constants c_j are in the range 112 to 127 and the constants m_j are prime numbers in the range 16718909 to 16776971 which are close to $2^{24} = 16777216$. These constants have been chosen so that they give good results with the spectral test, see Knuth [1] and Maclaren [2].

The period of each generator would be at least 2^{92} if it were not for common factors between $(m_1 - 1)$, $(m_2 - 1)$, $(m_3 - 1)$ and $(m_4 - 1)$. However, each should still have a period of at least 2^{80}. Further details of the generators can be obtained from NAG and further discussion of the properties of these generators is given in Maclaren [2] where it was shown that the generated pseudo-random sequences are essentially independent of one another according to the spectral test.

7 References

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

[2] Maclaren N M (1989) The generation of multiple independent sequences of pseudorandom numbers *Appl. Statist.* **38** 351–359

[3] Wichmann B A and Hill I D (1982) AS183 An efficient and portable pseudo-random number generator *Appl. Statist.* **31** 188–190

8 Example

This example generates a series of random numbers on each processor on a 2 by 2 logical grid of processors. The routine G05BBFP is used to initialise the seeds and the generators.

8.1 Example Text

```
*     G05BCFP Example Program Text
*     NAG Parallel Library Release 3. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          NOUT, NX
      PARAMETER        (NOUT=6,NX=10)
      INTEGER          MAG
      PARAMETER        (MAG=16909320)
*     .. Local Scalars ..
      DOUBLE PRECISION A, B
      INTEGER          I, ICNTXT, ICOFF, IFAIL, IGEN, MP, MYCOL, MYROW,
     +                 N, NP, NPCOL, NPROW
      LOGICAL          ROOT
      CHARACTER        CNUMOP, TITOP
      CHARACTER*20     FORMT
*     .. Local Arrays ..
      DOUBLE PRECISION WORK(NX), X(NX)
      INTEGER          IS(5), ISEED(4), IWORK(5)
*     .. External Functions ..
      LOGICAL          Z01ACFP
      EXTERNAL         Z01ACFP
*     .. External Subroutines ..
      EXTERNAL         G05BBFP, G05BCFP, X04BFFP, X04BMFP, Z01AAFP,
     +                 Z01ABFP, Z01ZAFP
*     .. Intrinsic Functions ..
      INTRINSIC        MOD
*     .. Executable Statements ..
      ROOT = Z01ACFP()
      IF (ROOT) THEN
         WRITE (NOUT,*) 'G05BCFP Example Program Results'
         WRITE (NOUT,*)
      END IF
*
      MP = 2
      NP = 2
*
*     Declare the processor grid
*
      IFAIL = 0
      CALL Z01AAFP(ICNTXT,MP,NP,IFAIL)
*
*     Initialise the seeds and the generator
      CALL Z01ZAFP(ICNTXT,NPROW,NPCOL,MYROW,MYCOL)
*
*     Initialize the seeds and choose a generator number that depends
*     on the processor position on the grid.
*
      ISEED(1) = 207*(50*MYROW+19*MYCOL) + 5678212
      ISEED(2) = 451*(70*MYROW+31*MYCOL) + 6252478
      ISEED(3) = 912*(39*MYROW+56*MYCOL) + 2626279
      ISEED(4) = 812*(69*MYROW+78*MYCOL) + 8932937
      IGEN = NP*MYROW*4 + MP*MYCOL*6
*
*     Make sure that the seeds are within the maximum value MAG
*
      DO 40 I = 1, 4
   20    IF (ISEED(I).GT.MAG) THEN
```

```
            ISEED(I) = ISEED(I)/2
            GO TO 20
         END IF
   40 CONTINUE
*
*     Make sure that the generator is valid
*
      IGEN = MOD(IGEN,273)
*
*     Print the seeds and the generator
*
      IS(1) = ISEED(1)
      IS(2) = ISEED(2)
      IS(3) = ISEED(3)
      IS(4) = ISEED(4)
      IS(5) = IGEN
      IF (ROOT) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Seeds and the generator'
         WRITE (NOUT,*)
      END IF
      FORMT = 'I10'
      TITOP = 'Y'
      CNUMOP = 'X'
      ICOFF = 0
      IFAIL = 0
      CALL X04BMFP(ICNTXT,NOUT,1,5,IS,1,FORMT,TITOP,CNUMOP,ICOFF,IWORK,
     +             1,IFAIL)
      CALL G05BBFP(ISEED,IGEN)
*
*
*     Set the lower and upper limits of the distribution
*     Set N (the number of random numbers per processor)
*
      A = 2.0D0
      B = 10.0D0
      N = 5
*
*     Now fill the vectors with random numbers
*
      CALL G05BCFP(A,B,N,X)
*
*     Print the vectors on the root processor
*
      IF (ROOT) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Random numbers on each processor'
         WRITE (NOUT,*)
      END IF
      FORMT = 'F12.5'
      TITOP = 'Y'
      CNUMOP = 'X'
      ICOFF = 0
      IFAIL = 0
      CALL X04BFFP(ICNTXT,NOUT,1,N,X,1,FORMT,TITOP,CNUMOP,ICOFF,WORK,1,
     +             IFAIL)

      IFAIL = 0
      CALL Z01ABFP(ICNTXT,'N',IFAIL)
*
      STOP
*
      END
```

8.2 Example Data

None.

8.3 Example Results

```
 G05BCFP Example Program Results


 Seeds and the generator

   Array from logical processor    0,   0

     5678212   6252478   2626279   8932937         0

   Array from logical processor    0,   1

     5682145   6266459   2677351   8996273        12

   Array from logical processor    1,   0

     5688562   6284048   2661847   8988965         8

   Array from logical processor    1,   1

     5692495   6298029   2712919   9052301        20


 Random numbers on each processor

   Array from logical processor    0,   0

       9.56336     2.84915     7.72301     6.21566     7.28020

   Array from logical processor    0,   1

       9.82620     9.52210     7.40150     2.36244     2.26696

   Array from logical processor    1,   0

       5.94202     7.37632     8.71823     3.59809     5.95273

   Array from logical processor    1,   1

       6.11450     5.30904     9.65975     6.32173     3.62686
```