C06MCFP

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

C06MCFP computes the Discrete Fourier Transform (DFT) of a sequence of complex data values z_j, where $j = 0, 1, \ldots, n-1$.

The DFT is here defined by:

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(\pm 2\pi i \frac{jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.) The **minus** sign is taken in the argument of exponential within the summation when the **forward** transform is required, and the **plus** sign is taken when the **backward** transform is required (see argument DIRECT). The above sequence and its DFT can also be expressed in the forms:

$$Z = [z_j] = X + iY; \quad \hat{Z} = [\hat{z}_k] = \hat{X} + i\hat{Y}$$

where X, Y, and $\hat{X}$, $\hat{Y}$ contain the real and the imaginary parts of Z and $\hat{Z}$ respectively.

This routine uses a two-dimensional formulation of the one-dimensional DFT, described in Bailey [1]. The dimension n must be previously factorized as $n = n_1 \times n_2$, where $n_1, n_2 \geq 1$. This factorization can be obtained by using the routine C06GXFP, which computes n_1 and n_2 with $n_1 (\geq \sqrt{n})$ as close as possible to $\sqrt{n}$. Therefore, the array Z and its transform $\hat{Z}$ are considered as matrices of dimension $n_1 \times n_2$ and $n_2 \times n_1$ respectively.

Optionally, the trigonometric coefficient generated by a previous call to C06MCFP for a problem of the same dimension (n) can be reused, thereby avoiding their repeated calculation.

2 Specification

```
 SUBROUTINE C06MCFP(ICNTXT, X, Y, N1, N2, INIT, TRIG, DIRECT, NHX,
1                   WORK, IFAIL)
 DOUBLE PRECISION   X(*), Y(*), TRIG(*), WORK(*)
 INTEGER            ICNTXT, N1, N2, NHX, IFAIL
 CHARACTER*1        INIT, DIRECT
```

3 Usage

3.1 Definitions

The following definitions are used in describing the data distribution within this document:

p	–	the total number of processors in the Library Grid.
N_b	–	the nominal number of rows of X and Y held locally on a logical processor.
$\hat{N}_b$	–	the nominal number of rows of $\hat{X}$ and $\hat{Y}$ held locally on a logical processor.
N_x	–	the actual number of rows of X and Y held locally on a logical processor, where $0 \leq N_x \leq N_b$,
$\hat{N}_x$	–	the actual number of rows of $\hat{X}$ and $\hat{Y}$ held locally on a logical processor, where $0 \leq \hat{N}_x \leq \hat{N}_b$.
$\lceil x \rceil$	–	the ceiling function of x, which gives the smallest integer which is not less than x.

3.2 Global and Local Arguments

The following global **input** arguments must have the same value on entry to the routine on each processor and the global **output** arguments will have the same value on exit from the routine on each processor:

Global input arguments: N1, N2, INIT, DIRECT, IFAIL

Global output arguments: IFAIL

The remaining arguments are local.

3.3 Distribution Strategy

Given the factorization $n = n_1 \times n_2$, the vectors X and Y are considered as matrices of **dimension** $n_1 \times n_2$ and the vectors $\hat{X}$ and $\hat{Y}$ as matrices of **dimension** $n_2 \times n_1$, as it follows:

$$\begin{array}{rcl} x_{j_1,j_2} & \equiv & x_{j_1+j_2 n_1} \\ y_{j_1,j_2} & \equiv & y_{j_1+j_2 n_1} \end{array} , \quad j_1 = 0,1,\ldots,n_1-1, \quad j_2 = 0,1,\ldots,n_2-1,$$

and

$$\begin{array}{rcl} \hat{x}_{k_1,k_2} & \equiv & \hat{x}_{k_1+k_2 n_2} \\ \hat{y}_{k_1,k_2} & \equiv & \hat{y}_{k_1+k_2 n_2} \end{array} , \quad k_1 = 0,1,\ldots,n_2-1, \quad k_2 = 0,1,\ldots,n_1-1.$$

(For simplicity the same notation is used for each one-dimensional sequence and the corresponding matrix.)

The matrices X and Y are distributed in row-block fashion, that is rows of X and Y are allocated to logical processors on the two-dimensional grid row by row (i.e. in row major ordering of the grid), starting from the $\{0,0\}$ logical processor. Each processor, that contains rows of X and Y, contains $N_b = \lceil n_1/p \rceil$ consecutive rows, except the last processor that actually contains data, for which the number of rows may be less than N_b. This processor holds $\mathrm{mod}(n_1, N_b)$ rows if $\mathrm{mod}(n_1, N_b) \neq 0$ and N_b rows otherwise.

The transformed matrices $\hat{X}$ and $\hat{Y}$ are also distributed in row-block fashion, but with n_2 playing the role of n_1 and $\hat{N}_b = \lceil n_2/p \rceil$ playing the role of N_b.

3.4 Related Routines

The Library provides many support routines for the generation/distribution and input/output of data in row-block form.

The following routines may be used in conjunction with C06MCFP:

Real matrix generation: F01ZMFP

Real matrix output: X04BFFP

4 Arguments

1: ICNTXT — INTEGER *Local Input*

On entry: the Library context, usually returned by a call to the Library Grid initialisation routine Z01AAFP.

Note: the value of ICNTXT **must not** be changed.

2: X(∗) — DOUBLE PRECISION array *Local Input/Local Output*

Note: the dimension of the array X must be at least $\max(\lceil n_1/p \rceil \times n_2, \lceil n_2/p \rceil \times n_1)$.

Distribution: array X is formally defined as a vector. However, you may find it more convenient to consider X as a two-dimensional array. On entry X is considered as X(0 : $N_x - 1$, 0 : N2−1), that is X(i,j) corresponds to X($1 + i + j \times N_x$); on exit X is considered as X(0 : $\hat{N}_x - 1$, 0 : N1−1), that is X(i,j) corresponds to X($1 + i + j \times \hat{N}_x$). The array X is not referenced if $N_x = 0$ and $\hat{N}_x = 0$.

On entry: the local parts of X (considered as a $n_1 \times n_2$ matrix) which define the real parts of the complex sequence Z, distributed in row-block fashion (see Section 3.3).

On exit: the local parts of $\hat{X}$ (considered as a $n_2 \times n_1$ matrix) distributed in row-block fashion (see Section 3.3).

3: Y(∗) — DOUBLE PRECISION array *Local Input/Local Output*

Note: the dimension of the array Y must be at least $\max(\lceil n_1/p \rceil \times n_2, \lceil n_2/p \rceil \times n_1)$.

Distribution: array Y is formally defined as a vector. However, you may find it more convenient to consider Y as a two-dimensional array. On entry Y is considered as Y($0 : N_x - 1, 0 :$ N2−1) that is Y(i,j) corresponds to Y($1 + i + j \times N_x$); on exit Y is considered as Y($0 : \hat{N}_x - 1, 0 :$ N1−1), that is Y(i,j) corresponds to Y($1 + i + j \times \hat{N}_x$). The array Y is not referenced if $N_x = 0$ and $\hat{N}_x = 0$.

On entry: the local parts of Y (considered as a $n_1 \times n_2$ matrix) which define the imaginary parts of the complex sequence Z, stored as a matrix in row-block distribution (see Section 3.3).

On exit: the local parts of $\hat{Y}$ (considered as a $n_2 \times n_1$ matrix) distributed in row-block fashion (see Section 3.3).

4: N1 — INTEGER *Global Input*

On entry: n_1, the first factor of n (may be returned by a call to C06GXFP).

Constraint: N1 ≥ 1.

5: N2 — INTEGER *Global Input*

On entry: n_2, the second factor of n (may be returned by a call to C06GXFP).

Constraint: N2 ≥ 1.

6: INIT — CHARACTER*1 *Global Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT is set to 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified values of N1 and N2 are supplied in the array TRIG, having been calculated in a previous call to C06MCFP.

Constraint: INIT = 'I' or 'S'.

7: TRIG(∗) — DOUBLE PRECISION array *Local Input/Local Output*

Note: the dimension of the array TRIG must be at least $\alpha = 2\times$(N1+N2) and $\beta = 2\times\lceil n_1/p \rceil\times$(N2−1) + 3. Users are advised not to change the elements of the array TRIG.

On entry: if INIT = 'S', TRIG must contain the required trigonometric coefficients, as computed in a previous call to the routine with the same values of N1 and N2.

In both cases, TRIG($1 :$ 2∗N1) and TRIG(2∗N1+1 : α) must contain the trigonometric coefficients related to the one-dimensional DFTs of length N1 and N2 respectively. TRIG($\alpha + 1 : \alpha + \beta$) must contain the remaining trigonometric coefficients contributed from the exponential factors related to both N1 and N2 which corresponds to the local parts of X and Y (see Section 6.1 and Section 6.2).

Note: with INIT = 'S', the trigonometric coefficients stored in TRIG($1 : \alpha$) (computed previously with the values of N1 and N2) can be reused when N1 and N2 are interchanged. However, in this case the exponential factors (as described in Section 6.2) are recomputed and do not need to be supplied on entry to the routine.

This situation commonly occurs when C06MCFP is used to perform a backward transform after a forward transform (see Section 8).

If INIT = 'I', TRIG does not need to be set.

On exit: the required trigonometric coefficients.

If the routine has been called with INIT = 'I' or 'S', then TRIG($1 :$ 2∗N1) contains the trigonometric coefficients related to the one-dimensional DFTs of length N1, TRIG(2∗N1+1 : α) contains the trigonometric coefficients related to the one-dimensional DFTs of length N2 and TRIG($\alpha + 1 :$ $\alpha + \beta$) contains the exponential factors related to both N1 and N2 which corresponds to the local parts of X and Y (see Section 6.1 and Section 6.2).

8: DIRECT — CHARACTER*1 *Global Input*

On entry: the direction of transform to be computed.

If DIRECT = 'F', then **F**orward transform as defined in Section 1 will be computed.
If DIRECT = 'B', then **B**ackward transform as defined in Section 1 will be computed.

Constraint: DIRECT = 'F' or 'B'.

9: NHX — INTEGER *Local Output*

On exit: $\hat{N}_x$, the actual number of rows of $\hat{X}$ and $\hat{Y}$ held on the logical processor.

10: WORK(∗) — DOUBLE PRECISION array *Local Workspace*

Note: the dimension of WORK, as declared in the (sub)program from which C06MCFP is called, must be at least $2\times\max(\lceil n_1/p\rceil \times n_2, n_1 \times \lceil n_2/p\rceil)$.

11: IFAIL — INTEGER *Global Input/Global Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in the Essential Introduction) the recommended values are:

IFAIL = 0, if multigridding is **not** employed;
IFAIL = −1, if multigridding is employed.

On exit: IFAIL = 0 unless the routine detects an error (see Section 5).

5 Errors and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output from the root processor (or processor {0,0} when the root processor is not available) on the current error message unit (as defined by X04AAF).

5.1 Full Error Checking Mode Only

IFAIL = −2000

The routine has been called with an invalid value of ICNTXT on one or more processors.

IFAIL = −1000

The logical processor grid and library mechanism (Library Grid) have not been correctly defined, see Z01AAFP.

IFAIL = $-i$

On entry, the ith argument was invalid. This error occured either because a global argument did not have the same value on all logical processors, or because its value on one or more processors was incorrect. An explanatory message distinguishes between these two cases.

6 Further Comments

6.1 Algorithmic Detail

The algorithm implemented in this routine is based on a *two-dimensional formulation* of the one-dimensional DFT, in Bailey [1] and in papers cited therein.

Given a factorization of the dimension of the transform, $n = n_1 \times n_2$, the input sequence and its transform are considered as an $n_1 \times n_2$ and $n_2 \times n_1$ matrices respectively, and the DFT is computed in the following steps:

(1) perform n_1 DFTs of length n_2, on the rows of the input matrix;
(2) multiply the (j_1, j_2)-th entry of the resulting matrix by the exponential factor $e^{\pm 2\pi i j_1 j_2/n}$, for $j_1 = 0, 1, \ldots, n_1$; $j_2 = 0, 1, \ldots, n_2$;
(3) transpose the resulting matrix into a $n_2 \times n_1$ matrix;
(4) perform n_2 DFTs of length n_1, on the rows of the transposed matrix.

The final matrix, considered as a one-dimensional array, gives the required DFT.

The FFT algorithm used in steps 1 and 4 is the *Stockham self-sorting algorithm*, described in Temperton [3].

6.2 Parallelism Detail

The parallel algorithm is based on a row-block distribution of the input and output sequences, considered as matrices, as described in Section 3.3.

The four steps described in Section 6.1 are carried out as it follows. Each processor performs Steps 1 and 2 on its local data, which contributes to the transposition of the global matrix in Step 3, and finally performs Step 4 on its new local data. The transposition of the distributed matrix is accomplished in two steps. Each processor, after dividing its block of rows into sub-blocks, (*i*) performs a local transposition of its sub-blocks, and (*ii*) participates to the transposition of the global matrix, where each sub-block is considered as a single element. More details are given in de Bono *et al.* [2].

The algorithm works with any factorization of n, $n = n_1 \times n_2$, with $n_1, n_2 \geq 1$. However, a factorization of n with n_1 and n_2 as close as possible to $\sqrt{n}$ leads usually to a more efficient distribution of workload among processors. Such a factorization can be obtained by calling C06GXFP prior to a call to C06MCFP.

6.3 Accuracy

An upper bound of the roundoff error in the computed DFT is given by:

$$\frac{\|\tilde{z} - \hat{z}\|}{\|\hat{z}\|} \leq 1.06\sqrt{n_1 n_2}\left(\sum_{i=1}^{r}(2a_i)^{\frac{3}{2}} + \sum_{j=1}^{s}(2b_j)^{\frac{3}{2}}\right)\varepsilon$$

where $\hat{z}$ is the *exact* DFT of z, $\tilde{z}$ is the *computed* one, $\|u\|$ is the following norm:

$$\|u\| = \left(\sum_{j=1}^{n} \mid u_{ij} \mid^2\right)^{\frac{1}{2}},$$

$n = n_1 \times n_2$, $n_1 = a_1 \times a_2 \times \ldots \times a_r$ and $n_2 = b_1 \times b_2 \times \ldots \times b_s$ are the factorizations of n_1 and n_2 used by the FFT algorithm, and ε is the machine precision. The input data are assumed to be exactly representable within the limits of machine precision and the roundoff errors in computing the trigonometric coefficients are not considered. However, experiments have shown that the actual error in the DFT computed by the routine has at most the same order of magnitude of the above error bound. For more details see de Bono *et al.* [2].

The routine computes first all the factors of 6 and 4 in n_1 and n_2, then all the factors of 2 and 3 and finally all the other factors, but does not give these factors in output. The value of ε can be computed using the routine X02AJF.

Some indication of accuracy can be otained performing a direct transform followed by an inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

6.4 Computational Costs

If the trigonometric coefficients and the exponential factors are precomputed, the number of floating-point operations per processor is approximately proportional to $\frac{n_1 \times n_2}{p} \times (\log n_2 + \log n_1 + 1)$, but also depends on the factorizations of n_1 and n_2 in the one-dimensional FFT algorithm used to perform steps 1 and 4 (see Section 6.1).

Each processor communicates with each other processor, exchanging approximately $2 \times \frac{n_1 \times n_2}{p^2}$ floating-point values.

7 References

[1] Bailey D H (1990) FFTs in External or Hierarchical Memory *J. Supercomputing* **4** 23–35

[**2**] de Bono I, de Cesare M L, di Serafino D and Perla F (1997) C06MCFP: a Parallel 1-D Mixed-Radix FFT Routine for MIMD Distributed-Memory Machines *Tech. Rep. TR-97-17* Center for Research on Parallel Computing and Supercomputers (CPS) – CNR, Naples, Italy

[**3**] Temperton C (1983) Self-sorting mixed-radix fast Fourier transforms *J. Comput. Phys.* **52** 1–23

8 Example

To compute the forward and the backward DFT of a complex sequence Z, where Z is given by

$$Z = X + iY = (x_j) + i(y_j), \quad j = 0, \ldots, n-1,$$

with

$$x_j = \frac{j}{n}, \quad y_j = \frac{n-j}{n},$$

and $n = 28$.

The routine C06GXFP is used to factorize n as $n = n_1 \times n_2$, with $n_1 = 7$ and $n_2 = 4$.

The routine F01ZMFP is used to generate X and Y, considered as matrices of dimension 7×4, on a 2 by 2 logical processor grid. The actual number of rows of the matrices X and Y on each logical processor, N_x, is equal to 2 on the logical processors {0,0}, {0,1} and {1,0}, and is equal to 1 on the logical processor {1,1}.

The routine C06MCFP is used to compute the forward and the backward DFT.

The routine X04BFFP is used to print the original sequence, the forward DFT and the backward DFT. The actual number of rows of the transformed matrices $\hat{X}$ and $\hat{Y}$, $\hat{N}_x$, is equal to 1 on all the logical processors.

8.1 Example Text

```
*     C06MCFP Example Program Text
*     NAG Parallel Library Release 3. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N
      PARAMETER        (N=28)
      INTEGER          MG, NG
      PARAMETER        (MG=2,NG=2)
      INTEGER          N1MAX, N2MAX, NXYMAX
      PARAMETER        (N1MAX=8,N2MAX=8,NXYMAX=16)
      INTEGER          LTRIG, LWORK
      PARAMETER        (LTRIG=2*N1MAX+2*N2MAX+2*NXYMAX-N2MAX+3,
     +                 LWORK=2*NXYMAX)
*     .. Scalars in Common ..
      INTEGER          N1
*     .. Local Scalars ..
      INTEGER          ICNTXT, ICOFF, IFAIL, IM, LDXY, MP, MPR, MYPCOL,
     +                 MYPROW, N2, NHX, NP, NPC, NUMPROCS
      LOGICAL          ROOT
      CHARACTER        DIRECT, INIT
      CHARACTER*80     CNUMOP, FORMAT, TITOP
*     .. Local Arrays ..
      DOUBLE PRECISION TRIG(LTRIG), WORK(LWORK), X(NXYMAX), Y(NXYMAX)
*     .. External Functions ..
      INTEGER          Z01CFFP
      LOGICAL          Z01ACFP
      EXTERNAL         Z01CFFP, Z01ACFP
*     .. External Subroutines ..
```

```
      EXTERNAL         C06GXFP, C06MCFP, F01ZMFP, GMATX, GMATY, X04BFFP,
     +                 Z01AAFP, Z01ABFP, Z01ZAFP
*     .. Common blocks ..
      COMMON           /DIM/N1
*     .. Executable Statements ..
*
      ROOT = Z01ACFP()
      IF (ROOT) WRITE (NOUT,*) 'C06MCFP Example Program Results'
*
*     Define 2D processor grid
*
      MP = MG
      NP = NG
      IFAIL = 0
      CALL Z01AAFP(ICNTXT,MP,NP,IFAIL)
*
*     Perform factorization of N, N = N1 x N2, and print results
*
      IFAIL = 0
      CALL C06GXFP(ICNTXT,N,N1,N2,IFAIL)
*
      IF (ROOT) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99999)
         WRITE (NOUT,99998) N, N1, N2
      END IF
*
*     Generate sequences X and Y, considered as matrices of
*     dimension N1 x N2
*
      CALL Z01ZAFP(ICNTXT,MPR,NPC,MYPROW,MYPCOL)
      NUMPROCS = MPR*NPC
      IM = MYPROW*NPC + MYPCOL
*
*
*     Z01CFFP compute the leading dimension of X and Y, LDXY, for use by
*
      LDXY = Z01CFFP(NUMPROCS,N1,IM)
*
*
      IFAIL = 0
      CALL F01ZMFP(ICNTXT,GMATX,N1,N2,X,LDXY,NHX,IFAIL)
*
      IFAIL = 0
      CALL F01ZMFP(ICNTXT,GMATY,N1,N2,Y,LDXY,NHX,IFAIL)
*
*     Print generated data
*
      IF (ROOT) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Generated data'
         WRITE (NOUT,*)
      END IF
*
      FORMAT = 'F7.3'
      TITOP = 'Y'
      CNUMOP = 'N'
      ICOFF = 0
```

```
*
      IF (ROOT) THEN
         WRITE (NOUT,*) 'X: Real'
         WRITE (NOUT,*)
      END IF
      IFAIL = 0
*
      CALL X04BFFP(ICNTXT,NOUT,NHX,N2,X,NHX,FORMAT,TITOP,CNUMOP,ICOFF,
     +             WORK,NHX,IFAIL)
*
      IF (ROOT) THEN
         WRITE (NOUT,*) 'Y: Imag'
         WRITE (NOUT,*)
      END IF
      IFAIL = 0
*
      CALL X04BFFP(ICNTXT,NOUT,NHX,N2,Y,NHX,FORMAT,TITOP,CNUMOP,ICOFF,
     +             WORK,NHX,IFAIL)
*
*     Compute forward transform
*
      INIT = 'I'
      DIRECT = 'F'
      IFAIL = 0
      CALL C06MCFP(ICNTXT,X,Y,N1,N2,INIT,TRIG,DIRECT,NHX,WORK,IFAIL)
*
*     Print results
*
      IF (ROOT) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of Discrete Fourier Transform'
         WRITE (NOUT,*)
      END IF
*
      IF (ROOT) THEN
         WRITE (NOUT,*) 'X: Real'
         WRITE (NOUT,*)
      END IF
      IFAIL = 0
*
      CALL X04BFFP(ICNTXT,NOUT,NHX,N1,X,NHX,FORMAT,TITOP,CNUMOP,ICOFF,
     +             WORK,NHX,IFAIL)
*
      IF (ROOT) THEN
         WRITE (NOUT,*) 'Y: Imag'
         WRITE (NOUT,*)
      END IF
      IFAIL = 0
*
      CALL X04BFFP(ICNTXT,NOUT,NHX,N1,Y,NHX,FORMAT,TITOP,CNUMOP,ICOFF,
     +             WORK,NHX,IFAIL)
*
*     Compute backward transform
*
      INIT = 'S'
      DIRECT = 'B'
      IFAIL = 0
      CALL C06MCFP(ICNTXT,X,Y,N2,N1,INIT,TRIG,DIRECT,NHX,WORK,IFAIL)
```

```
*
*     Print results
*
      IF (ROOT) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original sequence as restored by Backward',
     +     ' Transform'
         WRITE (NOUT,*)
      END IF
*
      IF (ROOT) THEN
         WRITE (NOUT,*) 'X: Real'
         WRITE (NOUT,*)
      END IF
      IFAIL = 0
*
      CALL X04BFFP(ICNTXT,NOUT,NHX,N2,X,NHX,FORMAT,TITOP,CNUMOP,ICOFF,
     +             WORK,NHX,IFAIL)
*
      IF (ROOT) THEN
         WRITE (NOUT,*) 'Y: Imag'
         WRITE (NOUT,*)
      END IF
      IFAIL = 0
*
      CALL X04BFFP(ICNTXT,NOUT,NHX,N2,Y,NHX,FORMAT,TITOP,CNUMOP,ICOFF,
     +             WORK,NHX,IFAIL)
*
*     Leave processor grid
*
      IFAIL = 0
      CALL Z01ABFP(ICNTXT,'N',IFAIL)
*
      STOP
*
99999 FORMAT (1X,' N',3X,'N1',3X,'N2')
99998 FORMAT (1X,3(I2,3X))
      END
*
*
      SUBROUTINE GMATX(I1,IL,N2,X,LDX)
*
*     GMATX generates the block X(I1:IL,1:N2) of the sequence
*     X(I) = I/N, where I = 0,...,N-1 and N = N1 X N2, considered
*     as a matrix of dimension N1 x N2.
*
*     .. Scalar Arguments ..
      INTEGER          I1, IL, LDX, N2
*     .. Array Arguments ..
      DOUBLE PRECISION X(LDX,*)
*     .. Scalars in Common ..
      INTEGER          N1
*     .. Local Scalars ..
      DOUBLE PRECISION PROD
      INTEGER          I, II, J, K
*     .. Intrinsic Functions ..
      INTRINSIC        DBLE
*     .. Common blocks ..
```

```
      COMMON           /DIM/N1
*     .. Executable Statements ..
*
      PROD = DBLE(N1*N2)
      DO 40 J = 1, N2
         K = 0
         DO 20 I = I1, IL
            K = K + 1
            II = I - 1 + N1*(J-1)
            X(K,J) = DBLE(II)/PROD
   20    CONTINUE
   40 CONTINUE
*
*     End of GMATX
*
      RETURN
      END
*
*
      SUBROUTINE GMATY(I1,IL,N2,Y,LDY)
*
*     GMATX generates the block Y(I1:IL,1:N2) of the sequence
*     Y(I) = (N-I)/N, where I = 0,...,N-1 and N = N1 X N2,
*     considered as a matrix of dimension N1 x N2.
*
*     .. Scalar Arguments ..
      INTEGER          I1, IL, LDY, N2
*     .. Array Arguments ..
      DOUBLE PRECISION Y(LDY,*)
*     .. Scalars in Common ..
      INTEGER          N1
*     .. Local Scalars ..
      DOUBLE PRECISION PROD
      INTEGER          I, II, J, K
*     .. Intrinsic Functions ..
      INTRINSIC        DBLE
*     .. Common blocks ..
      COMMON           /DIM/N1
*     .. Executable Statements ..
*
      PROD = DBLE(N1*N2)
      DO 40 J = 1, N2
         K = 0
         DO 20 I = I1, IL
            K = K + 1
            II = I - 1 + N1*(J-1)
            Y(K,J) = DBLE(PROD-II)/PROD
   20    CONTINUE
   40 CONTINUE
*
*     End of GMATY
*
      RETURN
      END
```

8.2 Example Data

None.

8.3 Example Results

```
C06MCFP Example Program Results

 N   N1   N2
28    7    4

Generated data

X: Real

  Array from logical processor    0,   0

   0.000  0.250  0.500  0.750
   0.036  0.286  0.536  0.786

  Array from logical processor    0,   1

   0.071  0.321  0.571  0.821
   0.107  0.357  0.607  0.857

  Array from logical processor    1,   0

   0.143  0.393  0.643  0.893
   0.179  0.429  0.679  0.929

  Array from logical processor    1,   1

   0.214  0.464  0.714  0.964

Y: Imag

  Array from logical processor    0,   0

   1.000  0.750  0.500  0.250
   0.964  0.714  0.464  0.214

  Array from logical processor    0,   1

   0.929  0.679  0.429  0.179
   0.893  0.643  0.393  0.143

  Array from logical processor    1,   0

   0.857  0.607  0.357  0.107
   0.821  0.571  0.321  0.071

  Array from logical processor    1,   1

   0.786  0.536  0.286  0.036


Components of Discrete Fourier Transform
```

```
X: Real

  Array from logical processor    0,   0

   2.551  0.102 -0.019 -0.073 -0.116 -0.170 -0.291

  Array from logical processor    0,   1

   0.744  0.056 -0.035 -0.084 -0.128 -0.189 -0.365

  Array from logical processor    1,   0

   0.320  0.024 -0.049 -0.094 -0.140 -0.213 -0.508

  Array from logical processor    1,   1

   0.176  0.000 -0.061 -0.105 -0.154 -0.245 -0.933
Y: Imag

  Array from logical processor    0,   0

   2.740  0.291  0.170  0.116  0.073  0.019 -0.102

  Array from logical processor    0,   1

   0.933  0.245  0.154  0.105  0.061  0.000 -0.176

  Array from logical processor    1,   0

   0.508  0.213  0.140  0.094  0.049 -0.024 -0.320

  Array from logical processor    1,   1

   0.365  0.189  0.128  0.084  0.035 -0.056 -0.744


Original sequence as restored by Backward Transform

X: Real

  Array from logical processor    0,   0

   0.000  0.250  0.500  0.750
   0.036  0.286  0.536  0.786

  Array from logical processor    0,   1

   0.071  0.321  0.571  0.821
   0.107  0.357  0.607  0.857

  Array from logical processor    1,   0

   0.143  0.393  0.643  0.893
   0.179  0.429  0.679  0.929

  Array from logical processor    1,   1
```

```
   0.214  0.464  0.714  0.964

Y: Imag

  Array from logical processor    0,   0

   1.000  0.750  0.500  0.250
   0.964  0.714  0.464  0.214

  Array from logical processor    0,   1

   0.929  0.679  0.429  0.179
   0.893  0.643  0.393  0.143

  Array from logical processor    1,   0

   0.857  0.607  0.357  0.107
   0.821  0.571  0.321  0.071

  Array from logical processor    1,   1

   0.786  0.536  0.286  0.036
```