NAG Library Routine Document

C06FPF

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

C06FPF computes the discrete Fourier transforms of m sequences, each containing n real data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06FPF (M, N, X, INIT, TRIG, WORK, IFAIL)

INTEGER             M, N, IFAIL
REAL (KIND=nag_wp)  X(M*N), TRIG(2*N), WORK(M*N)
CHARACTER(1)        INIT
```

3 Description

Given m sequences of n real data values x_j^p, for $j = 0, 1, \ldots, n-1$ and $p = 1, 2, \ldots, m$, C06FPF simultaneously calculates the Fourier transforms of all the sequences defined by

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j^p \times \exp\left(-i\frac{2\pi jk}{n}\right), \qquad k = 0, 1, \ldots, n-1 \text{ and } p = 1, 2, \ldots, m.$$

(Note the scale factor $\frac{1}{\sqrt{n}}$ in this definition.)

The transformed values $\hat{z}_k^p$ are complex, but for each value of p the $\hat{z}_k^p$ form a Hermitian sequence (i.e., $\hat{z}_{n-k}^p$ is the complex conjugate of $\hat{z}_k^p$), so they are completely determined by mn real numbers (see also the C06 Chapter Introduction).

The discrete Fourier transform is sometimes defined using a positive sign in the exponential term:

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j^p \times \exp\left(+i\frac{2\pi jk}{n}\right).$$

To compute this form, this routine should be followed by forming the complex conjugates of the $\hat{z}_k^p$; that is $x(k) = -x(k)$, for $k = (n/2+1) \times m + 1, \ldots, m \times n$.

The routine uses a variant of the fast Fourier transform (FFT) algorithm (see Brigham (1974)) known as the Stockham self-sorting algorithm, which is described in Temperton (1983). Special coding is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

Brigham E O (1974) *The Fast Fourier Transform* Prentice–Hall

Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: M – INTEGER *Input*

On entry: m, the number of sequences to be transformed.

Constraint: $M \geq 1$.

2: N – INTEGER *Input*

On entry: n, the number of real values in each sequence.

Constraint: $N \geq 1$.

3: X(M × N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension $(1 : \mathrm{M}, 0 : \mathrm{N} - 1)$; each of the m sequences is stored in a **row** of the array. In other words, if the data values of the pth sequence to be transformed are denoted by x_j^p, for $j = 0, 1, \ldots, n - 1$, then the mn elements of the array X must contain the values

$$x_0^1, x_0^2, \ldots, x_0^m, x_1^1, x_1^2, \ldots, x_1^m, \ldots, x_{n-1}^1, x_{n-1}^2, \ldots, x_{n-1}^m.$$

On exit: the m discrete Fourier transforms stored as if in a two-dimensional array of dimension $(1 : \mathrm{M}, 0 : \mathrm{N} - 1)$. Each of the m transforms is stored in a **row** of the array in Hermitian form, overwriting the corresponding original sequence. If the n components of the discrete Fourier transform $\hat{z}_k^p$ are written as $a_k^p + ib_k^p$, then for $0 \leq k \leq n/2$, a_k^p is contained in $\mathrm{X}(p, k)$, and for $1 \leq k \leq (n - 1)/2$, b_k^p is contained in $\mathrm{X}(p, n - k)$. (See also Section 2.1.2 in the C06 Chapter Introduction.)

4: INIT – CHARACTER(1) *Input*

On entry: indicates whether trigonometric coefficients are to be calculated.

INIT = 'I'
: Calculate the required trigonometric coefficients for the given value of n, and store in the array TRIG.

INIT = 'S' or 'R'
: The required trigonometric coefficients are assumed to have been calculated and stored in the array TRIG in a prior call to one of C06FPF, C06FQF or C06FRF. The routine performs a simple check that the current value of n is consistent with the values stored in TRIG.

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

5: TRIG(2 × N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIG must contain the required trigonometric coefficients that have been previously calculated. Otherwise TRIG need not be set.

On exit: contains the required coefficients (computed by the routine if INIT = 'I').

6: WORK(M × N) – REAL (KIND=nag_wp) array *Workspace*

7: 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, $\mathrm{M} < 1$.

IFAIL $= 2$

On entry, $\mathrm{N} < 1$.

IFAIL $= 3$

On entry, INIT $\neq$ 'I', 'S' or 'R'.

IFAIL $= 4$

Not used at this Mark.

IFAIL $= 5$

On entry, INIT $=$ 'S' or 'R', but the array TRIG and the current value of N are inconsistent.

IFAIL $= 6$

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

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

8 Further Comments

The time taken by C06FPF is approximately proportional to $nm \log n$, but also depends on the factors of n. C06FPF is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This example reads in sequences of real data values and prints their discrete Fourier transforms (as computed by C06FPF). The Fourier transforms are expanded into full complex form using and printed. Inverse transforms are then calculated by conjugating and calling C06FQF showing that the original sequences are restored.

9.1 Program Text

```
    Program c06fpfe

!     C06FPF Example Program Text

!     Mark 24 Release. NAG Copyright 2012.

!     .. Use Statements ..
      Use nag_library, Only: c06fpf, c06fqf, c06gqf, c06gsf, nag_wp
```

```
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                           :: i, ieof, ifail, j, m, n
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: trig(:), u(:), v(:), work(:), x(:)
!     .. Executable Statements ..
      Write (nout,*) 'C06FPF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
loop: Do
        Read (nin,*,Iostat=ieof) m, n
        If (ieof<0) Exit loop

        Allocate (trig(2*n),u(m*n),v(m*n),work(2*m*n),x(m*n))
        Do j = 1, m
          Read (nin,*)(x(i*m+j),i=0,n-1)
        End Do
        Write (nout,*)
        Write (nout,*) 'Original data values'
        Write (nout,*)
        Write (nout,99999)('     ',(x(i*m+j),i=0,n-1),j=1,m)

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call c06fpf(m,n,x,'Initial',trig,work,ifail)

        Write (nout,*)
        Write (nout,*) 'Discrete Fourier transforms in Hermitian format'
        Write (nout,*)
        Write (nout,99999)('     ',(x(i*m+j),i=0,n-1),j=1,m)
        Write (nout,*)
        Write (nout,*) 'Fourier transforms in full complex form'

        Call c06gsf(m,n,x,u,v,ifail)

        Do j = 1, m
          Write (nout,*)
          Write (nout,99999) 'Real ', (u(i*m+j),i=0,n-1)
          Write (nout,99999) 'Imag ', (v(i*m+j),i=0,n-1)
        End Do

        Call c06gqf(m,n,x,ifail)
        Call c06fqf(m,n,x,'Subsequent',trig,work,ifail)

        Write (nout,*)
        Write (nout,*) 'Original data as restored by inverse transform'
        Write (nout,*)
        Write (nout,99999)('     ',(x(i*m+j),i=0,n-1),j=1,m)
        Deallocate (trig,u,v,work,x)
      End Do loop

99999 Format (1X,A,6F10.4)
    End Program c06fpfe
```

9.2 Program Data

```
C06FPF Example Program Data
     3    6                                                        : m, n
     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815        : x
```

9.3 Program Results

```
C06FPF Example Program Results

Original data values

         0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
         0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
         0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

Discrete Fourier transforms in Hermitian format

         1.0737   -0.1041    0.1126   -0.1467   -0.3738   -0.0044
         1.3961   -0.0365    0.0780   -0.1521   -0.0607    0.4666
         1.1237    0.0914    0.3936    0.1530    0.3458   -0.0508

Fourier transforms in full complex form

Real     1.0737   -0.1041    0.1126   -0.1467    0.1126   -0.1041
Imag     0.0000   -0.0044   -0.3738    0.0000    0.3738    0.0044

Real     1.3961   -0.0365    0.0780   -0.1521    0.0780   -0.0365
Imag     0.0000    0.4666   -0.0607    0.0000    0.0607   -0.4666

Real     1.1237    0.0914    0.3936    0.1530    0.3936    0.0914
Imag     0.0000   -0.0508    0.3458    0.0000   -0.3458    0.0508

Original data as restored by inverse transform

         0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
         0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
         0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```