NAG Library Routine Document

C06PFF

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

C06PFF computes the discrete Fourier transform of one variable in a multivariate sequence of complex data values.

2 Specification

```
SUBROUTINE C06PFF (DIRECT, NDIM, L, ND, N, X, WORK, LWORK, IFAIL)

INTEGER                NDIM, L, ND(NDIM), N, LWORK, IFAIL
COMPLEX (KIND=nag_wp) X(N), WORK(LWORK)
CHARACTER(1)           DIRECT
```

3 Description

C06PFF computes the discrete Fourier transform of one variable (the lth say) in a multivariate sequence of complex data values $z_{j_1 j_2 \cdots j_m}$, where $j_1 = 0, 1, \ldots, n_1 - 1, \quad j_2 = 0, 1, \ldots, n_2 - 1$, and so on. Thus the individual dimensions are $n_1, n_2, \ldots, n_m$, and the total number of data values is $n = n_1 \times n_2 \times \cdots \times n_m$.

The routine computes n/n_l one-dimensional transforms defined by

$$\hat{z}_{j_1 \ldots k_l \ldots j_m} = \frac{1}{\sqrt{n_l}} \sum_{j_l=0}^{n_l-1} z_{j_1 \ldots j_l \ldots j_m} \times \exp\left(\pm \frac{2\pi i j_l k_l}{n_l}\right),$$

where $k_l = 0, 1, \ldots, n_l - 1$. The plus or minus sign in the argument of the exponential terms in the above definition determine the direction of the transform: a minus sign defines the **forward** direction and a plus sign defines the **backward** direction.

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

A call of C06PFF with DIRECT = 'F' followed by a call with DIRECT = 'B' will restore the original data.

The data values must be supplied in a one-dimensional complex array using column-major storage ordering of multidimensional data (i.e., with the first subscript j_1 varying most rapidly).

This routine calls C06PRF to perform one-dimensional discrete Fourier transforms. Hence, 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).

4 References

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

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

5 Parameters

1: DIRECT – CHARACTER(1) *Input*

On entry: if the forward transform as defined in Section 3 is to be computed, then DIRECT must be set equal to 'F'.

If the backward transform is to be computed then DIRECT must be set equal to 'B'.

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

2: NDIM – INTEGER *Input*

On entry: m, the number of dimensions (or variables) in the multivariate data.

Constraint: NDIM ≥ 1.

3: L – INTEGER *Input*

On entry: l, the index of the variable (or dimension) on which the discrete Fourier transform is to be performed.

Constraint: $1 \leq \mathrm{L} \leq \mathrm{NDIM}$.

4: ND(NDIM) – INTEGER array *Input*

On entry: the elements of ND must contain the dimensions of the NDIM variables; that is, ND(i) must contain the dimension of the ith variable.

Constraint: ND$(i) \geq 1$, for $i = 1, 2, \ldots,$ NDIM.

5: N – INTEGER *Input*

On entry: n, the total number of data values.

Constraint: N must equal the product of the first NDIM elements of the array ND.

6: X(N) – COMPLEX (KIND=nag_wp) array *Input/Output*

On entry: the complex data values. Data values are stored in X using column-major ordering for storing multidimensional arrays; that is, $z_{j_1 j_2 \cdots j_m}$ is stored in $\mathrm{X}(1 + j_1 + n_1 j_2 + n_1 n_2 j_3 + \cdots)$.

On exit: the corresponding elements of the computed transform.

7: WORK(LWORK) – COMPLEX (KIND=nag_wp) array *Workspace*

The workspace requirements as documented for C06PFF may be an overestimate in some implementations.

On exit: the real part of WORK(1) contains the minimum workspace required for the current value of N with this implementation.

8: LWORK – INTEGER *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which C06PFF is called.

Suggested value: LWORK $\geq$ N + ND(L) + 15.

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

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, NDIM < 1.

IFAIL $= 2$

On entry, L < 1 or L $>$ NDIM.

IFAIL $= 3$

On entry, DIRECT $\neq$ 'F' or 'B'.

IFAIL $= 4$

On entry, at least one of the first NDIM elements of ND is less than 1.

IFAIL $= 5$

On entry, N does not equal the product of the first NDIM elements of ND.

IFAIL $= 6$

On entry, LWORK is too small. The minimum amount of workspace required is returned in WORK(1).

IFAIL $= 8$

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

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

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 Parallelism and Performance

C06PFF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

C06PFF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken is approximately proportional to $n \times \log n_l$, but also depends on the factorization of n_l. C06PFF is faster if the only prime factors of n_l are 2, 3 or 5; and fastest of all if n_l is a power of 2.

10 Example

This example reads in a bivariate sequence of complex data values and prints the discrete Fourier transform of the second variable. It then performs an inverse transform and prints the sequence so obtained, which may be compared with the original data values.

10.1 Program Text

```
!   C06PFF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module c06pffe_mod

!     C06PFF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                               :: readx, writx
!     .. Parameters ..
      Integer, Parameter, Public           :: nin = 5, nout = 6
    Contains
      Subroutine readx(nin,x,n1,n2)
!       Read 2-dimensional complex data

!       .. Scalar Arguments ..
        Integer, Intent (In)               :: n1, n2, nin
!       .. Array Arguments ..
        Complex (Kind=nag_wp), Intent (Out)  :: x(n1,n2)
!       .. Local Scalars ..
        Integer                            :: i, j
!       .. Executable Statements ..
        Do i = 1, n1
          Read (nin,*)(x(i,j),j=1,n2)
        End Do
        Return
      End Subroutine readx

      Subroutine writx(nout,x,n1,n2)
!       Print 2-dimensional complex data

!       .. Scalar Arguments ..
        Integer, Intent (In)               :: n1, n2, nout
!       .. Array Arguments ..
        Complex (Kind=nag_wp), Intent (In)   :: x(n1,n2)
!       .. Local Scalars ..
        Integer                            :: i, j
!       .. Executable Statements ..
        Do i = 1, n1
```

```
          Write (nout,*)
          Write (nout,99999)(x(i,j),j=1,n2)
        End Do
        Return

99999   Format (1X,7(:1X,'(',F6.3,',',F6.3,')'))
      End Subroutine writx
    End Module c06pffe_mod

    Program c06pffe

!     C06PFF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: c06pff, nag_wp
      Use c06pffe_mod, Only: nin, nout, readx, writx
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Integer                              :: ieof, ifail, l, lwork, n, ndim
!     .. Local Arrays ..
      Complex (Kind=nag_wp), Allocatable   :: work(:), x(:)
      Integer, Allocatable                 :: nd(:)
!     .. Intrinsic Procedures ..
      Intrinsic                            :: product
!     .. Executable Statements ..
      Write (nout,*) 'C06PFF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
loop: Do
        Read (nin,*,Iostat=ieof) ndim
        If (ieof<0) Exit loop
        Allocate (nd(ndim))
        Read (nin,*) nd(1:ndim), l
        n = product(nd(1:ndim))
        lwork = n + nd(l) + 15
        Allocate (x(n),work(lwork))
        Call readx(nin,x,nd(1),nd(2))
        Write (nout,*)
        Write (nout,*) 'Original data'
        Call writx(nout,x,nd(1),nd(2))

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
!       Compute transform
        Call c06pff('F',ndim,l,nd,n,x,work,lwork,ifail)

        Write (nout,*)
        Write (nout,99999) 'Discrete Fourier transform of variable ', l
        Call writx(nout,x,nd(1),nd(2))

!       Compute inverse transform
        Call c06pff('B',ndim,l,nd,n,x,work,lwork,ifail)

        Write (nout,*)
        Write (nout,*) 'Original sequence as restored by inverse transform'
        Call writx(nout,x,nd(1),nd(2))
        Deallocate (nd,x,work)

      End Do loop

99999 Format (1X,A,I1)
    End Program c06pffe
```

10.2 Program Data

```
C06PFF Example Program Data
    2                  : ndim
    3    5    2        : nd(1), nd(2), l
     (1.000,0.000)
     (0.999,-0.040)
     (0.987,-0.159)
     (0.936,-0.352)
     (0.802,-0.597)
     (0.994,-0.111)
     (0.989,-0.151)
     (0.963,-0.268)
     (0.891,-0.454)
     (0.731,-0.682)
     (0.903,-0.430)
     (0.885,-0.466)
     (0.823,-0.568)
     (0.694,-0.720)
     (0.467,-0.884)     : x
```

10.3 Program Results

```
 C06PFF Example Program Results

 Original data

  ( 1.000, 0.000) ( 0.999,-0.040) ( 0.987,-0.159) ( 0.936,-0.352) ( 0.802,-0.597)

  ( 0.994,-0.111) ( 0.989,-0.151) ( 0.963,-0.268) ( 0.891,-0.454) ( 0.731,-0.682)

  ( 0.903,-0.430) ( 0.885,-0.466) ( 0.823,-0.568) ( 0.694,-0.720) ( 0.467,-0.884)

 Discrete Fourier transform of variable 2

  ( 2.113,-0.513) ( 0.288,-0.000) ( 0.126, 0.130) (-0.003, 0.190) (-0.287, 0.194)

  ( 2.043,-0.745) ( 0.286,-0.032) ( 0.139, 0.115) ( 0.018, 0.189) (-0.263, 0.225)

  ( 1.687,-1.372) ( 0.260,-0.125) ( 0.170, 0.063) ( 0.079, 0.173) (-0.176, 0.299)

 Original sequence as restored by inverse transform

  ( 1.000,-0.000) ( 0.999,-0.040) ( 0.987,-0.159) ( 0.936,-0.352) ( 0.802,-0.597)

  ( 0.994,-0.111) ( 0.989,-0.151) ( 0.963,-0.268) ( 0.891,-0.454) ( 0.731,-0.682)

  ( 0.903,-0.430) ( 0.885,-0.466) ( 0.823,-0.568) ( 0.694,-0.720) ( 0.467,-0.884)
```