# NAG Library Routine Document

# C06FJF

**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

C06FJF computes the multidimensional discrete Fourier transform of a multivariate sequence of complex data values.

## 2 Specification

```
SUBROUTINE C06FJF (NDIM, ND, N, X, Y, WORK, LWORK, IFAIL)
INTEGER            NDIM, ND(NDIM), N, LWORK, IFAIL
REAL (KIND=nag_wp) X(N), Y(N), WORK(LWORK)
```

## 3 Description

C06FJF computes the multidimensional discrete Fourier transform of a multidimensional sequence of complex data values $z_{j_1 j_2 \ldots j_m}$, 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 discrete Fourier transform is here defined (e.g., for $m = 2$) by:

$$\hat{z}_{k_1,k_2} = \frac{1}{\sqrt{n}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} z_{j_1 j_2} \times \exp\left(-2\pi i \left(\frac{j_1 k_1}{n_1} + \frac{j_2 k_2}{n_2}\right)\right),$$

where $k_1 = 0, 1, \ldots, n_1 - 1$, $k_2 = 0, 1, \ldots, n_2 - 1$.

The extension to higher dimensions is obvious. (Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.)

To compute the inverse discrete Fourier transform, defined with $\exp(+2\pi i(\ldots))$ in the above formula instead of $\exp(-2\pi i(\ldots))$, this routine should be preceded and followed by the complex conjugation of the data values and the transform (by negating the imaginary parts stored in $y$).

The data values must be supplied in a pair of one-dimensional arrays (real and imaginary parts separately), in accordance with the Fortran convention for storing multidimensional data (i.e., with the first subscript $j_1$ varying most rapidly).

This routine calls C06FCF to perform one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham (1974), and hence there are some restrictions on the values of the $n_i$ (see Section 5).

## 4 References

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

## 5 Arguments

1: NDIM – INTEGER *Input*

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

*Constraint*: NDIM $\geq 1$.

2: ND(NDIM) – INTEGER array *Input*

*On entry*: ND($i$) must contain $n_i$ (the dimension of the $i$th variable) , for $i = 1, 2, \ldots, m$. The largest prime factor of each ND($i$) must not exceed 19, and the total number of prime factors of ND($i$), counting repetitions, must not exceed 20.

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

3: N – INTEGER *Input*

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

*Constraint*: N = ND(1) $\times$ ND(2) $\times \cdots \times$ ND(NDIM).

4: X(N) – REAL (KIND=nag_wp) array *Input/Output*

*On entry*: X($1 + j_1 + n_1 j_2 + n_1 n_2 j_3 + \ldots$) must contain the real part of the complex data value $z_{j_1 j_2 \ldots j_m}$, for $0 \leq j_1 \leq n_1 - 1, 0 \leq j_2 \leq n_2 - 1, \ldots$; i.e., the values are stored in consecutive elements of the array according to the Fortran convention for storing multidimensional arrays.

*On exit*: the real parts of the corresponding elements of the computed transform.

5: Y(N) – REAL (KIND=nag_wp) array *Input/Output*

*On entry*: the imaginary parts of the complex data values, stored in the same way as the real parts in the array X.

*On exit*: the imaginary parts of the corresponding elements of the computed transform.

6: WORK(LWORK) – REAL (KIND=nag_wp) array *Workspace*

7: LWORK – INTEGER *Input*

*On entry*: the dimension of the array WORK as declared in the (sub)program from which C06FJF is called.

*Constraint*: LWORK $\geq 3 \times \max\{\text{ND}(i)\}$.

8: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation 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 argument, 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, N $\neq$ ND(1) $\times$ ND(2) $\times \cdots \times$ ND(NDIM).

IFAIL $= 10 \times l + 1$

At least one of the prime factors of ND($l$) is greater than 19.

IFAIL $= 10 \times l + 2$

ND($l$) has more than 20 prime factors.

IFAIL $= 10 \times l + 3$

On entry, ND($l$) $< 1$.

IFAIL $= 10 \times l + 4$

On entry, LWORK $< 3 \times$ ND($l$).

IFAIL $= -99$

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation 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

C06FJF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

C06FJF 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 $n \times \log(n)$, but also depends on the factorization of the individual dimensions ND($i$). C06FJF is faster if the only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

## 10 Example

This example reads in a bivariate sequence of complex data values and prints the two-dimensional Fourier transform. It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values.

## 10.1 Program Text

```
!   C06FJF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module c06fjfe_mod

!     C06FJF 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                           :: readxy, writxy
!     .. Parameters ..
      Integer, Parameter, Public       :: nin = 5, nout = 6
    Contains
      Subroutine readxy(nin,x,y,n1,n2)
!       Read 2-dimensional complex data

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

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

!       .. Scalar Arguments ..
        Integer, Intent (In)           :: n1, n2, nout
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x(n1,n2), y(n1,n2)
!       .. Local Scalars ..
        Integer                        :: i, j
!       .. Executable Statements ..
        Do i = 1, n1
          Write (nout,*)
          Write (nout,99999) 'Real ', (x(i,j),j=1,n2)
          Write (nout,99999) 'Imag ', (y(i,j),j=1,n2)
        End Do
        Return

99999   Format (1X,A,7F10.3,/,(6X,7F10.3))
      End Subroutine writxy
    End Module c06fjfe_mod

    Program c06fjfe

!     C06FJF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: c06fjf, nag_wp
      Use c06fjfe_mod, Only: nin, nout, readxy, writxy
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Integer                          :: ieof, ifail, lwork, n, ndim
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: work(:), x(:), y(:)
```

```
      Integer, Allocatable             :: nd(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: maxval, product
!     .. Executable Statements ..
      Write (nout,*) 'C06FJF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
loop: Do
        Read (nin,*,Iostat=ieof) ndim
        If (ieof<0) Then
          Exit loop
        End If

        Allocate (nd(ndim))
        Read (nin,*) nd(1:ndim)
        n = product(nd(1:ndim))
        lwork = 3*maxval(nd(1:ndim))
        Allocate (x(n),y(n),work(lwork))

        Call readxy(nin,x,y,nd(1),nd(2))
        Write (nout,*)
        Write (nout,*) 'Original data values'
        Call writxy(nout,x,y,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 c06fjf(ndim,nd,n,x,y,work,lwork,ifail)

        Write (nout,*)
        Write (nout,*) 'Components of discrete Fourier transform'
        Call writxy(nout,x,y,nd(1),nd(2))

!       Compute inverse transform
        y(1:n) = -y(1:n)
        Call c06fjf(ndim,nd,n,x,y,work,lwork,ifail)
        y(1:n) = -y(1:n)

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

      End Do loop

    End Program c06fjfe
```

## 10.2 Program Data

```
C06FJF Example Program Data
    2                                                    : ndim
    3    5                                               : nd(1), nd(2)
     1.000     0.999     0.987     0.936     0.802
     0.000    -0.040    -0.159    -0.352    -0.597
     0.994     0.989     0.963     0.891     0.731
    -0.111    -0.151    -0.268    -0.454    -0.682
     0.903     0.885     0.823     0.694     0.467
    -0.430    -0.466    -0.568    -0.720    -0.884      : x, y
```

## 10.3 Program Results

```
 C06FJF Example Program Results

 Original data values

 Real      1.000     0.999     0.987     0.936     0.802
 Imag      0.000    -0.040    -0.159    -0.352    -0.597

 Real      0.994     0.989     0.963     0.891     0.731
```

```
 Imag     -0.111    -0.151    -0.268    -0.454    -0.682

 Real      0.903     0.885     0.823     0.694     0.467
 Imag     -0.430    -0.466    -0.568    -0.720    -0.884

 Components of discrete Fourier transform

 Real      3.373     0.481     0.251     0.054    -0.419
 Imag     -1.519    -0.091     0.178     0.319     0.415

 Real      0.457     0.055     0.009    -0.022    -0.076
 Imag      0.137     0.032     0.039     0.036     0.004

 Real     -0.170    -0.037    -0.042    -0.038    -0.002
 Imag      0.493     0.058     0.008    -0.025    -0.083

 Original sequence as restored by inverse transform

 Real      1.000     0.999     0.987     0.936     0.802
 Imag     -0.000    -0.040    -0.159    -0.352    -0.597

 Real      0.994     0.989     0.963     0.891     0.731
 Imag     -0.111    -0.151    -0.268    -0.454    -0.682

 Real      0.903     0.885     0.823     0.694     0.467
 Imag     -0.430    -0.466    -0.568    -0.720    -0.884
```