NAG Library Routine Document

C06FAF

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

C06FAF calculates the discrete Fourier transform of a sequence of n real data values (using a work array for extra speed).

2 Specification

```
SUBROUTINE C06FAF (X, N, WORK, IFAIL)

INTEGER            N, IFAIL
REAL (KIND=nag_wp) X(N), WORK(N)
```

3 Description

Given a sequence of n real data values x_j, for $j = 0, 1, \ldots, n-1$, C06FAF calculates their discrete Fourier transform defined by

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

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.) The transformed values $\hat{z}_k$ are complex, but they form a Hermitian sequence (i.e., $\hat{z}_{n-k}$ is the complex conjugate of $\hat{z}_k$), so they are completely determined by n real numbers (see also the C06 Chapter Introduction).

To compute the inverse discrete Fourier transform defined by

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

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

C06FAF uses the fast Fourier transform (FFT) algorithm (see Brigham (1974)). There are some restrictions on the value of n (see Section 5).

4 References

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

5 Arguments

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

On entry: if X is declared with bounds $(0 : \mathrm{N} - 1)$ in the subroutine from which C06FAF is called, then $\mathrm{X}(j)$ must contain x_j, for $j = 0, 1, \ldots, n-1$.

On exit: the discrete Fourier transform stored in Hermitian form. If the components of the transform $\hat{z}_k$ are written as $a_k + ib_k$, and if X is declared with bounds $(0 : \mathrm{N} - 1)$ in the subroutine from which C06FAF is called, then for $0 \leq k \leq n/2$, a_k is contained in $\mathrm{X}(k)$, and for $1 \leq k \leq (n-1)/2$, b_k is contained in $\mathrm{X}(n-k)$. (See also Section 2.1.2 in the C06 Chapter Introduction and Section 10.)

2: N – INTEGER *Input*

On entry: n, the number of data values. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: $N > 1$.

3: WORK(N) – REAL (KIND=nag_wp) array *Workspace*

4: 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: $\text{IFAIL} = 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry $\text{IFAIL} = 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

$\text{IFAIL} = 1$

At least one of the prime factors of N is greater than 19.

$\text{IFAIL} = 2$

N has more than 20 prime factors.

$\text{IFAIL} = 3$

On entry, $N \leq 1$.

$\text{IFAIL} = 4$

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

$\text{IFAIL} = -99$

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.

$\text{IFAIL} = -399$

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.

$\text{IFAIL} = -999$

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

C06FAF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

C06FAF 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 n. C06FAF is faster if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

10 Example

This example reads in a sequence of real data values and prints their discrete Fourier transform (as computed by C06FAF), after expanding it from Hermitian form into a full complex sequence. It then performs an inverse transform using C06FBF and conjugation, and prints the sequence so obtained alongside the original data values.

10.1 Program Text

```
    Program c06fafe

!     C06FAF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: c06faf, c06fbf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                          :: ieof, ifail, j, n
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:), b(:), work(:), x(:), xx(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: mod
!     .. Executable Statements ..
      Write (nout,*) 'C06FAF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
loop: Do
        Read (nin,*,Iostat=ieof) n
        If (ieof<0) Then
          Exit loop
        End If
        Allocate (a(0:n-1),b(0:n-1),x(0:n-1),xx(0:n-1),work(n))
        Read (nin,*) x(0:n-1)
        xx(0:n-1) = x(0:n-1)

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
```

```
        ifail = 0
        Call c06faf(x,n,work,ifail)

        Write (nout,*)
        Write (nout,*) 'Components of discrete Fourier transform'
        Write (nout,*)
        Write (nout,*) '          Real      Imag'
        Write (nout,*)

!       Convert x to separated real and imaginary parts for printing.
        a(0:n/2) = x(0:n/2)
        a(n-1:n/2+1:-1) = x(1:n/2)
        b(0) = 0.0_nag_wp
        b(1:(n-1)/2) = x(n-1:n-(n-1)/2:-1)
        b(n-(n-1)/2:n-1) = -b((n-1)/2:1:-1)
        If (mod(n,2)==0) Then
          b(n/2) = 0.0_nag_wp
        End If

        Write (nout,99999)(j,a(j),b(j),j=0,n-1)

        x(n/2+1:n-1) = -x(n/2+1:n-1)
        Call c06fbf(x,n,work,ifail)

        Write (nout,*)
        Write (nout,*) 'Original sequence as restored by inverse transform'
        Write (nout,*)
        Write (nout,*) '        Original  Restored'
        Write (nout,*)
        Write (nout,99999)(j,xx(j),x(j),j=0,n-1)
        Deallocate (a,b,x,xx,work)
      End Do loop

99999 Format (1X,I5,2F10.5)
    End Program c06fafe
```

10.2 Program Data

```
C06FAF Example Program Data
    7              : n
  0.34907
  0.54890
  0.74776
  0.94459
  1.13850
  1.32850
  1.51370          : x
```

10.3 Program Results

```
 C06FAF Example Program Results

 Components of discrete Fourier transform

           Real      Imag

     0   2.48361   0.00000
     1  -0.26599   0.53090
     2  -0.25768   0.20298
     3  -0.25636   0.05806
     4  -0.25636  -0.05806
     5  -0.25768  -0.20298
     6  -0.26599  -0.53090

 Original sequence as restored by inverse transform

         Original  Restored

     0   0.34907   0.34907
     1   0.54890   0.54890
```

```
     2   0.74776   0.74776
     3   0.94459   0.94459
     4   1.13850   1.13850
     5   1.32850   1.32850
     6   1.51370   1.51370
```