NAG Library Routine Document

C06RFF

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

C06RFF computes the discrete Fourier cosine transforms of m sequences of real data values. The elements of each sequence and its transform are stored contiguously.

2 Specification

```
SUBROUTINE C06RFF (M, N, X, IFAIL)
INTEGER              M, N, IFAIL
REAL (KIND=nag_wp) X(0:N,M)
```

3 Description

Given m sequences of $n+1$ real data values x_j^p, for $j = 0, 1, \ldots, n$ and $p = 1, 2, \ldots, m$, C06RFF simultaneously calculates the Fourier cosine transforms of all the sequences defined by

$$\hat{x}_k^p = \sqrt{\tfrac{2}{n}}\left(\tfrac{1}{2}x_0^p + \sum_{j=1}^{n-1} x_j^p \times \cos\left(jk\frac{\pi}{n}\right) + \tfrac{1}{2}(-1)^k x_n^p\right), \quad k = 0, 1, \ldots, n \text{ and } p = 1, 2, \ldots, m.$$

(Note the scale factor $\sqrt{\frac{2}{n}}$ in this definition.)

This transform is also known as type-I DCT.

Since the Fourier cosine transform defined above is its own inverse, two consecutive calls of C06RFF will restore the original data.

The transform calculated by this routine can be used to solve Poisson's equation when the derivative of the solution is specified at both left and right boundaries (see Swarztrauber (1977)).

The routine uses a variant of the fast Fourier transform (FFT) algorithm (see Brigham (1974)) known as the Stockham self-sorting algorithm, described in Temperton (1983), together with pre- and post-processing stages described in Swarztrauber (1982). Special coding is provided for the factors 2, 3, 4 and 5.

4 References

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

Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19(3)** 490–501

Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computation* (ed G Rodrique) 51–83 Academic Press

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

5 Arguments

1: M – INTEGER *Input*

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

Constraint: M ≥ 1.

2: N – INTEGER *Input*

On entry: one less than the number of real values in each sequence, i.e., the number of values in each sequence is $n + 1$.

Constraint: N ≥ 1.

3: X(0 : N, M) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the data values of the pth sequence to be transformed, denoted by x_j^p, for $j = 0, 1, \ldots, n$ and $p = 1, 2, \ldots, m$, must be stored in X(j, p).

On exit: the $(n + 1)$ components of the pth Fourier cosine transform, denoted by $\hat{x}_k^p$, for $k = 0, 1, \ldots, n$ and $p = 1, 2, \ldots, m$, are stored in X(k, p), overwriting the corresponding original values.

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: 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, M $= \langle value \rangle$.
Constraint: M ≥ 1.

IFAIL $= 2$

On entry, N $= \langle value \rangle$.
Constraint: N ≥ 1.

IFAIL $= 3$

An internal error has occurred in this routine. Check the routine call and any array sizes. If the call is correct then please contact NAG for assistance.

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

C06RFF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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 by C06RFF is approximately proportional to $nm\log(n)$, but also depends on the factors of n. C06RFF 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. Workspace of order $O(n)$ is internally allocated by this routine.

10 Example

This example reads in sequences of real data values and prints their Fourier cosine transforms (as computed by C06RFF). It then calls C06RFF again and prints the results which may be compared with the original sequence.

10.1 Program Text

```
    Program c06rffe

!     C06RFF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: c06rff, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                          :: ieof, ifail, j, m, n, n1
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: x(:,:)
!     .. Executable Statements ..
      Write (nout,*) 'C06RFF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
loop: Do
        Read (nin,*,Iostat=ieof) m, n1
        n = n1 - 1
        If (ieof<0) Then
          Exit loop
        End If

        Allocate (x(n1,m))
        Read (nin,*)(x(1:n1,j),j=1,m)
        Write (nout,*)
```

```
        Write (nout,*) 'Original data values'
        Write (nout,*)
        Write (nout,99999)(x(1:n1,j),j=1,m)

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
!       -- Compute transform
        Call c06rff(m,n,x,ifail)

        Write (nout,*)
        Write (nout,*) 'Discrete Fourier cosine transforms'
        Write (nout,*)
        Write (nout,99999)(x(1:n1,j),j=1,m)

!       -- Compute inverse transform
        Call c06rff(m,n,x,ifail)

        Write (nout,*)
        Write (nout,*) 'Original data as restored by inverse transform'
        Write (nout,*)
        Write (nout,99999)(x(1:n1,j),j=1,m)
        Deallocate (x)
      End Do loop

99999 Format (1X,7F10.4)
    End Program c06rffe
```

10.2 Program Data

```
C06RFF Example Program Data
 3       7                                                : m, n+1
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  0.9562
 0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  0.4936
 0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  0.2057   : x
```

10.3 Program Results

```
 C06RFF Example Program Results

 Original data values

     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723    0.4936
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815    0.2057

 Discrete Fourier cosine transforms

     1.6833   -0.0482    0.0176    0.1368    0.3240   -0.5830   -0.0427
     1.9605   -0.4884   -0.0655    0.4444    0.0964    0.0856   -0.2289
     1.3838    0.1588   -0.0761   -0.1184    0.3512    0.5759    0.0110

 Original data as restored by inverse transform

     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723    0.4936
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815    0.2057
```