NAG Toolbox

nag_sum_fft_real_1d_multi_rfmt (c06fp)

1 Purpose

nag_sum_fft_real_1d_multi_rfmt (c06fp) computes the discrete Fourier transforms of m sequences, each containing n real data values. This function is designed to be particularly efficient on vector processors.

2 Syntax

```
[x, trig, ifail] = nag_sum_fft_real_1d_multi_rfmt(m, n, x, init, trig)
[x, trig, ifail] = c06fp(m, n, x, init, trig)
```

3 Description

Given m sequences of n real data values x_j^p, for $j = 0, 1, \ldots, n-1$ and $p = 1, 2, \ldots, m$, nag_sum_fft_real_1d_multi_rfmt (c06fp) simultaneously calculates the Fourier transforms of all the sequences defined by

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

(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 function 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 function 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 function 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

5.1 Compulsory Input Parameters

1: **m** – INTEGER

m, the number of sequences to be transformed.

Constraint: $\mathbf{m} \geq 1$.

2: **n** – INTEGER

n, the number of real values in each sequence.

Constraint: $\mathbf{n} \geq 1$.

3: $\mathbf{x}(\mathbf{m} \times \mathbf{n})$ – REAL (KIND=nag_wp) array

The data must be stored in **x** as if in a two-dimensional array of dimension $(1 : \mathbf{m}, 0 : \mathbf{n} - 1)$; 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

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

4: **init** – CHARACTER(1)

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 nag_sum_fft_real_1d_multi_rfmt (c06fp), nag_sum_fft_hermitian_1d_multi_rfmt (c06fq) or nag_sum_fft_complex_1d_multi_rfmt (c06fr). The function 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: $\mathbf{trig}(\mathbf{2} \times \mathbf{n})$ – REAL (KIND=nag_wp) array

If **init** = 'S' or 'R', **trig** must contain the required trigonometric coefficients that have been previously calculated. Otherwise **trig** need not be set.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

1: $\mathbf{x}(\mathbf{m} \times \mathbf{n})$ – REAL (KIND=nag_wp) array

The m discrete Fourier transforms stored as if in a two-dimensional array of dimension $(1 : \mathbf{m}, 0 : \mathbf{n} - 1)$. 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 $\mathbf{x}(p, k)$, and for $1 \leq k \leq (n-1)/2$, b_k^p is contained in $\mathbf{x}(p, n-k)$. (See also Section 2.1.2 in the C06 Chapter Introduction.)

2: $\mathbf{trig}(\mathbf{2} \times \mathbf{n})$ – REAL (KIND=nag_wp) array

Contains the required coefficients (computed by the function if **init** = 'I').

3: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, **m** < 1.

ifail $= 2$

On entry, **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 function calls and array dimensions. Seek expert help.

ifail $= -99$

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

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

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 nag_sum_fft_real_1d_multi_rfmt (c06fp) is approximately proportional to $nm\log(n)$, but also depends on the factors of n. nag_sum_fft_real_1d_multi_rfmt (c06fp) 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 nag_sum_fft_real_1d_multi_rfmt (c06fp)). The Fourier transforms are expanded into full complex form using and printed. Inverse transforms are then calculated by conjugating and calling nag_sum_fft_hermitian_1d_multi_rfmt (c06fq) showing that the original sequences are restored.

9.1 Program Text

```
function c06fp_example

fprintf('c06fp example results\n\n');

% 3 real sequences stored as rows
m = nag_int(3);
n = nag_int(6);
x = [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];

% Transform to get Hermitian sequences
init = 'Initial';
trig = zeros(2*n,1);
[xt, trig, ifail] = c06fp(m, n, x, init, trig);
disp('Discrete Fourier transforms in Hermitian format:');
disp(xt);

for j = 1:m
  zt(j,:) = nag_herm2complex(xt(j,:));
end
title = 'Discrete Fourier transforms in full complex format:';
[ifail] = x04da('General','Non-unit', zt, title);

% Restore data by conjugation and back transform
init = 'Subsequent';
nd = double(n);
xt(1:m,floor(nd/2)+2:n) = -xt(1:m,floor(nd/2)+2:n);
[xr, trig, ifail] = c06fq(m, n, xt, init, trig);

fprintf('\n');
disp('Original data as restored by inverse transform:');
disp(xr);

function [z] = nag_herm2complex(x);
  n = size(x,2);
  z(1) = complex(x(1));
  for j = 2:floor((n-1)/2) + 1
    z(j) = x(j) + i*x(n-j+2);
    z(n-j+2) = x(j) - i*x(n-j+2);
  end
  if (mod(n,2)==0)
    z(n/2+1) = complex(x(n/2+1));
  end
```

9.2 Program Results

```
     c06fp example results

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

 Discrete Fourier transforms in full complex format:
             1          2          3          4          5          6
 1      1.0737    -0.1041     0.1126    -0.1467     0.1126    -0.1041
        0.0000    -0.0044    -0.3738     0.0000     0.3738     0.0044

 2      1.3961    -0.0365     0.0780    -0.1521     0.0780    -0.0365
        0.0000     0.4666    -0.0607     0.0000     0.0607    -0.4666

 3      1.1237     0.0914     0.3936     0.1530     0.3936     0.0914
        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
```