

NAG Toolbox

nag_sum_fft_hermitian_1d_multi_rfmt (c06fq)

1 Purpose

nag_sum_fft_hermitian_1d_multi_rfmt (c06fq) computes the discrete Fourier transforms of m Hermitian sequences, each containing n complex data values. This function is designed to be particularly efficient on vector processors.

2 Syntax

```
[x, trig, ifail] = nag_sum_fft_hermitian_1d_multi_rfmt(m, n, x, init, trig)
[x, trig, ifail] = c06fq(m, n, x, init, trig)
```

3 Description

Given m Hermitian sequences of n complex data values z_j^p , for $j = 0, 1, \dots, n - 1$ and $p = 1, 2, \dots, m$, nag_sum_fft_hermitian_1d_multi_rfmt (c06fq) simultaneously calculates the Fourier transforms of all the sequences defined by

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

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

The transformed values are purely real (see also the C06 Chapter Introduction).

The discrete Fourier transform is sometimes defined using a positive sign in the exponential term

$$\hat{x}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \times \exp\left(+i\frac{2\pi j k}{n}\right).$$

To compute this form, this function should be preceded by forming the complex conjugates of the \hat{z}_k^p ; that is $x(k) = -x(k)$, for $k = (n/2 + 1) \times m + 1, \dots, m \times n$.

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 data values in each sequence.

Constraint: $n \geq 1$.

3: **x(m × n)** – REAL (KIND=nag_wp) array

The data must be stored in **x** as if in a two-dimensional array of dimension $(1 : m, 0 : n - 1)$; each of the m sequences is stored in a **row** of the array in Hermitian form. If the n data values z_j^p are written as $x_j^p + iy_j^p$, then for $0 \leq j \leq n/2$, x_j^p is contained in **x**(p, j), and for $1 \leq j \leq (n - 1)/2$, y_j^p is contained in **x**($p, n - j$). (See also Section 2.1.2 in the C06 Chapter Introduction.)

4: **init** – CHARACTER(1)

Indicates whether trigonometric coefficients are to be calculated.

init = 'T'

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** = 'T', 'S' or 'R'.

5: **trig(2 × 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: **x(m × n)** – REAL (KIND=nag_wp) array

The components of the m discrete Fourier transforms, stored as if in a two-dimensional array of dimension $(1 : m, 0 : n - 1)$. Each of the m transforms is stored as a **row** of the array, overwriting the corresponding original sequence. If the n components of the discrete Fourier transform are denoted by \hat{x}_k^p for $k = 0, 1, \dots, n - 1$, then the mn elements of the array **x** contain the values

$$\hat{x}_0^1, \hat{x}_0^2, \dots, \hat{x}_0^m, \hat{x}_1^1, \hat{x}_1^2, \dots, \hat{x}_1^m, \dots, \hat{x}_{n-1}^1, \hat{x}_{n-1}^2, \dots, \hat{x}_{n-1}^m.$$

2: **trig(2 × n)** – REAL (KIND=nag_wp) array

Contains the required coefficients (computed by the function if **init** = 'T').

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** ≠ 'T', '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_hermitian_1d_multi_rfmt` (c06fq) is approximately proportional to $n \log(n)$, but also depends on the factors of n . `nag_sum_fft_hermitian_1d_multi_rfmt` (c06fq) 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 which are assumed to be Hermitian sequences of complex data stored in Hermitian form. The sequences are expanded into full complex form and printed. The discrete Fourier transforms are then computed (using `nag_sum_fft_hermitian_1d_multi_rfmt` (c06fq)) and printed out. Inverse transforms are then calculated by conjugating and calling `nag_sum_fft_real_1d_multi_rfmt` (c06fp) showing that the original sequences are restored.

9.1 Program Text

```

function c06fq_example

fprintf('c06fq example results\n\n');

% 3 Hermitian sequences stored as rows in compact form
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];

disp('Original values in compact Hermitian form:');
disp(x);

for j = 1

    [u, v, ifail] = c06gs(m, n, x);
    z(j,:) = nag_herm2complex(x(j,:));
end
title = 'Original data written in full complex form:';
[ifail] = x04da('General','Non-unit', z, title);

% Transform to get Hermitian sequences
init = 'Initial';
trig = zeros(2*n,1);
[xt, trig, ifail] = c06fq(m, n, x, init, trig);
disp(' ');
disp('Discrete Fourier transforms (real values):');
disp(xt);

% Restore data by back transform and conjugation
init = 'Subsequent';
[xr, trig, ifail] = c06fp(m, n, xt, init, trig);
nd = double(n);
xr(1:m,floor(nd/2)+2:n) = -xr(1:m,floor(nd/2)+2: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

c06fq example results

```

Original values in compact Hermitian form:
 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

Original data written in full complex form:
      1       2       3       4       5       6
 1  0.3854  0.6772  0.1138  0.6751  0.1138  0.6772
 0.0000  0.1424  0.6362  0.0000 -0.6362 -0.1424

Discrete Fourier transforms (real values):
 1.0788   0.6623   -0.2391   -0.5783   0.4592   -0.4388
 0.8573   1.2261   0.3533   -0.2222   0.3413   -1.2291
 1.1825   0.2625   0.6744   0.5523   0.0540   -0.4790

```

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
