

## NAG Toolbox

### **nag\_sum\_fft\_complex\_3d\_sep (c06fx)**

## 1 Purpose

`nag_sum_fft_complex_3d_sep (c06fx)` computes the three-dimensional discrete Fourier transform of a trivariate sequence of complex data values. This function is designed to be particularly efficient on vector processors.

## 2 Syntax

```
[x, y, trign1, trign2, trign3, ifail] = nag_sum_fft_complex_3d_sep(n1, n2, n3,
x, y, init, trign1, trign2, trign3)
[x, y, trign1, trign2, trign3, ifail] = c06fx(n1, n2, n3, x, y, init, trign1,
trign2, trign3)
```

## 3 Description

`nag_sum_fft_complex_3d_sep (c06fx)` computes the three-dimensional discrete Fourier transform of a trivariate sequence of complex data values  $z_{j_1 j_2 j_3}$ , for  $j_1 = 0, 1, \dots, n_1 - 1$ ,  $j_2 = 0, 1, \dots, n_2 - 1$  and  $j_3 = 0, 1, \dots, n_3 - 1$ .

The discrete Fourier transform is here defined by

$$\hat{z}_{k_1 k_2 k_3} = \frac{1}{\sqrt{n_1 n_2 n_3}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} z_{j_1 j_2 j_3} \times \exp\left(-2\pi i \left(\frac{j_1 k_1}{n_1} + \frac{j_2 k_2}{n_2} + \frac{j_3 k_3}{n_3}\right)\right),$$

where  $k_1 = 0, 1, \dots, n_1 - 1$ ,  $k_2 = 0, 1, \dots, n_2 - 1$ ,  $k_3 = 0, 1, \dots, n_3 - 1$ .

(Note the scale factor of  $\frac{1}{\sqrt{n_1 n_2 n_3}}$  in this definition.)

To compute the inverse discrete Fourier transform, defined with  $\exp(+2\pi i(\dots))$  in the above formula instead of  $\exp(-2\pi i(\dots))$ , this function should be preceded and followed by forming the complex conjugates of the data values and the transform.

This function performs, for each dimension, multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm (see Brigham (1974)). It is designed to be particularly efficient on vector processors.

## 4 References

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

Temperton C (1983) Self-sorting mixed-radix fast Fourier transforms *J. Comput. Phys.* **52** 1–23

## 5 Parameters

### 5.1 Compulsory Input Parameters

1: **n1** – INTEGER

$n_1$ , the first dimension of the transform.

*Constraint:*  $\mathbf{n1} \geq 1$ .

2: **n2** – INTEGER

$n_2$ , the second dimension of the transform.

*Constraint:*  $\mathbf{n2} \geq 1$ .

3: **n3** – INTEGER

$n_3$ , the third dimension of the transform.

*Constraint:*  $\mathbf{n3} \geq 1$ .

4: **x(n1 × n2 × n3)** – REAL (KIND=nag\_wp) array

5: **y(n1 × n2 × n3)** – REAL (KIND=nag\_wp) array

The real and imaginary parts of the complex data values must be stored in arrays **x** and **y** respectively. If **x** and **y** are regarded as three-dimensional arrays of dimension  $(0 : \mathbf{n1} - 1, 0 : \mathbf{n2} - 1, 0 : \mathbf{n3} - 1)$ , then  $\mathbf{x}(j_1, j_2, j_3)$  and  $\mathbf{y}(j_1, j_2, j_3)$  must contain the real and imaginary parts of  $z_{j_1 j_2 j_3}$ .

6: **init** – CHARACTER(1)

Indicates whether trigonometric coefficients are to be calculated.

**init** = 'I'

Calculate the required trigonometric coefficients for the given values of  $n_1$ ,  $n_2$  and  $n_3$ , and store in the corresponding arrays **trign1**, **trign2** and **trign3**.

**init** = 'S' or 'R'

The required trigonometric coefficients are assumed to have been calculated and stored in the arrays **trign1**, **trign2** and **trign3** in a prior call to nag\_sum\_fft\_complex\_3d\_sep (c06fx). The function performs a simple check that the current values of  $n_1$ ,  $n_2$  and  $n_3$  are consistent with the corresponding values stored in **trign1**, **trign2** and **trign3**.

*Constraint:* **init** = 'I', 'S' or 'R'.

7: **trign1(2 × n1)** – REAL (KIND=nag\_wp) array

8: **trign2(2 × n2)** – REAL (KIND=nag\_wp) array

9: **trign3(2 × n3)** – REAL (KIND=nag\_wp) array

If **init** = 'S' or 'R', **trign1**, **trign2** and **trign3** must contain the required coefficients calculated in a previous call of the function. Otherwise **trign1**, **trign2** and **trign3** need not be set. If **n1** = **n2**, the same array may be supplied for **trign1** and **trign2**. Similar considerations apply if **n2** = **n3** or **n1** = **n3**.

## 5.2 Optional Input Parameters

None.

## 5.3 Output Parameters

1: **x(n1 × n2 × n3)** – REAL (KIND=nag\_wp) array

2: **y(n1 × n2 × n3)** – REAL (KIND=nag\_wp) array

The real and imaginary parts respectively of the corresponding elements of the computed transform.

3: **trign1(2 × n1)** – REAL (KIND=nag\_wp) array

4: **trign2(2 × n2)** – REAL (KIND=nag\_wp) array

5: **trign3(2 × n3)** – REAL (KIND=nag\_wp) array

**trign1**, **trign2** and **trign3** contain the required coefficients (computed by the function if **init** = 'I').

6:     **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, **n1** < 1.

**ifail** = 2

On entry, **n2** < 1.

**ifail** = 3

On entry, **n3** < 1.

**ifail** = 4

On entry, **init** ≠ 'T', 'S' or 'R'.

**ifail** = 5

Not used at this Mark.

**ifail** = 6

On entry, **init** = 'S' or 'R', but at least one of the arrays **trign1**, **trign2** and **trign3** is inconsistent with the current value of **n1**, **n2** or **n3**.

**ifail** = 7

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 is approximately proportional to  $n_1 n_2 n_3 \times \log(n_1 n_2 n_3)$ , but also depends on the factorization of the individual dimensions  $n_1$ ,  $n_2$  and  $n_3$ . **nag\_sum\_fft\_complex\_3d\_sep** (c06fx) is faster if the only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

## 9 Example

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

### 9.1 Program Text

```

function c06fx_example

fprintf('c06fx example results\n\n');

x = cat(3,[ 1.000  0.994  0.903;
            0.500  0.494  0.403],...
         [ 0.999  0.989  0.885;
           0.499  0.489  0.385],...
         [ 0.987  0.963  0.823;
           0.487  0.463  0.323],...
         [ 0.936  0.891  0.694;
           0.436  0.391  0.194]);
y = cat(3,[ 0.000 -0.111 -0.430;
            0.500  0.111  0.430],...
         [-0.040 -0.151 -0.466;
           0.040  0.151  0.466],...
         [-0.159 -0.268 -0.568;
           0.159  0.268  0.568],...
         [-0.352 -0.454 -0.720;
           0.352  0.454  0.720]);

nd = nag_int(size(x));

n1 = nd(1);
n2 = nd(2);
n3 = nd(3);

% Transform
init = 'Initial';
trign1 = zeros(2*n1,1);
trign2 = zeros(2*n2,1);
trign3 = zeros(2*n3,1);
[xt, yt, trign1, trign2, trign3, ifail] = ...
c06fx(...,
       n1, n2, n3, x, y, init, trign1, trign2, trign3);

% Restore
init = 'Subsequent';
[xr, yr, trign1, trign2, trign3, ifail] = ...
c06fx(...,
       n1, n2, n3, xt, -yt, init, trign1, trign2, trign3);

disp('Original Data:');
z = x + i*y;
disp(z);

disp('Components of discrete Fourier transform:');
zt = reshape(xt + i*yt,nd);
disp(zt);

disp('Original sequence as restored by inverse transform');
zr = reshape(xr + i*yr,nd);
disp(zr);

```

## 9.2 Program Results

c06fx example results

Original Data:

(:,:,1) =

$$\begin{array}{lll} 1.0000 + 0.0000i & 0.9940 - 0.1110i & 0.9030 - 0.4300i \\ 0.5000 + 0.5000i & 0.4940 + 0.1110i & 0.4030 + 0.4300i \end{array}$$

(:,:,2) =

$$\begin{array}{lll} 0.9990 - 0.0400i & 0.9890 - 0.1510i & 0.8850 - 0.4660i \\ 0.4990 + 0.0400i & 0.4890 + 0.1510i & 0.3850 + 0.4660i \end{array}$$

(:,:,3) =

$$\begin{array}{lll} 0.9870 - 0.1590i & 0.9630 - 0.2680i & 0.8230 - 0.5680i \\ 0.4870 + 0.1590i & 0.4630 + 0.2680i & 0.3230 + 0.5680i \end{array}$$

(:,:,4) =

$$\begin{array}{lll} 0.9360 - 0.3520i & 0.8910 - 0.4540i & 0.6940 - 0.7200i \\ 0.4360 + 0.3520i & 0.3910 + 0.4540i & 0.1940 + 0.7200i \end{array}$$

Components of discrete Fourier transform:

(:,:,1) =

$$\begin{array}{lll} 3.2921 + 0.1021i & 0.1433 - 0.0860i & 0.1433 + 0.2902i \\ 1.2247 - 1.6203i & 0.4243 + 0.3197i & -0.4243 + 0.3197i \end{array}$$

(:,:,2) =

$$\begin{array}{lll} 0.0506 - 0.0416i & 0.0155 + 0.1527i & -0.0502 + 0.1180i \\ 0.3548 + 0.0833i & 0.0204 - 0.1147i & 0.0070 - 0.0800i \end{array}$$

(:,:,3) =

$$\begin{array}{lll} 0.1127 + 0.1021i & -0.0245 + 0.1268i & -0.0245 + 0.0773i \\ -0.0000 + 0.1621i & 0.0134 - 0.0914i & -0.0134 - 0.0914i \end{array}$$

(:,:,4) =

$$\begin{array}{lll} 0.0506 + 0.2458i & -0.0502 + 0.0861i & 0.0155 + 0.0515i \\ -0.3548 + 0.0833i & -0.0070 - 0.0800i & -0.0204 - 0.1147i \end{array}$$

Original sequence as restored by inverse transform

(:,:,1) =

$$\begin{array}{lll} 1.0000 - 0.0000i & 0.9940 + 0.1110i & 0.9030 + 0.4300i \\ 0.5000 - 0.5000i & 0.4940 - 0.1110i & 0.4030 - 0.4300i \end{array}$$

(:,:,2) =

$$\begin{array}{lll} 0.9990 + 0.0400i & 0.9890 + 0.1510i & 0.8850 + 0.4660i \\ 0.4990 - 0.0400i & 0.4890 - 0.1510i & 0.3850 - 0.4660i \end{array}$$

(:,:,3) =

$$\begin{array}{lll} 0.9870 + 0.1590i & 0.9630 + 0.2680i & 0.8230 + 0.5680i \\ 0.4870 - 0.1590i & 0.4630 - 0.2680i & 0.3230 - 0.5680i \end{array}$$

(:,:,4) =

$$\begin{array}{lll} 0.9360 + 0.3520i & 0.8910 + 0.4540i & 0.6940 + 0.7200i \\ 0.4360 - 0.3520i & 0.3910 - 0.4540i & 0.1940 - 0.7200i \end{array}$$


---