NAG Toolbox

nag_sum_fft_complex_2d (c06pu)

1 Purpose

nag_sum_fft_complex_2d (c06pu) computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values (using complex data type).

2 Syntax

```
[x, ifail] = nag_sum_fft_complex_2d(direct, m, n, x)
[x, ifail] = c06pu(direct, m, n, x)
```

3 Description

nag_sum_fft_complex_2d (c06pu) computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values $z_{j_1 j_2}$, for $j1 = 0, 1, \ldots, m-1$ and $j2 = 0, 1, \ldots, n-1$.

The discrete Fourier transform is here defined by

$$\hat{z}_{k_1 k_2} = \frac{1}{\sqrt{mn}} \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} z_{j_1 j_2} \times \exp\left(\pm 2\pi i \left(\frac{j_1 k_1}{m} + \frac{j_2 k_2}{n}\right)\right),$$

where $k_1 = 0, 1, \ldots, m-1$ and $k_2 = 0, 1, \ldots, n-1$.

(Note the scale factor of $\frac{1}{\sqrt{mn}}$ in this definition.) The minus sign is taken in the argument of the exponential within the summation when the forward transform is required, and the plus sign is taken when the backward transform is required.

A call of nag_sum_fft_complex_2d (c06pu) with **direct** = 'F' followed by a call with **direct** = 'B' will restore the original data.

This function calls nag_sum_fft_complex_1d_multi_row (c06pr) to perform multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham (1974).

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: **direct** – CHARACTER(1)

If the forward transform as defined in Section 3 is to be computed, then **direct** must be set equal to `F'.

If the backward transform is to be computed then **direct** must be set equal to `B'.

Constraint: **direct** = 'F' or 'B'.

2: **m** – INTEGER

m, the first dimension of the transform.

Constraint: **m** ≥ 1.

3: **n** – INTEGER

n, the second dimension of the transform.

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

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

The complex data values. $\mathbf{x}(\mathbf{m} \times j_2 + j_1)$ must contain $z_{j_1 j_2}$, for $j1 = 1, 2, \ldots, \mathbf{m}$ and $j2 = 1, 2, \ldots, \mathbf{n}$.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

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

The corresponding elements of the computed transform.

2: **ifail** – INTEGER

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

6 Error Indicators and Warnings

Errors or warnings detected by the function:

$\mathbf{ifail} = 1$

On entry, $\mathbf{m} < 1$.

$\mathbf{ifail} = 2$

On entry, $\mathbf{n} < 1$.

$\mathbf{ifail} = 3$

On entry, **direct** $\neq$ 'F' or 'B'.

$\mathbf{ifail} = 6$

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

$\mathbf{ifail} = -99$

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

$\mathbf{ifail} = -399$

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

$\mathbf{ifail} = -999$

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 $mn \times \log(mn)$, but also depends on the factorization of the individual dimensions m and n. nag_sum_fft_complex_2d (c06pu) 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 bivariate sequence of complex data values and prints the two-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 c06pu_example

fprintf('c06pu example results\n\n');

m = nag_int(3);
n = nag_int(5);
x = [ 1.000     0.999     0.987     0.936     0.802;
      0.994     0.989     0.963     0.891     0.731;
      0.903     0.885     0.823     0.694     0.467];
y = [ 0.000    -0.040    -0.159    -0.352    -0.597;
     -0.111    -0.151    -0.268    -0.454    -0.682
     -0.430    -0.466    -0.568    -0.720    -0.884];
z = x + i*y;

% transform, then inverse transform to restore data
direct = 'F';
[zt, ifail] = c06pu(direct, m, n, z);
direct = 'B';
[zr, ifail] = c06pu(direct, m, n, zt);

title = 'Original data:';
[ifail] = x04da('General','Non-unit', z, title);
disp(' ');
title = 'Components of discrete Fourier transform:';
[ifail] = x04da('General','Non-unit', zt, title);
disp(' ');
title = 'Original data as restored by inverse transform';
[ifail] = x04da('General','Non-unit', zr, title);
```

9.2 Program Results

```
     c06pu example results

 Original data:
          1       2       3       4       5
 1   1.0000  0.9990  0.9870  0.9360  0.8020
     0.0000 -0.0400 -0.1590 -0.3520 -0.5970

 2   0.9940  0.9890  0.9630  0.8910  0.7310
    -0.1110 -0.1510 -0.2680 -0.4540 -0.6820

 3   0.9030  0.8850  0.8230  0.6940  0.4670
    -0.4300 -0.4660 -0.5680 -0.7200 -0.8840

 Components of discrete Fourier transform:
             1          2          3          4          5
 1      3.3731     0.4814     0.2507     0.0543    -0.4194
       -1.5187    -0.0907     0.1776     0.3188     0.4145

 2      0.4565     0.0549     0.0093    -0.0217    -0.0759
        0.1368     0.0317     0.0389     0.0356     0.0045

 3     -0.1705    -0.0375    -0.0423    -0.0377    -0.0022
```

```
      0.4927     0.0584     0.0082    -0.0255    -0.0829

 Original data as restored by inverse transform
          1       2       3       4       5
 1   1.0000  0.9990  0.9870  0.9360  0.8020
     0.0000 -0.0400 -0.1590 -0.3520 -0.5970

 2   0.9940  0.9890  0.9630  0.8910  0.7310
    -0.1110 -0.1510 -0.2680 -0.4540 -0.6820

 3   0.9030  0.8850  0.8230  0.6940  0.4670
    -0.4300 -0.4660 -0.5680 -0.7200 -0.8840
```