NAG Toolbox

nag_sum_fft_complex_2d_sep (c06fu)

1 Purpose

nag_sum_fft_complex_2d_sep (c06fu) computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values. This function is designed to be particularly efficient on vector processors.

Note: This function is scheduled to be withdrawn, please see c06fu in Advice on Replacement Calls for Withdrawn/Superseded Routines..

2 Syntax

```
[x, y, trigm, trign, ifail] = nag_sum_fft_complex_2d_sep(m, n, x, y, init, trigm, trign)
[x, y, trigm, trign, ifail] = c06fu(m, n, x, y, init, trigm, trign)
```

3 Description

nag_sum_fft_complex_2d_sep (c06fu) 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

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

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

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

To compute the inverse discrete Fourier transform, defined with $\exp(+2\pi i(\ldots))$ in the above formula instead of $\exp(-2\pi i(\ldots))$, this function should be preceded and followed by calls of nag_sum_conjugate_complex_sep (c06gc) to form the complex conjugates of the data values and the transform.

This function calls nag_sum_fft_complex_1d_multi_rfmt (c06fr) to perform multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in 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: **m** – INTEGER

m, the length of the first dimension of Z. Consider the matrix Z with elements $Z_{ij} = z_{i+1\,j+1}$, where z is the bivariate sequence defined in Section 3, then m is the number of rows of Z.

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

2: **n** – INTEGER

n, the length of the second dimension of Z. Consider the matrix Z with elements $Z_{ij} = z_{i+1j+1}$, where z is the bivariate sequence defined in Section 3, then n is the number of columns of Z.

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

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

4: **y**($\mathbf{m} \times \mathbf{n}$) – 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 two-dimensional arrays of dimension $(0 : \mathbf{m} - 1, 0 : \mathbf{n} - 1)$, then $\mathbf{x}(j_1, j_2)$ and $\mathbf{y}(j_1, j_2)$ must contain the real and imaginary parts of $z_{j_1 j_2}$.

5: **init** – CHARACTER(1)

Indicates whether trigonometric coefficients are to be calculated.

init = 'I'
: Calculate the required trigonometric coefficients for the given values of m and n, and store in the corresponding arrays **trigm** and **trign**.

init = 'S' or 'R'
: The required trigonometric coefficients are assumed to have been calculated and stored in the arrays **trigm** and **trign** in a prior call to nag_sum_fft_complex_2d_sep (c06fu). The function performs a simple check that the current values of m and n are consistent with the corresponding values stored in **trigm** and **trign**.

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

6: **trigm**($\mathbf{2} \times \mathbf{m}$) – REAL (KIND=nag_wp) array

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

If **init** = 'S' or 'R', **trigm** and **trign** must contain the required coefficients calculated in a previous call of the function. Otherwise **trigm** and **trign** need not be set.

If $m = n$ the same array may be supplied for **trigm** and **trign**.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

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

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

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

3: **trigm**($\mathbf{2} \times \mathbf{m}$) – REAL (KIND=nag_wp) array

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

trigm and **trign** contain the required coefficients (computed by the function if **init** = 'I').

5: **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 at least one of the arrays **trigm** and **trign** is inconsistent with the current value of **m** or **n**.

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 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_sep (c06fu) 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 c06fu_example

fprintf('c06fu 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];

% transform, then inverse transform to restore data
init = 'Initial';
trigm = zeros(6,1);
trign = zeros(10,1);
[xt, yt, trigm, trign, ifail] = c06fu(m, n, x, y, init, trigm, trign);
init = 'Subsequent';
[xr, yr, trigm, trign, ifail] = c06fu(m, n, xt, -yt, init, trigm, trign);

% Display as complex matrices
z = x + i*y;
nd = [m,n];
zt = reshape(xt+i*yt,nd);
zr = reshape(xr-i*yr,nd);

matrix = 'general';
diag = ' ';
usefrm = 'Above';
format = 'F9.3';
labrow = 'None';
labcol = 'None';
ncols  = nag_int(80);
indent = nag_int(0);

title  = 'Original data:';
[ifail] = x04db(...
    matrix, diag, z, usefrm, format, title, labrow, labcol, ncols, indent);
disp(' ');
title = 'Discrete Fourier transform:';
[ifail] = x04db(...
    matrix, diag, zt, usefrm, format, title, labrow, labcol, ncols, indent);
disp(' ');
title = 'Original sequence as restored by inverse transform:';
[ifail] = x04db(...
    matrix, diag, zr, usefrm, format, title, labrow, labcol, ncols, indent);
```

9.2 Program Results

```
     c06fu example results

 Original data:
     1.000    0.999    0.987    0.936    0.802
     0.000   -0.040   -0.159   -0.352   -0.597

     0.994    0.989    0.963    0.891    0.731
    -0.111   -0.151   -0.268   -0.454   -0.682

     0.903    0.885    0.823    0.694    0.467
    -0.430   -0.466   -0.568   -0.720   -0.884

 Discrete Fourier transform:
     3.373    0.481    0.251    0.054   -0.419
    -1.519   -0.091    0.178    0.319    0.415

     0.457    0.055    0.009   -0.022   -0.076
     0.137    0.032    0.039    0.036    0.004
```

```
    -0.170   -0.037   -0.042   -0.038   -0.002
     0.493    0.058    0.008   -0.025   -0.083

 Original sequence as restored by inverse transform:
     1.000    0.999    0.987    0.936    0.802
     0.000   -0.040   -0.159   -0.352   -0.597

     0.994    0.989    0.963    0.891    0.731
    -0.111   -0.151   -0.268   -0.454   -0.682

     0.903    0.885    0.823    0.694    0.467
    -0.430   -0.466   -0.568   -0.720   -0.884
```