nag_fft_multiple_complex (c06frc)

1. Purpose

nag_fft_multiple_complex (c06frc) computes the discrete Fourier transforms of m sequences, each containing n complex data values.

2. Specification

```
#include <nag.h>
#include <nagc06.h>

void nag_fft_multiple_complex(Integer m, Integer n, double x[],
     double y[], double trig[], NagError *fail)
```

3. Description

Given m sequences of n complex data values z_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, this function simultaneously calculates the Fourier transforms of all the sequences defined by

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \exp(-2\pi i jk/n), \qquad \text{for } k = 0, 1, \ldots, n-1; p = 1, 2, \ldots, m.$$

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

The first call of nag_fft_multiple_complex must be preceded by a call to nag_fft_init_trig (c06gzc) to initialise the array **trig** with trigonometric coefficients.

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

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \exp(+2\pi i jk/n).$$

To compute this form, this function should be preceded and followed by a call of nag_conjugate_complex (c06gcc) to form the complex conjugates of the z_j^p and the $\hat{z}_k^p$.

The function uses a variant of the Fast Fourier Transform algorithm (Brigham 1974) known as the Stockham self-sorting algorithm, which is described in Temperton (1983). Special code is provided for the factors 2, 3, 4, 5 and 6.

4. Parameters

m

Input: the number of sequences to be transformed, m.
Constraint: $\mathbf{m} \geq 1$.

n

Input: the number of complex values in each sequence, n.
Constraint: $\mathbf{n} \geq 1$.

x[m∗n]
y[m∗n]

Input: the real and imaginary parts of the complex data must be stored in **x** and **y** respectively. Each of the m sequences must be stored consecutively; hence if the real parts 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

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

The imaginary parts must be ordered similarly in **y**.
Output: **x** and **y** are overwritten by the real and imaginary parts of the complex transforms.

trig[2∗n]
Input: trigonometric coefficients as returned by a call of nag_fft_init_trig (c06gzc). nag_fft_multiple_complex makes a simple check to ensure that **trig** has been initialised and that the initialisation is compatible with the value of **n**.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **m** must not be less than 1: **m** = ⟨*value*⟩.
On entry, **n** must not be less than 1: **n** = ⟨*value*⟩.

NE_C06_NOT_TRIG
Value of **n** and **trig** array are incompatible or **trig** array not initialised.

NE_ALLOC_FAIL
Memory allocation failed.

6. Further Comments

The time taken by the function is approximately proportional to $nm \log n$, but also depends on the factors of n. The function 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.

6.1. 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).

6.2. 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.

7. See Also

nag_fft_multiple_hermitian (c06fqc)
nag_fft_init_trig (c06gzc)

8. Example

This program reads in sequences of complex data values and prints their discrete Fourier transforms (as computed by nag_fft_multiple_complex). Inverse transforms are then calculated using nag_conjugate_complex (c06gcc) and nag_fft_multiple_complex and printed out, showing that the original sequences are restored.

8.1. Program Text

```
/* nag_fft_multiple_complex(c06frc) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 1, 1990.
 *
 * Mark 3 revised, 1994.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagc06.h>

#define MMAX  5
#define NMAX 20
```

```
main()
{
  double trig[2*NMAX];
  Integer i, j, m, n;
  double x[MMAX*NMAX], y[MMAX*NMAX];

  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vprintf("c06frc Example Program Results\n");
  while (scanf("%ld%ld", &m, &n)!=EOF)
    if (m<=MMAX && n<=NMAX)
      {
        Vprintf("\n\nm = %2ld  n = %2ld\n", m, n);
        for (j = 0; j<m; ++j)
          {
            for (i = 0; i<n; ++i)
              Vscanf("%lf", &x[j*n + i]);
            for (i = 0; i<n; ++i)
              Vscanf("%lf", &y[j*n + i]);
          }
        Vprintf("\nOriginal data values\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("Real");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", x[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\nImag");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", y[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n\n");
          }
        /* Initialise trig array */
        c06gzc(n, trig, NAGERR_DEFAULT);
        /* Compute transforms */
        c06frc(m, n, x, y, trig, NAGERR_DEFAULT);
        Vprintf("\nDiscrete Fourier transforms\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("Real");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", x[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\nImag");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", y[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n\n");
          }
        /* Compute inverse transforms */
        c06gcc(m*n, y, NAGERR_DEFAULT);
        c06frc(m, n, x, y, trig, NAGERR_DEFAULT);
        c06gcc(m*n, y, NAGERR_DEFAULT);
        Vprintf ("\nOriginal data as restored by inverse transform\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("Real");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", x[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\nImag");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", y[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n\n");
          }
      }
    else
```

```
      {
        Vprintf("\nInvalid value of m or n.\n");
        exit(EXIT_FAILURE);
      }
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

```
c06frc Example Program Data
     3     6
     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
     0.9089    0.3118    0.3465    0.6198    0.2668    0.1614
     0.1156    0.0685    0.2060    0.8630    0.6967    0.2792
     0.6214    0.8681    0.7060    0.8652    0.9190    0.3355
```

8.3. Program Results

```
c06frc Example Program Results


m =  3  n =  6

Original data values

Real    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
Imag    0.5417    0.2983    0.1181    0.7255    0.8638    0.8723

Real    0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
Imag    0.9089    0.3118    0.3465    0.6198    0.2668    0.1614

Real    0.1156    0.0685    0.2060    0.8630    0.6967    0.2792
Imag    0.6214    0.8681    0.7060    0.8652    0.9190    0.3355


Discrete Fourier transforms

Real    1.0737   -0.5706    0.1733   -0.1467    0.0518    0.3625
Imag    1.3961   -0.0409   -0.2958   -0.1521    0.4517   -0.0321

Real    1.1237    0.1728    0.4185    0.1530    0.3686    0.0101
Imag    1.0677    0.0386    0.7481    0.1752    0.0565    0.1403

Real    0.9100   -0.3054    0.4079   -0.0785   -0.1193   -0.5314
Imag    1.7617    0.0624   -0.0695    0.0725    0.1285   -0.4335


Original data as restored by inverse transform

Real    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
Imag    0.5417    0.2983    0.1181    0.7255    0.8638    0.8723

Real    0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
Imag    0.9089    0.3118    0.3465    0.6198    0.2668    0.1614

Real    0.1156    0.0685    0.2060    0.8630    0.6967    0.2792
Imag    0.6214    0.8681    0.7060    0.8652    0.9190    0.3355
```