nag_fft_multiple_hermitian (c06fqc)

1. Purpose

nag_fft_multiple_hermitian (c06fqc) computes the discrete Fourier transforms of m Hermitian sequences, each containing n complex data values.

2. Specification

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

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

3. Description

Given m Hermitian 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

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

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

The transformed values are purely real.

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

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

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

For that form, this function should be preceded by a call to nag_multiple_conjugate_hermitian (c06gqc) to form the complex conjugates of the $\hat{z}_j^p$.

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 included 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 data values in each sequence, n.
Constraint: $\mathbf{n} \geq 1$.

x[m∗n]

Input: the m Hermitian sequences must be stored consecutively in **x** in Hermitian form. Sequence 1 should occupy the first n elements of **x**, sequence 2 the elements n to $2n-1$, so that in general sequence p occupies the array elements $(p-1)n$ to $pn-1$. 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 should be in array element $\mathbf{x}[(p-1)*n+j]$ and for $1 \leq j \leq (n-1)/2$, y_j^p should be in array element $\mathbf{x}[(p-1)*n+n-j]$.
Output: the components of the m discrete Fourier transforms, stored consecutively. Transform p occupies the elements $(p-1)n$ to $pn-1$ of **x** overwriting the corresponding original sequence; thus if the n components of the discrete Fourier transform are denoted by $\hat{x}_k^p$, for $k = 0, 1, \ldots, n-1$, then the mn elements of the array **x** contain the values

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

trig[2∗n]
Input: trigonometric coefficients as returned by a call of nag_fft_init_trig (c06gzc). nag_fft_multiple_hermitian 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) Fast Mixed-radix Real Fourier Transforms *J. Comput. Phys.* **52** 340–350.

7. See Also

nag_multiple_conjugate_hermitian (c06gqc)
nag_fft_init_trig (c06gzc)

8. Example

This program 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 using nag_multiple_hermitian_to_complex (c06gsc) and printed. The discrete Fourier transforms are then computed (using nag_fft_multiple_hermitian) and printed out. Inverse transforms are then calculated by calling nag_fft_multiple_real (c06fpc) followed by nag_multiple_conjugate_hermitian (c06gqc) showing that the original sequences are restored.

8.1. Program Text

```
/* nag_fft_multiple_hermitian(c06fqc) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 1, 1990.
 */

#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 u[MMAX*NMAX], v[MMAX*NMAX];
  double x[MMAX*NMAX];

  Vprintf("c06fqc Example Program Results\n");
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  while (scanf("%ld%ld", &m, &n)!=EOF)
    if (m<=MMAX && n<=NMAX)
      {
        Vprintf("\n\nm = %2ld  n = %2ld\n", m, n);
        /* Read in data and print out. */
        for (j = 0; j<m; ++j)
          for (i = 0; i<n; ++i)
            Vscanf("%lf", &x[j*n + i]);
        Vprintf("\nOriginal data values\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("    ");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", x[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n");
          }
        /* Calculate full complex form of Hermitian data sequences */
        c06gsc(m, n, x, u, v, NAGERR_DEFAULT);
        Vprintf("\nOriginal data written in full complex form\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("Real");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", u[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\nImag");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", v[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n\n");
          }
        /* Initialise trig array */
        c06gzc(n, trig, NAGERR_DEFAULT);
        /* Calculate transforms */
        c06fqc(m, n, x, trig, NAGERR_DEFAULT);
        Vprintf ("\nDiscrete Fourier transforms (real values)\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("    ");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", x[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n");
          }
        /* Calculate inverse transforms */
        c06fpc(m, n, x, trig, NAGERR_DEFAULT);
        c06gqc(m, n, x, NAGERR_DEFAULT);
        Vprintf ("\nOriginal data as restored by inverse transform\n\n");
        for (j = 0; j<m; ++j)
          {
            Vprintf("    ");
            for (i = 0; i<n; ++i)
              Vprintf("%10.4f%s", x[j*n + i],
                      (i%6==5 && i!=n-1 ? "\n     " : ""));
            Vprintf("\n");
          }
      }
    else
```

```
      Vfprintf(stderr,"\nInvalid value of m or n.\n");
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

```
c06fqc 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
```

8.3. Program Results

```
c06fqc Example Program Results


m =  3  n =  6

Original data values

        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

Real    0.3854    0.6772    0.1138    0.6751    0.1138    0.6772
Imag    0.0000    0.1424    0.6362    0.0000   -0.6362   -0.1424

Real    0.5417    0.2983    0.1181    0.7255    0.1181    0.2983
Imag    0.0000    0.8723    0.8638    0.0000   -0.8638   -0.8723

Real    0.9172    0.0644    0.6037    0.6430    0.6037    0.0644
Imag    0.0000    0.4815    0.0428    0.0000   -0.0428   -0.4815


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
```