nag_fft_multiple_qtr_cosine (c06hdc)

1. Purpose

nag_fft_multiple_qtr_cosine (c06hdc) computes the discrete quarter-wave Fourier cosine transforms of m sequences of real data values.

2. Specification

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

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

3. Description

Given m sequences of n real data values x_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, this function simultaneously calculates the quarter-wave Fourier cosine transforms of all the sequences defined by:

$$\hat{x}_k^p = \frac{1}{\sqrt{n}} \left\{ \frac{1}{2} x_0^p + \sum_{j=1}^{n-1} x_j^p \cos(j(2k-1)\frac{\pi}{2n}) \right\} \quad \text{if } \textbf{direct} = \textbf{Nag\_ForwardTransform},$$

or its inverse

$$x_k^p = \frac{2}{\sqrt{n}} \sum_{j=0}^{n-1} \hat{x}_j^p \cos((2j-1)k\frac{\pi}{2n}) \quad \text{if } \textbf{direct} = \textbf{Nag\_BackwardTransform},$$

for $k = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$.

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

A call of the function with **direct** = **Nag_ForwardTransform** followed by a call with **direct** = **Nag_BackwardTransform** will restore the original data (but see Section 6.1).

The transform calculated by this function can be used to solve Poisson's equation when the derivative of the solution is specified at the left boundary, and the solution is specified at the right boundary (Swarztrauber 1977).

The function uses a variant of the fast Fourier transform (FFT) algorithm (Brigham 1974) known as the Stockham self-sorting algorithm, described in Temperton (1983), together with pre- and post-processing stages described in Swarztrauber (1982). Special coding is provided for the factors 2, 3, 4, 5 and 6.

4. Parameters

direct

Input: if the forward transform as defined in Section 3 is to be computed, then **direct** must be set equal to **Nag_ForwardTransform**. If the backward transform is to be computed, that is the inverse, then **direct** must be set equal to **Nag_BackwardTransform**.
Constraint: **direct = Nag_ForwardTransform** or **Nag_BackwardTransform**.

m

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

n

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

x[m*n]

Input: the m data sequences stored in **x** consecutively. If the data values of the pth sequence to be transformed are denoted by x_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, then the first mn elements of the array **x** must contain the values

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

Output: the m quarter-wave cosine transforms stored consecutively overwriting the corresponding original sequence.

trig[2*n]

Input: trigonometric coefficients as returned by a call of nag_fft_init_trig (c06gzc). nag_fft_multiple_qtr_cosine 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 initialized.

NE_BAD_PARAM

On entry, parameter **direct** had an illegal value.

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 routine 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.

Swarztrauber P N (1977) The Methods of Cyclic Reduction, Fourier Analysis and the FACR Algorithm for the Discrete Solution of Poisson's Equation on a Rectangle *SIAM Review* **19** (3) 490–501.

Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computations* G Rodrigue (ed) Academic Press pp 51–83.

Temperton C (1983) Fast Mixed-radix Real Fourier Transforms *J. Comput. Phys.* **52** 340–350.

7. See Also

nag_fft_init_trig (c06gzc)

8. Example

This program reads in sequences of real data values and prints their quarter-wave cosine transforms as computed by nag_fft_multiple_qtr_cosine with **direct** = **Nag_ForwardTransform**. It then calls nag_fft_multiple_qtr_cosine again with **direct** = **Nag_BackwardTransform** and prints the results which may be compared with the original data.

8.1. Program Text

```
/* nag_fft_multiple_qtr_cosine(c06hdc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#define MMAX 5
#define NMAX 20
#define X(I,J) x[(I)*n + (J)]

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

  Vprintf("c06hdc Example Program Results\n");
  Vscanf(" %*[^\n]");    /* Skip heading in data file */
  while (scanf("%ld %ld", &m, &n) != EOF)
    if (m <= MMAX && n <= NMAX)
      {
        Vscanf(" %*[^\n]"); /* Skip text in data file */
        Vscanf(" %*[^\n]");
        for (i = 0; i < m; ++i)
          for (j = 0; j < n; ++j)
            Vscanf("%lf", &X(i,j));
        Vprintf("\nOriginal data values\n\n");
        for (i = 0; i < m; ++i)
          {
            for (j = 0; j < n; ++j)
              Vprintf(" %10.4f%s", X(i,j),
                      (j%7==6 && j!=n-1 ? "\n" : ""));
            Vprintf("\n");
          }
        /* Initialise trig array */
        c06gzc(n, trig, NAGERR_DEFAULT);
        /* Compute transform */
        c06hdc(Nag_ForwardTransform, m, n, x, trig, NAGERR_DEFAULT);

        Vprintf("\nDiscrete quarter-wave Fourier cosine transforms\n\n");
        for (i = 0; i < m; ++i)
          {
            for (j = 0; j < n; ++j)
              Vprintf(" %10.4f%s", X(i,j),
                      (j%7==6 && j!=n-1 ? "\n" : ""));
            Vprintf("\n");
          }
        /* Compute inverse transform */
        c06hdc(Nag_BackwardTransform, m, n, x, trig, NAGERR_DEFAULT);
        Vprintf("\nOriginal data as restored by inverse transform\n\n");
        for (i = 0; i < m; ++i)
          {
            for (j = 0; j < n; ++j)
              Vprintf(" %10.4f%s", X(i,j),
                      (j%7==6 && j!=n-1 ? "\n" : ""));
            Vprintf("\n");
          }
      }
    else Vfprintf(stderr,"\nInvalid value of m or n.\n");
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

```
c06hdc Example Program Data
3  6 : Number of sequences, m, and number of values in each sequence, n
Real data sequences
 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

```
c06hdc Example Program Results

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

Discrete quarter-wave Fourier cosine transforms

     0.7257    -0.2216     0.1011     0.2355    -0.1406    -0.2282
     0.7479    -0.6172     0.4112     0.0791     0.1331    -0.0906
     0.6713    -0.1363    -0.0064    -0.0285     0.4758     0.1475

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
```