NAG Library Function Document

nag_fft_multiple_cosine (c06hbc)

1 Purpose

nag_fft_multiple_cosine (c06hbc) computes the discrete Fourier cosine transforms of \( m \) sequences of real data values.

2 Specification

```c
#include <nag.h>
#include <nagc06.h>
void nag_fft_multiple_cosine (Integer m, Integer n, double x[],
const double trig[], NagError *fail)
```

3 Description

Given \( m \) sequences of \( n+1 \) real data values \( x^p_j \), for \( j = 0, 1, \ldots, n \) and \( p = 1, 2, \ldots, m \), this function simultaneously calculates the Fourier cosine transforms of all the sequences defined by

\[
\hat{x}^k_p = \sqrt{\frac{2}{n}} \left\{ \frac{1}{2} x^0_0 + \sum_{j=1}^{n-1} x^j_j \cos \left( \frac{jk\pi}{n} \right) + \frac{1}{2} (-1)^k x^n_n \right\}, \quad \text{for } k = 0, 1, \ldots, n; \ p = 1, 2, \ldots, m.
\]

(Note the scale factor \( \sqrt{\frac{2}{n}} \) in this definition.)

The Fourier cosine transform defined above is its own inverse, and two consecutive calls of this function with the same data will restore the original data (but see Section 9).

The transform calculated by this function can be used to solve Poisson’s equation when the solution is specified at both left and right boundaries (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 References


5 Arguments

1: \( \mathbf{m} \) – Integer  

*Input*

*On entry:* the number of sequences to be transformed, \( m \).

*Constraint:* \( m \geq 1 \).
2: \( n \) – Integer  
**Input**  
*On entry:* one less than the number of real values in each sequence, i.e., the number of values in each sequence is \( n + 1 \).  
*Constraint:* \( n \geq 1 \).

3: \( x[m \times (n + 1)] \) – double  
**Input/Output**  
*On entry:* the \( m \) data sequences stored in \( x \) consecutively. If the \( n + 1 \) data values of the \( p \)th sequence to be transformed are denoted by \( x^p_j \), for \( j = 0, 1, \ldots, n \) and \( p = 1, 2, \ldots, m \), then the first \( m(n + 1) \) elements of the array \( x \) must contain the values \( x^1_0, x^1_1, \ldots, x^1_n, x^2_0, x^2_1, \ldots, x^2_n, \ldots, x^m_0, x^m_1, \ldots, x^m_n \).  
*On exit:* the \( m \) Fourier cosine transforms stored consecutively, overwriting the corresponding original sequence.

4: \( \text{trig}[2 \times n] \) – const double  
**Input**  
*On entry:* trigonometric coefficients as returned by a call of \( \text{nag_fft_init_trig} \) (c06gzc). \( \text{nag_fft_multiple_cosine} \) (c06hbc) makes a simple check to ensure that \( \text{trig} \) has been initialized and that the initialization is compatible with the value of \( n \).

5: \( \text{fail} \) – NagError *  
**Input/Output**  
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 **Error Indicators and Warnings**

**NE_ALLOC_FAIL**  
Dynamic memory allocation failed.

**NE_C06_NOT_TRIG**  
Value of \( n \) and \( \text{trig} \) array are incompatible or \( \text{trig} \) array not initialized.

**NE_INT_ARG_LT**  
On entry, \( m = \langle \text{value} \rangle \).  
Constraint: \( m \geq 1 \).  
On entry, \( n = \langle \text{value} \rangle \).  
Constraint: \( n \geq 1 \).

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 **Parallelism and Performance**

Not applicable.

9 **Further Comments**

The time taken 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.
10  Example

This program reads in sequences of real data values and prints their Fourier cosine transforms (as computed by nag_fft_multiple_cosine (c06hbc)). It then calls nag_fft_multiple_cosine (c06hbc) again and prints the results which may be compared with the original sequence.

10.1 Program Text

/* nag_fft_multiple_cosine (c06hbc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 2, 1991. */
/* Mark 8 revised, 2004. */

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

#define X(I, J) x[(I) *row_len + (J)]

int main(void)
{
  Integer exit_status = 0, i, j, m, n, row_len;
  NagError fail;
  double *trig = 0, *x = 0;
  INIT_FAIL(fail);
  printf("nag_fft_multiple_cosine (c06hbc) Example Program Results\n");
  #ifdef _WIN32
  scanf_s(" %*[\n]"); /* Skip heading in data file */
  #else
  scanf(" %*[\n]"); /* Skip heading in data file */
  #endif
  #ifdef _WIN32
  scanf_s("%NAG_IFMT" "%NAG_IFMT"", &m, &n) != EOF)
  #else
  scanf("%NAG_IFMT" "%NAG_IFMT"", &m, &n) != EOF)
  #endif
  if (m >= 1 && n >= 1)
  {
    if (!(trig = NAG_ALLOC(2*n, double)) ||
        !(x = NAG_ALLOC(m*(n+1), double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
  }
  else
  {
    printf("Invalid m or n.\n");
    exit_status = 1;
    return exit_status;
  }
  row_len = n + 1;
  #ifdef _WIN32
  scanf_s(" %*[\n]"); /* Skip text in data file */
  #else
  scanf(" %*[\n]"); /* Skip text in data file */
  #endif
  #ifdef _WIN32
  scanf_s(" %*[\n]"); /* Skip text in data file */
  #else
  scanf(" %*[\n]");
  #endif
  #ifdef _WIN32
  scanf_s(" %*[\n]"); /* Skip text in data file */
  #else
  scanf(" %*[\n]");

Mark 25
for (i = 0; i < m; ++i)
for (j = 0; j < row_len; ++j)
    scanf_s("%lf", &X(i, j));
#else
    scanf("%lf", &X(i, j));
#endif
printf("Original data values\n\n");
for (i = 0; i < m; ++i)
{
    for (j = 0; j < row_len; ++j)
        printf(" %10.4f%s", X(i, j),
                (j%7 == 6 && j != row_len-1?"\n":""));
    printf("\n");
}
/* nag_fft_init_trig (c06gzc).
* Initialization function for other c06 functions
*/
nag_fft_init_trig(n, trig, &fail);
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_fft_init_trig (c06gzc).
%s
", fail.message);
        exit_status = 1;
        goto END;
    }
/* Initialise trig array */
/* nag_fft_multiple_cosine (c06hbc).
* Discrete cosine transform
*/
nag_fft_multiple_cosine(m, n, x, trig, &fail);
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_fft_multiple_cosine (c06hbc).
%s
", fail.message);
        exit_status = 1;
        goto END;
    }
/* Compute transform */
printf("Discrete Fourier cosine transforms\n\n");
for (i = 0; i < m; ++i)
{
    for (j = 0; j < row_len; ++j)
        printf(" %10.4f%s", X(i, j),
                (j%7 == 6 && j != row_len-1?"\n":""));
    printf("\n");
}
/* Compute inverse transform */
/* nag_fft_multiple_cosine (c06hbc), see above. */
nag_fft_multiple_cosine(m, n, x, trig, &fail);
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_fft_multiple_cosine (c06hbc).
%s
", fail.message);
        exit_status = 1;
        goto END;
    }
printf("Original data as restored by inverse transform\n\n");
for (i = 0; i < m; ++i)
{
    for (j = 0; j < row_len; ++j)
        printf(" %10.4f%s", X(i, j),
                (j%7 == 6 && j != row_len-1?"\n":""));
    printf("\n");
}
NAG_FREE(trig);
NAG_FREE(x);
END:
    NAG_FREE(trig);
    NAG_FREE(x);
    return exit_status;
}

10.2 Program Data

nag_fft_multiple_cosine (c06hbc) Example Program Data

3 6 : Number of sequences, m, (number of values in each sequence)-1, n
Real data sequences
  0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  0.9562
  0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  0.4936
  0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  0.2057

10.3 Program Results

nag_fft_multiple_cosine (c06hbc) Example Program Results

Original data values
  0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  0.9562
  0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  0.4936
  0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  0.2057
Discrete Fourier cosine transforms
  1.6833  -0.0482  0.0176  0.1368  0.3240  -0.5830  -0.0427
  1.9605  -0.4884  -0.0655  0.4444  0.0964  0.0856  -0.2289
  1.3838   0.1588  -0.0761  -0.1184  0.3512  0.5759  0.0110
Original data as restored by inverse transform
  0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  0.9562
  0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  0.4936
  0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  0.2057