NAG Library Function Document

nag_sum_fft_cosine (c06rfc)

1 Purpose

nag_sum_fft_cosine (c06rfc) computes the discrete Fourier cosine transforms of \( m \) sequences of real data values. The elements of each sequence and its transform are stored contiguously.

2 Specification

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

3 Description

Given \( m \) sequences of \( n + 1 \) real data values \( x_p^j \), for \( j = 0, 1, \ldots, n \) and \( p = 1, 2, \ldots, m \), nag_sum_fft_cosine (c06rfc) simultaneously calculates the Fourier cosine transforms of all the sequences defined by

\[
\hat{x}_k^p = \sqrt{\frac{2}{n}} \left( x_0^p + \sum_{j=1}^{n-1} x_j^p \cos \left( \frac{j\pi k}{n} \right) + \frac{1}{2} (-1)^k x_n^p \right), \quad k = 0, 1, \ldots, n \text{ and } p = 1, 2, \ldots, m.
\]

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

This transform is also known as type-I DCT.

Since the Fourier cosine transform defined above is its own inverse, two consecutive calls of nag_sum_fft_cosine (c06rfc) will restore the original data.

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

The function uses a variant of the fast Fourier transform (FFT) algorithm (see 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 and 5.

4 References


5 Arguments

1: \( m \) – Integer

*Input*

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

*Constraint:* \( m \geq 1 \).
2:  
   \textbf{n} \hspace{1em} \text{Integer} \hspace{1em} \text{Input}

   \textit{On entry:} one less than the number of real values in each sequence, i.e., the number of values in each sequence is \( n + 1 \).

   \textit{Constraint:} \( n \geq 1 \).

3:  
   \textbf{x}[(\text{n} + 1) \times \text{m}] \hspace{1em} \text{double} \hspace{1em} \text{Input/Output}

   \textit{On entry:} the \( m \) data sequences to be transformed. The \((n + 1)\) data values of the \( p \)th sequence to be transformed, denoted by \( x_p^j \), for \( j = 0, 1, \ldots, n \) and \( p = 1, 2, \ldots, m \), must be stored in \( x[(p - 1) \times (n + 1) + j] \).

   \textit{On exit:} the \( m \) Fourier cosine transforms, overwriting the corresponding original sequences. The \((n + 1)\) components of the \( p \)th Fourier cosine transform, denoted by \( \hat{x}_k^p \), for \( k = 0, 1, \ldots, n \) and \( p = 1, 2, \ldots, m \), are stored in \( x[(p - 1) \times (n + 1) + k] \).

4:  
   \textbf{fail} \hspace{1em} \text{NagError *} \hspace{1em} \text{Input/Output}

   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \ Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}

   Dynamic memory allocation failed.

   See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

   On entry, argument \langle value \rangle had an illegal value.

\textbf{NE_INT}

   On entry, \( m = \langle value \rangle \).

   \textit{Constraint:} \( m \geq 1 \).

   On entry, \( n = \langle value \rangle \).

   \textit{Constraint:} \( n \geq 1 \).

\textbf{NE_INTERNAL_ERROR}

   An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

   An unexpected error has been triggered by this function. Please contact NAG.

   See Section 3.6.6 in the Essential Introduction for further information.

\textbf{NE_NO_LICENCE}

   Your licence key may have expired or may not have been installed correctly.

   See Section 3.6.5 in the Essential Introduction for further information.

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

\texttt{nag_sum_fft_cosine (c06rfc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
9 Further Comments

The time taken by nag_sum_fft_cosine (c06rfc) is approximately proportional to $nm\log(n)$, but also depends on the factors of $n$. nag_sum_fft_cosine (c06rfc) 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. This function internally allocates a workspace of order $O(n)$ double values.

10 Example

This example reads in sequences of real data values and prints their Fourier cosine transforms (as computed by nag_sum_fft_cosine (c06rfc)). It then calls nag_sum_fft_cosine (c06rfc) again and prints the results which may be compared with the original sequence.

10.1 Program Text

/* nag_sum_fft_cosine (c06rfc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 24, 2013.
*/
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagc06.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0, j, m, n, n1;
    /* Arrays */
    double *x = 0;
    char title[60];
    /* Nag Types */
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sum_fft_cosine (c06rfc) Example Program Results\n");
    fflush(stdout);

    /* Read dimensions of array from data file. */
    #ifdef _WIN32
    scanf_s("%*\n\n" NAG_IFMT "%*\n\n", &m, &n1);
    #else
    scanf("%*\n\n" NAG_IFMT "%*\n\n", &m, &n1);
    #endif
    n=n1-1;
    if (!x = NAG_ALLOC((m * n1), double))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read array values from data file and print out. */
    for (j = 0; j < m*n1; j++)
    #ifdef _WIN32
    scanf_s("%lf", &x[j]);
    #else
    scanf("%lf", &x[j]);
    #endif

    /* Compute Fourier cosine transform. */
    ...
```c
#define _WIN32
scanf("%lf", &x[j]);
#endif
#ifndef _WIN32
sprintf_s(title, _countof(title), "\n Original data values\n");
#else
sprintf(title, "\n Original data values\n");
#endif
nag_gen_real_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
    Nag_NonUnitDiag, m, n1, x, n1, "%9.4f",
    title, Nag_NoLabels, 0, Nag_NoLabels,
    0, 80, 0, NULL, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).
    fail.message);
    exit_status = 1;
    goto END;
}
/* nag_sum_fft_cosine (c06rfc).
 * Discrete sine transforms
*/
nag_sum_fft_cosine(m, n, x, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sum_fft_cosine (c06rfc).
    fail.message);
    exit_status = 2;
    goto END;
}
#endif
sprintf_s(title, _countof(title), "\n Discrete Fourier cosine transforms\n");
#else
sprintf(title, "\n Discrete Fourier cosine transforms\n");
#endif
nag_gen_real_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
    Nag_NonUnitDiag, m, n1, x, n1, "%9.4f",
    title, Nag_NoLabels, 0, Nag_NoLabels,
    0, 80, 0, NULL, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).
    fail.message);
    exit_status = 3;
    goto END;
}
/* Two consecutive calls should restore the original data. */
nag_sum_fft_cosine(m, n, x, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sum_fft_cosine (c06rfc).
    fail.message);
    exit_status = 4;
    goto END;
}
#endif
sprintf_s(title, _countof(title), "\n Original data as restored by inverse transform\n");
#else
sprintf(title, "\n Original data as restored by inverse transform\n");
#endif
nag_gen_real_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
    Nag_NonUnitDiag, m, n1, x, n1, "%9.4f",
    title, Nag_NoLabels, 0, Nag_NoLabels,
    0, 80, 0, NULL, &fail);
if (fail.code != NE_NOERROR)
{
    
```
```c
printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n", fail.message);
exit_status = 5;
goto END;
}
END:
NAG_FREE(x);
return exit_status;
}

10.2 Program Data

nag_sum_fft_cosine (c06rfc) Example Program Data

3 7 : m, n+1

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3854</td>
<td>0.6772</td>
<td>0.1138</td>
<td>0.6751</td>
<td>0.6362</td>
<td>0.1424</td>
<td>0.9562</td>
</tr>
<tr>
<td>0.5417</td>
<td>0.2983</td>
<td>0.1181</td>
<td>0.7255</td>
<td>0.8638</td>
<td>0.8723</td>
<td>0.4936</td>
</tr>
<tr>
<td>0.9172</td>
<td>0.0644</td>
<td>0.6037</td>
<td>0.6430</td>
<td>0.0428</td>
<td>0.4815</td>
<td>0.2057</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_sum_fft_cosine (c06rfc) Example Program Results

Original data values

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3854</td>
<td>0.6772</td>
<td>0.1138</td>
<td>0.6751</td>
<td>0.6362</td>
<td>0.1424</td>
<td>0.9562</td>
</tr>
<tr>
<td>0.5417</td>
<td>0.2983</td>
<td>0.1181</td>
<td>0.7255</td>
<td>0.8638</td>
<td>0.8723</td>
<td>0.4936</td>
</tr>
<tr>
<td>0.9172</td>
<td>0.0644</td>
<td>0.6037</td>
<td>0.6430</td>
<td>0.0428</td>
<td>0.4815</td>
<td>0.2057</td>
</tr>
</tbody>
</table>

Discrete Fourier cosine transforms

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6833</td>
<td>-0.0482</td>
<td>0.0176</td>
<td>0.1368</td>
<td>0.3240</td>
<td>-0.5830</td>
<td>-0.0427</td>
</tr>
<tr>
<td>1.9605</td>
<td>-0.4884</td>
<td>-0.0655</td>
<td>0.4444</td>
<td>0.0964</td>
<td>0.0856</td>
<td>-0.2289</td>
</tr>
<tr>
<td>1.3838</td>
<td>0.1588</td>
<td>-0.0761</td>
<td>-0.1184</td>
<td>0.3512</td>
<td>0.5759</td>
<td>0.0110</td>
</tr>
</tbody>
</table>

Original data as restored by inverse transform

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3854</td>
<td>0.6772</td>
<td>0.1138</td>
<td>0.6751</td>
<td>0.6362</td>
<td>0.1424</td>
<td>0.9562</td>
</tr>
<tr>
<td>0.5417</td>
<td>0.2983</td>
<td>0.1181</td>
<td>0.7255</td>
<td>0.8638</td>
<td>0.8723</td>
<td>0.4936</td>
</tr>
<tr>
<td>0.9172</td>
<td>0.0644</td>
<td>0.6037</td>
<td>0.6430</td>
<td>0.0428</td>
<td>0.4815</td>
<td>0.2057</td>
</tr>
</tbody>
</table>
```