NAG Library Function Document

nag_fft_multiple_real (c06fpc)

1 Purpose

nag_fft_multiple_real (c06fpc) computes the discrete Fourier transforms of $m$ sequences, each containing $n$ real data values.

2 Specification

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

3 Description

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

$$ z_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j^p \exp(-2\pi i j k / n), \quad \text{for} \quad k = 0, 1, \ldots, n-1; p = 1, 2, \ldots, m. $$

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

The transformed values $z_k^p$ are complex, but for each value of $p$ the $z_k^p$ form a Hermitian sequence (i.e., $z_{n-k}^p$ is the complex conjugate of $z_k^p$), so they are completely determined by $mn$ real numbers. The first call of nag_fft_multiple_real (c06fpc) must be preceded by a call to nag_fft_init_trig (c06gzc) to initialize the array $\text{trig}$ with trigonometric coefficients according to the value of $n$.

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

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

To compute this form, this function should be followed by a call to nag_multiple_conjugate_hermitian (c06gqc) to form the complex conjugates of the $z_k^p$.

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 coding is provided for the factors 2, 3, 4, 5 and 6.

4 References


5 Arguments

1: m – Integer

*Input*

*On entry:* the number of sequences to be transformed, $m$.

*Constraint:* $m \geq 1$. 

Mark 25

Mark 25 c06fpc.1
2: \( n \) – Integer  
*Input*

*On entry:* the number of real values in each sequence, \( n \).

*Constraint:* \( n \geq 1 \).

3: \( x[\text{m} \times \text{n}] \) – double  
*Input/Output*

*On entry:* the \( m \) data sequences must be stored in \( x \) consecutively. If the data values of the \( p \)th sequence to be transformed are denoted by \( x^p_j \), for \( j = 0, 1, \ldots, n - 1 \), then the \( mn \) elements of the array \( x \) must contain the values

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

*On exit:* the \( m \) discrete Fourier transforms in Hermitian form, stored consecutively, overwriting the corresponding original sequences. If the \( n \) components of the discrete Fourier transform \( \tilde{z}_k^p \) are written as \( a_k^p + ib_k^p \), then for \( 0 \leq k \leq n/2 \), \( a_k^p \) is in array element \( x[(p - 1) \times n + k] \) and for \( 1 \leq k \leq (n-1)/2 \), \( b_k^p \) is in array element \( x[(p - 1) \times n + n - k] \).

4: \( \text{trig}[2 \times \text{n}] \) – const double  
*Input*

*On entry:* trigonometric coefficients as returned by a call of \text{nag_fft_init_trig} (c06gzc). \text{nag_fft_multiple_real} (c06fpc) makes a simple check to ensure that \text{trig} has been initialized and that the initialization is compatible with the value of \( n \).

5: \( \text{fail} \) – \text{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 discrete Fourier transforms (as computed by nag_fft_multiple_real (c06fpc)). The Fourier transforms are expanded into full complex form using nag_multiple_hermitian_to_complex (c06gsc) and printed. Inverse transforms are then calculated by calling nag_multiple_conjugate_hermitian (c06gqc) followed by nag_fft_multiple_hermitian (c06fqc) showing that the original sequences are restored.

10.1 Program Text

/* nag_fft_multiple_real (c06fpc) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 1, 1990.
 */

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

int main(void)
{
    Integer exit_status = 0, i, j, m, n;
    NagError fail;
    double *trig = 0, *u = 0, *v = 0, *x = 0;

    INIT_FAIL(fail);

    printf("nag_fft_multiple_real (c06fpc) Example Program Results\n");
    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*[^\n]");
    #else
        scanf("%*[^\n]");
    #endif
    #ifdef _WIN32
        while (scanf_s("%NAG_IFMT%NAG_IFMT", &m, &n) != EOF)
    #else
        while (scanf("%NAG_IFMT%NAG_IFMT", &m, &n) != EOF)
    #endif
    {
        if (m >= 1 && n >= 1)
        {
            if (!(trig = NAG_ALLOC(2*n, double)) ||
                !(u = NAG_ALLOC(m*n, double)) ||
                !(v = NAG_ALLOC(m*n, double)) ||
                !(x = NAG_ALLOC(m*n, double)))
            {
                printf("Allocation failure\n");
                exit_status = -1;
                goto END;
            }
        }
        else
        {
            printf("Invalid m or n.\n");
            exit_status = 1;
            return exit_status;
        }

        printf("\n\nm = %2"NAG_IFMT" n = %2"NAG_IFMT"\n", m, n);
        /* Read in data and print out. */
        for (j = 0; j < m; ++j)
            for (i = 0; i < n; ++i)
                scanf_s("%lf", &x[j*n + i]);

END:

```
#else
    scanf("%lf", &x[j*n + i]);
#endif

printf("\nOriginal data values\n\n");
for (j = 0; j < m; ++j)
{
    printf("  ");
    for (i = 0; i < n; ++i)
        printf("%10.4f%s", x[j*n + i],
            (i%6 == 5 && i != n-1?"\n   ":""));
    printf("\n");
}

/* nag_fft_init_trig (c06gzc).
 * Initialization function for other c06 functions
*/
    nag_fft_init_trig(n, trig, &fail); /* Initialise trig array */
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_fft_init_trig (c06gzc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Calculate transforms */
    /* nag_fft_multiple_real (c06fpc).
     * Multiple one-dimensional real discrete Fourier transforms
    */
    nag_fft_multiple_real(m, n, x, trig, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_fft_multiple_real (c06fpc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    printf("\nDiscrete Fourier transforms in Hermitian format\n\n");
    for (j = 0; j < m; ++j)
    {
        printf("  ");
        for (i = 0; i < n; ++i)
            printf("%10.4f%s", x[j*n + i],
                (i%6 == 5 && i != n-1?"\n   ":""));
        printf("\n");
    }

    /* Calculate full complex form of Hermitian result */
    /* nag_multiple_hermitian_to_complex (c06gsc).
     * Convert Hermitian sequences to general complex sequences
    */
    nag_multiple_hermitian_to_complex(m, n, x, u, v, &fail);
    printf("\nFourier transforms in full complex form\n\n");
    for (j = 0; j < m; ++j)
    {
        printf("Real");
        for (i = 0; i < n; ++i)
            printf("%10.4f", u[j*n + i],
                (i%6 == 5 && i != n-1?"\n   ":""));
        printf("\nImag");
        for (i = 0; i < n; ++i)
            printf("%10.4f", v[j*n + i],
                (i%6 == 5 && i != n-1?"\n   ":""));
        printf("\n\n");
    }

    /* Calculate inverse transforms */
    /* Conjugate Hermitian sequences of transforms */
    /* nag_multiple_conjugate_hermitian (c06gqc).
     * Complex conjugate of multiple Hermitian sequences
    */
    nag_multiple_conjugate_hermitian(m, n, x, &fail);
    /* Transform to give inverse transforms */
    /* nag_fft_multiple_hermitian (c06fqc).
     * Multiple one-dimensional Hermitian discrete Fourier
     * transforms
    */
c06 – Fourier Transforms

c06fpc

* transforms
*/

nag_fft_multiple_hermitian(m, n, x, trig, &fail);

printf("\nOriginal data as restored by inverse transform\n\n");

for (j = 0; j < m; ++j)
{
    printf(" ");
    for (i = 0; i < n; ++i)
    {
        printf("%10.4f%s", x[j*n + i],
            (i%6 == 5 && i != n-1?"\n ":""));
        printf("\n");
    }
}

END:
NAG_FREE(trig);
NAG_FREE(u);
NAG_FREE(v);
NAG_FREE(x);

return exit_status;
}

10.2 Program Data

nag_fft_multiple_real (c06fpc) Example Program Data

```
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
```

10.3 Program Results

nag_fft_multiple_real (c06fpc) 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

Discrete Fourier transforms in Hermitian format

1.0737  -0.1041  0.1126  -0.1467  -0.3738  -0.0044
1.3961  -0.0365  0.0780  -0.1521  -0.0607  0.4666
1.1237  0.0914  0.3936  0.1530  0.3458  -0.0508

Fourier transforms in full complex form

Real 1.0737  -0.1041  0.1126  -0.1467  0.1126  -0.1041
Imag 0.0000  -0.0044  -0.3738  0.0000  0.3738  0.0044

Real 1.3961  -0.0365  0.0780  -0.1521  0.0780  -0.0365
Imag 0.0000  0.4666  -0.0607  0.0000  0.0607  -0.4666

Real 1.1237  0.0914  0.3936  0.1530  0.3936  0.0914
Imag 0.0000  -0.0508  0.3458  0.0000  -0.3458  0.0508

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
```