NAG Library Function Document
nag_fft_multiple_complex (c06frc)

1 Purpose
nag_fft_multiple_complex (c06frc) computes the discrete Fourier transforms of \( m \) sequences, each containing \( n \) complex data values.

2 Specification
#include <nag.h>
#include <nagc06.h>
void nag_fft_multiple_complex (Integer m, Integer n, double x[], double y[],
const double trig[], NagError *fail)

3 Description
Given \( m \) sequences of \( n \) complex data values \( z^p_j \), 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
\[
\hat{z}^p_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z^p_j \exp(-2\pi ijk/n), \quad \text{for } k = 0, 1, \ldots, n - 1; p = 1, 2, \ldots, m.
\]
(Note the scale factor \( 1/\sqrt{n} \) in this definition.)
The first call of nag_fft_multiple_complex (c06frc) must be preceded by a call to nag_fft_init_trig (c06gzc) to initialize the array \( \text{trig} \) with trigonometric coefficients.
The discrete Fourier transform is sometimes defined using a positive sign in the exponential term
\[
\hat{z}^p_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z^p_j \exp(+2\pi ijk/n).
\]
To compute this form, this function should be preceded and followed by a call of nag_conjugate_complex (c06gcc) to form the complex conjugates of the \( z^p_j \) and the \( \hat{z}^p_k \).
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 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: \( m \geq 1. \)
2: n – Integer  

*Input*  

*On entry:* the number of complex values in each sequence, n.

*Constraint:* n ≥ 1.

3: x[m × n] – double  

*Input/Output*  

*On entry:* the real and imaginary parts of the complex data must be stored in x and y respectively. Each of the m sequences must be stored consecutively; hence if the real parts of the pth sequence to be transformed are denoted by x^p_j for j = 0, 1, ..., n – 1, then the mn elements of the array x must contain the values

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

The imaginary parts must be ordered similarly in y.

*On exit:* x and y are overwritten by the real and imaginary parts of the complex transforms.

4: y[m × n] – double  

*Input/Output*  

*On entry:* the real and imaginary parts of the complex data must be stored in x and y respectively. Each of the m sequences must be stored consecutively; hence if the real parts of the pth sequence to be transformed are denoted by x^p_j for j = 0, 1, ..., n – 1, then the mn elements of the array x must contain the values

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

The imaginary parts must be ordered similarly in y.

*On exit:* x and y are overwritten by the real and imaginary parts of the complex transforms.

5: trig[2 × n] – const double  

*Input*  

*On entry:* trigonometric coefficients as returned by a call of nag_fft_init_trig (c06gzc). nag_fft_multiple_complex (c06frc) makes a simple check to ensure that trig has been initialized and that the initialization is compatible with the value of n.

6: 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 trig array are incompatible or trig array not initialized.

**NE_INT_ARG_LT**  
On entry, m = ⟨value⟩.  
Constraint: m ≥ 1.

On entry, n = ⟨value⟩.  
Constraint: n ≥ 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 complex data values and prints their discrete Fourier transforms (as computed by nag_fft_multiple_complex (c06frc)). Inverse transforms are then calculated using nag_conjugate_complex (c06gcc) and nag_fft_multiple_complex (c06frc) and printed out, showing that the original sequences are restored.

10.1 Program Text

/* nag_fft_multiple_complex (c06frc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 1, 1990. */
/* Mark 3 revised, 1994. */
/* Mark 8 revised, 2004. */

#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, *x = 0, *y = 0;

    INIT_FAIL(fail);

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
    printf("nag_fft_multiple_complex (c06frc) Example Program Results\n");

    #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)) ||
            !(x = NAG_ALLOC(m*n, double)) ||
            !(y = 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);
        for (j = 0; j < m; ++j)
        {
            #ifdef _WIN32
            scanf_s("%lf", &x[j*n + i]);
            #else
            scanf("%lf", &x[j*n + i]);
            #endif
            //...
        }
    }

    #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);
    for (j = 0; j < m; ++j)
    {
        //...
    }
    #endif
}

END:
exit_status;
return exit_status;
```c
scanf("%lf", &x[i*n + j]);
#endif
for (i = 0; i < n; ++i)
#endif
scanf_s("%lf", &y[i*n + j]);
#else
scanf("%lf", &y[i*n + j]);
#endif
}
printf("\nOriginal data values\n\n");
for (j = 0; j < m; ++j)
{
printf("Real\n");
for (i = 0; i < n; ++i)
printf("%10.4f%s", x[i*n + j],
    (i%6 == 5 && i != n-1?"\n "":""));
printf("\nImag\n");
for (i = 0; i < n; ++i)
printf("%10.4f%s", y[i*n + j],
    (i%6 == 5 && i != n-1?"\n "":""));
printf("\n\n");
}
/* Initialise trig array */
/* nag_fft_init_trig (c06gzc). */
Initialization function for other C06 functions */
if (fail.code != NE_NOERROR)
{
printf("Error from nag_fft_init_trig (c06gzc).\n\n", fail.message);
exit_status = 1;
goto END;
}
/* Compute transforms */
/* nag_fft_multiple_complex (c06frc). */
Multiple one-dimensional complex discrete Fourier transforms */
nag_fft_multiple_complex(m, n, x, y, trig, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_fft_multiple_complex (c06frc).\n\n", fail.message);
exit_status = 1;
goto END;
}
printf("\nDiscrete Fourier transforms\n\n");
for (j = 0; j < m; ++j)
{
printf("Real\n");
for (i = 0; i < n; ++i)
printf("%10.4f%s", x[i*n + j],
    (i%6 == 5 && i != n-1?"\n "":""));
printf("\nImag\n");
for (i = 0; i < n; ++i)
printf("%10.4f%s", y[i*n + j],
    (i%6 == 5 && i != n-1?"\n "":""));
printf("\n\n");
}
/* Compute inverse transforms */
/* nag_conjugate_complex (c06gcc). */
Complex conjugate of complex sequence */
nag_conjugate_complex(m*n, y, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_conjugate_complex (c06gcc).\n\n", fail.message);
```
exit_status = 1;
goto END;
}

/* nag_fft_multiple_complex (c06frc), see above. */
nag_fft_multiple_complex(m, n, x, y, trig, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_fft_multiple_complex (c06frc).\n%s\n",
fail.message);
    exit_status = 1;
goto END;
}

/* nag_conjugate_complex (c06gcc), see above. */
nag_conjugate_complex(m*n, y, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_conjugate_complex (c06gcc).\n%s\n",
fail.message);
    exit_status = 1;
goto END;
}

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

END:
NAG_FREE(trig);
NAG_FREE(x);
NAG_FREE(y);
return exit_status;

10.2 Program Data

nag_fft_multiple_complex (c06frc) Example Program Data

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Real</th>
<th></th>
<th>Imag</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>0.3854</td>
<td></td>
<td>0.3854</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.6772</td>
<td></td>
<td>0.6772</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1138</td>
<td></td>
<td>0.1138</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.6751</td>
<td></td>
<td>0.6751</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.6362</td>
<td></td>
<td>0.6362</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1424</td>
<td></td>
<td>0.1424</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5417</td>
<td></td>
<td>0.5417</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.2983</td>
<td></td>
<td>0.2983</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1181</td>
<td></td>
<td>0.1181</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.7255</td>
<td></td>
<td>0.7255</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8638</td>
<td></td>
<td>0.8638</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8723</td>
<td></td>
<td>0.8723</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.9172</td>
<td></td>
<td>0.9172</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0644</td>
<td></td>
<td>0.0644</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.6037</td>
<td></td>
<td>0.6037</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.6430</td>
<td></td>
<td>0.6430</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0428</td>
<td></td>
<td>0.0428</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4815</td>
<td></td>
<td>0.4815</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_fft_multiple_complex (c06frc) Example Program Results

m = 3  n = 6
Original data values

<table>
<thead>
<tr>
<th>Real</th>
<th></th>
<th>Imag</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3854</td>
<td></td>
<td>0.3854</td>
<td></td>
</tr>
<tr>
<td>0.6772</td>
<td></td>
<td>0.6772</td>
<td></td>
</tr>
<tr>
<td>0.1138</td>
<td></td>
<td>0.1138</td>
<td></td>
</tr>
<tr>
<td>0.6751</td>
<td></td>
<td>0.6751</td>
<td></td>
</tr>
<tr>
<td>0.6362</td>
<td></td>
<td>0.6362</td>
<td></td>
</tr>
<tr>
<td>0.1424</td>
<td></td>
<td>0.1424</td>
<td></td>
</tr>
<tr>
<td>0.5417</td>
<td></td>
<td>0.5417</td>
<td></td>
</tr>
<tr>
<td>0.2983</td>
<td></td>
<td>0.2983</td>
<td></td>
</tr>
<tr>
<td>0.1181</td>
<td></td>
<td>0.1181</td>
<td></td>
</tr>
<tr>
<td>0.7255</td>
<td></td>
<td>0.7255</td>
<td></td>
</tr>
<tr>
<td>0.8638</td>
<td></td>
<td>0.8638</td>
<td></td>
</tr>
<tr>
<td>0.8723</td>
<td></td>
<td>0.8723</td>
<td></td>
</tr>
<tr>
<td>0.9172</td>
<td></td>
<td>0.9172</td>
<td></td>
</tr>
<tr>
<td>0.0644</td>
<td></td>
<td>0.0644</td>
<td></td>
</tr>
<tr>
<td>0.6037</td>
<td></td>
<td>0.6037</td>
<td></td>
</tr>
<tr>
<td>0.6430</td>
<td></td>
<td>0.6430</td>
<td></td>
</tr>
<tr>
<td>0.0428</td>
<td></td>
<td>0.0428</td>
<td></td>
</tr>
<tr>
<td>0.4815</td>
<td></td>
<td>0.4815</td>
<td></td>
</tr>
</tbody>
</table>
### Discrete Fourier transforms

<table>
<thead>
<tr>
<th></th>
<th>Real</th>
<th>Imag</th>
<th>Real</th>
<th>Imag</th>
<th>Real</th>
<th>Imag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.0737</td>
<td>-0.5706</td>
<td>0.1733</td>
<td>-0.1467</td>
<td>0.0518</td>
<td>0.3625</td>
</tr>
<tr>
<td></td>
<td>1.3961</td>
<td>-0.0409</td>
<td>-0.2958</td>
<td>-0.1521</td>
<td>0.4517</td>
<td>-0.0321</td>
</tr>
<tr>
<td></td>
<td>1.1237</td>
<td>0.1728</td>
<td>0.4185</td>
<td>0.1530</td>
<td>0.3686</td>
<td>0.0101</td>
</tr>
<tr>
<td></td>
<td>1.0677</td>
<td>0.0386</td>
<td>0.7481</td>
<td>0.1752</td>
<td>0.0565</td>
<td>0.1403</td>
</tr>
<tr>
<td></td>
<td>0.9100</td>
<td>-0.3054</td>
<td>0.4079</td>
<td>-0.0785</td>
<td>-0.1193</td>
<td>-0.5314</td>
</tr>
<tr>
<td></td>
<td>1.7617</td>
<td>0.0624</td>
<td>-0.0695</td>
<td>0.0725</td>
<td>0.1285</td>
<td>-0.4335</td>
</tr>
</tbody>
</table>

### Original data as restored by inverse transform

<table>
<thead>
<tr>
<th></th>
<th>Real</th>
<th>Imag</th>
<th>Real</th>
<th>Imag</th>
<th>Real</th>
<th>Imag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.3854</td>
<td>0.6772</td>
<td>0.1138</td>
<td>0.6751</td>
<td>0.6362</td>
<td>0.1424</td>
</tr>
<tr>
<td></td>
<td>0.5417</td>
<td>0.2983</td>
<td>0.1181</td>
<td>0.7255</td>
<td>0.8638</td>
<td>0.8723</td>
</tr>
<tr>
<td></td>
<td>0.9172</td>
<td>0.0644</td>
<td>0.6037</td>
<td>0.6430</td>
<td>0.0428</td>
<td>0.4815</td>
</tr>
<tr>
<td></td>
<td>0.9089</td>
<td>0.3118</td>
<td>0.3465</td>
<td>0.6198</td>
<td>0.2668</td>
<td>0.1614</td>
</tr>
<tr>
<td></td>
<td>0.1156</td>
<td>0.0685</td>
<td>0.2060</td>
<td>0.8630</td>
<td>0.6967</td>
<td>0.2792</td>
</tr>
<tr>
<td></td>
<td>0.6214</td>
<td>0.8681</td>
<td>0.7060</td>
<td>0.8652</td>
<td>0.9190</td>
<td>0.3355</td>
</tr>
</tbody>
</table>