NAG Library Function Document

nag_sum_fft_complex_1d (c06pcc)

1 Purpose
nag_sum_fft_complex_1d (c06pcc) calculates the discrete Fourier transform of a sequence of \( n \) complex data values (using complex data type).

2 Specification
#include <nag.h>
#include <nagc06.h>
void nag_sum_fft_complex_1d (Nag_TransformDirection direct, Complex x[],
Integer n, NagError *fail)

3 Description
Given a sequence of \( n \) complex data values \( z_j \) for \( j = 0,1,\ldots,n-1 \), nag_sum_fft_complex_1d (c06pcc) calculates their (forward or backward) discrete Fourier transform (DFT) defined by

\[
\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \exp\left(\pm \frac{2\pi i j k}{n}\right), \quad k = 0,1,\ldots,n-1.
\]

(Note the scale factor of \( \frac{1}{\sqrt{n}} \) in this definition.) The minus sign is taken in the argument of the exponential within the summation when the forward transform is required, and the plus sign is taken when the backward transform is required.

A call of nag_sum_fft_complex_1d (c06pcc) with \( \text{direct} = \text{Nag\_ForwardTransform} \) followed by a call with \( \text{direct} = \text{Nag\_BackwardTransform} \) will restore the original data.

nag_sum_fft_complex_1d (c06pcc) uses a variant of the fast Fourier transform (FFT) algorithm (see Brigham (1974)) known as the Stockham self-sorting algorithm, which is described in Temperton (1983). If \( n \) is a large prime number or if \( n \) contains large prime factors, then the Fourier transform is performed using Bluestein’s algorithm (see Bluestein (1968)), which expresses the DFT as a convolution that in turn can be efficiently computed using FFTs of highly composite sizes.

4 References
Bluestein L I (1968) A linear filtering approach to the computation of the discrete Fourier transform 
Northeast Electronics Research and Engineering Meeting Record 10 218–219

5 Arguments
1: \textbf{direct} – Nag\_TransformDirection

\textit{Input}

\textit{On entry:} if the forward transform as defined in Section 3 is to be computed, then \textbf{direct} must be set equal to Nag\_ForwardTransform.

If the backward transform is to be computed then \textbf{direct} must be set equal to Nag\_BackwardTransform.

\textit{Constraint:} \textbf{direct} = Nag\_ForwardTransform or Nag\_BackwardTransform.
2: \( \mathbf{x}[\mathbf{n}] \) – Complex \hspace{1cm} \text{Input/Output}

*On entry:* \( \mathbf{x}[j] \) must contain \( z_j \), for \( j = 0, 1, \ldots, n - 1 \).

*On exit:* the components of the discrete Fourier transform. \( \hat{z}_k \) is contained in \( \mathbf{x}[k] \), for \( 0 \leq k \leq n - 1 \).

3: \textbf{n} – Integer \hspace{1cm} \text{Input}

*On entry:* \( n \), the number of data values.

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

4: \textbf{fail} – NagError * 

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

6 \hspace{0.5cm} \textbf{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 \( \langle \text{value} \rangle \) had an illegal value.

\( \langle \text{value} \rangle \) is an invalid value of \textbf{direct}.

\textbf{NE_INT}

On entry, \( \textbf{n} = \langle \text{value} \rangle \).

*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 \hspace{0.5cm} \textbf{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 \hspace{0.5cm} \textbf{Parallelism and Performance}

\texttt{nag\_sum\_fft\_complex\_1d (c06pcc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag\_sum\_fft\_complex\_1d (c06pcc)} makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.
Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken is approximately proportional to $n \times \log(n)$, but also depends on the factorization of $n$. nag_sum_fft_complex_1d (c06pcc) is faster if the only prime factors of $n$ are 2, 3 or 5; and fastest of all if $n$ is a power of 2. This function internally allocates a workspace of $2n + 15$ Complex values. When the Bluestein’s FFT algorithm is in use, an additional Complex workspace of size approximately $8n$ is allocated.

10 Example

This example reads in a sequence of complex data values and prints their discrete Fourier transform (as computed by nag_sum_fft_complex_1d (c06pcc) with direct = Nag_ForwardTransform). It then performs an inverse transform using nag_sum_fft_complex_1d (c06pcc) with direct = Nag_BackwardTransform, and prints the sequence so obtained alongside the original data values.

10.1 Program Text

/* nag_sum_fft_complex_1d (c06pcc) 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, i, n;
    /* Arrays */
    Complex *x = 0, *z = 0, *x_back = 0;
    /* Nag Types */
    NagError fail;
    #ifdef NAG_LOAD_FP
    /* The following line is needed to force the Microsoft linker 
     * to load floating point support */
    float force_loading_of_ms_float_support = 0;
    #endif /* NAG_LOAD_FP */
    INIT_FAIL(fail);
    printf("nag_sum_fft_complex_1d (c06pcc) Example Program Results\n");
    /* Read dimensions of array and array values from data file. */
    /* */
    #ifdef _WIN32
    scanf_s("%*[\n] %"NAG_IFMT "%*[\n]", &n);
    #else
    scanf("%*[\n] %"NAG_IFMT "%*[\n]", &n);
    #endif
    if (!(x = NAG_ALLOC(n, Complex)) ||
        !(z = NAG_ALLOC(n, Complex)) ||
        !(x_back = NAG_ALLOC(n, Complex)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* ... of program body ... */

"
for (i = 0; i < n; ++i) {
    #ifdef _WIN32
        scanf_s(" ( %lf, %lf )", &x[i].re, &x[i].im);
    #else
        scanf(" ( %lf, %lf )", &x[i].re, &x[i].im);
    #endif
    z[i] = x[i];
}

/* Compute discrete Fourier transform of complex array x using
   * nag_sum_fft_complex_1d (c06pcc).
*/
    nag_sum_fft_complex_1d(Nag_ForwardTransform, z, n, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_sum_fft_complex_1d (c06pcc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    for (i = 0; i < n; ++i)
        x_back[i] = z[i];

/* Compute inverse discrete Fourier transform of complex array z using
   * nag_sum_fft_complex_1d (c06pcc).
*/
    nag_sum_fft_complex_1d(Nag_BackwardTransform, x_back, n, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_sum_fft_complex_1d (c06pcc).\n%s\n", fail.message);
        exit_status = 2;
        goto END;
    }

    printf("%2s%13s%28s%22s\n","i","x","z = FFT(x)","InvFFT(z)");
    for (i=0;i<n;i++)
        printf(" %NAG_IFMT%8.5f, %8.5f ) (%8.5f, %8.5f ) (%8.5f, %8.5f )\n", i, x[i].re, x[i].im, z[i].re, z[i].im, x_back[i].re, x_back[i].im);

END:
    NAG_FREE(x);
    NAG_FREE(z);
    NAG_FREE(x_back);

    return exit_status;
}

10.2 Program Data

nag_sum_fft_complex_1d (c06pcc) Example Program Data

<table>
<thead>
<tr>
<th>n</th>
<th>x[0:n-1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>0.34907, -0.37168</td>
<td></td>
</tr>
<tr>
<td>0.54890, -0.35669</td>
<td></td>
</tr>
<tr>
<td>0.74776, -0.31175</td>
<td></td>
</tr>
<tr>
<td>0.94459, -0.23702</td>
<td></td>
</tr>
<tr>
<td>1.13850, -0.13274</td>
<td></td>
</tr>
<tr>
<td>1.32850, 0.00074</td>
<td></td>
</tr>
<tr>
<td>1.51370, 0.16298</td>
<td></td>
</tr>
</tbody>
</table>
### 10.3 Program Results

**nag_sum_fft_complex_1d (c06pcc) Example Program Results**

<table>
<thead>
<tr>
<th>i</th>
<th>x</th>
<th>z = FFT(x)</th>
<th>InvFFT(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0.34907, -0.37168)</td>
<td>(2.48361, -0.47100)</td>
<td>(0.34907, -0.37168)</td>
</tr>
<tr>
<td>1</td>
<td>(0.54890, -0.35669)</td>
<td>(-0.55180, 0.49684)</td>
<td>(0.54890, -0.35669)</td>
</tr>
<tr>
<td>2</td>
<td>(0.74776, -0.31175)</td>
<td>(-0.36711, 0.09756)</td>
<td>(0.74776, -0.31175)</td>
</tr>
<tr>
<td>3</td>
<td>(0.94459, -0.23702)</td>
<td>(-0.28767, -0.05865)</td>
<td>(0.94459, -0.23702)</td>
</tr>
<tr>
<td>4</td>
<td>(1.13850, -0.13274)</td>
<td>(-0.22506, -0.17477)</td>
<td>(1.13850, -0.13274)</td>
</tr>
<tr>
<td>5</td>
<td>(1.32850, 0.00074)</td>
<td>(-0.14825, -0.30840)</td>
<td>(1.32850, 0.00074)</td>
</tr>
<tr>
<td>6</td>
<td>(1.51370, 0.16298)</td>
<td>(0.01983, -0.56496)</td>
<td>(1.51370, 0.16298)</td>
</tr>
</tbody>
</table>