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

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

(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 **direct** = Nag_ForwardTransform followed by a call with **direct** = 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

Brigham E O (1974) *The Fast Fourier Transform* Prentice–Hall

Temperton C (1983) Self-sorting mixed-radix fast Fourier transforms *J. Comput. Phys.* **52** 1–23

5 Arguments

1: **direct** – Nag_TransformDirection *Input*

On entry: if the forward transform as defined in Section 3 is to be computed, then **direct** must be set equal to Nag_ForwardTransform.

If the backward transform is to be computed then **direct** must be set equal to Nag_BackwardTransform.

Constraint: **direct** = Nag_ForwardTransform or Nag_BackwardTransform.

2: **x**[**n**] – Complex *Input/Output*

On entry: **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 **x**[k], for $0 \leq k \leq n - 1$.

3: **n** – Integer *Input*

On entry: n, the number of data values.

Constraint: **n** ≥ 1.

4: **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_BAD_PARAM

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

"$\langle$*value*$\rangle$" is an invalid value of **direct**.

NE_INT

On entry, **n** $= \langle$*value*$\rangle$.
Constraint: **n** ≥ 1.

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.

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

nag_sum_fft_complex_1d (c06pcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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 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 2013 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;

  INIT_FAIL(fail);

  printf("nag_sum_fft_complex_1d (c06pcc) Example Program Results\n");

  /* Read dimensions of array and array values from data file. */
  scanf("%*[^\n] %ld%*[^\n]", &n);
  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;
    }
  for (i = 0; i < n; ++i) {
    scanf(" ( %lf, %lf )", &x[i].re, &x[i].im);
    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("\n%2s%13s%28s%22s\n","i","x","z = FFT(x)","InvFFT(z)");
   for (i=0;i<n;i++)
     printf("%21d (%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
    7                        : n
 ( 0.34907,  -0.37168 )
 ( 0.54890,  -0.35669 )
 ( 0.74776,  -0.31175 )
 ( 0.94459,  -0.23702 )
 ( 1.13850,  -0.13274 )
 ( 1.32850,   0.00074 )
 ( 1.51370,   0.16298 )      : x[0:n-1]
```

10.3 Program Results

```
nag_sum_fft_complex_1d (c06pcc) Example Program Results

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