NAG Library Function Document

nag_fft_complex (c06ecc)

1 Purpose

nag_fft_complex (c06ecc) calculates the discrete Fourier transform of a sequence of n complex data values.

2 Specification

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

3 Description

Given a sequence of n complex data values z_j, for $j = 0, 1, \ldots, n-1$, nag_fft_complex (c06ecc) calculates their discrete Fourier transform defined by

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

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

To compute the inverse discrete Fourier transform defined by

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

this function should be preceded and followed by calls of nag_conjugate_complex (c06gcc) to form the complex conjugates of the z_j and the $\hat{z}_k$.

nag_fft_complex (c06ecc) uses the fast Fourier transform (FFT) algorithm (see Brigham (1974)). There are some restrictions on the value of n (see Section 5).

4 References

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

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of data values.

Constraints:

n > 1;
The largest prime factor of **n** must not exceed 19, and the total number of prime factors of **n**, counting repetitions, must not exceed 20.

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

On entry: **x**[j] must contain x_j, the real part of z_j, for $j = 0, 1, \ldots, n-1$.

On exit: the real parts a_k of the components of the discrete Fourier transform. a_k is contained in **x**[k], for $k = 0, 1, \ldots, n-1$.

3: **y**[**n**] – double *Input/Output*

On entry: **y**[j] must contain y_j, the imaginary part of z_j, for $j = 0, 1, \ldots, n-1$.

On exit: the imaginary parts b_k of the components of the discrete Fourier transform. b_k is contained in **y**[k], for $k = 0, 1, \ldots, n-1$.

4: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_C06_FACTOR_GT

At least one of the prime factors of **n** is greater than 19.

NE_C06_TOO_MANY_FACTORS

n has more than 20 prime factors.

NE_INT_ARG_LE

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 $n\log(n)$, but also depends on the factorization of n. nag_fft_complex (c06ecc) is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

On the other hand, nag_fft_complex (c06ecc) is particularly slow if n has several unpaired prime factors, i.e., if the 'square-free' part of n has several factors.

10 Example

This example reads in a sequence of complex data values and prints their discrete Fourier transform. It then performs an inverse transform using nag_fft_complex (c06ecc) and nag_conjugate_complex (c06gcc), and prints the sequence so obtained alongside the original data values.

10.1 Program Text

```
/* nag_fft_complex (c06ecc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 1, 1990.
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
```

```
#include <nag_stdlib.h>
#include <nagc06.h>

int main(void)
{
  Integer  exit_status = 0, j, n;
  NagError fail;
  double   *x = 0, *xx = 0, *y = 0, *yy = 0;

  INIT_FAIL(fail);

  printf("nag_fft_complex (c06ecc) 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"", &n) != EOF)
#else
  while (scanf("%"NAG_IFMT"", &n) != EOF)
#endif
    {
      if (n > 1)
        {
          if (!(x = NAG_ALLOC(n, double)) ||
              !(y = NAG_ALLOC(n, double)) ||
              !(xx = NAG_ALLOC(n, double)) ||
              !(yy = NAG_ALLOC(n, double)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }
        }
      else
        {
          printf("Invalid n.\n");
          exit_status = 1;
          return exit_status;
        }
      for (j = 0; j < n; ++j)
        {
#ifdef _WIN32
          scanf_s("%lf%lf", &x[j], &y[j]);
#else
          scanf("%lf%lf", &x[j], &y[j]);
#endif
          xx[j] = x[j];
          yy[j] = y[j];
        }
      /* Compute transform */
      /* nag_fft_complex (c06ecc).
       * Single one-dimensional complex discrete Fourier transform
       */
      nag_fft_complex(n, x, y, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_fft_complex (c06ecc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
      printf("\nComponents of discrete Fourier transform\n\n");
      printf("         Real       Imag\n\n");
      for (j = 0; j < n; ++j)
        printf("%3"NAG_IFMT" %10.5f %10.5f\n", j, x[j], y[j]);
      /* Compute inverse transform */
      /* Conjugate the transform */
      /* nag_conjugate_complex (c06gcc).
```

```
       * Complex conjugate of complex sequence
       */
      nag_conjugate_complex(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;
        }
      /* Transform */
      /* nag_fft_complex (c06ecc), see above. */
      nag_fft_complex(n, x, y, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_fft_complex (c06ecc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
      /* Conjugate to give inverse transform */
      /* nag_conjugate_complex (c06gcc), see above. */
      nag_conjugate_complex(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 sequence as restored by inverse transform\n");
      printf("\n             Original                 Restored\n");
      printf("         Real       Imag           Real       Imag\n\n");
      for (j = 0; j < n; ++j)
        printf("%3"NAG_IFMT" %10.5f %10.5f     %10.5f %10.5f\n",
                j, xx[j], yy[j], x[j], y[j]);
 END:
      NAG_FREE(x);
      NAG_FREE(y);
      NAG_FREE(xx);
      NAG_FREE(yy);
    }
  return exit_status;
}
```

10.2 Program Data

```
nag_fft_complex (c06ecc) Example Program Data
    7
  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
```

10.3 Program Results

```
nag_fft_complex (c06ecc) Example Program Results

Components of discrete Fourier transform

         Real       Imag

  0    2.48361   -0.47100
  1   -0.55180    0.49684
  2   -0.36711    0.09756
  3   -0.28767   -0.05865
  4   -0.22506   -0.17477
```

```
  5   -0.14825   -0.30840
  6    0.01983   -0.56496

Original sequence as restored by inverse transform

            Original                 Restored
         Real       Imag          Real       Imag

  0    0.34907   -0.37168        0.34907   -0.37168
  1    0.54890   -0.35669        0.54890   -0.35669
  2    0.74776   -0.31175        0.74776   -0.31175
  3    0.94459   -0.23702        0.94459   -0.23702
  4    1.13850   -0.13274        1.13850   -0.13274
  5    1.32850    0.00074        1.32850    0.00074
  6    1.51370    0.16298        1.51370    0.16298
```