NAG Library Function Document

nag_fft_3d (c06pxc)

1 Purpose

nag_fft_3d (c06pxc) computes the three-dimensional discrete Fourier transform of a trivariate sequence of complex data values (using complex data type).

2 Specification

```
#include <nag.h>
#include <nagc06.h>
void nag_fft_3d (Nag_TransformDirection direct, Integer n1, Integer n2,
     Integer n3, Complex x[], NagError *fail)
```

3 Description

nag_fft_3d (c06pxc) computes the three-dimensional discrete Fourier transform of a trivariate sequence of complex data values $z_{j_1 j_2 j_3}$, for $j_1 = 0, 1, \ldots, n_1 - 1$, $j_2 = 0, 1, \ldots, n_2 - 1$ and $j_3 = 0, 1, \ldots, n_3 - 1$.

The discrete Fourier transform is here defined by

$$\hat{z}_{k_1 k_2 k_3} = \frac{1}{\sqrt{n_1 n_2 n_3}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} z_{j_1 j_2 j_3} \times \exp\left(\pm 2\pi i \left(\frac{j_1 k_1}{n_1} + \frac{j_2 k_2}{n_2} + \frac{j_3 k_3}{n_3}\right)\right),$$

where $k_1 = 0, 1, \ldots, n_1 - 1$, $k_2 = 0, 1, \ldots, n_2 - 1$ and $k_3 = 0, 1, \ldots, n_3 - 1$.

(Note the scale factor of $\frac{1}{\sqrt{n_1 n_2 n_3}}$ 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_fft_3d (c06pxc) with **direct** = Nag_ForwardTransform followed by a call with **direct** = Nag_BackwardTransform will restore the original data.

This function performs multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm (see Brigham (1974)).

4 References

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: **n1** – Integer *Input*

On entry: n_1, the first dimension of the transform.

Constraint: **n1** ≥ 1.

3: **n2** – Integer *Input*

On entry: n_2, the second dimension of the transform.

Constraint: **n2** ≥ 1.

4: **n3** – Integer *Input*

On entry: n_3, the third dimension of the transform.

Constraint: **n3** ≥ 1.

5: **x**[**n1** $\times$ **n2** $\times$ **n3**] – Complex *Input/Output*

On entry: the complex data values. Data values are stored in **x** using column-major ordering for storing multidimensional arrays; that is, $z_{j_1 j_2 j_3}$ is stored in **x**$[j_1 + n_1 j_2 + n_1 n_2 j_3]$.

On exit: the corresponding elements of the computed transform.

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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n1** = ⟨*value*⟩.
Constraint: **n1** ≥ 1.

On entry, **n2** = ⟨*value*⟩.
Constraint: **n2** ≥ 1.

On entry, **n3** = ⟨*value*⟩.
Constraint: **n3** ≥ 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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

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 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_fft_3d (c06pxc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_fft_3d (c06pxc) 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 $n_1 n_2 n_3 \times \log(n_1 n_2 n_3)$, but also depends on the factorization of the individual dimensions n_1, n_2 and n_3. nag_fft_3d (c06pxc) is faster if the only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

10 Example

This example reads in a trivariate sequence of complex data values and prints the three-dimensional Fourier transform. It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values.

10.1 Program Text

```
/* nag_fft_3d (c06pxc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2002.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagc06.h>
#include <nagx04.h>

static Integer writex(Integer n1, Integer n2, Integer n3, Complex *x)
{
  /* Routine to print 3D matrix in 2D slices. */
  Integer  k;
  NagError fail;

  INIT_FAIL(fail);

  for (k = 1; k <= n3; k++)
    {
      char title[30];
#ifdef _WIN32
      sprintf_s(title, _countof(title), "X(i,j,k) for k = %" NAG_IFMT, k);
#else
      sprintf(title, "X(i,j,k) for k = %" NAG_IFMT, k);
#endif
      fflush(stdout);
      nag_gen_complx_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                    Nag_NonUnitDiag, n1, n2, &x[(k-1)*n1*n2],
                                    n1,
```

```
                                    Nag_BracketForm, "%6.3f",
                                    title, Nag_NoLabels, 0,
                                    Nag_NoLabels, 0, 90, 0, 0,
                                    &fail);
      printf("\n");
    }
  if (fail.code != NE_NOERROR)
    {
      printf(
        "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
        fail.message);
      return 1;
    }
  return 0;
}

int main(void)
{
  /* Scalars */
  Integer  i, j, k, n1, n2, n3;
  Integer  exit_status = 0;
  NagError fail;
  /* Arrays */
  Complex  *x = 0;
#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 */
#define X(I, J, K) x[(K-1)*n2*n1 + (J-1)*n1 + I - 1]

  INIT_FAIL(fail);

  printf("nag_fft_3d (c06pxc) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"", &n1, &n2, &n3);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"", &n1, &n2, &n3);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  if (n1*n2*n3 >= 1)
    {
      /* Allocate memory */
      if (!(x = NAG_ALLOC(n1 * n2 * n3, Complex)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* Read in complex data and print out. */
      for (k = 1; k <= n3; ++k)
        {
          for (i = 1; i <= n1; ++i)
            {
              for (j = 1; j <= n2; ++j)
                {
#ifdef _WIN32
                  scanf_s(" ( %lf, %lf ) ", &X(i, j, k).re,
                          &X(i, j, k).im);
#else
```

```
                  scanf(" ( %lf, %lf ) ", &X(i, j, k).re,
                          &X(i, j, k).im);
#endif
                }
              }
        }
      printf("\nOriginal data values\n\n");
      exit_status = writex(n1, n2, n3, x);
      if (exit_status != 0)
        {
          goto END;
        }
      /* Compute transform */
      /* nag_fft_3d (c06pxc).
       * Three-dimensional complex discrete Fourier transform,
       * complex data format
       */
      nag_fft_3d(Nag_ForwardTransform, n1, n2, n3, x, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_fft_3d (c06pxc).\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      printf("\nComponents of discrete Fourier transforms\n\n");
      exit_status = writex(n1, n2, n3, x);
      if (exit_status != 0)
        {
          goto END;
        }
      /* Compute inverse transform */
      /* nag_fft_3d (c06pxc), see above. */
      nag_fft_3d(Nag_BackwardTransform, n1, n2, n3, x, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_fft_3d (c06pxc).\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      printf("\nOriginal data as restored by inverse transform\n\n");
      exit_status = writex(n1, n2, n3, x);
    }
  else
    printf("\nInvalid value of n1, n2 or n3.\n");

 END:
  NAG_FREE(x);

  return exit_status;
}
```

10.2 Program Data

```
nag_fft_3d (c06pxc) Example Program Data
2 3 4  : values of n1, n2, n3
   ( 1.000, 0.000)  ( 0.994,-0.111)  ( 0.903,-0.430)
   ( 0.500, 0.500)  ( 0.494, 0.111)  ( 0.403, 0.430)

   ( 0.999,-0.040)  ( 0.989,-0.151)  ( 0.885,-0.466)
   ( 0.499, 0.040)  ( 0.489, 0.151)  ( 0.385, 0.466)

   ( 0.987,-0.159)  ( 0.963,-0.268)  ( 0.823,-0.568)
   ( 0.487, 0.159)  ( 0.463, 0.268)  ( 0.323, 0.568)

   ( 0.936,-0.352)  ( 0.891,-0.454)  ( 0.694,-0.720)
   ( 0.436, 0.352)  ( 0.391, 0.454)  ( 0.194, 0.720)
```

10.3 Program Results

```
nag_fft_3d (c06pxc) Example Program Results

Original data values

 X(i,j,k) for k = 1
   ( 1.000, 0.000)  ( 0.994,-0.111)  ( 0.903,-0.430)
   ( 0.500, 0.500)  ( 0.494, 0.111)  ( 0.403, 0.430)

 X(i,j,k) for k = 2
   ( 0.999,-0.040)  ( 0.989,-0.151)  ( 0.885,-0.466)
   ( 0.499, 0.040)  ( 0.489, 0.151)  ( 0.385, 0.466)

 X(i,j,k) for k = 3
   ( 0.987,-0.159)  ( 0.963,-0.268)  ( 0.823,-0.568)
   ( 0.487, 0.159)  ( 0.463, 0.268)  ( 0.323, 0.568)

 X(i,j,k) for k = 4
   ( 0.936,-0.352)  ( 0.891,-0.454)  ( 0.694,-0.720)
   ( 0.436, 0.352)  ( 0.391, 0.454)  ( 0.194, 0.720)


Components of discrete Fourier transforms

 X(i,j,k) for k = 1
   ( 3.292, 0.102)  ( 0.143,-0.086)  ( 0.143, 0.290)
   ( 1.225,-1.620)  ( 0.424, 0.320)  (-0.424, 0.320)

 X(i,j,k) for k = 2
   ( 0.051,-0.042)  ( 0.016, 0.153)  (-0.050, 0.118)
   ( 0.355, 0.083)  ( 0.020,-0.115)  ( 0.007,-0.080)

 X(i,j,k) for k = 3
   ( 0.113, 0.102)  (-0.024, 0.127)  (-0.024, 0.077)
   ( 0.000, 0.162)  ( 0.013,-0.091)  (-0.013,-0.091)

 X(i,j,k) for k = 4
   ( 0.051, 0.246)  (-0.050, 0.086)  ( 0.016, 0.051)
   (-0.355, 0.083)  (-0.007,-0.080)  (-0.020,-0.115)


Original data as restored by inverse transform

 X(i,j,k) for k = 1
   ( 1.000,-0.000)  ( 0.994,-0.111)  ( 0.903,-0.430)
   ( 0.500, 0.500)  ( 0.494, 0.111)  ( 0.403, 0.430)

 X(i,j,k) for k = 2
   ( 0.999,-0.040)  ( 0.989,-0.151)  ( 0.885,-0.466)
   ( 0.499, 0.040)  ( 0.489, 0.151)  ( 0.385, 0.466)

 X(i,j,k) for k = 3
   ( 0.987,-0.159)  ( 0.963,-0.268)  ( 0.823,-0.568)
   ( 0.487, 0.159)  ( 0.463, 0.268)  ( 0.323, 0.568)

 X(i,j,k) for k = 4
   ( 0.936,-0.352)  ( 0.891,-0.454)  ( 0.694,-0.720)
   ( 0.436, 0.352)  ( 0.391, 0.454)  ( 0.194, 0.720)
```