NAG Library Function Document

nag_sum_fft_complex_2d (c06puc)

1 Purpose

nag_sum_fft_complex_2d (c06puc) computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values (using complex data type).

2 Specification

```
#include <nag.h>
#include <nagc06.h>
void nag_sum_fft_complex_2d (Nag_TransformDirection direct, Integer m,
     Integer n, Complex x[], NagError *fail)
```

3 Description

nag_sum_fft_complex_2d (c06puc) computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values $z_{j_1 j_2}$, for $j_1 = 0, 1, \ldots, m-1$ and $j_2 = 0, 1, \ldots, n-1$.

The discrete Fourier transform is here defined by

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

where $k_1 = 0, 1, \ldots, m-1$ and $k_2 = 0, 1, \ldots, n-1$.

(Note the scale factor of $\frac{1}{\sqrt{mn}}$ 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_2d (c06puc) 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 in Brigham (1974) and Temperton (1983).

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: **m** – Integer *Input*

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

Constraint: $\mathbf{m} \geq 1$.

3: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 1$.

4: **x**[**m** × **n**] – Complex *Input/Output*

On entry: the complex data values. $\mathbf{x}[\mathbf{m} \times j_2 + j_1]$ must contain $z_{j_1 j_2}$, for $j_1 = 0, 1, \ldots, \mathbf{m} - 1$ and $j_2 = 0, 1, \ldots, \mathbf{n} - 1$.

On exit: the corresponding elements of the computed transform.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

⟨*value*⟩ is an invalid value of **direct**.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: $\mathbf{m} \geq 1$.

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 1$.

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 How to Use the NAG Library and its Documentation 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 How to Use the NAG Library and its Documentation 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_sum_fft_complex_2d (c06puc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sum_fft_complex_2d (c06puc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The time taken is approximately proportional to $mn \times \log(mn)$, but also depends on the factorization of the individual dimensions m and n. nag_sum_fft_complex_2d (c06puc) is faster if the only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2. This function internally allocates a workspace of $mn + n + m + 30$ Complex values.

10 Example

This example reads in a bivariate sequence of complex data values and prints the two-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_sum_fft_complex_2d (c06puc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
  /* Scalars */
  Integer exit_status = 0, i, m, n;
  /* Arrays */
  Complex *x = 0;
  char title[60];
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_sum_fft_complex_2d (c06puc) Example Program Results\n");
  fflush(stdout);

  /* Read dimensions of array and array values from data file. */
#ifdef _WIN32
  scanf_s("%*[^\n] %" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m, &n);
#else
  scanf("%*[^\n] %" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m, &n);
#endif

  if (!(x = NAG_ALLOC(m * n, Complex)))
  {
```

```
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  /* The data is read in order as n groups of m complex values. */
  for (i = 0; i < m * 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

  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive).
   * The data can be viewed as an n-by-m matrix stored in row order, or as an
   * m-by-n matrix stored in column order. We choose the former here.
   */
#ifdef _WIN32
  sprintf_s(title, (unsigned)_countof(title), "\n Original data values\n");
#else
  sprintf(title, "\n Original data values\n");
#endif
  nag_gen_complx_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
                                Nag_NonUnitDiag, n, m, x, m, Nag_BracketForm,
                                "%6.3f", title, Nag_NoLabels, 0, Nag_NoLabels,
                                0, 80, 0, NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Compute two-dimensional complex discrete Fourier transform using
   * nag_sum_fft_complex_2d (c06puc) and print out.
   */
  nag_sum_fft_complex_2d(Nag_ForwardTransform, m, n, x, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sum_fft_complex_2d (c06puc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }

#ifdef _WIN32
  sprintf_s(title, (unsigned)_countof(title),
            "\n Components of discrete Fourier transform\n");
#else
  sprintf(title, "\n Components of discrete Fourier transform\n");
#endif
  nag_gen_complx_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
                                Nag_NonUnitDiag, n, m, x, m, Nag_BracketForm,
                                "%6.3f", title, Nag_NoLabels, 0, Nag_NoLabels,
                                0, 80, 0, NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 3;
    goto END;
  }

  /* Compute two-dimensional complex discrete Fourier backward transform using
   * nag_sum_fft_complex_2d (c06puc) and print out.
   */
  nag_sum_fft_complex_2d(Nag_BackwardTransform, m, n, x, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sum_fft_complex_2d (c06puc).\n%s\n", fail.message);
    exit_status = 4;
    goto END;
  }

#ifdef _WIN32
```

```
  sprintf_s(title, (unsigned)_countof(title),
            "\n Original sequence as restored by inverse transform\n");
#else
  sprintf(title, "\n Original sequence as restored by inverse transform\n");
#endif
  nag_gen_complx_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
                                Nag_NonUnitDiag, n, m, x, m, Nag_BracketForm,
                                "%6.3f", title, Nag_NoLabels, 0, Nag_NoLabels,
                                0, 80, 0, NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 5;
  }

END:
  NAG_FREE(x);
  return exit_status;
}
```

10.2 Program Data

```
nag_sum_fft_complex_2d (c06puc) Example Program Data
      3      5                                        : m, n
 ( 1.000, 0.000)  ( 0.994,-0.111)  ( 0.903,-0.430)
 ( 0.999,-0.040)  ( 0.989,-0.151)  ( 0.885,-0.466)
 ( 0.987,-0.159)  ( 0.963,-0.268)  ( 0.823,-0.568)
 ( 0.936,-0.352)  ( 0.891,-0.454)  ( 0.694,-0.720)
 ( 0.802,-0.597)  ( 0.731,-0.682)  ( 0.467,-0.884)    : x
```

10.3 Program Results

```
nag_sum_fft_complex_2d (c06puc) Example Program Results

 Original data values

   ( 1.000, 0.000)  ( 0.994,-0.111)  ( 0.903,-0.430)
   ( 0.999,-0.040)  ( 0.989,-0.151)  ( 0.885,-0.466)
   ( 0.987,-0.159)  ( 0.963,-0.268)  ( 0.823,-0.568)
   ( 0.936,-0.352)  ( 0.891,-0.454)  ( 0.694,-0.720)
   ( 0.802,-0.597)  ( 0.731,-0.682)  ( 0.467,-0.884)

 Components of discrete Fourier transform

   ( 3.373,-1.519)  ( 0.457, 0.137)  (-0.170, 0.493)
   ( 0.481,-0.091)  ( 0.055, 0.032)  (-0.037, 0.058)
   ( 0.251, 0.178)  ( 0.009, 0.039)  (-0.042, 0.008)
   ( 0.054, 0.319)  (-0.022, 0.036)  (-0.038,-0.025)
   (-0.419, 0.415)  (-0.076, 0.004)  (-0.002,-0.083)

 Original sequence as restored by inverse transform

   ( 1.000, 0.000)  ( 0.994,-0.111)  ( 0.903,-0.430)
   ( 0.999,-0.040)  ( 0.989,-0.151)  ( 0.885,-0.466)
   ( 0.987,-0.159)  ( 0.963,-0.268)  ( 0.823,-0.568)
   ( 0.936,-0.352)  ( 0.891,-0.454)  ( 0.694,-0.720)
   ( 0.802,-0.597)  ( 0.731,-0.682)  ( 0.467,-0.884)
```