Module 7.1: nag_fft
Discrete Fourier Transforms

`nag_fft` provides procedures for computations involving the one-dimensional *discrete Fourier transform* of real, Hermitian or complex data values and the two-dimensional and three-dimensional discrete Fourier transform of complex data values. It also provides utility procedures for computations involving Fourier transforms.

Contents

Introduction

1 The Discrete Fourier Transform (DFT)

The one-dimensional DFT of a sequence of n values $z_j, j = 0, 1, \ldots, n-1$, is defined in this module by

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \exp\left(-\frac{2\pi ijk}{n}\right), \tag{1}$$

for $k = 0, 1, \ldots, n-1$. The original values z_j and the transformed values $\hat{z}_k$ are, in general, complex.

The two-dimensional DFT of a bivariate sequence of data values $z_{j_1j_2}$, for $j_1 = 0, 1, \ldots, m-1$; $j_2 = 0, 1, \ldots, n-1$, is defined similarly by

$$\hat{z}_{k_1k_2} = \frac{1}{\sqrt{mn}} \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} z_{j_1j_2} \exp\left(-2\pi i\left(\frac{j_1k_1}{m} + \frac{j_2k_2}{n}\right)\right), \tag{2}$$

for $k_1 = 0, 1, \ldots, m-1$; $k_2 = 0, 1, \ldots, n-1$.

The three-dimensional DFT of a trivariate sequence of data values $z_{j_1j_2j_3}$, where $j_1 = 0, 1, \ldots, m-1$, $j_2 = 0, 1, \ldots, n-1$, $j_3 = 0, 1, \ldots, p-1$, is defined similarly by

$$\hat{z}_{k_1k_2k_3} = \frac{1}{\sqrt{mnp}} \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} \sum_{j_3=0}^{p-1} z_{j_1j_2j_3} \times \exp\left(-2\pi i\left(\frac{j_1k_1}{m} + \frac{j_2k_2}{n} + \frac{j_3k_3}{p}\right)\right), \tag{3}$$

for $k_1 = 0, 1, \ldots, m-1$, $k_2 = 0, 1, \ldots, n-1$, $k_3 = 0, 1, \ldots, p-1$.

The DFT is sometimes defined using a positive sign in the exponential term, which gives the inverse of (1), (2) and (3). Note also the scale factors $1/\sqrt{n}$ in (1), $1/\sqrt{mn}$ in (2) and $1/\sqrt{mnp}$ in (3); often the DFT is defined without these scale factors, but with scale factors $1/n$, $1/mn$ and $1/mnp$ in the inverses.

The procedures in this module evaluate the DFT by using the fast Fourier transform (FFT) algorithm (see Gentleman and Sande [5] and Brigham [1]); for general advice on the use of Fourier transforms see Hamming [4].

2 Types of Sequences

The sequence of data values in (1) can be either real, complex or complex *Hermitian*. Hermitian sequences, $z_j, j = 0, 1, \ldots, n-1$, of length n have the property that z_0 is always real and $z_j = \overline{z}_{n-j}$, for $j = 1, \ldots, n/2$. This means that if n is even $z_{n/2}$ is real. Hermitian sequences are sometimes also called *half-complex* or *conjugate symmetric*.

The following shows how the transformed sequence depends on the original data.

- A complex data sequence gives a complex transformed sequence.
- A Hermitian sequence gives a real transformed sequence.
- A real data sequence gives a Hermitian transformed sequence.

The FFT procedures in this module are classified as either *basic* or *general*. The basic procedures overwrite the Fourier transform on the original input data, and also represent Hermitian sequences with the compact real storage scheme outlined in Section 3. The general procedures do not use compact storage; they are functions which return either the Fourier transform or its inverse.

The general procedures are functions and they are as follows.

- The procedure `nag_fft_1d` computes the transform (1) applied to a single complex sequence or m complex sequences, all of the same length.

- The procedure `nag_fft_1d_real` computes the transform (1) applied to real or Hermitian sequences. It can compute the transform of a single sequence or m sequences, all of the same length.
- The procedure `nag_fft_2d` computes the 2-d transform (2) for a complex bivariate sequence of data values.
- The procedure `nag_fft_3d` computes the 3-d transform (3) for a complex trivariate sequence of data values.

The basic procedures are as follows.

- The procedure `nag_fft_1d_basic` computes the transform (1) applied to complex, real or Hermitian sequences. It can compute the transform of a single sequence or m sequences, all of the same length. It is similar to `nag_fft_1d` and `nag_fft_1d_real` but it is a subroutine that overwrites the Fourier transform on the input data.
- The procedure `nag_fft_2d_basic` computes the same transform as `nag_fft_2d` but it is a subroutine that overwrites the Fourier transform on the input data.
- The procedure `nag_fft_3d_basic` computes the same transform as `nag_fft_3d` but it is a subroutine that overwrites the Fourier transform on the input data.

3 Compact Storage of Hermitian Sequences

The basic procedures in this module represent a Hermitian sequence z_k of length n by using only n real data values. Let $z_k = x_k + iy_k$, x_k and y_k being the real and imaginary parts. For all Hermitian sequences, z_0 is always real; in addition, if n is even, $z_{n/2}$ is also real. If these values are stored in an array z, declared with bounds $(0 : n-1)$, the storage is arranged as follows.

If n is *even*:

index	0	1	2	$\dots$	$n/2$	$\dots$	$n-2$	$n-1$
Sequence	x_0	x_1+iy_1	x_2+iy_2	$\dots$	$x_{n/2}$	$\dots$	x_2-iy_2	x_1-iy_1
Stored values	x_0	x_1	x_2	$\dots$	$x_{n/2}$	$\dots$	y_2	y_1

$$\begin{aligned} \mathtt{z}(k) &= x_k, && \text{for } k = 0,1,\dots,n/2, \text{ and} \\ \mathtt{z}(n-k) &= y_k, && \text{for } k = 1,2,\dots,n/2-1. \end{aligned}$$

If $n = 2s+1$ (i.e., *odd*):

index	0	1	2	$\dots$	s	$s+1$	$\dots$	$n-2$	$n-1$
Sequence	x_0	x_1+iy_1	x_2+iy_2	$\dots$	x_s+iy_s	x_s-iy_s	$\dots$	x_2-iy_2	x_1-iy_1
Stored values	x_0	x_1	x_2	$\dots$	x_s	y_s	$\dots$	y_2	y_1

$$\begin{aligned} \mathtt{z}(k) &= x_k, && \text{for } k = 0,1,\dots,s, \text{ and} \\ \mathtt{z}(n-k) &= y_k, && \text{for } k = 1,2,\dots,s. \end{aligned}$$

Procedures `nag_herm_to_cmplx` and `nag_cmplx_to_herm` convert Hermitian sequences between this compact storage scheme and conventional storage as complex sequences.

4 Inverse Transforms

The inverse transform to (1) is defined as:

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \exp\left(+\frac{2\pi ijk}{n}\right), \quad k = 0,1,\dots,n-1. \tag{4}$$

The *general* procedures `nag_fft_1d`, `nag_fft_1d_real`, `nag_fft_2d` and `nag_fft_3d` are all capable of returning either the forward transform (1) or the inverse transform (4).

If you are using one of the *basic* procedures, and require an inverse transform, you should proceed as follows.

1. Form the complex conjugate of the original sequence.
2. Compute the DFT of the conjugated sequence.
3. Form the complex conjugate of the transformed sequence.

For general complex sequences, the Fortran 90 intrinsic function `CONJG` can be used for steps 1 and 3. (This also applies to two-dimensional transforms.)

If the original sequence is real, step 1 is redundant and the transformed sequence is Hermitian. Step 3 can be performed using `CONJG` if `nag_fft_1d_real` was used in step 2, or using procedure `nag_conj_herm` if `nag_fft_1d_basic` was used in step 2.

If the original sequence is Hermitian, step 3 becomes redundant. Step 1 can be performed using `CONJG` or `nag_conj_herm` depending on the storage scheme used to store the Hermitian data.

5 Trigonometric Coefficients

Computing a DFT involves computation of a number of trigonometric coefficients, which can take a significant proportion of the total CPU-time. The procedures in this module can either compute the trigonometric coefficients internally (in which case they are recomputed at each call) or they allow the coefficients to be pre-computed by `nag_fft_trig` and supplied in an optional argument (which is more efficient if several calls are made to compute transforms of the same length).

Procedure: nag_fft_1d

1 Description

`nag_fft_1d` returns the discrete Fourier transform (DFT), or its inverse, of either a single sequence or m sequences (all of the same length n) of complex data values.

See Section 1 of the Module Introduction for the definition of the DFT, and Section 4 of the Module Introduction for the definition of the inverse transform.

2 Usage

```
USE nag_fft
[value =] nag_fft_1d(z  [, optional arguments])
```

The function result is an array of type complex(kind=*wp*) and the same shape as that of `z`.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '`x`(n)' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of sequences to be transformed

$n \geq 1$ — the number of data values in each sequence

3.1 Mandatory Argument

z(n) / **z**(m, n) — complex(kind=*wp*), intent(in)

Input: if `z` has rank 1, it must hold the single sequence to be transformed. If `z` has rank 2, it must hold the m sequences to be transformed, with each sequence stored in a row of the array.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

inverse — logical, intent(in), optional

Input: specifies whether the inverse transform is to be calculated.

If `inverse` = `.false.`, then the forward transform is calculated;

if `inverse` = `.true.`, then the inverse transform is calculated.

Default: `inverse` = `.false.`.

trig($2n$) — real(kind=*wp*), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of length n.

Default: if `trig` is not present, the coefficients are computed internally.

Constraints: `trig` must have been set by a prior call to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2] and Temperton [3]. Special coding is provided for the cases where n has factors 2, 3, 4, 5, and 6.

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $nm \log n$, but also depends on the factors of n. The procedure is fastest if the only prime factors of n are 2, 3, and 5, and is particularly slow if n is a large prime, or has large prime factors.

If several calls are made to this procedure to compute transforms of the same length n, supplying trigonometric coefficients through the optional argument `trig` results in a saving of CPU-time.

Procedure: nag_fft_1d_real

1 Description

`nag_fft_1d_real` returns the discrete Fourier transform (DFT), or its inverse, of either a single sequence or m sequences (all of the same length n) of real or Hermitian data values.

See Section 1 of the Module Introduction for the definition of the DFT, and Section 4 of the Module Introduction for the definition of the inverse transform.

2 Usage

`USE nag_fft`

`[`*value* `=] nag_fft_1d_real(z [,` *optional arguments*`])`

The function result is an array of the same shape as that of `z`. If `z` is of type real(kind=*wp*) the result is of type complex(kind=*wp*). If `z` is of type complex(kind=*wp*) the result is of type real(kind=*wp*).

2.1 Interfaces

Distinct interfaces are provided for each of the four combinations of the following cases.

Hermitian / real data

Hermitian data:	`z` is of type complex(kind=*wp*) and the result is of type real(kind=*wp*).
Real data:	`z` is of type real(kind=*wp*) and the result is of type complex(kind=*wp*).

Single sequence / multiple sequences

Single sequence:	`z` is a rank-1 array.
Multiple sequences:	`z` is a rank-2 array.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of sequences to be transformed

$n \geq 1$ — the number of data values in each sequence

3.1 Mandatory Argument

z(n) / **z**(m, n) — real(kind=*wp*)/complex(kind=*wp*), intent(in)

Input: if `z` has rank 1, it must hold the single sequence to be transformed. If `z` has rank 2, it must hold the m sequences to be transformed, with each sequence stored in a row of the array.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

inverse — logical, intent(in), optional

Input: specifies whether the inverse transform is to be calculated.

If `inverse` = `.false.`, then the forward transform is calculated;

if `inverse` = `.true.`, then the inverse transform is calculated.

Default: `inverse` = `.false.`.

trig($2n$) — real(kind=wp), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of length n.

Default: if `trig` is not present, the coefficients are computed internally if needed.

Constraints: `trig` must have been set by a prior call to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 3 of this module document.

6 Further Comments

6.1 Algorithmic Detail

For all the different data types, the procedure uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2] and Temperton [3]. Special coding is provided for the cases where n has factors 2, 3, 4, 5, and 6.

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $nm \log n$, but also depends on the factors of n. The procedure is fastest if the only prime factors of n are 2, 3, and 5, and is particularly slow if n is a large prime, or has large prime factors.

The time taken for real or Hermitian sequences is roughly half the time taken for general complex sequences.

If several calls are made to this procedure to compute transforms of the same length n, supplying trigonometric coefficients through the optional argument `trig` results in a saving of CPU-time.

Procedure: nag_fft_1d_basic

1 Description

`nag_fft_1d_basic` is a generic procedure which computes the discrete Fourier transform (DFT) of either a single sequence or m sequences (all of the same length n).

This procedure handles the following types of problem, which are distinguished by the values of the argument `nag_key`.

- `nag_key` = `nag_key_cmplx`: the input sequences and the transformed sequences are general complex sequences; the argument `z` must be of type complex(kind=*wp*).
- `nag_key` = `nag_key_herm`: the input sequences are Hermitian, and the transformed sequences are real; the argument `z` must be of type real(kind=*wp*).
- `nag_key` = `nag_key_real`: the input sequences are real, and the transformed sequences are Hermitian; the argument `z` must be of type real(kind=*wp*).

See Section 1 of the Module Introduction for the definition of the DFT, and Section 2 and Section 3 of the Module Introduction for the definition of Hermitian sequences and their storage. See Section 4 of the Module Introduction for the computation of inverse transforms.

2 Usage

```
USE nag_fft
CALL nag_fft_1d_basic(nag_key, z  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for each of the six combinations of the following cases.

Complex / Hermitian / real data

Complex data:	`nag_key` = `nag_key_cmplx` and `z` is of type complex(kind=*wp*).
Hermitian data:	`nag_key` = `nag_key_herm` and `z` is of type real(kind=*wp*).
Real data:	`nag_key` = `nag_key_real` and `z` is of type real(kind=*wp*).

Single sequence / multiple sequences

Single sequence:	`z` is a rank-1 array.
Multiple sequences:	`z` is a rank-2 array.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of sequences to be transformed
$n \geq 1$ — the number of data values in each sequence

3.1 Mandatory Arguments

nag_key — a 'key' argument, intent(in)

Input: must have one of the following three values (which are named constants, each of a different derived type, defined by the Library, and accessible from this module):

`nag_key_cmplx`: to transform general complex sequences (`z` is complex);

`nag_key_herm`: to transform Hermitian sequences to real sequences (`z` is real);

`nag_key_real`: to transform real sequences to Hermitian sequences (`z` is real).

For further explanation of 'key' arguments, see the Essential Introduction.

z(n) / **z**(m, n) — complex(kind=*wp*)/real(kind=*wp*), intent(inout)

Input: if `z` has rank 1, it must hold the single sequence to be transformed. If `z` has rank 2, it must hold the m sequences to be transformed, with each sequence stored in a row of the array. See Section 3 of the Module Introduction for details of the compact storage of Hermitian sequences (`nag_key` = `nag_key_herm`).

Output: the transformed sequence or sequences, each overwriting the corresponding input sequence. Note that if the input sequences are real (`nag_key` = `nag_key_real`), then the output sequences are Hermitian; if the input sequences are Hermitian (`nag_key` = `nag_key_herm`), then the output sequences are real.

Constraints:

if `nag_key` = `nag_key_cmplx`, `z` must be of type complex(kind=*wp*);

if `nag_key` = `nag_key_real` or `nag_key_herm`, `z` must be of type real(kind=*wp*).

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

trig$(2n)$ — real(kind=*wp*), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of length n.

Default: if `trig` is not present, the coefficients are computed internally.

Constraints: `trig` must have been set by a prior call to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 2, 4 and 5 of this module document.

6 Further Comments

6.1 Algorithmic Detail

For all the different data types, the procedure uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2] and Temperton [3]. Special coding is provided for the cases where n has factors 2, 3, 4, 5, and 6.

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $nm \log n$, but also depends on the factors of n. The procedure is fastest if the only prime factors of n are 2, 3, and 5, and is particularly slow if n is a large prime, or has large prime factors.

The time taken for real or Hermitian sequences is roughly half the time taken for general complex sequences.

If several calls are made to this procedure to compute transforms of the same length n, supplying trigonometric coefficients through the optional argument `trig` results in a saving of CPU-time.

Procedure: nag_fft_2d

1 Description

`nag_fft_2d` returns the two-dimensional discrete Fourier transform (DFT), or its inverse, of a bivariate $m \times n$ sequence of complex data values.

See Section 1 of the Module Introduction for the definition of the DFT, and Section 4 of the Module Introduction for the definition of the inverse transform.

2 Usage

`USE nag_fft`

`[`*value* `=] nag_fft_2d(z  [,` *optional arguments*`])`

The function result is an array of type complex(kind=*wp*) and the same shape as that of `z`.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of rows in the sequence

$n \geq 1$ — the number of columns in the sequence

3.1 Mandatory Argument

z(m, n) — complex(kind=*wp*), intent(in)

Input: the bivariate sequence to be transformed.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

inverse — logical, intent(in), optional

Input: specifies whether the inverse transform is to be calculated.

If `inverse` = `.false.`, then the forward transform is calculated;

if `inverse` = `.true.`, then the inverse transform is calculated.

Default: `inverse` = `.false.`.

trig_m$(2m)$ — real(kind=*wp*), intent(in), optional

trig_n$(2n)$ — real(kind=*wp*), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of lengths m and n respectively.

Note: if $m = n$, only one of these arguments need be present.

Default: if either `trig_m` or `trig_n` is not present, the corresponding coefficients are computed internally if needed.

Constraints: `trig_m` and `trig_n` must have been set by prior calls to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 6 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure performs multiple one-dimensional complex discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham [1].

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $nm\log(nm)$, but also depends on the factorisation of the individual dimensions m and n. The procedure is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

If several calls are made to this procedure to compute transforms with the same dimensions m and n, supplying trigonometric coefficients through the optional arguments `trig_m` and `trig_n` results in a saving of CPU-time.

Procedure: nag_fft_2d_basic

1 Description

`nag_fft_2d_basic` computes the two-dimensional discrete Fourier transform (DFT) of a bivariate $m \times n$ sequence of complex data values.

See Section 1 of the Module Introduction for the definition of the DFT, and Section 4 of the Module Introduction for the computation of an inverse transform.

2 Usage

```
USE nag_fft
CALL nag_fft_2d_basic(z  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of rows in the sequence

$n \geq 1$ — the number of columns in the sequence

3.1 Mandatory Argument

z(m, n) — complex(kind=*wp*), intent(inout)

Input: the bivariate sequence to be transformed.

Output: the discrete Fourier transform.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

trig_m$(2m)$ — real(kind=*wp*), intent(in), optional

trig_n$(2n)$ — real(kind=*wp*), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of lengths m and n respectively.

Note: if $m = n$, only one of these arguments need be present.

Default: if either `trig_m` or `trig_n` is not present, the corresponding coefficients are computed internally if needed.

Constraints: `trig_m` and `trig_n` must have been set by prior calls to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 7 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure performs multiple one-dimensional complex discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham [1].

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $nm\log(nm)$, but also depends on the factorisation of the individual dimensions m and n. The procedure is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

If several calls are made to this procedure to compute transforms with the same dimensions m and n, supplying trigonometric coefficients through the optional arguments `trig_m` and `trig_n` results in a saving of CPU-time.

Procedure: nag_fft_3d

1 Description

`nag_fft_3d` returns the three-dimensional discrete Fourier transform (DFT), or its inverse, of a trivariate $m \times n \times p$ sequence of complex data values.

See Section 1 of the Module Introduction for the definition of the DFT, and Section 4 of the Module Introduction for the definition of the inverse transform.

2 Usage

```
USE nag_fft
[value =] nag_fft_3d(z  [, optional arguments])
```

The function result is an array of type complex(kind=*wp*) and the same shape as that of `z`.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$	— the first dimension of the sequence
$n \geq 1$	— the second dimension of the sequence
$p \geq 1$	— the third dimension of the sequence

3.1 Mandatory Argument

z(m, n, p) — complex(kind=*wp*), intent(in)

Input: the sequence to be transformed.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

inverse — logical, intent(in), optional

Input: specifies whether the inverse transform is to be calculated.

If `inverse` = `.false.`, then the forward transform is calculated;

if `inverse` = `.true.`, then the inverse transform is calculated.

Default: `inverse` = `.false.`.

trig_1$(2m)$ — real(kind=*wp*), intent(in), optional
trig_2$(2n)$ — real(kind=*wp*), intent(in), optional
trig_3$(2p)$ — real(kind=*wp*), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of lengths m, n and p respectively.

Note:

if $m = n = p$, only one of these arguments need be present;

if $m = n$, only one of `trig_1` or `trig_2` need be present;

if $m = p$, only one of `trig_1` or `trig_3` need be present;

if $n = p$, only one of `trig_2` or `trig_3` need be present.

Default: if either `trig_1`, `trig_2` or `trig_3` is not present, the corresponding coefficients are computed internally if needed.

Constraints: `trig_1`, `trig_2` and `trig_3` must have been set by prior calls to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 8 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure performs multiple one-dimensional complex discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham [1].

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $mnp \log(mnp)$, but also depends on the factorisation of the individual dimensions m, n and p. The procedure is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

If several calls are made to this procedure to compute transforms with the same dimensions m, n and p, supplying trigonometric coefficients through the optional arguments `trig_1`, `trig_2` and `trig_3` results in a saving of CPU-time.

Procedure: nag_fft_3d_basic

1 Description

`nag_fft_3d_basic` computes the three-dimensional discrete Fourier transform (DFT) of a trivariate $m \times n \times p$ sequence of complex data values.

See Section 1 of the Module Introduction for the definition of the DFT, and Section 4 of the Module Introduction for the computation of an inverse transform.

2 Usage

```
USE nag_fft
CALL nag_fft_3d_basic(z  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the first dimension of the sequence

$n \geq 1$ — the second dimension of the sequence

$p \geq 1$ — the third dimension of the sequence

3.1 Mandatory Argument

z(m, n, p) — complex(kind=wp), intent(inout)

Input: the sequence to be transformed.

Output: the discrete Fourier transform.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

trig_1$(2m)$ — real(kind=wp), intent(in), optional

trig_2$(2n)$ — real(kind=wp), intent(in), optional

trig_3$(2p)$ — real(kind=wp), intent(in), optional

Input: trigonometric coefficients required for the computation of transforms of lengths m, n and p respectively.

Note:

if $m = n = p$, only one of these arguments need be present;

if $m = n$, only one of `trig_1` or `trig_2` need be present;

if $m = p$, only one of `trig_1` or `trig_3` need be present;

if $n = p$, only one of `trig_2` or `trig_3` need be present.

Default: if either `trig_1`, `trig_2` or `trig_3` is not present, the corresponding coefficients are computed internally if needed.

Constraints: `trig_1`, `trig_2` and `trig_3` must have been set by prior calls to `nag_fft_trig`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 9 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure performs multiple one-dimensional complex discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham [1].

6.2 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).

6.3 Timing

The time taken by the procedure is approximately proportional to $mnp \log(mnp)$, but also depends on the factorisation of the individual dimensions m, n and p. The procedure is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

If several calls are made to this procedure to compute transforms with the same dimensions m, n and p, supplying trigonometric coefficients through the optional arguments `trig_1`, `trig_2` and `trig_3` results in a saving of CPU-time.

Procedure: nag_fft_trig

1 Description

nag_fft_trig computes the trigonometric coefficients required in the computation of Fourier transforms and is designed to be called in conjunction with the procedures nag_fft_1d_basic, nag_fft_1d, nag_fft_2d_basic, nag_fft_2d, nag_fft_3d_basic and nag_fft_3d. It may also be used in conjunction with the procedures in the modules nag_sym_fft (7.2) and nag_conv (7.3), and is available through the USE statements for those modules.

2 Usage

```
USE nag_fft
CALL nag_fft_trig(trig  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$n \geq 1$ — the length of the transforms for which the coefficients are required

3.1 Mandatory Argument

trig($2n$) — real(kind=wp), intent(out)

Output: the trigonometric coefficients which may be used in the computation of discrete Fourier transforms of length n.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
302	An array argument has an invalid shape.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document. Further illustrations can be found in the example programs for the modules nag_sym_fft (7.2) and nag_conv (7.3).

Procedure: nag_herm_to_cmplx

1 Description

nag_herm_to_cmplx converts a single Hermitian sequence, or m Hermitian sequences, all of length n, from compact storage using n real values to storage as general complex sequences using n complex values. See Section 3 of the Module Introduction.

2 Usage

```
USE nag_fft
CALL nag_herm_to_cmplx(z_herm, z_cmplx  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathtt{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of sequences

$n \geq 1$ — the length of each sequence

3.1 Mandatory Arguments

z_herm(n) / **z_herm**(m, n) — real(kind=*wp*), intent(in)

Input: if z_herm has rank 1, it must hold a single Hermitian sequence. If z_herm has rank 2, it must hold m Hermitian sequences, with each sequence stored in a row of the array. Each sequence is stored compactly using only n real values, as described in Section 3 of the Module Introduction.

z_cmplx(n) / **z_cmplx**(m, n) — complex(kind=*wp*), intent(out)

Output: the same Hermitian sequences, stored as general sequences, using n complex values for each sequence.

Constraints: z_cmplx must have exactly the same shape as z_herm.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 4 of this module document.

Procedure: nag_cmplx_to_herm

1 Description

`nag_cmplx_to_herm` converts a single Hermitian sequence, or m Hermitian sequences, all of length n, from storage as general complex sequences using n complex values, to compact storage using n real values. See Section 3 of the Module Introduction.

2 Usage

```
USE nag_fft
CALL nag_cmplx_to_herm(z_cmplx, z_herm  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of sequences
$n \geq 1$ — the length of each sequence

3.1 Mandatory Arguments

z_cmplx(n) / **z_cmplx**(m, n) — complex(kind=wp), intent(in)

Input: if `z_cmplx` has rank 1, it must hold a single Hermitian sequence. If `z_cmplx` has rank 2, it must hold m Hermitian sequences, with each sequence stored in a row of the array. Each sequence is stored as a general sequence using n complex values.

Constraints: each sequence in the array `z_cmplx` must be Hermitian.

z_herm(n) / **z_herm**(m, n) — real(kind=wp), intent(out)

Output: the same Hermitian sequences, stored compactly using n real values for each sequence.

Constraints: `z_herm` must have exactly the same shape as `z_cmplx`.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 5 of this module document.

Procedure: nag_conj_herm

1 Description

`nag_conj_herm` forms the complex conjugate of a single Hermitian sequence, or of m Hermitian sequences, all of length n. Each Hermitian sequence is stored compactly using n real data values, as described in Section 3 of the Module Introduction.

2 Usage

```
USE nag_fft
CALL nag_conj_herm(z  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of sequences

$n \geq 1$ — the length of each sequence

3.1 Mandatory Argument

z(n) / **z**(m, n) — real(kind=*wp*), intent(inout)

Input: if `z` has rank 1, it must hold a single Hermitian sequence. If `z` has rank 2, it must hold m Hermitian sequences, with each sequence stored in a row of the array. Each sequence is stored compactly using n real data values, as described in Section 3 of the Module Introduction.

Output: each sequence is overwritten by its complex conjugate.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
302	An array argument has an invalid shape.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 4 and 5 of this module document.

Example 1: Discrete Fourier Transforms of 1-d Complex Sequences

This program reads in sequences of complex data values and prints their discrete Fourier transforms (as computed by `nag_fft_1d`). Inverse transforms are then calculated using `nag_fft_1d` and printed out, showing that the original sequences are restored. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex01

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_1d, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:)
  COMPLEX (wp), ALLOCATABLE :: z(:,:), zhat(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),zhat(m,n),trig(2*n)) ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z(i,:)
  END DO
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Original data')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)

  zhat = nag_fft_1d(z,trig=trig)

  CALL nag_write_gen_mat(zhat,format='f7.4',int_row_labels=.TRUE., &
   title='Transformed data')

  WRITE (nag_std_out,*)
  z = nag_fft_1d(zhat,inverse=.TRUE.,trig=trig)
```

```
      CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
       title='Original data restored by inverse transform')

      DEALLOCATE (z,zhat,trig)     ! Deallocate storage

    END PROGRAM nag_fft_ex01
```

2 Program Data

```
Example Program Data for nag_fft_ex01
3 4                                                                 : m, n
(0.3854,0.5417) (0.6772,0.2983) (0.1138,0.1181) (0.6751,0.7255) : sequence 1
(0.9172,0.9089) (0.0644,0.3118) (0.6037,0.3465) (0.6430,0.6198) : sequence 2
(0.1156,0.6214) (0.0685,0.8681) (0.2060,0.7060) (0.8630,0.8652) : sequence 3
```

3 Program Results

```
 Example Program Results for nag_fft_ex01

 Original data
 1  ( 0.3854, 0.5417) ( 0.6772, 0.2983) ( 0.1138, 0.1181) ( 0.6751, 0.7255)
 2  ( 0.9172, 0.9089) ( 0.0644, 0.3118) ( 0.6037, 0.3465) ( 0.6430, 0.6198)
 3  ( 0.1156, 0.6214) ( 0.0685, 0.8681) ( 0.2060, 0.7060) ( 0.8630, 0.8652)

 Transformed data
 1  ( 0.9258, 0.8418) (-0.0778, 0.2107) (-0.4265,-0.1820) ( 0.3494, 0.2128)
 2  ( 1.1141, 1.0935) ( 0.0028, 0.5705) ( 0.4068, 0.1619) ( 0.3107,-0.0081)
 3  ( 0.6265, 1.5303) (-0.0437, 0.3549) (-0.3049,-0.2029) (-0.0467,-0.4395)

 Original data restored by inverse transform
 1  ( 0.3854, 0.5417) ( 0.6772, 0.2983) ( 0.1138, 0.1181) ( 0.6751, 0.7255)
 2  ( 0.9172, 0.9089) ( 0.0644, 0.3118) ( 0.6037, 0.3465) ( 0.6430, 0.6198)
 3  ( 0.1156, 0.6214) ( 0.0685, 0.8681) ( 0.2060, 0.7060) ( 0.8630, 0.8652)
```

Example 2: Discrete Fourier Transforms of 1-d Complex Sequences Using Basic Procedures

This program reads in sequences of complex data values and prints their discrete Fourier transforms (as computed by `nag_fft_1d_basic`). Inverse transforms are then calculated using the Fortran 90 intrinsic function `CONJG` and `nag_fft_1d_basic` and printed out, showing that the original sequences are restored. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex02

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_1d_basic, nag_key_cmplx, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC CONJG, KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:)
  COMPLEX (wp), ALLOCATABLE :: z(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex02'
  WRITE (nag_std_out,*)

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),trig(2*n))  ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z(i,:)
  END DO

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Original data values')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)

  CALL nag_fft_1d_basic(nag_key_cmplx,z,trig=trig)

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Discrete Fourier transforms')

  WRITE (nag_std_out,*)

  z = CONJG(z)
```

```
      CALL nag_fft_1d_basic(nag_key_cmplx,z,trig=trig)

      z = CONJG(z)

      CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
       title='Original data restored by inverse transform')

      DEALLOCATE (z,trig)          ! Deallocate storage

    END PROGRAM nag_fft_ex02
```

2 Program Data

```
Example Program Data for nag_fft_ex02
3   4                                                         : m, n
(0.3854,0.5417) (0.6772,0.2983) (0.1138,0.1181) (0.6751,0.7255) : sequence 1
(0.9172,0.9089) (0.0644,0.3118) (0.6037,0.3465) (0.6430,0.6198) : sequence 2
(0.1156,0.6214) (0.0685,0.8681) (0.2060,0.7060) (0.8630,0.8652) : sequence 3
```

3 Program Results

```
 Example Program Results for nag_fft_ex02

 Original data values
 1  ( 0.3854, 0.5417) ( 0.6772, 0.2983) ( 0.1138, 0.1181) ( 0.6751, 0.7255)
 2  ( 0.9172, 0.9089) ( 0.0644, 0.3118) ( 0.6037, 0.3465) ( 0.6430, 0.6198)
 3  ( 0.1156, 0.6214) ( 0.0685, 0.8681) ( 0.2060, 0.7060) ( 0.8630, 0.8652)

 Discrete Fourier transforms
 1  ( 0.9258, 0.8418) (-0.0778, 0.2107) (-0.4265,-0.1820) ( 0.3494, 0.2128)
 2  ( 1.1141, 1.0935) ( 0.0028, 0.5705) ( 0.4068, 0.1619) ( 0.3107,-0.0081)
 3  ( 0.6265, 1.5303) (-0.0437, 0.3549) (-0.3049,-0.2029) (-0.0467,-0.4395)

 Original data restored by inverse transform
 1  ( 0.3854, 0.5417) ( 0.6772, 0.2983) ( 0.1138, 0.1181) ( 0.6751, 0.7255)
 2  ( 0.9172, 0.9089) ( 0.0644, 0.3118) ( 0.6037, 0.3465) ( 0.6430, 0.6198)
 3  ( 0.1156, 0.6214) ( 0.0685, 0.8681) ( 0.2060, 0.7060) ( 0.8630, 0.8652)
```

Example 3: Discrete Fourier Transforms of 1-d Real Sequences

This program reads in sequences of real data values and prints their discrete Fourier transforms (as computed by `nag_fft_1d_real`). The original sequences are then restored by using `nag_fft_1d_real` to calculate the inverse transforms. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex03

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_1d_real, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:), z(:,:)
  COMPLEX (wp), ALLOCATABLE :: zhat(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex03'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),zhat(m,n),trig(2*n)) ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z(i,:)
  END DO
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Original data')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)

  zhat = nag_fft_1d_real(z,trig=trig)

  CALL nag_write_gen_mat(zhat,format='f7.4',int_row_labels=.TRUE., &
   title='Transformed data')

  WRITE (nag_std_out,*)
  z = nag_fft_1d_real(zhat,inverse=.TRUE.,trig=trig)

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Original data restored by inverse transform')
```

```
      DEALLOCATE (z,zhat,trig)     ! Deallocate storage

    END PROGRAM nag_fft_ex03
```

2 Program Data

```
Example Program Data for nag_fft_ex03
3 4                                  : m, n
0.3854    0.6772    0.1138    0.6751 : sequence 1
0.5417    0.2983    0.1181    0.7255 : sequence 2
0.9172    0.0644    0.6037    0.6430 : sequence 3
```

3 Program Results

```
 Example Program Results for nag_fft_ex03

 Original data
 1  0.3854 0.6772 0.1138 0.6751
 2  0.5417 0.2983 0.1181 0.7255
 3  0.9172 0.0644 0.6037 0.6430

 Transformed data
 1  ( 0.9258, 0.0000) ( 0.1358,-0.0010) (-0.4265, 0.0000) ( 0.1358, 0.0010)
 2  ( 0.8418, 0.0000) ( 0.2118, 0.2136) (-0.1820, 0.0000) ( 0.2118,-0.2136)
 3  ( 1.1141, 0.0000) ( 0.1568, 0.2893) ( 0.4068, 0.0000) ( 0.1568,-0.2893)

 Original data restored by inverse transform
 1  0.3854 0.6772 0.1138 0.6751
 2  0.5417 0.2983 0.1181 0.7255
 3  0.9172 0.0644 0.6037 0.6430
```

Example 4: Discrete Fourier Transforms of 1-d Real Sequences Using Basic Procedures

This program reads in sequences of real data values and prints their discrete Fourier transforms (as computed by `nag_fft_1d_basic`). The Fourier transforms are expanded into full complex form using `nag_herm_to_cmplx`. Inverse transforms are then calculated by calling `nag_conj_herm` followed by `nag_fft_1d_basic`, showing that the original sequences are restored. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex04

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_1d_basic, nag_key_real, nag_key_herm, &
   nag_fft_trig, nag_conj_herm, nag_herm_to_cmplx
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:), z(:,:)
  COMPLEX (wp), ALLOCATABLE :: z_cmplx(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex04'
  WRITE (nag_std_out,*)

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),z_cmplx(m,n),trig(2*n)) ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z(i,:)
  END DO

  CALL nag_write_gen_mat(z,format='f10.4',int_row_labels=.TRUE., &
   title='Original data values')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)

  CALL nag_fft_1d_basic(nag_key_real,z,trig=trig)

  CALL nag_write_gen_mat(z,format='f10.4',int_row_labels=.TRUE., &
   title='Discrete Fourier transforms in Hermitian form')

  WRITE (nag_std_out,*)
```

```
    CALL nag_herm_to_cmplx(z,z_cmplx)

    CALL nag_write_gen_mat(z_cmplx,format='f7.4',int_row_labels=.TRUE., &
     title='Fourier transforms in full complex form')

    WRITE (nag_std_out,*)

    CALL nag_conj_herm(z)

    CALL nag_fft_1d_basic(nag_key_herm,z,trig=trig)

    CALL nag_write_gen_mat(z,format='f10.4',int_row_labels=.TRUE., &
     title='Original data restored by inverse transform')

    DEALLOCATE (z,z_cmplx,trig)  ! Deallocate storage

  END PROGRAM nag_fft_ex04
```

2 Program Data

```
Example Program Data for nag_fft_ex04
3   4                                     : m, n
  0.3854    0.6772    0.1138    0.6751    : first sequence
  0.5417    0.2983    0.1181    0.7255    : second sequence
  0.9172    0.0644    0.6037    0.6430    : third sequence
```

3 Program Results

```
 Example Program Results for nag_fft_ex04

 Original data values
 1     0.3854    0.6772    0.1138    0.6751
 2     0.5417    0.2983    0.1181    0.7255
 3     0.9172    0.0644    0.6037    0.6430

 Discrete Fourier transforms in Hermitian form
 1     0.9258    0.1358   -0.4265   -0.0010
 2     0.8418    0.2118   -0.1820    0.2136
 3     1.1141    0.1568    0.4068    0.2893

 Fourier transforms in full complex form
 1  ( 0.9258, 0.0000) ( 0.1358,-0.0010) (-0.4265, 0.0000) ( 0.1358, 0.0010)
 2  ( 0.8418, 0.0000) ( 0.2118, 0.2136) (-0.1820, 0.0000) ( 0.2118,-0.2136)
 3  ( 1.1141, 0.0000) ( 0.1568, 0.2893) ( 0.4068, 0.0000) ( 0.1568,-0.2893)

 Original data restored by inverse transform
 1     0.3854    0.6772    0.1138    0.6751
 2     0.5417    0.2983    0.1181    0.7255
 3     0.9172    0.0644    0.6037    0.6430
```

Example 5: Discrete Fourier Transforms of 1-d Hermitian Sequences

This program reads in sequences of Hermitian data values stored as general complex sequences. The sequences are converted to compact storage in a real array using `nag_cmplx_to_herm` and printed. The discrete Fourier transforms are then computed (using `nag_fft_1d_basic`) and printed. Inverse Fourier transforms are then calculated by calling `nag_fft_1d_basic` followed by `nag_conj_herm`, showing that the original sequences are restored. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex05

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_1d_basic, nag_key_real, nag_key_herm, &
   nag_fft_trig, nag_conj_herm, nag_cmplx_to_herm
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:), z(:,:)
  COMPLEX (wp), ALLOCATABLE :: z_cmplx(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex05'
  WRITE (nag_std_out,*)

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),z_cmplx(m,n),trig(2*n)) ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z_cmplx(i,:)
  END DO

  CALL nag_write_gen_mat(z_cmplx,format='f7.4',int_row_labels=.TRUE., &
   title='Original data')

  WRITE (nag_std_out,*)

  CALL nag_cmplx_to_herm(z_cmplx,z)

  CALL nag_write_gen_mat(z,format='f10.4',int_row_labels=.TRUE., &
   title='Original data in compact storage')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)
```

```
      CALL nag_fft_1d_basic(nag_key_herm,z,trig=trig)

      CALL nag_write_gen_mat(z,format='f10.4',int_row_labels=.TRUE., &
       title='Discrete Fourier transforms (real values)')

      WRITE (nag_std_out,*)

      CALL nag_fft_1d_basic(nag_key_real,z,trig=trig)

      CALL nag_conj_herm(z)

      CALL nag_write_gen_mat(z,format='f10.4',int_row_labels=.TRUE., &
       title='Original data in compact storage, restored by inverse transform' &
       )

      DEALLOCATE (z,z_cmplx,trig)  ! Deallocate storage

    END PROGRAM nag_fft_ex05
```

2 Program Data

```
Example Program Data for nag_fft_ex05
3   4                                                 : m, n
( 0.3854, 0.0000) ( 0.6772, 0.6751) ( 0.1138, 0.0000)
( 0.6772,-0.6751)                                     : first sequence
( 0.5417, 0.0000) ( 0.2983, 0.7255) ( 0.1181, 0.0000)
( 0.2983,-0.7255)                                     : second sequence
( 0.9172, 0.0000) ( 0.0644, 0.6430) ( 0.6037, 0.0000)
( 0.0644,-0.6430)                                     : third sequence
```

3 Program Results

```
 Example Program Results for nag_fft_ex05

 Original data
 1  ( 0.3854, 0.0000) ( 0.6772, 0.6751) ( 0.1138, 0.0000) ( 0.6772,-0.6751)
 2  ( 0.5417, 0.0000) ( 0.2983, 0.7255) ( 0.1181, 0.0000) ( 0.2983,-0.7255)
 3  ( 0.9172, 0.0000) ( 0.0644, 0.6430) ( 0.6037, 0.0000) ( 0.0644,-0.6430)

 Original data in compact storage
 1      0.3854    0.6772    0.1138    0.6751
 2      0.5417    0.2983    0.1181    0.7255
 3      0.9172    0.0644    0.6037    0.6430

 Discrete Fourier transforms (real values)
 1      0.9268    0.8109   -0.4276   -0.5393
 2      0.6282    0.9373    0.0316   -0.5137
 3      0.8249    0.7997    0.6961   -0.4863

 Original data in compact storage, restored by inverse transform
 1      0.3854    0.6772    0.1138    0.6751
 2      0.5417    0.2983    0.1181    0.7255
 3      0.9172    0.0644    0.6037    0.6430
```

Example 6: Discrete Fourier Transforms of 2-d Complex Sequences

This program reads in a bivariate sequence of complex data values and prints the 2-d Fourier transform (as computed by `nag_fft_2d`). It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex06

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_2d, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trigm(:), trign(:)
  COMPLEX (wp), ALLOCATABLE :: z(:,:), zhat(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex06'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),zhat(m,n),trigm(2*m),trign(2*n)) ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z(i,:)
  END DO
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Original data')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trigm)

  CALL nag_fft_trig(trign)

  zhat = nag_fft_2d(z,trig_m=trigm,trig_n=trign)

  CALL nag_write_gen_mat(zhat,format='f7.4',int_row_labels=.TRUE., &
   title='Transformed data')

  WRITE (nag_std_out,*)
```

```
      z = nag_fft_2d(zhat,inverse=.TRUE.,trig_m=trigm,trig_n=trign)

      CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
       title='Original data restored by inverse transform')

      DEALLOCATE (z,zhat,trigm,trign) ! Deallocate storage

    END PROGRAM nag_fft_ex06
```

2 Program Data

```
Example Program Data for nag_fft_ex06
3 4                                                      : m,n
(1.000, 0.000) (0.999,-0.040) (0.987,-0.159) (0.936,-0.352) : sequence 1
(0.994,-0.111) (0.989,-0.151) (0.963,-0.268) (0.891,-0.454) : sequence 2
(0.903,-0.430) (0.885,-0.466) (0.823,-0.568) (0.694,-0.720) : sequence 3
```

3 Program Results

```
 Example Program Results for nag_fft_ex06

 Original data
 1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
 2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
 3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)

 Transformed data
 1  ( 3.1939,-1.0736) ( 0.2867, 0.0294) ( 0.0797, 0.1868) (-0.2151, 0.2327)
 2  ( 0.4013, 0.1652) ( 0.0254, 0.0268) (-0.0078, 0.0250) (-0.0404, 0.0043)
 3  (-0.1987, 0.4312) (-0.0306, 0.0268) (-0.0268,-0.0100) (-0.0034,-0.0447)

 Original data restored by inverse transform
 1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
 2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
 3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)
```

Example 7: Discrete Fourier Transforms of 2-d Complex Sequences Using Basic Procedures

This program reads in a bivariate sequence of complex data values and prints the 2-d Fourier transform (as computed by `nag_fft_2d_basic`). It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex07

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_2d_basic, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC CONJG, KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig_m(:), trig_n(:)
  COMPLEX (wp), ALLOCATABLE :: z(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex07'
  WRITE (nag_std_out,*)

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ALLOCATE (z(m,n),trig_m(2*m),trig_n(2*n)) ! Allocate storage

  DO i = 1, m
    READ (nag_std_in,*) z(i,:)
  END DO

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Original data values')

  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig_m)

  CALL nag_fft_trig(trig_n)

  CALL nag_fft_2d_basic(z,trig_m=trig_m,trig_n=trig_n)

  CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
   title='Components of discrete Fourier transform')

  WRITE (nag_std_out,*)
```

```
      z = CONJG(z)

      CALL nag_fft_2d_basic(z,trig_m=trig_m,trig_n=trig_n)

      z = CONJG(z)

      CALL nag_write_gen_mat(z,format='f7.4',int_row_labels=.TRUE., &
       title='Original data restored by inverse transform')

      DEALLOCATE (z,trig_m,trig_n) ! Deallocate storage

    END PROGRAM nag_fft_ex07
```

2 Program Data

```
Example Program Data for nag_fft_ex07
 3   4                   : Number of rows, m, and columns, n, in z
 (1.000, 0.000) (0.999,-0.040) (0.987,-0.159) (0.936,-0.352)
 (0.994,-0.111) (0.989,-0.151) (0.963,-0.268) (0.891,-0.454)
 (0.903,-0.430) (0.885,-0.466) (0.823,-0.568) (0.694,-0.720)  : z
```

3 Program Results

```
 Example Program Results for nag_fft_ex07

 Original data values
 1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
 2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
 3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)

 Components of discrete Fourier transform
 1  ( 3.1939,-1.0736) ( 0.2867, 0.0294) ( 0.0797, 0.1868) (-0.2151, 0.2327)
 2  ( 0.4013, 0.1652) ( 0.0254, 0.0268) (-0.0078, 0.0250) (-0.0404, 0.0043)
 3  (-0.1987, 0.4312) (-0.0306, 0.0268) (-0.0268,-0.0100) (-0.0034,-0.0447)

 Original data restored by inverse transform
 1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
 2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
 3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)
```

Example 8: Discrete Fourier Transforms of 3-d Complex Sequences

This program reads in a trivariate sequence of complex data values and prints the 3-d Fourier transform (as computed by `nag_fft_3d`). It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex08

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_3d, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, m, n, p
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig_1(:), trig_2(:), trig_3(:)
  COMPLEX (wp), ALLOCATABLE :: z(:,:,:), z_hat(:,:,:)
  CHARACTER (20), ALLOCATABLE :: title(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex08'
  WRITE (nag_std_out,*)

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n, p

  ALLOCATE (z(m,n,p),trig_1(2*m),trig_2(2*n),trig_3(2*p),title(m), &
   z_hat(m,n,p))               ! Allocate storage

  DO i = 1, m
    DO j = 1, n
      READ (nag_std_in,*) z(i,j,:)
    END DO
  END DO

  title(1) = 'Sequence z(1,:,:)'
  title(2) = 'Sequence z(2,:,:)'

  WRITE (nag_std_out,*) 'Original data'
  DO i = 1, m

    CALL nag_write_gen_mat(z(i,:,:),format='f7.4',int_row_labels=.TRUE., &
     title=title(i))

  END DO
  WRITE (nag_std_out,*)
```

```
    CALL nag_fft_trig(trig_1)

    CALL nag_fft_trig(trig_2)

    CALL nag_fft_trig(trig_3)

    z_hat = nag_fft_3d(z,trig_1=trig_1,trig_2=trig_2,trig_3=trig_3)

    WRITE (nag_std_out,*) 'Components of discrete Fourier transform'
    DO i = 1, m

      CALL nag_write_gen_mat(z_hat(i,:,:),format='f7.4', &
       int_row_labels=.TRUE.,title=title(i))

    END DO
    WRITE (nag_std_out,*)

    z = nag_fft_3d(z_hat,inverse=.TRUE.,trig_1=trig_1,trig_2=trig_2, &
     trig_3=trig_3)

    WRITE (nag_std_out,*) 'Original data restored by inverse transform'
    DO i = 1, m

      CALL nag_write_gen_mat(z(i,:,:),format='f7.4',int_row_labels=.TRUE., &
       title=title(i))

    END DO
    DEALLOCATE (z,z_hat,trig_1,trig_2,trig_3) ! Deallocate storage

  END PROGRAM nag_fft_ex08
```

2 Program Data

```
Example Program Data for nag_fft_ex08
 2   3   4                       : m, n, p  where z is of shape (m,n,p)
 (1.000, 0.000) (0.999,-0.040) (0.987,-0.159) (0.936,-0.352) : z(1,1,:)
 (0.994,-0.111) (0.989,-0.151) (0.963,-0.268) (0.891,-0.454) : z(1,2,:)
 (0.903,-0.430) (0.885,-0.466) (0.823,-0.568) (0.694,-0.720) : z(1,3,:)
 (0.500, 0.500) (0.499, 0.040) (0.487, 0.159) (0.436, 0.352) : z(2,1,:)
 (0.494, 0.111) (0.489, 0.151) (0.463, 0.268) (0.391, 0.454) : z(2,2,:)
 (0.403, 0.430) (0.385, 0.466) (0.323, 0.568) (0.194, 0.720) : z(2,3,:)
```

3 Program Results

```
 Example Program Results for nag_fft_ex08

 Original data
 Sequence z(1,:,:)
 1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
 2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
 3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)
 Sequence z(2,:,:)
 1  ( 0.5000, 0.5000) ( 0.4990, 0.0400) ( 0.4870, 0.1590) ( 0.4360, 0.3520)
 2  ( 0.4940, 0.1110) ( 0.4890, 0.1510) ( 0.4630, 0.2680) ( 0.3910, 0.4540)
 3  ( 0.4030, 0.4300) ( 0.3850, 0.4660) ( 0.3230, 0.5680) ( 0.1940, 0.7200)

 Components of discrete Fourier transform
 Sequence z(1,:,:)
 1  ( 3.2921, 0.1021) ( 0.0506,-0.0416) ( 0.1127, 0.1021) ( 0.0506, 0.2458)
 2  ( 0.1433,-0.0860) ( 0.0155, 0.1527) (-0.0245, 0.1268) (-0.0502, 0.0861)
```

```
3  ( 0.1433, 0.2902) (-0.0502, 0.1180) (-0.0245, 0.0773) ( 0.0155, 0.0515)
Sequence z(2,:,:)
1  ( 1.2247,-1.6203) ( 0.3548, 0.0833) ( 0.0000, 0.1621) (-0.3548, 0.0833)
2  ( 0.4243, 0.3197) ( 0.0204,-0.1147) ( 0.0134,-0.0914) (-0.0070,-0.0800)
3  (-0.4243, 0.3197) ( 0.0070,-0.0800) (-0.0134,-0.0914) (-0.0204,-0.1147)

Original data restored by inverse transform
Sequence z(1,:,:)
1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)
Sequence z(2,:,:)
1  ( 0.5000, 0.5000) ( 0.4990, 0.0400) ( 0.4870, 0.1590) ( 0.4360, 0.3520)
2  ( 0.4940, 0.1110) ( 0.4890, 0.1510) ( 0.4630, 0.2680) ( 0.3910, 0.4540)
3  ( 0.4030, 0.4300) ( 0.3850, 0.4660) ( 0.3230, 0.5680) ( 0.1940, 0.7200)
```

Example 9: Discrete Fourier Transforms of 3-d Complex Sequences Using Basic Procedures

This program reads in a trivariate sequence of complex data values and prints the 3-d Fourier transform (as computed by `nag_fft_3d_basic`). It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values. This example program also shows the use of procedure `nag_fft_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_fft_ex09

  ! Example Program Text for nag_fft
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_fft, ONLY : nag_fft_3d_basic, nag_fft_trig
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC CONJG, KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, m, n, p
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig_1(:), trig_2(:), trig_3(:)
  COMPLEX (wp), ALLOCATABLE :: z(:,:,:)
  CHARACTER (20), ALLOCATABLE :: title(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_fft_ex09'
  WRITE (nag_std_out,*)

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n, p

  ALLOCATE (z(m,n,p),trig_1(2*m),trig_2(2*n),trig_3(2*p),title(m)) ! Allocate
  ! storage

  DO i = 1, m
    DO j = 1, n
      READ (nag_std_in,*) z(i,j,:)
    END DO
  END DO

  title(1) = 'Sequence z(1,:,:)'
  title(2) = 'Sequence z(2,:,:)'

  WRITE (nag_std_out,*) 'Original data'
  DO i = 1, m

    CALL nag_write_gen_mat(z(i,:,:),format='f7.4',int_row_labels=.TRUE., &
     title=title(i))

  END DO
  WRITE (nag_std_out,*)
```

```
      CALL nag_fft_trig(trig_1)

      CALL nag_fft_trig(trig_2)

      CALL nag_fft_trig(trig_3)

      CALL nag_fft_3d_basic(z,trig_1=trig_1,trig_2=trig_2,trig_3=trig_3)

      WRITE (nag_std_out,*) 'Components of discrete Fourier transform'
      DO i = 1, m

        CALL nag_write_gen_mat(z(i,:,:),format='f7.4',int_row_labels=.TRUE., &
         title=title(i))

      END DO
      WRITE (nag_std_out,*)

      z = CONJG(z)

      CALL nag_fft_3d_basic(z,trig_1=trig_1,trig_2=trig_2,trig_3=trig_3)

      z = CONJG(z)

      WRITE (nag_std_out,*) 'Original data restored by inverse transform'
      DO i = 1, m

        CALL nag_write_gen_mat(z(i,:,:),format='f7.4',int_row_labels=.TRUE., &
         title=title(i))

      END DO
      DEALLOCATE (z,trig_1,trig_2,trig_3) ! Deallocate storage

    END PROGRAM nag_fft_ex09
```

2 Program Data

```
Example Program Data for nag_fft_ex09
 2   3   4                         : m, n, p  where z is of shape (m,n,p)
 (1.000, 0.000) (0.999,-0.040) (0.987,-0.159) (0.936,-0.352) : z(1,1,:)
 (0.994,-0.111) (0.989,-0.151) (0.963,-0.268) (0.891,-0.454) : z(1,2,:)
 (0.903,-0.430) (0.885,-0.466) (0.823,-0.568) (0.694,-0.720) : z(1,3,:)
 (0.500, 0.500) (0.499, 0.040) (0.487, 0.159) (0.436, 0.352) : z(2,1,:)
 (0.494, 0.111) (0.489, 0.151) (0.463, 0.268) (0.391, 0.454) : z(2,2,:)
 (0.403, 0.430) (0.385, 0.466) (0.323, 0.568) (0.194, 0.720) : z(2,3,:)
```

3 Program Results

```
 Example Program Results for nag_fft_ex09

 Original data
 Sequence z(1,:,:)
 1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
 2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
 3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)
 Sequence z(2,:,:)
 1  ( 0.5000, 0.5000) ( 0.4990, 0.0400) ( 0.4870, 0.1590) ( 0.4360, 0.3520)
 2  ( 0.4940, 0.1110) ( 0.4890, 0.1510) ( 0.4630, 0.2680) ( 0.3910, 0.4540)
 3  ( 0.4030, 0.4300) ( 0.3850, 0.4660) ( 0.3230, 0.5680) ( 0.1940, 0.7200)

 Components of discrete Fourier transform
```

```
Sequence z(1,:,:)
1  ( 3.2921, 0.1021) ( 0.0506,-0.0416) ( 0.1127, 0.1021) ( 0.0506, 0.2458)
2  ( 0.1433,-0.0860) ( 0.0155, 0.1527) (-0.0245, 0.1268) (-0.0502, 0.0861)
3  ( 0.1433, 0.2902) (-0.0502, 0.1180) (-0.0245, 0.0773) ( 0.0155, 0.0515)
Sequence z(2,:,:)
1  ( 1.2247,-1.6203) ( 0.3548, 0.0833) ( 0.0000, 0.1621) (-0.3548, 0.0833)
2  ( 0.4243, 0.3197) ( 0.0204,-0.1147) ( 0.0134,-0.0914) (-0.0070,-0.0800)
3  (-0.4243, 0.3197) ( 0.0070,-0.0800) (-0.0134,-0.0914) (-0.0204,-0.1147)

Original data restored by inverse transform
Sequence z(1,:,:)
1  ( 1.0000, 0.0000) ( 0.9990,-0.0400) ( 0.9870,-0.1590) ( 0.9360,-0.3520)
2  ( 0.9940,-0.1110) ( 0.9890,-0.1510) ( 0.9630,-0.2680) ( 0.8910,-0.4540)
3  ( 0.9030,-0.4300) ( 0.8850,-0.4660) ( 0.8230,-0.5680) ( 0.6940,-0.7200)
Sequence z(2,:,:)
1  ( 0.5000, 0.5000) ( 0.4990, 0.0400) ( 0.4870, 0.1590) ( 0.4360, 0.3520)
2  ( 0.4940, 0.1110) ( 0.4890, 0.1510) ( 0.4630, 0.2680) ( 0.3910, 0.4540)
3  ( 0.4030, 0.4300) ( 0.3850, 0.4660) ( 0.3230, 0.5680) ( 0.1940, 0.7200)
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_fft_ex10`
: Discrete Fourier transform of 1-d Hermitian sequences (using `nag_fft_1d_real`).

`nag_fft_ex11`
: Discrete Fourier transform of a single 1-d complex sequence (using `nag_fft_1d_basic`).

`nag_fft_ex12`
: Discrete Fourier transform of a single 1-d complex sequence (using `nag_fft_1d`).

`nag_fft_ex13`
: Discrete Fourier transform of single 1-d real sequence (using `nag_fft_1d_real`).

`nag_fft_ex14`
: Discrete Fourier transform of a single 1-d Hermitian sequence (using `nag_fft_1d_real`).

`nag_fft_ex15`
: Discrete Fourier transform of a single 1-d real sequence (using `nag_fft_1d_basic`).

`nag_fft_ex16`
: Discrete Fourier transform of a single 1-d Hermitian sequence (using `nag_fft_1d_basic`).

References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

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

[4] Hamming R W (1962) *Numerical Methods for Scientists and Engineers* McGraw-Hill

[5] Gentleman W S and Sande G (1966) Fast Fourier transforms for fun and profit *Proc. Joint Computer Conference, AFIPS* **29** 563–578