Module 7.2: nag_sym_fft
Symmetric Discrete Fourier Transforms

`nag_sym_fft` provides procedures for computations involving one-dimensional *real symmetric discrete Fourier transforms*.

Contents

Introduction

1 The Discrete Fourier Transform (DFT)

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

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

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

2 Real Symmetric Transforms

In many applications the sequence x_j will not only be real, but may also possess additional symmetries which we may exploit to reduce further the computing time and storage requirements.

If the sequence x_j is *odd* ($x_j = -x_{n-j}$), then the discrete Fourier transform of x_j contains only sine terms. Rather than compute the transform of an odd sequence, we define the *discrete sine transform (DST)* (Van Loan [7]) of a real sequence by

$$\hat{x}_k = \sqrt{\frac{2}{n}} \left\{ \sum_{j=1}^{n-1} x_j \sin\left(\frac{\pi jk}{n}\right) \right\}, k = 1, 2, \ldots, n-1, \tag{2}$$

which could have been computed using the Fourier transform of a real odd sequence of length $2n$. In this case the x_j are arbitrary, and the symmetry only becomes apparent when the sequence is extended.

Similarly we define the *discrete cosine transform* (also called DCT; see Van Loan [7]) of a real sequence by

$$\hat{x}_k = \sqrt{\frac{2}{n}} \left\{ \frac{1}{2}x_0 + \sum_{j=1}^{n-1} x_j \cos\left(\frac{\pi jk}{n}\right) + \frac{1}{2}(-1)^k x_n \right\}, \ k = 0, \ldots, n, \tag{3}$$

which could have been computed using the Fourier transform of a real *even* sequence of length $2n$.

In addition to these 'half-wave' symmetries described above, sequences arise in practice with 'quarter-wave' symmetries. We define the *discrete quarter-wave sine transform* by

$$\hat{x}_k = \frac{1}{\sqrt{n}} \left\{ \sum_{j=1}^{n-1} x_j \sin\left(\frac{\pi j(2k-1)}{2n}\right) + \frac{1}{2}(-1)^{k-1} x_n \right\}, \ k = 1, \ldots, n. \tag{4}$$

Similarly we define the *discrete quarter-wave cosine transform* by

$$\hat{x}_k = \frac{1}{\sqrt{n}} \left\{ \frac{1}{2}x_0 + \sum_{j=1}^{n-1} x_j \cos\left(\frac{\pi j(2k-1)}{2n}\right) \right\}, \ k = 0, \ldots, n-1. \tag{5}$$

3 Real Symmetric Inverse Transforms

The sine or cosine Fourier transform is its own inverse.

The inverse of the quarter-wave sine transform (also called DST-II; see Van Loan [7]) is:

$$x_k = \frac{2}{\sqrt{n}} \left\{ \sum_{j=1}^{n} \hat{x}_j \sin\left(\frac{\pi k(2j-1)}{2n}\right) \right\} \tag{6}$$

and the inverse of the quarter-wave cosine transform (also called DCT-II; see Van Loan [7]):

$$x_k = \frac{2}{\sqrt{n}} \left\{ \sum_{j=0}^{n-1} \hat{x}_j \cos\left(\frac{\pi k(2j-1)}{2n}\right) \right\}. \tag{7}$$

4 Applications to Solving Partial Differential Equations

One application of Fourier transforms of symmetric sequences is in the solution of elliptic partial differential equations. If an equation is discretised using finite differences, then it is sometimes possible to reduce the problem of solving the resulting large system of linear equations to that of solving a number of tridiagonal systems of linear equations. This is accomplished by uncoupling the equations using Fourier transforms, where the nature of the boundary conditions determines the choice of transforms. This approach is used for the solution of the 3-d Helmholtz equation in the module `nag_pde_helm` (13.1). Full details of the Fourier method for the solution of partial differential equations may be found in Swarztrauber [4] and Swarztrauber [6].

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 the procedure `nag_fft_trig` from the module `nag_fft` (7.1) and supplied in an optional argument (which is more efficient if several calls are made to compute transforms of the same length).

If you are using procedures from this module, and wish to pre-compute the trigonometric coefficients by calling `nag_fft_trig`, it is not necessary to add a separate `USE` statement for the module `nag_fft` (7.1). The procedure `nag_fft_trig` is also available through the `USE` statement for this module.

Procedure: nag_fft_sin

1 Description

nag_fft_sin returns the discrete Fourier sine transform of order n of either a single sequence or m sequences (all of the same length $n-1$).

The Fourier sine transform is its own inverse and two consecutive calls of this procedure will restore the original data.

2 Usage

```
USE nag_sym_fft
[value =] nag_fft_sin(x  [, optional arguments])
```

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

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 order of the transform (one more than the number of data values in each sequence)

3.1 Mandatory Argument

x$(n-1)$ / **x**$(m, n-1)$ — real(kind=*wp*), intent(in)

Input: if x has rank 1, it must hold the single sequence to be transformed. If x 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.

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

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

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

Constraints: trig must have been set by a prior call to the procedure nag_fft_trig from the module nag_fft (7.1). This procedure is accessible through the USE statement for this module.

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 (see Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2] and Temperton [3], together with pre- and post-processing stages described in Swarztrauber [5]. Special coding is provided for the cases where n has factors 2, 3, 4, 5, or 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 order n, supplying trigonometric coefficients through the optional argument `trig` results in a saving of CPU-time.

Procedure: nag_fft_cos

1 Description

nag_fft_cos returns the discrete Fourier cosine transform of order n of either a single sequence or m sequences (all of the same length $n + 1$).

The Fourier cosine transform is its own inverse and two consecutive calls of this procedure will restore the original data.

2 Usage

```
USE nag_sym_fft
[value =] nag_fft_cos(x  [, optional arguments])
```

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

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 order of the transform (one less than the number of data values in each sequence)

3.1 Mandatory Argument

x($n + 1$) / **x**($m, n + 1$) — real(kind=*wp*), intent(in)

Input: if x has rank 1, it must hold the single sequence to be transformed. If x 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.

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

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

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

Constraints: trig must have been set by a prior call to the procedure nag_fft_trig from module nag_fft (7.1). This procedure is accessible through the USE statement for this module.

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 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure uses a variant of the fast Fourier transform (FFT) algorithm (see Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2] and Temperton [3], together with pre- and post-processing stages described in Swarztrauber [5]. Special coding is provided for the cases where n has factors 2, 3, 4, 5, or 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 order n, supplying trigonometric coefficients through the optional argument `trig` results in a saving of CPU-time.

Procedure: nag_fft_qtr_sin

1 Description

nag_fft_qtr_sin returns the discrete quarter-wave Fourier sine transform, or its inverse, of either a single sequence or m sequences (all of the same length n).

2 Usage

```
USE nag_sym_fft
[value =] nag_fft_qtr_sin(x  [, optional arguments])
```

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

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

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

Input: if x has rank 1, it must hold the single sequence to be transformed. If x 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.

See Section 2 and Section 3 of the Module Introduction for definitions of the forward quarter-wave sine transform and its inverse, respectively.

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 the procedure nag_fft_trig from module nag_fft (7.1). This procedure is accessible through the USE statement for this module.

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

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

1 Description

nag_fft_qtr_cos returns the discrete quarter-wave Fourier cosine transform, or its inverse, of either a single sequence or m sequences (all of the same length n).

2 Usage

```
USE nag_sym_fft
[value =] nag_fft_qtr_cos(x  [, optional arguments])
```

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

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

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

Input: if x has rank 1, it must hold the single sequence to be transformed. If x 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.

See Section 2 and Section 3 of the Module Introduction for definitions of the forward quarter-wave cosine transform and its inverse, respectively.

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 the procedure nag_fft_trig from module nag_fft (7.1). This procedure is accessible through the USE statement for this module.

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 4 of this module document.

6 Further Comments

6.1 Algorithmic Detail

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

Example 1: Discrete Fourier sine transform of a single 1-d sequence

This program reads a sequence of real data values and prints its discrete Fourier sine transform. The inverse transform is then calculated and printed out, showing that the original sequence is restored. This example program also shows the use of the procedure `nag_fft_trig` from module `nag_fft` (7.1). Note that `nag_fft_trig` is accessed through the `USE` statement for this module.

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_sym_fft_ex01

  ! Example Program Text for nag_sym_fft
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_fft, ONLY : nag_fft_sin, nag_fft_trig
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:), x(:), xhat(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_fft_ex01'

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

  ALLOCATE (x(n-1),xhat(n-1),trig(2*n)) ! Allocate storage

  READ (nag_std_in,*) x
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Original data'
  WRITE (nag_std_out,'(7f10.4)') x
  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)

  xhat = nag_fft_sin(x,trig=trig)

  WRITE (nag_std_out,*) 'Transformed data'
  WRITE (nag_std_out,'(7f10.4)') xhat
  WRITE (nag_std_out,*)

  x = nag_fft_sin(xhat,trig=trig)

  WRITE (nag_std_out,*) 'Original data restored by inverse transform'
  WRITE (nag_std_out,'(7f10.4)') x
  WRITE (nag_std_out,*)

  DEALLOCATE (x,xhat,trig)     ! Deallocate storage
```

```
    END PROGRAM nag_sym_fft_ex01
```

2 Program Data

```
Example Program Data for nag_sym_fft_ex01
  8                                                    : n
0.3854 0.6772  0.1138  0.6751  0.6362  0.1424 0.9562 : (z(i),i=1,n-1)
```

3 Program Results

```
 Example Program Results for nag_sym_fft_ex01

 Original data
    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562

 Transformed data
    1.2305   -0.1191    0.4285   -0.0242    0.5240   -0.6539   -0.0242

 Original data restored by inverse transform
    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
```

Example 2: Discrete Fourier cosine transforms of multiple 1-d sequences

This program reads sequences of real data values and prints their discrete Fourier cosine transforms. The inverse transforms are then calculated and printed out, showing that the original sequences are restored.

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_sym_fft_ex02

  ! Example Program Text for nag_sym_fft
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_fft, ONLY : nag_fft_cos
  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 :: x(:,:), xhat(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_fft_ex02'

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

  ALLOCATE (x(m,n+1),xhat(m,n+1)) ! Allocate storage

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

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

  WRITE (nag_std_out,*)

  xhat = nag_fft_cos(x)

  CALL nag_write_gen_mat(xhat,format='f10.4',int_row_labels=.TRUE., &
   title='Transformed data')

  WRITE (nag_std_out,*)

  x = nag_fft_cos(xhat)

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

  DEALLOCATE (x,xhat)          ! Deallocate storage
```

```
    END PROGRAM nag_sym_fft_ex02
```

2 Program Data

```
Example Program Data for nag_sym_fft_ex02
3  6                                                    : m, n
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  0.9562 : (z(1,i),i=1,n+1)
 0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  0.4936 : (z(2,i),i=1,n+1)
 0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  0.2057 : (z(3,i),i=1,n+1)
```

3 Program Results

```
 Example Program Results for nag_sym_fft_ex02

 Original data
 1     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
 2     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723    0.4936
 3     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815    0.2057

 Transformed data
 1     1.6833   -0.0482    0.0176    0.1368    0.3240   -0.5830   -0.0427
 2     1.9605   -0.4884   -0.0655    0.4444    0.0964    0.0856   -0.2289
 3     1.3838    0.1588   -0.0761   -0.1184    0.3512    0.5759    0.0110

 Original data restored by inverse transform
 1     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
 2     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723    0.4936
 3     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815    0.2057
```

Example 3: Discrete quarter-wave Fourier sine transforms of multiple 1-d sequences

This program reads sequences of real data values and prints their discrete quarter-wave Fourier sine transforms. The inverse transforms are then calculated and printed out, showing that the original sequences are restored.

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_sym_fft_ex03

  ! Example Program Text for nag_sym_fft
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_fft, ONLY : nag_fft_qtr_sin
  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 :: x(:,:), xhat(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_fft_ex03'

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

  ALLOCATE (x(m,n),xhat(m,n))  ! Allocate storage

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

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

  WRITE (nag_std_out,*)

  xhat = nag_fft_qtr_sin(x)

  CALL nag_write_gen_mat(xhat,format='f10.4',int_row_labels=.TRUE., &
   title='Transformed data')

  WRITE (nag_std_out,*)

  x = nag_fft_qtr_sin(xhat,inverse=.TRUE.)

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

```
      DEALLOCATE (x,xhat)          ! Deallocate storage

    END PROGRAM nag_sym_fft_ex03
```

2 Program Data

```
Example Program Data for nag_sym_fft_ex03
3  6                                                : m, n
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  : (z(1,i),i=1,n)
 0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  : (z(2,i),i=1,n)
 0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  : (z(3,i),i=1,n)
```

3 Program Results

```
 Example Program Results for nag_sym_fft_ex03

 Original data
 1     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
 2     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
 3     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

 Transformed data
 1     0.7304    0.2078    0.1150    0.2577   -0.2869   -0.0815
 2     0.9274   -0.1152    0.2532    0.2883   -0.0026   -0.0635
 3     0.6268    0.3547    0.0760    0.3078    0.4987   -0.0507

 Original data restored by inverse transform
 1     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
 2     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
 3     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

Example 4: Discrete quarter-wave Fourier cosine transform of a single 1-d sequence

This program reads a sequence of real data values and prints its discrete quarter-wave Fourier cosine transform. The inverse transform is then calculated and printed out, showing that the original sequence is restored. This example program also shows the use of the procedure `nag_fft_trig` from module `nag_fft` (7.1). Note that `nag_fft_trig` is accessed through the `USE` statement for `nag_sym_fft`.

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_sym_fft_ex04

  ! Example Program Text for nag_sym_fft
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_fft, ONLY : nag_fft_qtr_cos, nag_fft_trig
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: trig(:), x(:), xhat(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_fft_ex04'

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

  ALLOCATE (x(n),xhat(n),trig(2*n)) ! Allocate storage

  READ (nag_std_in,*) x
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Original data'
  WRITE (nag_std_out,'(7f10.4)') x
  WRITE (nag_std_out,*)

  CALL nag_fft_trig(trig)

  xhat = nag_fft_qtr_cos(x,trig=trig)

  WRITE (nag_std_out,*) 'Transformed data'
  WRITE (nag_std_out,'(7f10.4)') xhat
  WRITE (nag_std_out,*)

  x = nag_fft_qtr_cos(xhat,inverse=.TRUE.,trig=trig)

  WRITE (nag_std_out,*) 'Original data restored by inverse transform'
  WRITE (nag_std_out,'(7f10.4)') x
  WRITE (nag_std_out,*)

  DEALLOCATE (x,xhat,trig)     ! Deallocate storage
```

```
   END PROGRAM nag_sym_fft_ex04
```

2 Program Data

```
Example Program Data for nag_sym_fft_ex04
 6                                                 : n
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424 : (z(i),i=1,n)
```

3 Program Results

```
 Example Program Results for nag_sym_fft_ex04

 Original data
    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424

 Transformed data
    0.7257   -0.2216    0.1011    0.2355   -0.1406   -0.2282

 Original data restored by inverse transform
    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
```

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_sym_fft_ex05`

Discrete Fourier sine transforms of multiple 1-d sequences

`nag_sym_fft_ex06`

Discrete Fourier cosine transform of a single 1-d sequence

`nag_sym_fft_ex07`

Discrete quarter-wave Fourier sine transform of a single 1-d sequence

`nag_sym_fft_ex08`

Discrete quarter-wave Fourier cosine transforms of multiple 1-d sequences

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] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[5] Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computation* (ed G Rodrique) Academic Press 51–83

[6] Swarztrauber P N (1984) Fast Poisson solvers *Studies in Numerical Analysis* (ed G H Golub) Mathematical Association of America

[7] Van Loan C (1992) *Computational Frameworks for the Fast Fourier Transform* SIAM, Philadelphia