Module 6.1: nag_sym_eig
Standard Symmetric Eigenvalue Problems

`nag_sym_eig` provides procedures for solving standard eigenvalue problems

$$Az = \lambda z$$

where the matrix A is *real symmetric* or *complex Hermitian*. It also provides procedures for performing various computational sub-tasks involved in solving such problems.

Contents

Introduction

1 Notation

The *symmetric eigenvalue problem* is to find the *eigenvalues*, λ_i, and the corresponding eigenvectors, z_i, of a real symmetric matrix A such that

$$Az_i = \lambda_i z_i \text{ for } i = 1, \ldots, n,$$

where n is the order of A. The *Hermitian eigenvalue problem* is defined likewise for complex Hermitian matrices. For both problems the eigenvalues λ_i are real.

We can also write:

$A = Z\Lambda Z^T$ with Z orthogonal, if A is real;

$A = Z\Lambda Z^H$ with Z unitary, if A is complex,

where Λ is a real diagonal matrix whose diagonal elements are the eigenvalues, and the columns of Z are the eigenvectors. This is called the *spectral factorization* of A.

2 Choice of Procedures

The procedures `nag_sym_eig_all` and `nag_sym_eig_sel` have been designed to meet most requirements. They solve the most frequent types of problems in a single call, namely:

All eigenvalues of A (`nag_sym_eig_all`)

All eigenvalues and eigenvectors of A (`nag_sym_eig_all` with optional argument)

Selected eigenvalues of A (`nag_sym_eig_sel`)

Selected eigenvalues and the corresponding eigenvectors of A (`nag_sym_eig_sel` with optional argument)

The procedures `nag_sym_eig_all` and `nag_sym_eig_sel` call a number of lower-level procedures, each to perform a distinct computational step. You can accomplish a wider variety of tasks by calling the lower-level procedures directly, either individually or in various combinations.

The lower-level procedures are the following.

`nag_sym_tridiag_reduc` reduces the matrix A to a real symmetric tridiagonal matrix T by an orthogonal similarity transformation so that A is factorized as $A = QTQ^T$, where Q is orthogonal (or $A = QTQ^H$ with Q unitary, if A is complex).

`nag_sym_tridiag_orth` (using information returned by `nag_sym_tridiag_reduc`) either forms the matrix Q or computes the product of Q and a given matrix C; this is useful in particular when C is a matrix of eigenvectors of T computed by `nag_sym_tridiag_eig_vec` and QC is the matrix of the corresponding eigenvectors of A.

`nag_sym_tridiag_eig_all` computes all the eigenvalues, and optionally the eigenvectors, of the real symmetric tridiagonal matrix T; or, in other words, computes the spectral factorization of T as $T = S\Lambda S^T$, where S is orthogonal. The eigenvalues of T are the eigenvalues of A, and the eigenvectors of A are given by $Z = QS$.

`nag_sym_tridiag_eig_val` computes selected eigenvalues of T by bisection.

`nag_sym_tridiag_eig_vec` computes selected eigenvectors of T, corresponding to given eigenvalues, by inverse iteration.

The lower-level procedures are intended for use by more experienced users.

3 Storage of Matrices

The procedures in this module allow a choice of storage schemes for the symmetric or Hermitian matrix A: conventional storage or packed storage. The choice is determined by the rank of the corresponding argument `a`.

3.1 Conventional Storage

`a` is a rank-2 array, of shape (n,n). Matrix element a_{ij} is stored in `a(`i, j`)`. Only the elements of either the upper or the lower triangle need be stored, as specified by the argument `uplo`; the remaining elements of `a` need not be set.

This storage scheme is more straightforward and carries less risk of user error than packed storage; on some machines it may result in more efficient execution. But it requires almost twice as much memory, although the other triangle of `a` can sometimes be used to store other data, and if the matrix Z of eigenvectors is required, it can also be stored in `a`, overwriting the matrix A.

3.2 Packed Storage

`a` is a rank-1 array of shape $(n(n+1)/2)$. The elements of either the upper or the lower triangle of A, as specified by `uplo`, are packed by columns into contiguous elements of `a`.

Packed storage is more economical in use of memory than conventional storage, but if all eigenvectors are required, a separate rank-2 array `z` must be supplied to store them. Packed storage may also result in less efficient execution on some machines.

The details of packed storage are as follows:

- if `uplo` = `'u'` or `'U'`, a_{ij} is stored in `a(`$i + j(j-1)/2$`)`, for $i \leq j$;
- if `uplo` = `'l'` or `'L'`, a_{ij} is stored in `a(`$i + (2n-j)(j-1)/2$`)`, for $i \geq j$.

For example

`uplo`	Hermitian Matrix	Packed storage in array `a`
`'u'` or `'U'`	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \overline{a}_{12} & a_{22} & a_{23} & a_{24} \\ \overline{a}_{13} & \overline{a}_{23} & a_{33} & a_{34} \\ \overline{a}_{14} & \overline{a}_{24} & \overline{a}_{34} & a_{44} \end{pmatrix}$	a_{11} $\underbrace{a_{12}\, a_{22}}$ $\underbrace{a_{13}\, a_{23}\, a_{33}}$ $\underbrace{a_{14}\, a_{24}\, a_{34}\, a_{44}}$
`'l'` or `'L'`	$\begin{pmatrix} a_{11} & \overline{a}_{21} & \overline{a}_{31} & \overline{a}_{41} \\ a_{21} & a_{22} & \overline{a}_{32} & \overline{a}_{42} \\ a_{31} & a_{32} & a_{33} & \overline{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}\, a_{21}\, a_{31}\, a_{41}}$ $\underbrace{a_{22}\, a_{32}\, a_{42}}$ $\underbrace{a_{33}\, a_{43}}$ a_{44}

Note that for symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. For Hermitian matrices, packing the upper triangle by columns is equivalent to packing the conjugate of the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the conjugate of the upper triangle by rows.

Procedure: nag_sym_eig_all

1 Description

nag_sym_eig_all is a generic procedure which computes all the eigenvalues, and optionally all the eigenvectors, of a real symmetric or complex Hermitian matrix A of order n. It allows either conventional or packed storage for A (see the Module Introduction).

By default, only eigenvalues are computed. If the optional argument z_on_a is present and set to .true., the eigenvectors are computed and overwritten on the original matrix A (this option is only available if conventional storage is used); otherwise if the optional argument z is present, the eigenvectors are computed and stored in z.

We write:

$$Az_i = \lambda_i z_i \text{ for } i = 1, \ldots, n,$$

where λ_i is an eigenvalue of A, and z_i is the corresponding eigenvector. We also write

$A = Z\Lambda Z^T$ with Z orthogonal, if A is real;

$A = Z\Lambda Z^H$ with Z unitary, if A is complex;

where Λ is a real diagonal matrix whose diagonal elements are the eigenvalues, and the columns of Z are the eigenvectors.

Each eigenvector z_i is normalized so that $\|z_i\|_2 = 1$, and its component of largest absolute value is (real and) positive.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_eig_all(uplo, a, lambda  [, optional arguments])
```

2.1 Interfaces

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

Real / complex data

Real data: a and the optional argument z are of type real(kind=*wp*).

Complex data: a and the optional argument z are of type complex(kind=*wp*).

Conventional / packed storage (see the Module Introduction)

Conventional: a is a rank-2 array.

Packed: a is a rank-1 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 '**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 — the order of the matrix A

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied.

If `uplo` = `'u'` or `'U'`, the upper triangle is supplied;

if `uplo` = `'l'` or `'L'`, the lower triangle is supplied.

Constraints: `uplo` = `'u'`, `'U'`, `'l'` or `'L'`.

a(n, n) **/ a**$(n(n+1)/2)$ — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the matrix A.

Conventional storage (`a` has shape (n, n))

If `uplo` = `'u'`, the upper triangle of A must be stored, and elements below the diagonal need not be set;

if `uplo` = `'l'`, the lower triangle of A must be stored, and elements above the diagonal need not be set.

Packed storage (`a` has shape $(n(n+1)/2)$)

If `uplo` = `'u'`, the upper triangle of A must be stored, packed by columns, with a_{ij} in `a(`$i + j(j-1)/2$`)` for $i \leq j$;

if `uplo` = `'l'`, the lower triangle of A must be stored, packed by columns, with a_{ij} in `a(`$i + (2n-j)(j-1)/2$`)` for $i \geq j$.

Output: if `z_on_a` is present and set to `.true.`(conventional storage only), `a` is overwritten by the matrix Z of eigenvectors; otherwise (by default), `a` is overwritten by intermediate results.

lambda(n) — real(kind=*wp*), intent(out)

Output: the eigenvalues in ascending order.

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.

z_on_a — logical, intent(in), optional

Input: specifies whether the matrix Z of eigenvectors is to be overwritten on `a`.

If `z_on_a` = `.false.`, Z is not computed unless `z` is present;

if `z_on_a` = `.true.`, Z is overwritten on `a`.

Default: `z_on_a` = `.false.`.

Constraints: `z_on_a` must *not* be present if packed storage is used (`a` has rank 1).

z(n, n) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the matrix Z of eigenvectors. The ith column `z(`$:, i$`)` holds the eigenvector corresponding to the eigenvalue `lambda(`i`)`.

Note: if `z_on_a` is present and set to `.true.`, and `z` is also present, then `z` is not used and a warning is raised.

Constraints: `z` must be of the same type as `a`.

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.

Failures (error%level = 2):

error%code	Description
201	Failure to converge. (This error is not likely to occur.) The QR algorithm failed to compute all the eigenvalues in the permitted number of iterations.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. z is present when z_on_a is .true.; the eigenvectors are returned in a, and z is not used.

5 Examples of Usage

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

That example uses the following call statement to compute all the eigenvalues and eigenvectors of A, storing the eigenvectors in an array z:

```
CALL nag_sym_eig_all(uplo,a,lambda,z=z)
```

To overwrite the eigenvectors on a (only possible if conventional storage is used for A), the call statement should be changed to:

```
CALL nag_sym_eig_all(uplo,a,lambda,z_on_a=.true.)
```

6 Further Comments

6.1 Algorithmic Detail

The procedure first calls nag_sym_tridiag_reduc to reduce A to real symmetric tridiagonal form T, forming the transformation matrix Q if eigenvectors are required. It then calls nag_sym_tridiag_eig_all to compute the eigenvalues of T and, if required, the eigenvectors, using the QR algorithm. See the documents of those procedures for more details, and Chapter 8 of Golub and Van Loan [3] or Parlett [4] for background.

The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The computed eigenvalues and eigenvectors are exact for a nearby matrix $A + E$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ = EPSILON(1.0_*wp*).

If λ_i is an exact eigenvalue, and $\tilde{\lambda}_i$ is the corresponding computed value, then

$$|\tilde{\lambda}_i - \lambda_i| \leq c(n)\epsilon\|A\|_2,$$

where $c(n)$ is a modestly increasing function of n.

If z_i is the corresponding exact eigenvector and $\tilde{z}_i$ the computed eigenvector, then the angle $\theta(\tilde{z}_i, z_i)$ between them is bounded as follows:

$$\theta(\tilde{z}_i, z_i) \leq \frac{c(n)\epsilon\|A\|_2}{\min_{i\neq j}|\lambda_i - \lambda_j|}.$$

Thus the accuracy of a computed eigenvector depends on the gap between its eigenvalue and all the other eigenvalues.

6.3 Timing

The time taken by the procedure is approximately proportional to n^3. Computing both eigenvectors and eigenvalues is likely to take about 5 times as long as computing eigenvalues alone.

Procedure: nag_sym_eig_sel

1 Description

nag_sym_eig_sel is a generic procedure which computes selected eigenvalues, and optionally the corresponding eigenvectors, of a real symmetric or complex Hermitian matrix A of order n. It allows either conventional or packed storage for A (see the Module Introduction).

We write:

$$Az_i = \lambda_i z_i \text{ for } i = 1, \ldots, n,$$

where λ_i is an eigenvalue of A, and z_i is the corresponding eigenvector. The eigenvalues are indexed in ascending order:

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n.$$

Eigenvalues may be selected either by index or by value (but not by a combination of the two). If either or both of the optional arguments il and iu are present, the procedure computes those eigenvalues λ_i whose indices i satisfy

$$\texttt{il} \leq i \leq \texttt{iu}.$$

If either or both of the optional arguments vl and vu are present, it computes those eigenvalues λ which satisfy

$$\texttt{vl} < \lambda \leq \texttt{vu}.$$

By default, only eigenvalues are computed. The eigenvectors corresponding to the selected eigenvalues are computed only if the optional argument z is present.

The number of selected eigenvalues is denoted by m. The argument lambda and the optional arguments z and fail are pointer arrays, because, if eigenvalues are selected by value, the number of them in the specified range may not be known in advance. If eigenvalues are selected by index, $m =$ iu $-$ il $+$ 1. The procedure allocates the required amount of memory to lambda, z and fail; on exit from the procedure, $m =$ SIZE(lambda).

Each eigenvector z_i is normalized so that $\|z_i\|_2 = 1$, and its component of largest absolute value is (real and) positive.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_eig_sel(uplo, a, lambda  [, optional arguments])
```

2.1 Interfaces

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

Real / complex data

Real data: a and the optional argument z are of type real(kind=*wp*).

Complex data: a and the optional argument z are of type complex(kind=*wp*).

Conventional / packed storage (see the Module Introduction)

Conventional: `a` is a rank-2 array.

Packed: `a` is a rank-1 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 '**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 — the order of the matrix A

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied.

If `uplo` = `'u'` or `'U'`, the upper triangle is supplied;

if `uplo` = `'l'` or `'L'`, the lower triangle is supplied.

Constraints: `uplo` = `'u'`, `'U'`, `'l'` or `'L'`.

a(n, n) / **a**($n(n+1)/2$) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the matrix A.

Conventional storage (`a` has shape (n, n))

If `uplo` = `'u'`, the upper triangle of A must be stored, and elements below the diagonal need not be set;

if `uplo` = `'l'`, the lower triangle of A must be stored, and elements above the diagonal need not be set.

Packed storage (`a` has shape $(n(n+1)/2)$)

If `uplo` = `'u'`, the upper triangle of A must be stored, packed by columns, with a_{ij} in `a(`$i + j(j-1)/2$`)` for $i \leq j$;

if `uplo` = `'l'`, the lower triangle of A must be stored, packed by columns, with a_{ij} in `a(`$i + (2n-j)(j-1)/2$`)` for $i \geq j$.

Output: `a` is overwritten by intermediate results.

lambda(:) — real(kind=*wp*), pointer

Output: the m selected eigenvalues in ascending order.

Note: the procedure creates a target array of shape (m). If there are no eigenvalues in the selected interval, then $m = 0$.

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.

il — integer, intent(in), optional

iu — integer, intent(in), optional

Input: the first and last indices, respectively, of the selected eigenvalues, where the eigenvalues are indexed in ascending order. An eigenvalue λ_i is selected if `il` $\leq i \leq$ `iu`.

Default: `il` = 1, `iu` = n.

Constraints: Min(n, 1) $\leq$ `il` $\leq$ `iu` $\leq n$; `il` and `iu` must not be present if either `vl` or `vu` is present.

vl — real(kind=*wp*), intent(in), optional

vu — real(kind=*wp*), intent(in), optional

Input: the lower and upper bounds, respectively, on the selected eigenvalues. An eigenvalue λ is selected if $\texttt{vl} < \lambda \leq \texttt{vu}$.

Default: $\texttt{vl} = -\infty$, $\texttt{vu} = +\infty$.

Constraints: $\texttt{vl} \leq \texttt{vu}$; vl and vu must not be present if either il or iu is present.

abs_tol — real(kind=*wp*), intent(in), optional

Input: the absolute tolerance for the eigenvalues. An eigenvalue (or cluster) is accepted if it has been determined to lie in an interval whose width is less than or equal to abs_tol. If $\texttt{abs\_tol} \leq 0$, then the default value is used.

Default: $\texttt{abs\_tol} = \epsilon \|A\|_1$, where $\epsilon =$ EPSILON(1.0_*wp*).

z(:,:) — real(kind=*wp*) / complex(kind=*wp*), pointer, optional

Output: the m selected eigenvectors. The ith column $\texttt{z}(:, i)$ holds the eigenvector corresponding to the eigenvalue $\texttt{lambda}(i)$. See also fail.

Note: the procedure creates a target array of shape (n, m). If there are no eigenvalues in the selected interval, then $m = 0$.

Constraints: z must be of the same type as a.

fail(:) — integer, pointer, optional

Output: on successful exit, all elements of fail are set to 0. If error code 202 is returned, the leading elements of fail hold the column indices (in z) of those eigenvectors which failed to converge, and the remaining elements are set to 0. For example, if fail(1) = 2, the eigenvector in column 2 of z failed to converge.

Note: the procedure creates a target array of shape (m).

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.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Failure to converge. The bisection algorithm failed to find all the specified eigenvalues.
202	Failure to converge. The inverse iteration algorithm failed to converge to one or more eigenvectors in 5 iterations; the most recent iterate is stored in the corresponding column of z. If k eigenvectors failed to converge, their indices are returned in $\texttt{fail}(1:k)$ (if present).

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

That example uses the following call statement to select eigenvalues by index, with indices from `il` to `iu`:

```
CALL nag_sym_eig_sel(uplo,a,lambda,il=il,iu=iu,z=z)
```

To select eigenvalues in the range -1.0 to $+1.0$, the call statement should be changed to:

```
CALL nag_sym_eig_sel(uplo,a,lambda,vl=-1.0_wp,vu=+1.0_wp,z=z)
```

6 Further Comments

6.1 Algorithmic Detail

The procedure first calls `nag_sym_tridiag_reduc` to reduce A to real symmetric tridiagonal form T. It then calls `nag_sym_tridiag_eig_val` to compute the specified eigenvalues of T. If eigenvectors are required, it calls `nag_sym_tridiag_eig_vec` to compute the corresponding eigenvectors of T, and then calls `nag_sym_tridiag_orth` to transform them to eigenvectors of A. See the documents of those procedures for more details, and Chapter 8 of Golub and Van Loan [3] or Parlett [4] for background.

The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

If λ_i is an exact eigenvalue, and $\tilde{\lambda}_i$ is the corresponding computed value, then

$$|\tilde{\lambda}_i - \lambda_i| \leq c(n)\epsilon\|A\|_2,$$

where $c(n)$ is a modestly increasing function of n.

Each computed eigenvector z_i is the exact eigenvector of a nearby matrix $A + E_i$, such that $\|E_i\|_2 = O(\epsilon)\|A\|$, where $\epsilon =$ `EPSILON(1.0_wp)`. Hence the residual is small:

$$\|Az_i - \lambda_i z_i\|_2 = O(\epsilon)\|A\|_2.$$

However, a set of eigenvectors computed by this procedure may not be orthogonal to so high a degree of accuracy as those computed by `nag_sym_eig_all`.

Procedure: nag_sym_tridiag_reduc

1 Description

nag_sym_tridiag_reduc is a generic procedure which reduces a real symmetric or complex Hermitian matrix A of order n to a real symmetric tridiagonal matrix T by an orthogonal or unitary similarity transformation. It allows either conventional or packed storage for A (see the Module Introduction).

The transformation is written

$A = QTQ^T$ with Q orthogonal, if A is real;

$A = QTQ^H$ with Q unitary, if A is complex.

By default, the transformation matrix Q is represented as the product of $n-1$ elementary reflectors (see the Module Introduction for details); this representation can be passed to the procedure nag_sym_tridiag_orth to perform further operations with Q.

Optionally the matrix Q can be formed explicitly (this may be required for a subsequent call to nag_sym_tridiag_eig_all). If the optional argument q_on_a is present and set to .true., Q is overwritten on the original matrix A (this option is only available if conventional storage is used); otherwise if the optional argument q is present, Q is stored in q.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_tridiag_reduc(uplo, a, d, e  [, optional arguments])
```

2.1 Interfaces

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

Real / complex data

Real data: a and the optional arguments q and tau are of type real(kind=*wp*).

Complex data: a and the optional arguments q and tau are of type complex(kind=*wp*).

Conventional / packed storage (see the Module Introduction)

Conventional: a is a rank-2 array.

Packed: a is a rank-1 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 '**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 — the order of the matrix A

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied.

If uplo = 'u' or 'U', the upper triangle is supplied;

if uplo = 'l' or 'L', the lower triangle is supplied.

Constraints: uplo = 'u', 'U', 'l' or 'L'.

a(n,n) / **a**$(n(n+1)/2)$ — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the matrix A.

Conventional storage (`a` has shape (n,n))

If `uplo` = `'u'`, the upper triangle of A must be stored, and elements below the diagonal need not be set;

if `uplo` = `'l'`, the lower triangle of A must be stored, and elements above the diagonal need not be set.

Packed storage (`a` has shape $(n(n+1)/2)$)

If `uplo` = `'u'`, the upper triangle of A must be stored, packed by columns, with a_{ij} in `a(`$i+j(j-1)/2$`)` for $i \leq j$;

if `uplo` = `'l'`, the lower triangle of A must be stored, packed by columns, with a_{ij} in `a(`$i+(2n-j)(j-1)/2$`)` for $i \geq j$.

Output: if `q_on_a` is present and set to `.true.`(conventional storage only), `a` is overwritten by the transformation matrix Q; otherwise (by default), `a` is overwritten by details of Q, represented as a product of elementary reflectors; this form may subsequently be passed (with `tau`) to `nag_sym_tridiag_orth`.

d(n) — real(kind=*wp*), intent(out)

Output: the diagonal elements of the tridiagonal matrix T.

e$(n-1)$ — real(kind=*wp*), intent(out)

Output: the off-diagonal elements of the tridiagonal matrix T.

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.

q_on_a — logical, intent(in), optional

Input: specifies whether the transformation matrix Q is to be overwritten on `a`.

If `q_on_a` = `.false.`, Q is not formed unless `q` is present;

if `q_on_a` = `.true.`, Q is overwritten on `a`.

Default: `q_on_a` = `.false.`.

Constraints: `q_on_a` must *not* be present if packed storage is used (`a` has rank 1).

q(n,n) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the transformation matrix Q.

Note: if `q_on_a` is present and set to `.true.`, and `q` is also present, then `q` is not used and a warning is raised.

Constraints: `q` must be of the same type as `a`.

tau(n) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: further details of the transformation matrix Q; `a` and `tau` together may be required for passing to `nag_sym_tridiag_orth`.

Constraints: `tau` must be of the same type as `a`.

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.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. q is present when q_on_a is .true.: the matrix Q is returned in a, and q is not used.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 3 and 4 of this module document.

The first example uses the following call statement to overwrite the transformation matrix Q on a (only possible if conventional storage is used for A):

```
CALL nag_sym_tridiag_reduc(uplo,a,d,e,q_on_a=.true.)
```

To store Q in a separate array of shape (n, n), the call statement should be changed to:

```
CALL nag_sym_tridiag_reduc(uplo,a,d,e,q=q)
```

If it is not convenient to form Q at the same time as the reduction to tridiagonal form, the call to this procedure could be replaced by a call to this procedure followed by a call to nag_sym_tridiag_orth; an array tau of shape (n) is needed to pass extra information about Q from one procedure to the other:

```
CALL nag_sym_tridiag_reduc(uplo,a,d,e,tau=tau)
 . . .
CALL nag_sym_tridiag_orth(uplo,a,tau,q_on_a=.true.)
```

6 Further Comments

6.1 Algorithmic Detail

The reduction is performed by applying elementary reflectors (Householder matrices), as described in Golub and Van Loan [3], Section 8.2.

The algorithm is derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The computed tridiagonal matrix T is exactly similar to a nearby matrix $A + E$, where

$$\|E\|_2 = c(n)\epsilon\|A\|_2,$$

$c(n)$ is a modestly increasing function of n, and ϵ = EPSILON(1.0_*wp*).

The elements of T themselves may be sensitive to small perturbations in A or to rounding errors in the computation, but this does not affect the stability of the eigenvalues and eigenvectors.

The computed matrix Q differs from an exactly orthogonal or unitary matrix by a matrix E such that $\|E\|_2 = O(\epsilon)$.

6.3 Timing

For real data, the total number of floating-point operations performed is roughly as follows:

to reduce A to tridiagonal form:	$(4/3)n^3$
to form the transformation matrix Q:	$(4/3)n^3$

For complex data, 4 times as many (real) floating-point operations are performed.

Procedure: nag_sym_tridiag_orth

1 Description

nag_sym_tridiag_orth is intended for use following a call to nag_sym_tridiag_reduc.

nag_sym_tridiag_reduc reduces a real symmetric or complex Hermitian matrix A of order n to a real symmetric tridiagonal matrix T by an orthogonal similarity transformation. The transformation is written

$A = QTQ^T$ with Q orthogonal, if A is real;

$A = QTQ^H$ with Q unitary, if A is complex.

The transformation matrix Q is represented as the product of $n-1$ elementary reflectors:

$$Q = H_1 H_2 \dots H_{n-1}$$

where

$$H_i = I - \tau_i v_i v_i^H;$$

the vector v_i is stored in the ith column of the upper or lower triangle of a, and the scalar τ_i is stored in tau(i).

nag_sym_tridiag_orth is a generic procedure which accepts this representation of a real orthogonal or complex unitary matrix Q and either:

forms the matrix Q explicitly, or

applies Q to a given real or complex matrix C from the left or right, overwriting C with QC, CQ, Q^TC or CQ^T (Q^HC or CQ^H if Q is complex).

This procedure allows either conventional or packed storage for the input representation of Q, depending on which storage scheme was used for the matrix A on input to the preceding call to nag_sym_tridiag_reduc.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_tridiag_orth(uplo, a, tau  [, optional arguments])
```

2.1 Interfaces

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

Real / complex data

Real data: a, tau, and the optional arguments q and c, are of type real(kind=*wp*).

Complex data: a, tau, and the optional arguments q and c, are of type complex(kind=*wp*).

Conventional / packed storage (see the Module Introduction)

Conventional: a is a rank-2 array.

Packed: a is a rank-1 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 '**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.

n — the order of the matrix Q

m_C — the number of rows of the matrix C

n_C — the number of columns of the matrix C

If Q is applied from the left, $m_C = n$; if Q is applied from the right, $n_C = n$.

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: must have the same value as in the previous call of `nag_sym_tridiag_reduc`. It specifies whether the upper or lower triangle of A was supplied to `nag_sym_tridiag_reduc`.

If `uplo` = `'u'` or `'U'`, the upper triangle was supplied;

if `uplo` = `'l'` or `'L'`, the lower triangle was supplied.

Constraints: `uplo` = `'u'`, `'U'`, `'l'` or `'L'`.

a(n, n) / **a**($n(n+1)/2$) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: details of the representation of Q as returned by `nag_sym_tridiag_reduc` with `q_on_a` = `.false.`.

Output: if `q_on_a` is present and set to `.true.`(conventional storage only), `a` is overwritten by the transformation matrix Q; otherwise (by default), `a` is unchanged.

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

Input: further details of the representation of Q as returned by `nag_sym_tridiag_reduc`.

Constraints: `tau` must be of the same type as `a`.

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.

q_on_a — logical, intent(in), optional

Input: specifies whether the transformation matrix Q is to be overwritten on `a`.

If `q_on_a` = `.true.`, Q is overwritten on `a`;

if `q_on_a` = `.false.`, Q is returned in `q` if present, or else is not formed explicitly.

Default: `q_on_a` = `.false.`.

Constraints: `q_on_a` must *not* be present if packed storage is used (`a` has rank 1).

q(n, n) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the transformation matrix Q.

Note: if `q_on_a` is present and set to `.true.`, and `q` is also present, then `q` is not used and a warning is raised.

Constraints: `q` must be of the same type as `a`.

c(m_C, n_C) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: the matrix C.

Output: overwritten by QC, Q^TC, Q^HC, CQ, CQ^T or CQ^H, according to the values of `side` and `trans`.

Constraints: `c` must be of the same type as `a`.

side — character(len=1), intent(in), optional

Input: specifies whether Q (or Q^T or Q^H) is to be applied to C from the left or right.

If `side` = `'l'` or `'L'`, from the left;

if `side` = `'r'` or `'R'`, from the right.

Default: `side` = `'l'`.

Constraints: `side` = `'l'`, `'L'`, `'r'` or `'R'`; `side` must not be present unless `c` is present.

trans — character(len=1), intent(in), optional

Input: specifies whether Q, Q^T or Q^H is to be applied to C.

If `trans` = `'n'` or `'N'`, Q is applied;

if `trans` = `'t'` or `'T'` (real matrices only), Q^T is applied;

if `trans` = `'c'` or `'C'` (complex matrices only), Q^H is applied.

Default: `trans` = `'n'`.

Constraints: for real matrices `trans` = `'n'`, `'N'`, `'t'` or `'T'`; for complex matrices `trans` = `'n'`, `'N'`, `'c'` or `'C'`; `trans` must not be present unless `c` is present.

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.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. `q` is present when `q_on_a` is `.true.`; the matrix Q is returned in `a`, and `q` is not used.
102	No computation performed. `q_on_a` is not present or is set to `.false.`, and neither `q` nor `c` is present; no computation has been requested.

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 algorithm is derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The computed matrix Q differs from an exactly orthogonal or unitary matrix by a matrix E such that $\|E\|_2 = O(\epsilon)$.

The computed result of applying Q to C differs from the exact result by a matrix E such that $\|E\|_2 = O(\epsilon)\|C\|_2$.

6.3 Timing

For real data, the total number of floating-point operations performed is roughly as follows:

to form the transformation matrix Q:	$(4/3)n^3$
to compute QC or Q^HC:	$2n^2n_C$
to compute CQ or CQ^H:	$2m_Cn^2$

For complex data, 4 times as many (real) floating-point operations are performed.

Procedure: nag_sym_tridiag_eig_all

1 Description

nag_sym_tridiag_eig_all computes all the eigenvalues, and, optionally all the eigenvectors, of a real symmetric tridiagonal matrix T of order n.

We write:

$$Ts_i = \lambda_i s_i \text{ for } i = 1, \ldots, n,$$

where λ_i is an eigenvalue of T, and s_i is the corresponding eigenvector. We also write

$$T = S\Lambda S^T,$$

where Λ is a real diagonal matrix whose diagonal elements are the eigenvalues, and S is an orthogonal matrix whose columns are the eigenvectors of T.

By default, only eigenvalues are computed. If the optional argument z is present, then eigenvectors are also computed and stored in z.

By default, the computed eigenvectors are those of T. However, the procedure can also be used to compute all the eigenvectors of a real symmetric or complex Hermitian matrix A which has been reduced to tridiagonal form by nag_sym_tridiag_reduc:

$$A = QTQ^T = (QS)\Lambda(S^TQ^T), \text{ if } A \text{ is real;}$$

$$A = QTQ^H = (QS)\Lambda(S^TQ^H), \text{ if } A \text{ is complex;}$$

and so $Z = QS$ is the matrix of eigenvectors of A.

To compute the eigenvectors of A, the optional argument z_tridiag must be present and set to .false., and on entry z must contain the matrix Q returned by nag_sym_tridiag_reduc (or possibly by nag_sym_tridiag_orth).

Each eigenvector z_i is normalized so that $\|z_i\|_2 = 1$, and its component of largest absolute value is (real and) positive.

A choice of algorithms is provided; see Section 6.1.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_tridiag_eig_all(nag_key, d, e  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for the following cases.

Real / complex data

Real data: nag_key = nag_key_real, and the optional argument z is of type real(kind=*wp*).

Complex data: nag_key = nag_key_cmplx, and the optional argument z is of type complex(kind=*wp*).

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 — the order of the matrix T

3.1 Mandatory Arguments

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

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

`nag_key_real` if the matrix Z of eigenvectors (and the original matrix A) is real;

`nag_key_cmplx` if the matrix Z of eigenvectors (and the original matrix A) is complex.

If `z` is not present, either value may be supplied. For further explanation of "key" arguments, see the Essential Introduction.

d(n) — real(kind=*wp*), intent(inout)

Input: the diagonal elements of the matrix T.

Output: the eigenvalues of T, arranged in ascending order.

e($n-1$) — real(kind=*wp*), intent(inout)

Input: the off-diagonal elements of the matrix T.

Output: overwritten by intermediate results.

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.

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

Input: if `z_tridiag` is `.true.`(the default), `z` need not be set on entry; if `z_tridiag` is `.false.`, `z` must contain the transformation matrix Q used by `nag_sym_tridiag_reduc` to reduce A to tridiagonal form.

Output: the matrix Z of eigenvectors of the tridiagonal matrix T, if `z_tridiag` is `.true.`, or of the original matrix A, if `z_tridiag` is `.false.`.
The ith column `z(:,`i`)` holds the eigenvector corresponding to the ith eigenvalue in `d(`i`)`.

Constraints: `z` must be of type real(kind=*wp*) if `nag_key` = `nag_key_real`, and of type complex(kind=*wp*) if `nag_key` = `nag_key_cmplex`.

z_tridiag — logical, intent(in), optional

Input: specifies whether the eigenvectors of the tridiagonal matrix T or of an original matrix A are to be computed.

If `z_tridiag` = `.true.`(the default), the eigenvectors of T are computed, and `z` need not be set on entry;

if `z_tridiag` = `.false.`, the eigenvectors of A are computed, and on entry `z` must contain the transformation matrix Q.

Default: `z_tridiag` = `.true.`.

Constraints: `z_tridiag` must not be present unless `z` is present.

method — character(len=1), intent(in), optional

Input: the method to be used (see also Section 6.1).

If `method` = `'q'` or `'Q'`, the QR algorithm with implicit shift;

if `method` = `'r'` or `'R'`, a root-free variant of the QR algorithm, suitable for eigenvalues only;

if `method` = `'s'` or `'S'`, the SVD algorithm applied to the Cholesky factor of T, suitable only for positive-definite matrices.

Note: if the specified algorithm is unsuitable, the default algorithm is used and a warning is raised.

Default: for eigenvalues only, `method` = `'r'`; for eigenvalues and eigenvectors, `method` = `'q'`.

Constraints: `method` = `'q'`, `'Q'`, `'r'`, `'R'`, `'s'` or `'S'`.

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.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Failure to converge. (This error is unlikely to occur.) The bidiagonal SVD algorithm (used if method = 's') failed to compute all the eigenvalues in the permitted number of iterations.
202	Failure to converge. (This error is unlikely to occur.) The tridiagonal QR algorithm or its root-free variant (used if method = 'q' or 'r') failed to compute all the eigenvalues in the permitted number of iterations. The symmetric tridiagonal matrix returned in d and e is orthogonally similar to T, and gives information about those eigenvalues which have converged.

Warnings (error%level = 1):

error%code	Description
101	The specified algorithm could not be used. The root-free QR algorithm (specified by method = 'r') could not be used because eigenvectors were requested; the ordinary QR algorithm was used.
102	The specified algorithm could not be used. The SVD algorithm (specified by method = 's') could not be used because T was not positive definite; the QR algorithm or its root-free variant was used according to whether or not the eigenvectors were required.

5 Examples of Usage

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

That example uses the following call statement to compute all the eigenvalues and eigenvectors of a full complex Hermitian matrix A which has been reduced to tridiagonal form by a preceding call to nag_sym_tridiag_reduc:

```
CALL nag_sym_tridiag_eig_all(nag_key_cmplx,d,e,z_tridiag=.false.,z=a)
```

To compute the eigenvalues and eigenvectors of the real symmetric tridiagonal matrix stored in `d` and `e`, an array `z` of type `real(kind=wp)` and shape (n, n) should be declared, and the call statement changed to:

```
CALL nag_sym_tridiag_eig_all(nag_key_real,d,e,z=z)
```

6 Further Comments

6.1 Algorithmic Detail

A choice of algorithms is provided, specified by the optional argument `method`

`method = 'q'`: the QR algorithm with implicit shift (the default if eigenvectors are required, that is, if `z` is present);

`method = 'r'`: a root-free variant of the QR algorithm with explicit shift (the default if only eigenvalues are required);

`method = 's'`: (assuming T is positive definite) performing a Cholesky factorization of T as $T = LL^T$ and then computing part of the singular value decomposition (SVD) of the bidiagonal matrix L: $L = U\Sigma V^T$, so that $T = U\Sigma^2 U^T$. This algorithm computes the small eigenvalues of T to high relative accuracy (see Demmel and Kahan [2]), but can only be used when T is positive definite. If this algorithm is attempted and the Cholesky factorization fails because T is not positive definite, the procedure uses the default algorithm instead.

All three algorithms switch between QR and QL variants in order to handle graded matrices effectively. The algorithms are derived from LAPACK (Anderson *et al.* [1]). See also Chapter 8 of Parlett [4] or Section 8.2 of Golub and Van Loan [3].

6.2 Accuracy

The computed eigenvalues and eigenvectors are exact for a nearby matrix $T + E$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and $\epsilon =$ `EPSILON(1.0_wp)`.

If λ_i is an exact eigenvalue and $\tilde{\lambda}_i$ is the corresponding computed value, then

$$|\tilde{\lambda}_i - \lambda_i| \leq c(n)\epsilon\|T\|_2,$$

where $c(n)$ is a modestly increasing function of n.

If z_i is the corresponding exact eigenvector and $\tilde{z}_i$ the computed eigenvector, then the angle $\theta(\tilde{z}_i, z_i)$ between them is bounded as follows:

$$\theta(\tilde{z}_i, z_i) \leq \frac{c(n)\epsilon\|A\|_2}{\min_{i\neq j}|\lambda_i - \lambda_j|}.$$

Thus the accuracy of a computed eigenvector depends on the gap between its eigenvalues and all the other eigenvalues.

If the SVD algorithm is used (`method = 's'`), stronger results hold good. The eigenvalues and eigenvectors are all computed to high relative accuracy, so that any small eigenvalues (and the corresponding eigenvectors) are computed more accurately than by the QR algorithm. However, if T was obtained by reducing a full positive definite matrix A to tridiagonal form, then rounding errors in the reduction may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of A, if its eigenvalues vary widely in magnitude.

The stronger bounds which hold when the SVD algorithm is used are as follows:

$$|\tilde{\lambda}_i - \lambda_i| \leq c(n)\epsilon\kappa_2(H)\lambda_i,$$

where H is the tridiagonal matrix defined by $h_{ij} = t_{ij}/\sqrt{t_{ii}t_{jj}}$, and $\kappa_2(H) = \|H\|_2\|H^{-1}\|_2$ (the 2-norm condition number of H);

$$\theta(\tilde{z}_i, z_i) \leq \frac{c(n)\epsilon\kappa_2(H)}{\text{relgap}_i},$$

where relgap_i is the relative gap between λ_i and the other eigenvalues, defined by

$$\text{relgap}_i = \min_{i \neq j} \frac{|\lambda_i - \lambda_j|}{\lambda_i + \lambda_j}.$$

6.3 Timing

The total number of floating-point operations performed by the different algorithms is typically as follows, but depends on how rapidly the algorithms converge:

	Eigenvalues only	Real eigenvectors	Complex eigenvectors
QR algorithm	$24n^2$	$7n^3$	$14n^3$
root-free QR algorithm	$14n^2$		
SVD algorithm	$30n^2$	$6n^3$	$12n^3$

When eigenvalues only are required, the operations are all performed in scalar mode; the additional operations to compute the eigenvectors can be vectorized and on some machines may be performed much faster.

Procedure: nag_sym_tridiag_eig_val

1 Description

nag_sym_tridiag_eig_val computes selected eigenvalues of a real symmetric tridiagonal matrix T of order n, using bisection.

Eigenvalues may be selected either by index or by value (but not by a combination of the two). If either or both of the optional arguments il and iu are present, the procedure computes those eigenvalues λ_i whose indices i satisfy

$$\texttt{il} \leq i \leq \texttt{iu}.$$

If either or both of the optional arguments vl and vu are present, it computes those eigenvalues λ which satisfy

$$\texttt{vl} < \lambda \leq \texttt{vu}.$$

The procedure searches for zero or negligible off-diagonal elements of T to see if the matrix splits into block diagonal form:

$$T = \begin{pmatrix} T_1 & & & \\ & T_2 & & \\ & & \ddots & \\ & & & T_{n_s} \end{pmatrix}.$$

It performs bisection on each of the blocks T_i, for $i = 1, \ldots, n_s$ and, if required, can return the block index of each computed eigenvalue in block, so that a subsequent call to nag_sym_tridiag_eig_vec to compute the corresponding eigenvectors can also take advantage of the block structure.

The number of selected eigenvalues is denoted by m. The argument lambda and the optional argument block are pointer arrays, because, if eigenvalues are selected by value, the number of them in the specified range may not be known in advance. If eigenvalues are selected by index, $m =$ iu $-$ il $+$ 1. The procedure allocates the required amount of memory to lambda and block; on exit from the procedure, $m =$ SIZE(lambda).

The selected eigenvalues are returned in ascending order for the entire matrix, irrespective of the block structure.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_tridiag_eig_val(d, e, lambda  [, 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 — the order of the matrix T

3.1 Mandatory Arguments

d(n) — real(kind=*wp*), intent(in)

Input: the diagonal elements of the matrix T.

e($n-1$) — real(kind=*wp*), intent(in)

Input: the off-diagonal elements of the matrix T.

lambda(:) — real(kind=*wp*), pointer

Output: the m selected eigenvalues, in ascending order.

Note: the procedure creates a target array of shape (m). If there are no eigenvalues in the selected interval then $m = 0$.

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.

il — integer, intent(in), optional

iu — integer, intent(in), optional

Input: the first and last indices, respectively, of the selected eigenvalues, where the eigenvalues are indexed in ascending order. An eigenvalue λ_i is selected if `il` $\leq i \leq$ `iu`.

Default: `il` $= 1$, `iu` $= n$.

Constraints: $1 \leq$ `il` $\leq$ `iu` $\leq n$ (or if $n = 0$, `il` $= 1$ and `iu` $= 0$) ; `il` and `iu` must not be present if either `vl` or `vu` is present.

vl — real(kind=*wp*), intent(in), optional

vu — real(kind=*wp*), intent(in), optional

Input: the lower and upper bounds, respectively, on the selected eigenvalues. An eigenvalue λ is selected if `vl` $< \lambda \leq$ `vu`.

Default: `vl` $= -\infty$, `vu` $= +\infty$.

Constraints: `vl` $\leq$ `vu`; `vl` and `vu` must not be present if either `il` or `iu` is present.

block(:) — integer, pointer, optional

Output: `block(`i`)` specifies the index of the block to which the ith eigenvalue `lambda(`i`)` belongs. If error code 201 is returned, `block(`i`)` < 0 indicates that the ith eigenvalue, returned in `lambda(`i`)`, failed to converge to the required accuracy.

Note: the procedure creates a target array of shape (m).

abs_tol — real(kind=*wp*), intent(in), optional

Input: the absolute tolerance for the eigenvalues. An eigenvalue (or cluster) is accepted if it has been determined to lie in an interval whose width $\leq$ `abs_tol`. If `abs_tol` ≤ 0, then the default value is used.

Default: `abs_tol` $= \epsilon \|T\|_1$, where $\epsilon =$ `EPSILON(1.0_`*wp*`)`.

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.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Failure to converge.
	The bisection algorithm has failed to compute some (or all) of the selected eigenvalues to the desired accuracy; if `lambda(`i`)` did not converge, then on exit `block(`i`)` < 0.
202	Failure to find all the specified eigenvalues.
	Eigenvalues were selected by index and the algorithm has failed to compute some or all of them; try calling the procedure again with different values of `il` and `iu`, or try using `nag_sym_tridiag_eig_all` to compute all the eigenvalues.
203	Error during the bisection.
	No eigenvalues have been computed; the floating point arithmetic on your computer is not behaving as expected.

These failures are unlikely to occur. If they are causing persistent trouble and you have checked that the procedure is being called correctly, please contact NAG.

5 Examples of Usage

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

That example uses the following call statement to compute the eigenvalues of T in the interval (`vl`,`vu`], and to store information in an integer array `block` for passing to a subsequent call of `nag_sym_tridiag_eig_vec`:

```
CALL nag_sym_tridiag_eig_val(d,e,lambda,vl=vl,vu=vu,block=block)
```

To compute the 3 largest eigenvalues of T (that is, those with indices $n-2$ to n), the call statement should be changed to:

```
CALL nag_sym_tridiag_eig_val(d,e,lambda,il=n-2)
```

6 Further Comments

6.1 Algorithmic Detail

For background on the bisection method, see Chapter 3 of Parlett [4] or Section 8.4.1 of Golub and Van Loan [3].

The algorithm is derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The eigenvalues of T are computed to high relative accuracy which means that if they vary widely in magnitude, then any small eigenvalues will be computed more accurately than, for example, with the standard QR algorithm. However, if T was obtained by reducing a full matrix A to tridiagonal form, then rounding errors in the reduction may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of A, if its eigenvalues vary widely in magnitude.

6.3 Timing

The total number of floating-point operations is $O(mn)$, but depends on how rapidly the algorithm converges.

Procedure: nag_sym_tridiag_eig_vec

1 Description

nag_sym_tridiag_eig_vec computes selected eigenvectors of a real symmetric tridiagonal matrix T, corresponding to specified eigenvalues, using inverse iteration.

The relevant eigenvalues of T must be supplied. They may be computed, for example, by the procedure nag_sym_tridiag_eig_val or nag_sym_tridiag_eig_all.

nag_sym_tridiag_eig_val also returns in the array block the index of the diagonal block to which each eigenvalue belongs; this also should be passed to this procedure because, if T splits into block diagonal form, it can reduce the amount of work and improve the orthogonality of the computed eigenvectors.

This procedure is generic, allowing the array z which holds the computed eigenvectors to be of type real(kind=*wp*) or complex(kind=*wp*) (although the values of the eigenvectors are purely real). The eigenvectors can subsequently be transformed, by calling nag_sym_tridiag_orth, to eigenvectors of a real symmetric or complex Hermitian matrix A, which has been reduced to tridiagonal form T by nag_sym_tridiag_reduc.

Each eigenvector z_i is normalized so that $\|z_i\|_2 = 1$, and its component of largest absolute value is positive.

2 Usage

```
USE nag_sym_eig
CALL nag_sym_tridiag_eig_vec(d, e, lambda, 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 '**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.

n — the order of the matrix T

m — the number of selected eigenvalues

3.1 Mandatory Arguments

d(n) — real(kind=*wp*), intent(in)

Input: the diagonal elements of the matrix T.

e($n-1$) — real(kind=*wp*), intent(in)

Input: the off-diagonal elements of the matrix T.

lambda(m) — real(kind=*wp*), intent(in)

Input: the selected eigenvalues.

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

Output: the selected eigenvectors. The ith column z(:, i) holds the eigenvector corresponding to the eigenvalue lambda(i). See also fail.

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.

block(m) — integer, intent(in), optional

Input: `block(`i`)` specifies the index of the block to which the ith eigenvalue `lambda(`i`)` belongs, as returned by `nag_sym_tridiag_eig_val`.

Default: `block(`i`)` $= 1$ for $i = 1, 2, \ldots, m$; that is, T is treated as a single block.

fail(m) — integer, intent(out), optional

Output: on successful exit, all elements of `fail` are set to 0. If error code 201 is returned, the leading elements of `fail` hold the column indices (in `z`) of those eigenvectors which failed to converge, and the remaining elements are set to 0. For example, if `fail(1)` $= 2$, the eigenvector in column 2 of `z` failed to converge.

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.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Failure to converge. Some eigenvectors have failed to converge in 5 iterations; the most recent iterate is stored in the corresponding column of `z`. If k eigenvectors failed to converge, their indices are returned in `fail(`$1:k$`)` (if present).

5 Examples of Usage

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

That example shows a call to this procedure to compute the eigenvectors corresponding to the eigenvalues stored in the array `lambda` by a preceding call to the procedure `nag_sym_tridiag_eig_val`. The relevant statements are:

```
CALL nag_sym_tridiag_eig_val(d,e,lambda,vl=vl,vu=vu,block=block)
 . . .
ALLOCATE (z(n,size(lambda)))
CALL nag_sym_trid_eig_vec(d,e,lambda,z,block=block)
```

You may wish to compute *all* the eigenvalues of T by a call to `nag_sym_tridiag_eig_all`, before selecting those eigenvalues for which you wish to compute the corresponding eigenvectors. If so, you must save a copy of the arrays `d` and `e`, because they are overwritten by `nag_sym_tridiag_eig_all`. Suppose in this case that you wish to compute the eigenvectors corresponding to the 2 smallest eigenvalues. Then arrays `dd` and `ee` should be declared of type `real(kind=`*wp*`)` and shape (n) and $(n-1)$ respectively, and the statements should be changed to:

```
dd = d
ee = e
CALL nag_sym_tridiag_eig_all(nag_key_real,dd,ee)
 . . .
ALLOCATE (z(n,2))
CALL nag_sym_tridiag_eig_vec(d,e,dd(1:2),z)
```

6 Further Comments

6.1 Algorithmic Detail

The algorithm is derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

Each computed eigenvector z_i is the exact eigenvector of a nearby matrix $T + E_i$, such that $\|E_i\|_2 = O(\epsilon)\|T\|_2$, where $\epsilon =$ `EPSILON(1.0_wp)`. Hence, the residual is small:

$$\|Tz_i - \lambda_i z_i\|_2 = O(\epsilon)\|T\|_2.$$

However, a set of eigenvectors computed by this procedure may not be orthogonal to so high a degree of accuracy as those computed by `nag_sym_tridiag_eig_all`.

6.3 Timing

The total number of floating-point operations performed is $O(mn)$, but depends on how rapidly the algorithm converges.

Example 1: All eigenvalues and eigenvectors of a real symmetric matrix

Compute all the eigenvalues and eigenvectors of a real symmetric matrix A. This example calls the single procedure `nag_sym_eig_all`, using conventional storage for A.

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_eig_ex01

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_eig, ONLY : nag_sym_eig_all
  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, n
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), lambda(:), z(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_eig_ex01'

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

  ALLOCATE (a(n,n),lambda(n),z(n,n)) ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    READ (nag_std_in,*) (a(i,:i),i=1,n)
  CASE ('U','u')
    READ (nag_std_in,*) (a(i,i:),i=1,n)
  END SELECT

  ! Compute the eigenvalues and eigenvectors of A

  CALL nag_sym_eig_all(uplo,a,lambda,z=z)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Eigenvalues'
  WRITE (nag_std_out,'(2X,6(F9.3:))') lambda
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(z,format='(F9.3)',title='Eigenvectors')

  DEALLOCATE (a,lambda,z)      ! Deallocate storage

END PROGRAM nag_sym_eig_ex01
```

2 Program Data

```
Example Program Data for nag_sym_eig_ex01
 'U'                        : Value of uplo
 4                          : Value of n
 2.07   3.87   4.20  -1.15
       -0.21   1.87   0.63
               1.15   2.06
                     -1.81 : End of Matrix A (Upper triangle)
```

3 Program Results

```
 Example Program Results for nag_sym_eig_ex01

 Eigenvalues
     -5.003   -1.999    0.201    8.001

 Eigenvectors
      0.566   -0.233   -0.397    0.684
     -0.348    0.799   -0.178    0.456
     -0.474   -0.409    0.538    0.565
      0.578    0.374    0.722    0.068
```

Example 2: Selected eigenvalues and eigenvectors of a complex Hermitian matrix

Compute selected eigenvalues and the corresponding eigenvectors of a complex Hermitian matrix A. The eigenvalues are selected by index: eigenvalues with indices from `il` to `iu` are computed, the values of `il` and `iu` being read from the data file. This example calls the single procedure `nag_sym_eig_sel`, using packed storage for A.

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_eig_ex02

  ! Example Program Text for nag_sym_eig
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_write_mat, ONLY : nag_write_gen_mat
  USE nag_sym_eig, ONLY : nag_sym_eig_sel
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, il, iu, j, n
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), POINTER :: lambda(:)
  COMPLEX (wp), ALLOCATABLE :: a(:)
  COMPLEX (wp), POINTER :: z(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_eig_ex02'

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

  ALLOCATE (a(n*(n+1)/2))      ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    DO i = 1, n
      READ (nag_std_in,*) (a(i+(2*n-j)*(j-1)/2),j=1,i)
    END DO
  CASE ('U','u')
    DO i = 1, n
      READ (nag_std_in,*) (a(i+j*(j-1)/2),j=i,n)
    END DO
  END SELECT

  ! Compute the eigenvalues and eigenvectors of A

  CALL nag_sym_eig_sel(uplo,a,lambda,il=il,iu=iu,z=z)
```

```
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Selected eigenvalues'
      WRITE (nag_std_out,'(12X,5(F6.3:,10X))') lambda
      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(z,format='(F6.3)',title='Selected eigenvectors')

      DEALLOCATE (a,lambda,z)      ! Deallocate storage

      NULLIFY (lambda,z)

    END PROGRAM nag_sym_eig_ex02
```

2 Program Data

```
Example Program Data for nag_sym_eig_ex02
 'L'                                                    : Value of uplo
 4                                                      : Value of n
 1 3                                                    : Values of il, iu
( 2.17, 0.00)
( 0.74,-1.33) (-2.28, 0.00)
(-2.06,-1.17) ( 1.78, 2.03) (-1.12, 0.00)
( 1.28,-1.64) ( 2.26,-0.10) ( 0.01,-0.43) (-0.37, 0.00) : End of Matrix A
```

3 Program Results

```
 Example Program Results for nag_sym_eig_ex02

 Selected eigenvalues
            -5.472          -2.591           1.916

 Selected eigenvectors
   (-0.221,-0.146) (-0.354,-0.274) (-0.293,-0.078)
   ( 0.739, 0.000) (-0.265,-0.142) ( 0.189,-0.482)
   (-0.373,-0.455) (-0.158,-0.348) ( 0.664, 0.000)
   (-0.186,-0.051) ( 0.751, 0.000) (-0.051,-0.443)
```

Example 3: All eigenvalues and eigenvectors of a complex Hermitian matrix, using lower-level procedures

Compute all the eigenvalues and eigenvectors of a complex Hermitian matrix A. This example calls the lower-level procedures `nag_sym_tridiag_reduc` and `nag_sym_tridiag_eig_all`, using conventional storage for A. It calls `nag_sym_tridiag_reduc` to reduce A to tridiagonal form T, and to compute the unitary transformation matrix Q, overwriting it on A. It then calls `nag_sym_tridiag_eig_all` to compute the eigenvalues of T, and to overwrite Q with the matrix of eigenvectors of A.

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_eig_ex03

  ! Example Program Text for nag_sym_eig
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_write_mat, ONLY : nag_write_gen_mat
  USE nag_sym_eig, ONLY : nag_key_cmplx, nag_sym_tridiag_reduc, &
   nag_sym_tridiag_eig_all
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, n
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: d(:), e(:)
  COMPLEX (wp), ALLOCATABLE :: a(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_eig_ex03'

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

  ALLOCATE (a(n,n),d(n),e(n-1)) ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    READ (nag_std_in,*) (a(i,:i),i=1,n)
  CASE ('U','u')
    READ (nag_std_in,*) (a(i,i:),i=1,n)
  END SELECT

  ! Reduce A to real symmetric tridiagonal form

  CALL nag_sym_tridiag_reduc(uplo,a,d,e,q_on_a=.TRUE.)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Diagonal of the tridiagonal matrix T'
  WRITE (nag_std_out,'(12X,5(F6.3:,10X))') d
  WRITE (nag_std_out,*) 'Super-diagonal of the tridiagonal matrix T'
```

```
      WRITE (nag_std_out,'(12X,5(F6.3:,10X))') e

      ! Compute all the eigenvalues and eigenvectors of A

      CALL nag_sym_tridiag_eig_all(nag_key_cmplx,d,e,z_tridiag=.FALSE.,z=a)

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Eigenvalues'
      WRITE (nag_std_out,'(12X,5(F6.3:,10X))') d
      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(a,format='(F6.3)',title='Eigenvectors')

      DEALLOCATE (a,d,e)           ! Deallocate storage

    END PROGRAM nag_sym_eig_ex03
```

2 Program Data

```
Example Program Data for nag_sym_eig_ex03
 'U'                                                    : Value of uplo
 4                                                      : Value of n
( 2.17, 0.00) ( 0.74, 1.33) (-2.06, 1.17) ( 1.28, 1.64)
              (-2.28, 0.00) ( 1.78,-2.03) ( 2.26, 0.10)
                            (-1.12, 0.00) ( 0.01, 0.43)
                                          (-0.37, 0.00) : End of Matrix A
```

3 Program Results

```
 Example Program Results for nag_sym_eig_ex03

 Diagonal of the tridiagonal matrix T
           -1.178          -1.249           1.197          -0.370
 Super-diagonal of the tridiagonal matrix T
            3.733          -2.162          -3.103

 Eigenvalues
           -5.472          -2.591           1.916           4.546

 Eigenvectors
   (-0.221,-0.146) (-0.354,-0.274) (-0.293,-0.078) ( 0.798, 0.000)
   ( 0.739, 0.000) (-0.265,-0.142) ( 0.189,-0.482) ( 0.061,-0.302)
   (-0.373,-0.455) (-0.158,-0.348) ( 0.664, 0.000) (-0.133,-0.223)
   (-0.186,-0.051) ( 0.751, 0.000) (-0.051,-0.443) ( 0.210,-0.395)
```

Example 4: Selected eigenvalues and eigenvectors of a real symmetric matrix, using lower-level procedures

Compute selected eigenvalues and the corresponding eigenvectors of a real symmetric matrix A. Eigenvalues in the interval (vl,vu] are selected, the values of vl and vu being read from the data file. This example calls the lower-level procedures nag_sym_tridiag_reduc, nag_sym_tridiag_eig_val, nag_sym_tridiag_eig_vec and nag_sym_tridiag_orth, using packed storage for A. It first calls nag_sym_tridiag_reduc to reduce A to tridiagonal form T, then nag_sym_tridiag_eig_val to compute selected eigenvalues of T, nag_sym_tridiag_eig_vec to compute the corresponding eigenvectors of T, and finally nag_sym_tridiag_orth to transform the eigenvectors of T to eigenvectors of A.

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_eig_ex04

  ! Example Program Text for nag_sym_eig
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_eig, ONLY : nag_sym_tridiag_reduc, nag_sym_tridiag_eig_val, &
   nag_sym_tridiag_eig_vec, nag_sym_tridiag_orth
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, SIZE
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, n
  REAL (wp) :: vl, vu
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  INTEGER, POINTER :: block(:)
  REAL (wp), ALLOCATABLE :: a(:), d(:), e(:), tau(:), z(:,:)
  REAL (wp), POINTER :: lambda(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_eig_ex04'

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

  ALLOCATE (a(n*(n+1)/2),d(n),e(n-1),tau(n)) ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    DO i = 1, n
      READ (nag_std_in,*) (a(i+(2*n-j)*(j-1)/2),j=1,i)
    END DO
  CASE ('U','u')
    DO i = 1, n
      READ (nag_std_in,*) (a(i+j*(j-1)/2),j=i,n)
    END DO
  END SELECT
```

```
      ! Reduce A to real symmetric tridiagonal form

      CALL nag_sym_tridiag_reduc(uplo,a,d,e,tau=tau)

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Diagonal of the tridiagonal matrix T'
      WRITE (nag_std_out,'(2X,6(F9.3:))') d
      WRITE (nag_std_out,*) 'Super-diagonal of the tridiagonal matrix T'
      WRITE (nag_std_out,'(2X,6(F9.3:))') e

      ! Compute the selected eigenvalues of T

      CALL nag_sym_tridiag_eig_val(d,e,lambda,vl=vl,vu=vu,block=block)

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Selected eigenvalues'
      WRITE (nag_std_out,'(2X,6(F9.3:))') lambda

      ALLOCATE (z(n,SIZE(lambda))) ! Allocate storage for the eigenvectors

      ! Compute the selected eigenvectors of T

      CALL nag_sym_tridiag_eig_vec(d,e,lambda,block=block,z=z)

      ! Transform the eigenvectors of T into eigenvectors of A

      CALL nag_sym_tridiag_orth(uplo,a,tau,c=z)

      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(z,format='(F9.3)',title='Selected eigenvectors')

      DEALLOCATE (a,d,e,block,lambda,tau,z) ! Deallocate storage

      NULLIFY (block,lambda)

    END PROGRAM nag_sym_eig_ex04
```

2 Program Data

```
Example Program Data for nag_sym_eig_ex04
 'L'                              : Value of uplo
 4                                : Value of n
-2.00  10.00                      : Values of vl and vu
 2.07
 3.87  -0.21
 4.20   1.87   1.15
-1.15   0.63   2.06  -1.81        : Matrix A (lower triangle)
```

3 Program Results

```
 Example Program Results for nag_sym_eig_ex04

 Diagonal of the tridiagonal matrix T
      2.070    1.474   -0.649   -1.695
 Super-diagonal of the tridiagonal matrix T
     -5.826    2.624    0.916

 Selected eigenvalues
     -1.999    0.201    8.001
```

```
 Selected eigenvectors
     -0.233    0.397   -0.684
      0.799    0.178   -0.456
     -0.409   -0.538   -0.565
      0.374   -0.722   -0.068
```

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_eig_ex05`
: All eigenvalues and eigenvectors of a complex Hermitian matrix, using `nag_sym_eig_all` with conventional storage.

`nag_sym_eig_ex06`
: Selected eigenvalues and eigenvectors of a real symmetric matrix, using `nag_sym_eig_sel` with packed storage.

`nag_sym_eig_ex07`
: All eigenvalues and eigenvectors of a real symmetric matrix, using lower-level procedures with conventional storage.

`nag_sym_eig_ex08`
: Selected eigenvalues and eigenvectors of a complex Hermitian matrix, using lower-level procedures with packed storage.

`nag_sym_eig_ex09`
: All eigenvalues and eigenvectors of a real symmetric matrix, using `nag_sym_eig_all` with packed storage.

`nag_sym_eig_ex10`
: All eigenvalues and eigenvectors of a real symmetric matrix, using lower-level procedures with packed storage.

`nag_sym_eig_ex11`
: Selected eigenvalues and eigenvectors of a real symmetric matrix, using `nag_sym_eig_sel` with conventional storage.

`nag_sym_eig_ex12`
: Selected eigenvalues and eigenvectors of a real symmetric matrix, using lower-level procedures with conventional storage.

`nag_sym_eig_ex13`
: Selected eigenvalues and eigenvectors of a complex Hermitian matrix, using `nag_sym_eig_sel` with conventional storage.

`nag_sym_eig_ex14`
: Selected eigenvalues and eigenvectors of a complex Hermitian matrix, using lower-level procedures with conventional storage.

`nag_sym_eig_ex15`
: All eigenvalues and eigenvectors of a complex Hermitian matrix, using `nag_sym_eig_all` with packed storage.

`nag_sym_eig_ex16`
: All eigenvalues and eigenvectors of a complex Hermitian matrix, using lower-level procedures with packed storage.

References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

[3] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition)

[4] Parlett B N (1980) *The Symmetric Eigenvalue Problem* Prentice-Hall