Module 6.2: nag_nsym_eig
Standard Nonsymmetric Eigenvalue Problems

`nag_nsym_eig` provides procedures for solving standard nonsymmetric eigenvalue problems

$$Ax = \lambda x$$

where A is a general real or complex square matrix, and for computing the related Schur factorization of A.

Contents

Introduction

1 Notation and Background

The *standard eigenvalue problem* is to find the *eigenvalues*, λ_i, and the corresponding eigenvectors, x_i, of a general real or complex matrix A such that

$$Ax_i = \lambda_i x_i \text{ for } i = 1, \ldots, n, \tag{1}$$

where n is the order of A.

If A is known to be real symmetric or complex Hermitian, you should turn to the module `nag_sym_eig` (6.1) because in these cases the problem has special properties and it is desirable to take advantage of them (in particular, the eigenvalues are real and the eigenvectors are mutually orthogonal).

This module is intended for standard *nonsymmetric eigenvalue problems*, when A is not known to be symmetric or Hermitian. (Strictly speaking, when A is complex, we should talk of non-Hermitian problems.)

The eigenvalues and eigenvectors may be complex, even when A is real. If A is real, complex eigenvalues and eigenvectors always occur in complex conjugate pairs; if x is an eigenvector corresponding to the complex eigenvalue λ, then the complex conjugate vector $\bar{x}$ is the eigenvector corresponding to $\bar{\lambda}$:

$$Ax = \lambda x \text{ and } A\bar{x} = \bar{\lambda}\bar{x}.$$

The eigenvectors x_i defined in (1) are sometimes referred to as *right eigenvectors*. The *left eigenvectors* y_i are defined by:

$$y_i^H A = \lambda_i y_i^H \text{ or equivalently } A^H y_i = \bar{\lambda}_i y_i.$$

Thus a left eigenvector of A is a right eigenvector of A^H ($= A^T$ if A is real).

2 Schur Factorization

An important tool for solving nonsymmetric eigenvalue problems is the *Schur factorization* of A (also referred to as the Schur decomposition). This is defined as:

$A = ZTZ^T$, with T real upper *quasi-triangular* and Z orthogonal, if A is real;

$A = ZTZ^H$ with T complex upper triangular and Z unitary, if A is complex.

The matrix T is called the *Schur form* of A. If A is complex, or if A is real *and* all its eigenvalues are real, then T is upper triangular, and the diagonal elements of T are the eigenvalues of A. If A is real and has complex conjugate pairs of eigenvalues, then, corresponding to each such pair, T has a 2×2 block on the diagonal; the eigenvalues of the diagonal blocks are the complex eigenvalues of A.

The columns of Z are the Schur vectors of A. They are mutually orthogonal, unlike the eigenvectors, so they are often more satisfactory to work with in numerical computation.

The eigenvalues of T are the same as the eigenvalues of A, and if x is an eigenvector of T, then Zx is an eigenvector of A.

3 Choice of Procedures

Two procedures are provided in this module. They can be used for the following computations:

All eigenvalues of A (`nag_nsym_eig_all`)

All eigenvalues and eigenvectors (right, left or both) of A (`nag_nsym_eig_all` with optional arguments)

Schur form of A (`nag_schur_fac`)

Schur form and Schur vectors of A (`nag_schur_fac` with optional argument)

Procedure: nag_nsym_eig_all

1 Description

nag_nsym_eig_all is a generic procedure which computes all the eigenvalues, and optionally all the left or right eigenvectors, of a real or complex general matrix A of order n.

We write:

$Ax_i = \lambda_i x_i$ for $i = 1, \ldots, n$, for the right eigenvectors x_i;
$y_i^H A = \lambda_i y_i^H$ for $i = 1, \ldots, n$, for the left eigenvectors y_i.

The eigenvalues and eigenvectors may be complex, even when A is real. They are always returned in complex arrays.

By default, only the eigenvalues are computed. Optionally, either the right or left eigenvectors, or both, may be computed.

Each (left or right) eigenvector x is normalized so that $\|x\|_2 = 1$ and the element of largest absolute value is real and positive.

2 Usage

```
USE nag_nsym_eig
CALL nag_nsym_eig_all(a, 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 A

3.1 Mandatory Arguments

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

Input: the general matrix A.

Output: overwritten by intermediate results.

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

Output: the eigenvalues of A. They may occur in any 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.

vr(n, n) — complex(kind=*wp*), intent(out), optional

Output: the right eigenvectors of A. The ith column vr(:, i) holds the right eigenvector corresponding to the eigenvalue lambda(i).

vl(n, n) — complex(kind=*wp*), intent(out), optional

Output: the left eigenvectors of A. The ith column `vl(:,i)` holds the left eigenvector corresponding to the eigenvalue `lambda(i)`.

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

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

4 Error Codes

Fatal errors (error%level = 3):

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

Failures (error%level = 2):

error%code	Description
201	Failure to converge. The QR algorithm failed to compute all the eigenvalues in the permitted number of iterations.

5 Examples of Usage

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

In that example, the following call statement is used to compute eigenvalues and right eigenvectors:

```
CALL nag_nsym_eig_all( a, lambda, vr=vr )
```

To compute eigenvalues only, the call statement should be changed to:

```
CALL nag_nsym_eig_all( a, lambda )
```

To compute left eigenvectors as well as right eigenvectors, a suitable array `vl` must be declared, and the call statement changed to:

```
CALL nag_nsym_eig_all( a, lambda, vr=vr, vl=vl )
```

6 Further Comments

6.1 Algorithmic Detail

The procedure performs the following steps (see Chapter 7 of Golub and Van Loan [2] for more details).

1. It balances the matrix, using a diagonal similarity transformation to reduce its norm.
2. It reduces the balanced matrix $\tilde{A}$ to upper Hessenberg form H, using an orthogonal or unitary similarity transformation: $\tilde{A} = QHQ^H$. If eigenvectors are required, it forms the matrix Q.
3. If only eigenvalues are required, it applies the QR algorithm to compute the eigenvalues of H, which are the same as the eigenvalues of A.

4. If eigenvectors are required, it applies the QR algorithm to compute the Schur factorization of H: $H = STS^H$, forming the matrix $Z = QS$ which is the matrix of Schur vectors of A.
5. It then computes the eigenvectors of the Schur form T by back-substitution, pre-multiplies them by Z to form the eigenvectors of the balanced matrix $\tilde{A}$, and finally transforms them to those of the original matrix A.

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 computed eigenvalue, then

$$|\lambda_i - \tilde{\lambda}_i| < \frac{c(n)\epsilon\|A\|_2}{s_i}, \tag{2}$$

where $c(n)$ is a modestly increasing function of n, and $\epsilon =$ `EPSILON(1.0_wp)`; s_i is the reciprocal condition number of λ_i, which is defined by

$$s_i = y_i^H x_i,$$

where y_i and x_i are the left and right eigenvectors corresponding to λ_i. The bound (2) may be an overestimate if A is badly scaled.

6.3 Timing

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

Procedure: nag_schur_fac

1 Description

nag_schur_fac computes part or all of the Schur factorization of a real or complex general matrix A of order n.

We write the Schur factorization as follows:

$A = ZTZ^T$, with T real upper *quasi-triangular* and Z orthogonal, if A is real;

$A = ZTZ^H$ with T complex upper triangular and Z unitary, if A is complex.

See the Module Introduction for more details.

By default, only the Schur form T is computed. Optionally, the matrix Z of Schur vectors may be computed, and the eigenvalues may be extracted from T and stored in a separate complex array. If *only* the eigenvalues are required, it is more efficient to call nag_nsym_eig_all.

Each Schur vector z_i is normalized so that $\|z_i\|_2 = 1$ and the element of largest absolute value is (real and) positive.

2 Usage

```
USE nag_nsym_eig
CALL nag_schur_fac(a  [, 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 A

3.1 Mandatory Argument

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

Input: the general matrix A.

Output: the Schur form 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.

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

Output: the eigenvalues of A.

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

Output: the matrix Z.

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
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. The QR algorithm failed to compute the Schur form in the permitted number of iterations.

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 performs the following steps (see Chapter 7 of Golub and Van Loan [2] for more details).

1. It reduces A to upper Hessenberg form H, using an orthogonal or unitary similarity transformation: $A = QHQ^H$. If the Schur vectors are required, it forms the matrix Q.
2. If only the Schur form is required, it applies the QR algorithm to compute the Schur form of H, which is the same as the Schur form of A.
3. If the Schur vectors are required, it applies the QR algorithm to compute the whole of the Schur factorization of H: $H = STS^H$, forming the matrix $Z = QS$ which is the matrix of Schur vectors of A.

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

6.2 Accuracy

The computed Schur factorization is the exact factorization of a nearby matrix $A + E$, where

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

and ϵ = EPSILON(1.0_*wp*).

6.3 Timing

The time taken by the procedure is approximately proportional to n^3. Computing both the Schur form and the Schur vectors is likely to take about twice as long as computing the Schur form alone.

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

Compute all eigenvalues and the right eigenvectors of a real matrix 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_nsym_eig_ex01

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_nsym_eig, ONLY : nag_nsym_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
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:)
  COMPLEX (wp), ALLOCATABLE :: lambda(:), vr(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_nsym_eig_ex01'

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

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

  READ (nag_std_in,*) (a(i,:),i=1,n)

  ! Compute eigenvalues and eigenvectors

  CALL nag_nsym_eig_all(a,lambda=lambda,vr=vr)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Eigenvalues'
  WRITE (nag_std_out,'(3X,8("(",F7.4,",",F7.4,")":1X))') lambda
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(vr,int_col_labels=.TRUE.,format='(F7.4)', &
   title='Matrix of right eigenvectors (one vector per column)')

  DEALLOCATE (a,lambda,vr)     ! Deallocate storage

END PROGRAM nag_nsym_eig_ex01
```

2 Program Data

```
Example Program Data for nag_nsym_eig_ex01
  4                           :Value of n
  0.35   0.45  -0.14  -0.17
  0.09   0.07  -0.54   0.35
 -0.44  -0.33  -0.03   0.17
  0.25  -0.32  -0.13   0.11   :End of matrix A
```

3 Program Results

```
 Example Program Results for nag_nsym_eig_ex01

 Eigenvalues
   ( 0.7995, 0.0000) (-0.0994, 0.4008) (-0.0994,-0.4008) (-0.1007, 0.0000)

 Matrix of right eigenvectors (one vector per column)
                   1                 2                 3                 4
   ( 0.6551, 0.0000) (-0.1933, 0.2546) (-0.1933,-0.2546) ( 0.1253, 0.0000)
   ( 0.5236, 0.0000) ( 0.2519,-0.5224) ( 0.2519, 0.5224) ( 0.3320, 0.0000)
   (-0.5362, 0.0000) ( 0.0972,-0.3084) ( 0.0972, 0.3084) ( 0.5938, 0.0000)
   ( 0.0956, 0.0000) ( 0.6760, 0.0000) ( 0.6760, 0.0000) ( 0.7221, 0.0000)
```

Example 2: Schur factorization of a real matrix

Compute the Schur factorization of a real matrix 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_nsym_eig_ex02

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_nsym_eig, ONLY : nag_schur_fac
  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
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), z(:,:)
  COMPLEX (wp), ALLOCATABLE :: lambda(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_nsym_eig_ex02'

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

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

  READ (nag_std_in,*) (a(i,:),i=1,n)

  ! Compute the Schur factorization

  CALL nag_schur_fac(a,lambda=lambda,z=z)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Eigenvalues'
  WRITE (nag_std_out,'(1X,"(",F8.4,",",F8.4,")")') lambda
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(a,int_col_labels=.TRUE.,title='Schur matrix')

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(z,int_col_labels=.TRUE., &
   title='Matrix of Schur vectors (one vector per column)')

  DEALLOCATE (a,lambda,z)      ! Deallocate storage

END PROGRAM nag_nsym_eig_ex02
```

2 Program Data

```
Example Program Data for nag_nsym_eig_ex02
  4                          : Value of n
  0.35   0.45  -0.14  -0.17
  0.09   0.07  -0.54   0.35
 -0.44  -0.33  -0.03   0.17
  0.25  -0.32  -0.13   0.11 : End of Matrix A
```

3 Program Results

```
 Example Program Results for nag_nsym_eig_ex02

 Eigenvalues
 (  0.7995,  0.0000)
 ( -0.0994,  0.4008)
 ( -0.0994, -0.4008)
 ( -0.1007,  0.0000)

 Schur matrix
         1       2       3       4
    0.7995 -0.1144 -0.0060  0.0336
    0.0000 -0.0994 -0.2478  0.3474
    0.0000  0.6483 -0.0994 -0.2026
    0.0000  0.0000  0.0000 -0.1007

 Matrix of Schur vectors (one vector per column)
         1       2       3       4
    0.6551  0.1037 -0.3450  0.6641
    0.5236 -0.5807  0.6141 -0.1068
   -0.5362 -0.3073  0.2935  0.7293
    0.0956  0.7467  0.6463  0.1249
```

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_nsym_eig_ex03`

Schur factorization of a complex matrix.

`nag_nsym_eig_ex04`

All eigenvalues and right eigenvectors of a complex matrix.

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