Module 6.6: nag_nsym_gen_eig
Nonsymmetric Generalized Eigenvalue Problems

`nag_nsym_gen_eig` provides procedures for solving nonsymmetric generalized eigenvalue problems

$$Ax = \lambda Bx,$$

where A and B are general real or complex square matrices, and for computing the related generalized Schur factorization of the pencil $A - \lambda B$.

Contents

Introduction 6.6.3

Procedures

`nag_nsym_gen_eig_all` 6.6.5
All eigenvalues, and optionally eigenvectors, of a real or complex nonsymmetric generalized eigenvalue problem

`nag_gen_schur_fac` 6.6.9
Generalized Schur factorization of a real or complex matrix pencil

Examples

Example 1: Eigenvalues and eigenvectors of a real nonsymmetric generalized eigenvalue problem 6.6.13

Example 2: Generalized Schur factorization of a real matrix pencil 6.6.15

Additional Examples 6.6.19

References 6.6.20

Introduction

1 Notation and Background

The *generalized eigenvalue problem* is to find the *eigenvalues* λ, and the corresponding eigenvectors x, satisfying

$$Ax = \lambda Bx, \quad x \neq 0, \tag{1}$$

where A and B are real or complex square matrices. We also refer to the *pencil* $A - \lambda B$ (that is the set of all matrices $A - \lambda B$ where λ is a complex scalar), and to the eigenvalues of the pencil which are those values λ such that $\det(A - \lambda B) = 0$.

If A and B are known to be real symmetric or complex Hermitian, and B positive definite, you should turn to the module `nag_sym_gen_eig` (6.5) because in these cases the problem has special properties and it is desirable to take advantage of them (in particular, the eigenvalues are real).

This module is intended for *nonsymmetric generalized eigenvalue problems*, when A and B are not known to satisfy the conditions for using `nag_sym_gen_eig`. (Strictly speaking, when A and B are complex, we should talk of non-Hermitian problems.)

The eigenvalues and eigenvectors may be complex, even when A and B are real. If A and B are 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 $\bar{\lambda}$:

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

An eigenvector x defined in (1) is sometimes referred to as a *right eigenvector*. A *left eigenvector* y is defined by:

$$y^H A = \lambda y^H B \text{ or equivalently } A^H y = \bar{\lambda} B^H y.$$

Thus a left eigenvector of $A - \lambda B$ is a right eigenvector of $A^H - \lambda B^H$ ($= A^T - \lambda B^T$ if A and B are real).

Generalized eigenvalue problems exhibit a wider range of behaviour than standard eigenvalue problems (which result when $B = I$).

1. *Infinite eigenvalues:* the equation (1) may be re-written

   $$\mu Ax = Bx, \text{ where } \mu = 1/\lambda.$$

   We say that λ is an infinite eigenvalue of the pencil $A - \lambda B$ if and only if μ is a zero eigenvalue of the pencil $B - \mu A$. In this case, if x is the corresponding right eigenvector, then $Bx = 0$, so B must be singular and x is a null vector of B.

2. *Singular pencils:* if A and B are both singular and have a common null space, then for any vector x in this null space, $Ax = \lambda Bx = 0$ for *all* values of λ. The pencil $A - \lambda B$ is called a singular pencil. See Wilkinson [4] for a discussion of the difficulties that can arise with pencils that are singular or almost singular.

To cope with infinite eigenvalues and singular pencils, the procedures in this module do not compute the eigenvalues λ_i directly, but return scalars α_i and β_i such that

$$\beta_i Ax_i = \alpha_i Bx_i, \text{ for } i = 1, \ldots, n.$$

For finite eigenvalues, $\lambda_i = \alpha_i/\beta_i$.

Infinite eigenvalues are indicated by a zero value (or in practice a very small value) of β_i.

Singular pencils are generally indicated by zero values (or in practice very small values) of both α_i and β_i.

2 Generalized Schur Factorization

An important tool for solving nonsymmetric generalized eigenvalue problems is the *generalized Schur factorization* of the pencil $A - \lambda B$ (also referred to as the generalized Schur decomposition). This is defined as follows:

if A and B are real,

$$A = QTZ^T \text{ and } B = QSZ^T,$$

with T real upper *quasi-triangular*, S real upper triangular, and Q and Z orthogonal;

if A and B are complex,

$$A = QTZ^H \text{ and } B = QSZ^H,$$

with T and S complex upper triangular, and Q and Z unitary.

The pencil $T - \lambda S$ is called the *generalized Schur form* of $A - \lambda B$. The diagonal elements of S are the scalars β_i referred to above. If $A - \lambda B$ is complex, or if $A - \lambda B$ is real *and* all its eigenvalues are real, then T is upper triangular, and the diagonal elements of T are the scalars α_i. If $A - \lambda B$ 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 of $T - \lambda S$ are the complex eigenvalues of $A - \lambda B$.

The columns of Q are the *left Schur vectors*, and the columns of Z are the *right Schur vectors*. The Schur vectors are mutually orthogonal, unlike the eigenvectors, so they are often more satisfactory to work with in numerical computation.

The eigenvalues of $T - \lambda S$ are the same as the eigenvalues of $A - \lambda B$; if x is a right eigenvector of $T - \lambda S$, then Zx is a right eigenvector of $A - \lambda B$; if y is a left eigenvector of $T - \lambda S$, then Qy is a left eigenvector of $A - \lambda B$.

3 Choice of Procedures

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

All eigenvalues of $A - \lambda B$ (`nag_nsym_gen_eig_all`)

All eigenvalues and eigenvectors of $A - \lambda B$ (right, left or both) (`nag_nsym_gen_eig_all` with optional arguments)

Generalized Schur form of $A - \lambda B$ (`nag_gen_schur_fac`)

Generalized Schur form and vectors (right, left or both) of $A - \lambda B$ (`nag_gen_schur_fac` with optional arguments)

Procedure: nag_nsym_gen_eig_all

1 Description

nag_nsym_gen_eig_all is a generic procedure which computes all the eigenvalues, and optionally all the left or right eigenvectors, of a real or complex generalized eigenvalue problem.

Let A and B be real or complex square matrices of order n. We write:

$\beta_i A x_i = \alpha_i B x_i$, for $i = 1, \ldots, n$, for the right eigenvectors x_i;

$\beta_i y_i^H A = \alpha_i y_i^H B$, for $i = 1, \ldots, n$, for the left eigenvectors y_i.

In exact arithmetic:

- $\beta_i \neq 0$ indicates a finite eigenvalue $\lambda_i = \alpha_i / \beta_i$;
- $\beta_i = 0$ and $\alpha_i \neq 0$ indicates an infinite eigenvalue;
- $\alpha_i = \beta_i = 0$ indicates that the pencil is singular.

In finite-precision arithmetic, particular care must be taken if α_i and β_i are both *small* (compared with $\|A\|$ and $\|B\|$): this indicates that the pencil is almost singular, and it may be that no reliance can be placed on the values of any of the other eigenvalues $\lambda_j = \alpha_j / \beta_j$ (see Wilkinson [4]).

The values α_i and the eigenvectors x_i or y_i may be complex, even when A and B are real. They are always returned in complex arrays.

By default, only values α_i and β_i 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_gen_eig
CALL nag_nsym_gen_eig_all(a, b, alpha, beta  [, optional arguments])
```

3 Arguments

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

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

n — the order of the matrices A and B

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.

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

Input: the general matrix B.

Output: overwritten by intermediate results.

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

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

Output: the values α_i.

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

Output: the values β_i.

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. The ith column `vr(:,i)` holds the right eigenvector corresponding to the eigenvalue λ_i $(= \alpha_i/\beta_i)$.

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

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

Output: the left eigenvectors. The ith column `vl(:,i)` holds the left eigenvector corresponding to the eigenvalue λ_i $(= \alpha_i/\beta_i)$.

Constraints: `vl` 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 QZ 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_gen_nsym_eig_all( a, b, alpha, beta, vr=vr )
```

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

```
CALL nag_gen_nsym_eig_all( a, b, alpha, beta )
```

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, b, alpha, beta, 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 pencil, using diagonal transformations to try to make the norms of the rows and columns of A and B as close to 1 as possible.
2. It reduces the balanced pencil $\tilde{A} - \lambda\tilde{B}$ by an orthogonal or unitary transformation to a pencil $H - \lambda K$, where H is upper Hessenberg and K is upper triangular: $\tilde{A} = Q_1 H Z_1^H$ and $\tilde{B} = Q_1 K Z_1^H$. $H - \lambda K$ has the same eigenvalues as $A - \lambda B$. If left eigenvectors are required, it forms the matrix Q_1; if right eigenvectors are required, it forms the matrix Z_1.
3. If only eigenvalues are required, it applies the QZ algorithm to compute the values α_i and β_i.
4. If eigenvectors are required, it applies the QZ algorithm to reduce the upper Hessenberg matrix H to quasi-triangular form while maintaining K in upper triangular form. This gives the generalized Schur factorization of $H - \lambda K$: $H = Q_2 T Z_2^H$ and $K = Q_2 S Z_2^H$. If left eigenvectors are required, the matrix $Q = Q_1 Q_2$ is formed; if right eigenvectors are required, the matrix $Z = Z_1 Z_2$ is formed.
5. It then computes the required eigenvectors of the pencil $T - \lambda S$ by backward substitution; it pre-multiplies left eigenvectors by Q and right eigenvectors by Z to give eigenvectors of the balanced pencil $\tilde{A} - \lambda\tilde{B}$. Finally it transforms them to eigenvectors of the original pencil $A - \lambda B$.

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

6.2 Accuracy

The computed eigenvalues are always exact for a perturbed problem $(A + \delta A)x = \lambda(B + \delta B)x$, where $\|\delta A\|/\|A\|$ and $\|\delta B\|/\|B\|$ are of the order of the machine precision, `EPSILON(1.0_wp)`. For a discussion of the sensitivity of the problem to perturbations in the data, see Stewart and Sun [3].

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 twice as long as computing eigenvalues alone.

Procedure: nag_gen_schur_fac

1 Description

nag_gen_schur_fac is a generic procedure which computes part or all of the generalized Schur factorization of a matrix pencil $A - \lambda B$, where A and B are real or complex square matrices of order n.

We write the generalized Schur factorization as follows:

if A and B are real,

$$A = QTZ^T \text{ and } B = QSZ^T,$$

with T real upper *quasi-triangular*, S real upper triangular, and Q and Z orthogonal;

if A and B are complex,

$$A = QTZ^H \text{ and } B = QSZ^H,$$

with T and S complex upper triangular, and Q and Z unitary.

See the Module Introduction for more details.

By default, only the matrices T and S of the generalized Schur form are computed. Optionally, the matrix Q of left Schur vectors, or the matrix Z of right Schur vectors, or both, may be computed; and the values α_i and β_i may be extracted from T and S and stored in separate arrays. If *only* the eigenvalues are required, it is more efficient to call nag_nsym_gen_eig_all.

2 Usage

```
USE nag_nsym_gen_eig
CALL nag_gen_schur_fac(a, b  [, optional arguments])
```

3 Arguments

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

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

n — the order of the matrices A and B

3.1 Mandatory Arguments

a(n, n) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)
Input: the general matrix A.
Output: the matrix T of the generalized Schur form.

b(n, n) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)
Input: the general matrix B.
Output: the matrix S of the generalized Schur form.
Constraints: b 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.

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

Output: the values α_i.

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

Output: the values β_i.

Constraints: `alpha` and `beta` must both be present or both absent.

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

Output: the matrix Q of left Schur vectors.

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

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

Output: the matrix Z of right Schur vectors.

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.
305	Invalid absence 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 QZ 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 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 the pencil $A - \lambda B$ by an orthogonal or unitary transformation to a pencil $H - \lambda K$, where H is upper Hessenberg and K is upper triangular: $A = Q_1 H Z_1^H$ and $B = Q_1 K Z_1^H$. If left Schur vectors are required, it forms the matrix Q_1; if right Schur vectors are required, it forms the matrix Z_1.

2. It applies the QZ algorithm to reduce the upper Hessenberg matrix H to quasi-triangular form while maintaining K in upper triangular form. This gives the generalized Schur factorization of $H - \lambda K$: $H = Q_2 T Z_2^H$ and $K = Q_2 S Z_2^H$. If left Schur vectors are required, the matrix $Q = Q_1 Q_2$ is formed; if right Schur vectors are required, the matrix $Z = Z_1 Z_2$ is formed.

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

6.2 Accuracy

The computed decomposition is the exact decomposition of a perturbed pencil $(A + \delta A) - \lambda(B + \delta B)$, where $\|\delta A\|/\|A\|$ and $\|\delta B\|/\|B\|$ are of the order of the machine precision, `EPSILON(1.0_wp)`. For a discussion of the sensitivity of the problem to perturbations in the data, see Stewart and Sun [3].

6.3 Timing

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

Example 1: Eigenvalues and eigenvectors of a real nonsymmetric generalized eigenvalue problem

Compute all the eigenvalues and the right eigenvectors of a real nonsymmetric generalized eigenvalue problem $Az = \lambda Bz$.

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_gen_eig_ex01

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_nsym_gen_eig, ONLY : nag_nsym_gen_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(:,:), b(:,:), beta(:)
  COMPLEX (wp), ALLOCATABLE :: alpha(:), vr(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) &
   'Example Program Results for nag_nsym_gen_eig_ex01'

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

  ALLOCATE (a(n,n),b(n,n),alpha(n),beta(n),vr(n,n)) ! Allocate storage

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

  ! Compute the right eigenvectors

  CALL nag_nsym_gen_eig_all(a,b,alpha,beta,vr=vr)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Eigenvalues'
  WRITE (nag_std_out,'(1X,"(",F7.4,",",F7.4,")")') alpha/beta
  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,b,alpha,beta,vr) ! Deallocate storage

END PROGRAM nag_nsym_gen_eig_ex01
```

2 Program Data

```
Example Program Data for nag_nsym_gen_eig_ex01
  4                          :Value of n
   3.9  12.5 -34.5  -0.5
   4.3  21.5 -47.5   7.5
   4.3  21.5 -43.5   3.5
   4.4  26.0 -46.0   6.0     :End of matrix A
   1.0   2.0  -3.0   1.0
   1.0   3.0  -5.0   4.0
   1.0   3.0  -4.0   3.0
   1.0   3.0  -4.0   4.0     :End of matrix B
```

3 Program Results

```
 Example Program Results for nag_nsym_gen_eig_ex01

 Eigenvalues
 ( 2.0000, 0.0000)
 ( 3.0000, 4.0000)
 ( 3.0000,-4.0000)
 ( 4.0000, 0.0000)

 Matrix of right eigenvectors (one vector per column)
                  1                 2                 3                 4
   ( 0.9961, 0.0000) ( 0.9449, 0.0000) ( 0.9449, 0.0000) ( 0.9875, 0.0000)
   ( 0.0057, 0.0000) ( 0.1890, 0.0000) ( 0.1890, 0.0000) ( 0.0110, 0.0000)
   ( 0.0626, 0.0000) ( 0.1134,-0.1512) ( 0.1134, 0.1512) (-0.0329, 0.0000)
   ( 0.0626, 0.0000) ( 0.1134,-0.1512) ( 0.1134, 0.1512) ( 0.1536, 0.0000)
```

Example 2: Generalized Schur factorization of a real matrix pencil

Compute the generalized Schur factorization of a real matrix pencil $A - \lambda B$.

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_gen_eig_ex02

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

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_nsym_gen_eig, ONLY : nag_gen_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(:,:), b(:,:), beta(:), q(:,:), z(:,:)
  COMPLEX (wp), ALLOCATABLE :: alpha(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) &
   'Example Program Results for nag_nsym_gen_eig_ex02'

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

  ALLOCATE (a(n,n),b(n,n),alpha(n),beta(n),q(n,n), &
   z(n,n))                     ! Allocate storage

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

  ! Compute the generalized Schur factorization

  CALL nag_gen_schur_fac(a,b,alpha=alpha,beta=beta,q=q,z=z)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Eigenvalues'
  WRITE (nag_std_out,'(1X,"(",F7.4,",",F7.4,")")') alpha/beta
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(a,int_col_labels=.TRUE.,format='(F11.4)', &
   title='generalized Schur form of A')

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(b,int_col_labels=.TRUE.,format='(F11.4)', &
   title='generalized Schur form of B')

  WRITE (nag_std_out,*)
```

```
      CALL nag_write_gen_mat(q,int_col_labels=.TRUE.,format='(F11.4)', &
       title='Matrix of left Schur vectors (one vector per column)')

      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(z,int_col_labels=.TRUE.,format='(F11.4)', &
       title='Matrix of right Schur vectors (one vector per column)')

      DEALLOCATE (a,b,alpha,beta,q,z) ! Deallocate storage

    END PROGRAM nag_nsym_gen_eig_ex02
```

2 Program Data

```
Example Program Data for nag_nsym_gen_eig_ex02
  4                          :Value of n
   3.9  12.5 -34.5  -0.5
   4.3  21.5 -47.5   7.5
   4.3  21.5 -43.5   3.5
   4.4  26.0 -46.0   6.0     :End of matrix A
   1.0   2.0  -3.0   1.0
   1.0   3.0  -5.0   4.0
   1.0   3.0  -4.0   3.0
   1.0   3.0  -4.0   4.0     :End of matrix B
```

3 Program Results

```
 Example Program Results for nag_nsym_gen_eig_ex02

 Eigenvalues
 ( 2.0000, 0.0000)
 ( 3.0000, 4.0000)
 ( 3.0000,-4.0000)
 ( 4.0000, 0.0000)

 generalized Schur form of A
           1          2          3          4
      3.8009   -31.3260    61.4846   -66.8359
      0.0000     3.3505     7.0744    -6.6922
      0.0000    -1.1918     1.4098    -4.3790
      0.0000     0.0000     0.0000     4.0000

 generalized Schur form of B
           1          2          3          4
      1.9005     1.0777     5.6252    -9.9873
      0.0000     1.1761     0.0000    -1.7511
      0.0000     0.0000     0.4474    -1.0901
      0.0000     0.0000     0.0000     1.0000

 Matrix of left Schur vectors (one vector per column)
           1          2          3          4
      0.4642    -0.8116    -0.3547     0.0000
      0.5002     0.0697     0.4950    -0.7071
      0.5002     0.0697     0.4950     0.7071
      0.5331     0.5758    -0.6198     0.0000

 Matrix of right Schur vectors (one vector per column)
           1          2          3          4
      0.9961    -0.0818     0.0343     0.0000
      0.0057     0.4445     0.8957     0.0000
```

```
    0.0626     0.6307    -0.3134     0.7071
    0.0626     0.6307    -0.3134    -0.7071
```

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_gen_eig_ex03`

Computes the generalized Schur factorization of a complex matrix pencil.

`nag_nsym_gen_eig_ex04`

Computes all the eigenvalues and both the right and the left eigenvectors of a complex nonsymmetric generalized eigenvalue problem $Az = \lambda Bz$.

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)

[3] Stewart G W and Sun J-G (1990) *Matrix Perturbation Theory* Academic Press

[4] Wilkinson J H (1979) Kronecker's canonical form and the QZ algorithm *Linear Algebra Appl.* **28** 285–303