

# NAG Library Function Document

## nag\_eigen\_real\_gen\_quad (f02jcc)

### 1 Purpose

nag\_eigen\_real\_gen\_quad (f02jcc) solves the quadratic eigenvalue problem

$$(\lambda^2 A + \lambda B + C)x = 0,$$

where  $A$ ,  $B$  and  $C$  are real  $n$  by  $n$  matrices.

The function returns the  $2n$  eigenvalues,  $\lambda_j$ , for  $j = 1, 2, \dots, 2n$ , and can optionally return the corresponding right eigenvectors,  $x_j$  and/or left eigenvectors,  $y_j$  as well as estimates of the condition numbers of the computed eigenvalues and backward errors of the computed right and left eigenvectors. A left eigenvector satisfies the equation

$$y^H(\lambda^2 A + \lambda B + C) = 0,$$

where  $y^H$  is the complex conjugate transpose of  $y$ .

$\lambda$  is represented as the pair  $(\alpha, \beta)$ , such that  $\lambda = \alpha/\beta$ . Note that the computation of  $\alpha/\beta$  may overflow and indeed  $\beta$  may be zero.

### 2 Specification

```
#include <nag.h>
#include <nagf02.h>

void nag_eigen_real_gen_quad (Nag_ScaleType scal, Nag_LeftVecsType jobvl,
    Nag_RightVecsType jobvr, Nag_CondErrType sense, double tol, Integer n,
    double a[], Integer pda, double b[], Integer pdb, double c[],
    Integer pdc, double alphas[], double alphas_i[], double beta[],
    double vl[], Integer pdvl, double vr[], Integer pdvr, double s[],
    double bevl[], double bevr[], Integer *iwarn, NagError *fail)
```

### 3 Description

The quadratic eigenvalue problem is solved by linearizing the problem and solving the resulting  $2n$  by  $2n$  generalized eigenvalue problem. The linearization is chosen to have favourable conditioning and backward stability properties. An initial preprocessing step is performed that reveals and deflates the zero and infinite eigenvalues contributed by singular leading and trailing matrices.

The algorithm is backward stable for problems that are not too heavily damped, that is  $\|B\| \leq \sqrt{\|A\| \cdot \|C\|}$ .

Further details on the algorithm are given in Hammarling *et al.* (2013).

### 4 References

Fan H -Y, Lin W.-W and Van Dooren P. (2004) Normwise scaling of second order polynomial matrices. *SIAM J. Matrix Anal. Appl.* **26**, **1** 252–256

Gaubert S and Sharify M (2009) Tropical scaling of polynomial matrices *Lecture Notes in Control and Information Sciences Series* **389** 291–303 Springer–Verlag

Hammarling S, Munro C J and Tisseur F (2013) An algorithm for the complete solution of quadratic eigenvalue problems. *ACM Trans. Math. Software.* **39**(3):18:1–18:119 (<http://eprints.ma.man.ac.uk/1815/>)

## 5 Arguments

- 1: **scal** – Nag\_ScaleType *Input*
- On entry:* determines the form of scaling to be performed on  $A$ ,  $B$  and  $C$ .
- scal** = Nag\_NoScale  
No scaling.
- scal** = Nag\_CondFanLinVanDooren (the recommended value)  
Fan, Lin and Van Dooren scaling if  $\frac{\|B\|}{\sqrt{\|A\| \times \|C\|}} < 10$  and no scaling otherwise where  $\|Z\|$  is the Frobenius norm of  $Z$ .
- scal** = Nag\_FanLinVanDooren  
Fan, Lin and Van Dooren scaling.
- scal** = Nag\_TropicalLargest  
Tropical scaling with largest root.
- scal** = Nag\_TropicalSmallest  
Tropical scaling with smallest root.
- Constraint:* **scal** = Nag\_NoScale, Nag\_CondFanLinVanDooren, Nag\_FanLinVanDooren, Nag\_TropicalLargest or Nag\_TropicalSmallest.
- 2: **jobvl** – Nag\_LeftVecsType *Input*
- On entry:* if **jobvl** = Nag\_NotLeftVecs, do not compute left eigenvectors.
- If **jobvl** = Nag\_LeftVecs, compute the left eigenvectors.
- If **sense** = Nag\_CondOnly, Nag\_BackErrLeft, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth, **jobvl** must be set to Nag\_LeftVecs.
- Constraint:* **jobvl** = Nag\_NotLeftVecs or Nag\_LeftVecs.
- 3: **jobvr** – Nag\_RightVecsType *Input*
- On entry:* if **jobvr** = Nag\_NotRightVecs, do not compute right eigenvectors.
- If **jobvr** = Nag\_RightVecs, compute the right eigenvectors.
- If **sense** = Nag\_CondOnly, Nag\_BackErrRight, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth, **jobvr** must be set to Nag\_RightVecs.
- Constraint:* **jobvr** = Nag\_NotRightVecs or Nag\_RightVecs.
- 4: **sense** – Nag\_CondErrType *Input*
- On entry:* determines whether, or not, condition numbers and backward errors are computed.
- sense** = Nag\_NoCondBackErr  
Do not compute condition numbers, or backward errors.
- sense** = Nag\_CondOnly  
Just compute condition numbers for the eigenvalues.
- sense** = Nag\_BackErrLeft  
Just compute backward errors for the left eigenpairs.
- sense** = Nag\_BackErrRight  
Just compute backward errors for the right eigenpairs.
- sense** = Nag\_BackErrBoth  
Compute backward errors for the left and right eigenpairs.
- sense** = Nag\_CondBackErrLeft  
Compute condition numbers for the eigenvalues and backward errors for the left eigenpairs.

**sense** = Nag\_CondBackErrRight

Compute condition numbers for the eigenvalues and backward errors for the right eigenpairs.

**sense** = Nag\_CondBackErrBoth

Compute condition numbers for the eigenvalues and backward errors for the left and right eigenpairs.

*Constraint:* **sense** = Nag\_NoCondBackErr, Nag\_CondOnly, Nag\_BackErrLeft, Nag\_BackErrRight, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth.

5: **tol** – double *Input*

*On entry:* **tol** is used as the tolerance for making decisions on rank in the deflation procedure. If **tol** is zero on entry then  $n \times \max(\|A\|, \|B\|, \|C\|) \times \mathit{machine\ precision}$  is used in place of **tol**, where  $\|Z\|$  is the Frobenius norm of the (scaled) matrix  $Z$  and  $\mathit{machine\ precision}$  is as returned by function nag\_machine\_precision (X02AJC). If **tol** is  $-1.0$  on entry then no deflation is attempted. The recommended value for **tol** is zero.

6: **n** – Integer *Input*

*On entry:* the order of the matrices  $A$ ,  $B$  and  $C$ .

*Constraint:*  $n \geq 0$ .

7: **a**[*dim*] – double *Input/Output*

**Note:** the dimension, *dim*, of the array **a** must be at least  $\mathbf{pda} \times \mathbf{n}$ .

The ( $i, j$ )th element of the matrix  $A$  is stored in  $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ .

*On entry:* the  $n$  by  $n$  matrix  $A$ .

*On exit:* **a** is used as internal workspace, but if **jobvl** = Nag\_LeftVecs or **jobvr** = Nag\_RightVecs, then **a** is restored on exit.

8: **pda** – Integer *Input*

*On entry:* the stride separating matrix row elements in the array **a**.

*Constraint:*  $\mathbf{pda} \geq \mathbf{n}$ .

9: **b**[*dim*] – double *Input/Output*

**Note:** the dimension, *dim*, of the array **b** must be at least  $\mathbf{pdb} \times \mathbf{n}$ .

The ( $i, j$ )th element of the matrix  $B$  is stored in  $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ .

*On entry:* the  $n$  by  $n$  matrix  $B$ .

*On exit:* **b** is used as internal workspace, but is restored on exit.

10: **pdb** – Integer *Input*

*On entry:* the stride separating matrix row elements in the array **b**.

*Constraint:*  $\mathbf{pdb} \geq \mathbf{n}$ .

11: **c**[*dim*] – double *Input/Output*

**Note:** the dimension, *dim*, of the array **c** must be at least  $\mathbf{pdc} \times \mathbf{n}$ .

The ( $i, j$ )th element of the matrix  $C$  is stored in  $\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$ .

*On entry:* the  $n$  by  $n$  matrix  $C$ .

*On exit:* **c** is used as internal workspace, but if **jobvl** = Nag\_LeftVecs or **jobvr** = Nag\_RightVecs, **c** is restored on exit.

- 12: **pdc** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **c**.  
*Constraint:* **pdc**  $\geq$  **n**.
- 13: **alphar**[2  $\times$  **n**] – double *Output*  
*On exit:* **alphar**[ $j - 1$ ], for  $j = 1, 2, \dots, 2n$ , contains the real part of  $\alpha_j$  for the  $j$ th eigenvalue pair  $(\alpha_j, \beta_j)$  of the quadratic eigenvalue problem.
- 14: **alphai**[2  $\times$  **n**] – double *Output*  
*On exit:* **alphai**[ $j - 1$ ], for  $j = 1, 2, \dots, 2n$ , contains the imaginary part of  $\alpha_j$  for the  $j$ th eigenvalue pair  $(\alpha_j, \beta_j)$  of the quadratic eigenvalue problem. If **alphai**[ $j - 1$ ] is zero then the  $j$ th eigenvalue is real; if **alphai**[ $j - 1$ ] is positive then the  $j$ th and  $(j + 1)$ th eigenvalues are a complex conjugate pair, with **alphai**[ $j$ ] negative.
- 15: **beta**[2  $\times$  **n**] – double *Output*  
*On exit:* **beta**[ $j - 1$ ], for  $j = 1, 2, \dots, 2n$ , contains the second part of the  $j$ th eigenvalue pair  $(\alpha_j, \beta_j)$  of the quadratic eigenvalue problem, with  $\beta_j \geq 0$ . Infinite eigenvalues have  $\beta_j$  set to zero.
- 16: **vl**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **vl** must be at least  $2 \times \mathbf{n}$  when **jobvl** = Nag\_LeftVecs.  
 Where **VL**( $i, j$ ) appears in this document, it refers to the array element **vl**[ $(j - 1) \times \mathbf{pdvl} + i - 1$ ].  
*On exit:* if **jobvl** = Nag\_LeftVecs, the left eigenvectors  $y_j$  are stored one after another in **vl**, in the same order as the corresponding eigenvalues. If the  $j$ th eigenvalue is real, then  $y_j = \mathbf{VL}(:, j)$ , the  $j$ th column of **VL**. If the  $j$ th and  $(j + 1)$ th eigenvalues form a complex conjugate pair, then  $y_j = \mathbf{VL}(:, j) + i \times \mathbf{VL}(:, j + 1)$  and  $y_{j+1} = \mathbf{VL}(:, j) - i \times \mathbf{VL}(:, j + 1)$ . Each eigenvector will be normalized with length unity and with the element of largest modulus real and positive.  
 If **jobvl** = Nag\_NotLeftVecs, **vl** is not referenced and may be **NULL**.
- 17: **pdvl** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **vl**.  
*Constraint:* **pdvl**  $\geq$  **n**.
- 18: **vr**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **vr** must be at least  $2 \times \mathbf{n}$  when **jobvr** = Nag\_RightVecs.  
 Where **VR**( $i, j$ ) appears in this document, it refers to the array element **vr**[ $(j - 1) \times \mathbf{pdvr} + i - 1$ ].  
*On exit:* if **jobvr** = Nag\_RightVecs, the right eigenvectors  $x_j$  are stored one after another in **vr**, in the same order as the corresponding eigenvalues. If the  $j$ th eigenvalue is real, then  $x_j = \mathbf{VR}(:, j)$ , the  $j$ th column of **VR**. If the  $j$ th and  $(j + 1)$ th eigenvalues form a complex conjugate pair, then  $x_j = \mathbf{VR}(:, j) + i \times \mathbf{VR}(:, j + 1)$  and  $x_{j+1} = \mathbf{VR}(:, j) - i \times \mathbf{VR}(:, j + 1)$ . Each eigenvector will be normalized with length unity and with the element of largest modulus real and positive.  
 If **jobvr** = Nag\_NotRightVecs, **vr** is not referenced and may be **NULL**.
- 19: **pdvr** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **vr**.  
*Constraint:* **pdvr**  $\geq$  **n**.

- 20: **s**[*dim*] – double *Output*
- Note:** the dimension, *dim*, of the array **s** must be at least  $2 \times \mathbf{n}$  when **sense** = Nag\_CondOnly, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth.
- Also: computing the condition numbers of the eigenvalues requires that both the left and right eigenvectors be computed.
- On exit:* if **sense** = Nag\_CondOnly, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth, **s**[*j* – 1] contains the condition number estimate for the *j*th eigenvalue (large condition numbers imply that the problem is near one with multiple eigenvalues). Infinite condition numbers are returned as the largest model double number (nag\_real\_largest\_number (X02ALC)).
- If **sense** = Nag\_NoCondBackErr, Nag\_BackErrLeft, Nag\_BackErrRight or Nag\_BackErrBoth, **s** is not referenced and may be **NULL**.
- 21: **bevl**[*dim*] – double *Output*
- Note:** the dimension, *dim*, of the array **bevl** must be at least  $2 \times \mathbf{n}$  when **sense** = Nag\_BackErrLeft, Nag\_BackErrBoth, Nag\_CondBackErrLeft or Nag\_CondBackErrBoth.
- On exit:* if **sense** = Nag\_BackErrLeft, Nag\_BackErrBoth, Nag\_CondBackErrLeft or Nag\_CondBackErrBoth, **bevl**[*j* – 1] contains the backward error estimate for the computed left eigenpair  $(\lambda_j, y_j)$ .
- If **sense** = Nag\_NoCondBackErr, Nag\_CondOnly, Nag\_BackErrRight or Nag\_CondBackErrRight, **bevl** is not referenced and may be **NULL**.
- 22: **bevr**[*dim*] – double *Output*
- Note:** the dimension, *dim*, of the array **bevr** must be at least  $2 \times \mathbf{n}$  when **sense** = Nag\_BackErrRight, Nag\_BackErrBoth, Nag\_CondBackErrRight or Nag\_CondBackErrBoth.
- On exit:* if **sense** = Nag\_BackErrRight, Nag\_BackErrBoth, Nag\_CondBackErrRight or Nag\_CondBackErrBoth, **bevr**[*j* – 1] contains the backward error estimate for the computed right eigenpair  $(\lambda_j, x_j)$ .
- If **sense** = Nag\_NoCondBackErr, Nag\_CondOnly, Nag\_BackErrLeft or Nag\_CondBackErrLeft, **bevr** is not referenced and may be **NULL**.
- 23: **iwarn** – Integer \* *Output*
- On exit:* **iwarn** will be positive if there are warnings, otherwise **iwarn** will be 0.
- If **fail.code** = NE\_NOERROR then:
- if **iwarn** = 1 then one, or both, of the matrices *A* and *C* is zero. In this case no scaling is performed, even if **scal** > 0;
  - if **iwarn** = 2 then the matrices *A* and *C* are singular, or nearly singular, so the problem is potentially ill-posed;
  - if **iwarn** = 3 then both the conditions for **iwarn** = 1 and **iwarn** = 2 above, apply.
- If **fail.code** = NE\_ITERATIONS\_QZ, nag\_dgges (f08xac) has flagged that **iwarn** eigenvalues are invalid.
- If **fail.code** = NE\_ITERATIONS\_QZ, nag\_dggeev (f08wac) has flagged that **iwarn** eigenvalues are invalid.
- 24: **fail** – NagError \* *Input/Output*
- The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_ARRAY\_SIZE

On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
Constraint: **pda**  $\geq$  **n**.

On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
Constraint: **pdb**  $\geq$  **n**.

On entry, **pdv** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
Constraint: **pdv**  $\geq$  **n**.

On entry, **pdvl** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$  and **jobvl** =  $\langle value \rangle$ .  
Constraint: when **jobvl** = Nag\_LeftVecs, **pdvl**  $\geq$  **n**.

On entry, **pdvr** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$  and **jobvr** =  $\langle value \rangle$ .  
Constraint: when **jobvr** = Nag\_RightVecs, **pdvr**  $\geq$  **n**.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .  
Constraint: **n**  $\geq$  0.

### NE\_INTERNAL\_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

### NE\_INVALID\_VALUE

On entry, **sense** =  $\langle value \rangle$  and **jobvl** =  $\langle value \rangle$ .  
Constraint: when **jobvl** = Nag\_NotLeftVecs, **sense** = Nag\_NoCondBackErr or Nag\_BackErrRight, when **jobvl** = Nag\_LeftVecs, **sense** = Nag\_CondOnly, Nag\_BackErrLeft, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth.

On entry, **sense** =  $\langle value \rangle$  and **jobvr** =  $\langle value \rangle$ .  
Constraint: when **jobvr** = Nag\_NotRightVecs, **sense** = Nag\_NoCondBackErr or Nag\_BackErrLeft, when **jobvr** = Nag\_RightVecs, **sense** = Nag\_CondOnly, Nag\_BackErrRight, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth.

### NE\_ITERATIONS\_QZ

The *QZ* iteration failed in nag\_dggeev (f08wac).

**iwarn** returns the value of **info** returned by nag\_dggeev (f08wac). This failure is unlikely to happen, but if it does, please contact NAG.

The *QZ* iteration failed in nag\_dgges (f08xac).

**iwarn** returns the value of **info** returned by nag\_dgges (f08xac). This failure is unlikely to happen, but if it does, please contact NAG.

### NE\_SINGULAR

The quadratic matrix polynomial is nonregular (singular).

## 7 Accuracy

The algorithm is backward stable for problems that are not too heavily damped, that is  $\|B\| \leq \sqrt{\|A\| \cdot \|C\|}$ .

## 8 Parallelism and Performance

nag\_eigen\_real\_gen\_quad (f02jcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_eigen\_real\_gen\_quad (f02jcc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

## 9 Further Comments

None.

## 10 Example

To solve the quadratic eigenvalue problem

$$(\lambda^2 A + \lambda B + C)x = 0$$

where

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 3 & 1 & 1 \\ 3 & 2 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 1 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix}.$$

The example also returns the left eigenvectors, condition numbers for the computed eigenvalues and backward errors of the computed right and left eigenpairs.

### 10.1 Program Text

```

/* nag_eigen_real_gen_quad (f02jcc) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf02.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>
#include <math.h>

int main(void)
{
    /* Integer scalar and array declarations */
    Integer          i, iwarn, j, pda, pdb, pdc, pdvl, pdvr, n;
    Integer          exit_status = 0;

    /* NAG structures and types */
    NagError         fail;
    Nag_ScaleType    scal;
    Nag_LeftVecsType jobvl;
    Nag_RightVecsType jobvr;

```

```

Nag_CondErrType  sense;

/* Double scalar and array declarations */
double           bmax, inf, tmp;
double           tol = 0.0;
double           *a = 0, *alphai= 0, *alphan= 0, *b = 0, *beta= 0, *bevl= 0;
double           *bevr= 0, *c = 0, *ei= 0, *er= 0, *s= 0, *vl = 0, * vr = 0;

/* Character scalar declarations */
char             cjobvl[40], cjobvr[40], cscal[40], csense[40];

/* Initialise the error structure */
INIT_FAIL(fail);

printf("nag_eigen_real_gen_quad (f02jcc) Example Program Results\n\n");

/* Skip heading in data file */
scanf("%*[\n] ");

/* Read in the problem size, scaling and output required */
scanf("%ld%39s%39s%*[\n] ", &n, cscal, csense);
scal = (Nag_ScaleType) nag_enum_name_to_value(cscal);
sense = (Nag_CondErrType) nag_enum_name_to_value(csense);
scanf("%39s%39s%*[\n] ", cjobvl, cjobvr);
jobvl = (Nag_LeftVecsType) nag_enum_name_to_value(cjobvl);
jobvr = (Nag_RightVecsType) nag_enum_name_to_value(cjobvr);

pda = n;
pdb = n;
pdc = n;
pdvl = n;
pdvr = n;

if (!(a = NAG_ALLOC(n*pda, double)) ||
    !(b = NAG_ALLOC(n*pdb, double)) ||
    !(c = NAG_ALLOC(n*pdc, double)) ||
    !(alphai = NAG_ALLOC(2*n, double)) ||
    !(alphan = NAG_ALLOC(2*n, double)) ||
    !(beta = NAG_ALLOC(2*n, double)) ||
    !(ei = NAG_ALLOC(2*n, double)) ||
    !(er = NAG_ALLOC(2*n, double)) ||
    !(vl = NAG_ALLOC(2*n*pdvl, double)) ||
    !(vr = NAG_ALLOC(2*n*pdvr, double)) ||
    !(s = NAG_ALLOC(2*n, double)) ||
    !(bevr = NAG_ALLOC(2*n, double)) ||
    !(bevl = NAG_ALLOC(2*n, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read in the matrix A */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        scanf("%lf", &a[j*pda+i]);
scanf("%*[\n] ");

/* Read in the matrix B */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        scanf("%lf", &b[j*pdb+i]);
scanf("%*[\n] ");

/* Read in the matrix C */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        scanf("%lf", &c[j*pdc+i]);
scanf("%*[\n] ");

```



```

/* nag_eigen_real_gen_quad (f02jcc): Solve the quadratic eigenvalue problem */
nag_eigen_real_gen_quad(scal, jobvl, jobvr, sense, tol, n, a, pda, b, pdb, c,
                        pdc, alphas, alphas, beta, vl, pdvl, vr, pdvr, s,
                        bevl, bevr, &iwarn, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_eigen_real_gen_quad (f02jcc).\n%s\n", fail.message);
    exit_status = -1;
    goto END;
} else if (iwarn!=0) {
    printf("Warning from nag_eigen_real_gen_quad (f02jcc).");
    printf(" iwarn = %ld\n", iwarn);
}

/* Infinity */
inf = X02ALC;

/* Display eigenvalues */
for (j = 0; j < 2*n; j++) {
    if (beta[j] >= 1.0) {
        er[j] = alphas[j]/beta[j];
        ei[j] = alphas[j]/beta[j];
    } else {
        tmp = inf*beta[j];
        if ((fabs(alphas[j])<tmp) && (fabs(alphas[j])<tmp)) {
            er[j] = alphas[j]/beta[j];
            ei[j] = alphas[j]/beta[j];
        } else {
            er[j] = inf;
            ei[j] = 0.0;
        }
    }
    if (er[j]<inf) {
        printf("Eigenvalue(%3ld) = (%11.4e, %11.4e)\n",j+1,er[j],ei[j]);
    } else {
        printf("Eigenvalue(%3ld) is infinite\n",j+1);
    }
}

if (jobvr == Nag_RightVecs) {
    printf("\n");
    /* x04cac: Print out the right eigenvectors */
    nag_gen_real_mat_print (Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                            n, 2*n, vr, pdvr, "Right eigenvectors (matrix VR)",
                            NULL, &fail);
}

if (jobvl == Nag_LeftVecs && fail.code == NE_NOERROR) {
    printf("\n");
    /* x04cac: Print out the left eigenvectors */
    nag_gen_real_mat_print (Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                            n, 2*n, vl, pdvl, "Left eigenvectors (matrix VL)",
                            NULL, &fail);
}

if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Display condition numbers and errors, as required */
if (sense==Nag_CondOnly || sense==Nag_CondBackErrLeft ||
    sense==Nag_CondBackErrRight || sense==Nag_CondBackErrBoth) {
    printf("\n");
    printf("Eigenvalue Condition numbers\n");
    for (j = 0 ; j < 2*n; j++)
        printf("%2ld %11.4e\n", j+1, s[j]);
}

if (sense==Nag_BackErrRight || sense==Nag_BackErrBoth ||
    sense==Nag_CondBackErrRight || sense==Nag_CondBackErrBoth) {
    /* nag_dmax_val (f16jnc).
    * Get maximum value (bmax) and location of that value (j) of bevr.

```

```

    */
    nag_dmax_val(2*n, bevr, 1, &j, &bmax, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dmax_val (f16jnc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    printf("\n");
    printf("Backward errors for eigenvalues and right eigenvectors\n");
    if (bmax<10.0*X02AJC) {
        printf(" All errors are less than 10 times machine precision\n");
    } else {
        for (j = 0; j < 2*n; j++)
            printf("%11.4e\n", bevr[j]);
    }
}

if (sense==Nag_BackErrLeft || sense==Nag_BackErrBoth ||
    sense==Nag_CondBackErrLeft || sense==Nag_CondBackErrBoth) {
    /* nag_dmax_val (f16jnc).
    * Get maximum value (bmax) and location of that value (j) of bevl.
    */
    nag_dmax_val(2*n, bevl, 1, &j, &bmax, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dmax_val (f16jnc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    printf("\n");
    printf("Backward errors for eigenvalues and left eigenvectors\n");
    if (bmax<10.0*X02AJC) {
        printf(" All errors are less than 10 times machine precision\n");
    } else {
        for (j = 0; j < 2*n; j++)
            printf("%11.4e\n", bevl[j]);
    }
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(alphai);
NAG_FREE(alphar);
NAG_FREE(beta);
NAG_FREE(ei);
NAG_FREE(er);
NAG_FREE(vl);
NAG_FREE(vr);
NAG_FREE(s);
NAG_FREE(bevr);
NAG_FREE(bevl);

return(exit_status);
}

```

## 10.2 Program Data

```

nag_eigen_real_gen_quad (f02jcc) Example Program Data
 3 Nag_CondFanLinVanDooren Nag_CondBackErrBoth : n, scal, sense
Nag_LeftVecs Nag_RightVecs : jobvl, jobvr

1.0 2.0 2.0
3.0 1.0 1.0
3.0 2.0 1.0 : a

3.0 2.0 1.0
2.0 1.0 3.0
1.0 3.0 2.0 : b

```

```

1.0  1.0  2.0
2.0  3.0  1.0
3.0  1.0  2.0

```

: c

### 10.3 Program Results

nag\_eigen\_real\_gen\_quad (f02jcc) Example Program Results

```

Eigenvalue( 1) = (-3.8513e+00,  0.0000e+00)
Eigenvalue( 2) = (-5.9217e-01,  8.0280e-01)
Eigenvalue( 3) = (-5.9217e-01, -8.0280e-01)
Eigenvalue( 4) = ( 5.2326e-01,  6.2251e-01)
Eigenvalue( 5) = ( 5.2326e-01, -6.2251e-01)
Eigenvalue( 6) = ( 7.8909e-01,  0.0000e+00)

```

Right eigenvectors (matrix VR)

	1	2	3	4	5	6
1	-0.2108	0.3751	-0.1877	-0.6593	0.0424	-0.3478
2	0.7695	0.5020	-0.2433	0.0302	0.0197	0.8277
3	-0.6028	0.7162	0.0000	0.7498	0.0000	-0.4405

Left eigenvectors (matrix VL)

	1	2	3	4	5	6
1	0.1052	0.7816	0.0000	0.8079	0.0000	0.0358
2	0.7381	0.5075	-0.1352	-0.1124	-0.0314	0.7072
3	-0.6664	0.3202	-0.1038	-0.5704	0.0913	-0.7061

Eigenvalue Condition numbers

1	2.3092e+00
2	7.0275e-01
3	7.0275e-01
4	2.7013e+00
5	2.7013e+00
6	2.0144e+00

Backward errors for eigenvalues and right eigenvectors

All errors are less than 10 times machine precision

Backward errors for eigenvalues and left eigenvectors

All errors are less than 10 times machine precision

---