

## NAG Library Function Document

### nag\_eigen\_complex\_gen\_quad (f02jqc)

#### 1 Purpose

nag\_eigen\_complex\_gen\_quad (f02jqc) solves the quadratic eigenvalue problem

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

where  $A$ ,  $B$  and  $C$  are complex  $n$  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_complex_gen_quad (Nag_ScaleType scal, Nag_LeftVecsType jobvl,
    Nag_RightVecsType jobvr, Nag_CondErrType sense, double tol, Integer n,
    Complex a[], Integer pda, Complex b[], Integer pdb, Complex c[],
    Integer pdc, Complex alpha[], Complex beta[], Complex vl[],
    Integer pdvl, Complex 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  $\|B\| \leq 10\sqrt{\|A\| \cdot \|C\|}$ .

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  $n \times \mathit{machine\ precision}$  is used in place of **tol**, where *machine precision* is as returned by function nag\_machine\_precision (X02AJC). A diagonal element of a triangular matrix,  $R$ , is regarded as zero if  $|r_{jj}| \leq \mathit{tol} \times \mathit{size}(X)$ , or  $n \times \mathit{machine\ precision} \times \mathit{size}(X)$  when **tol** is zero, where  $\mathit{size}(X)$  is based on the size of the absolute values of the elements of the matrix  $X$  containing the matrix  $R$ . See Hammarling *et al.* (2013) for the motivation. 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:*  $\mathbf{n} \geq 0$ .

7: **a**[*dim*] – Complex *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 **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*] – Complex *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 **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*] – Complex *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 **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: **pdv** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **c**.  
*Constraint:* **pdv**  $\geq$  **n**.
- 13: **alpha**[ $2 \times \mathbf{n}$ ] – Complex *Output*  
*On exit:* **alpha**[ $j - 1$ ], for  $j = 1, 2, \dots, 2n$ , contains the first part of the the  $j$ th eigenvalue pair  $(\alpha_j, \beta_j)$  of the quadratic eigenvalue problem.
- 14: **beta**[ $2 \times \mathbf{n}$ ] – Complex *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. Although **beta** is declared complex, it is actually real and non-negative. Infinite eigenvalues have  $\beta_j$  set to zero.
- 15: **vl**[*dim*] – Complex *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**[( $j - 1$ )  $\times$  **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. 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**.
- 16: **pdvl** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **vl**.  
*Constraint:* **pdvl**  $\geq$  **n**.
- 17: **vr**[*dim*] – Complex *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**[( $j - 1$ )  $\times$  **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. 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**.
- 18: **pdvr** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **vr**.  
*Constraint:* **pdvr**  $\geq$  **n**.
- 19: **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.  
**Note:** 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 real 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**.

20: **bevl**[*dim*] – double *Output*

**Note:** the dimension, *dim*, of the array **bevl** must be at least  $2 \times n$  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**.

21: **bevr**[*dim*] – double *Output*

**Note:** the dimension, *dim*, of the array **bevr** must be at least  $2 \times n$  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**.

22: **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** = Nag\_CondFanLinVanDooren;

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 **iwarn** = 4,  $\|b\| \geq 10\sqrt{\|A\| \cdot \|C\|}$  and backward stability cannot be guaranteed.

If **fail.code** = NE\_ITERATIONS\_QZ, nag\_zgges (f08xnc) has flagged that **iwarn** eigenvalues are invalid.

If **fail.code** = NE\_ITERATIONS\_QZ, nag\_zggeev (f08wnc) has flagged that **iwarn** eigenvalues are invalid.

23: **fail** – NagError \* *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

**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.

An unexpected error has been triggered by this function. Please contact NAG.  
 See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

**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\_zggeev (f08wnc).  
**iwarn** returns the value of **info** returned by nag\_zggeev (f08wnc). This failure is unlikely to happen, but if it does, please contact NAG.

The *QZ* iteration failed in nag\_zgges (f08xnc).  
**iwarn** returns the value of **info** returned by nag\_zgges (f08xnc). This failure is unlikely to happen, but if it does, please contact NAG.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
 See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

**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\_complex\_gen\_quad (f02jqc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_eigen\_complex\_gen\_quad (f02jqc) 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 x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also 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

$$A = \begin{pmatrix} 2i & 4i & 4i \\ 6i & 2i & 2i \\ 6i & 4i & 2i \end{pmatrix}, \quad B = \begin{pmatrix} 3+3i & 2+2i & 1+i \\ 2+2i & 1+i & 3+3i \\ 1+i & 3+3i & 2+2i \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix}.$$

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

### 10.1 Program Text

```

/* nag_eigen_complex_gen_quad (f02jqc) Example Program.
*
* NAGPRODCODE Version.
*
* Copyright 2016 Numerical Algorithms Group.
*
* Mark 26, 2016.
*/

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf02.h>
#include <nagx02.h>
#include <nagx04.h>
#include <nagm01.h>
#include <math.h>

#ifdef __cplusplus
extern "C"
{
#endif
    static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b);
#ifdef __cplusplus
}
#endif

```

```

#define COMPLEX(A)      A.re, A.im

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

    size_t      *indices = 0;

    /* Nag Types */
    NagError      fail;
    Nag_ScaleType  scal;
    Nag_LeftVecsType  jobvl;
    Nag_RightVecsType  jobvr;
    Nag_CondErrType  sense;

    /* Double scalar and array declarations */
    double      rbetaj;
    double      tol = 0.0;
    double      *bevl = 0, *bevr = 0, *s = 0;

    /* Complex scalar and array declarations */
    Complex      *a = 0, *alpha = 0, *b = 0, *beta = 0,
                *c = 0, *e = 0, *vl = 0, *vr = 0, *cvr = 0;

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

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

    printf("nag_eigen_complex_gen_quad (f02jqc) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    /* Read in the problem size, scaling and output required */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%39s%39s%*[\n] ", &n, cscal,
            (unsigned)_countof(cscal), csense,
            (unsigned)_countof(csense));
#else
    scanf("%" NAG_IFMT "%39s%39s%*[\n] ", &n, cscal, csense);
#endif
    scal = (Nag_ScaleType) nag_enum_name_to_value(cscal);
    sense = (Nag_CondErrType) nag_enum_name_to_value(csense);

#ifdef _WIN32
    scanf_s("%39s%39s%*[\n] ", cjobvl, (unsigned)_countof(cjobvl), cjobvr,
            (unsigned)_countof(cjobvr));
#else
    scanf("%39s%39s%*[\n] ", cjobvl, cjobvr);
#endif
    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, Complex)) ||
        !(b = NAG_ALLOC(n * pdb, Complex)) ||
        !(c = NAG_ALLOC(n * pdc, Complex)) ||

```



```

!(alpha = NAG_ALLOC(2 * n, Complex)) ||
!(beta = NAG_ALLOC(2 * n, Complex)) ||
!(e = NAG_ALLOC(2 * n, Complex)) ||
!(vl = NAG_ALLOC(2 * n * pdvl, Complex)) ||
!(vr = NAG_ALLOC(2 * n * pdvr, Complex)) ||
!(s = NAG_ALLOC(2 * n, double)) ||
!(bevr = NAG_ALLOC(2 * n, double)) ||
!(bevl = NAG_ALLOC(2 * n, double)) ||
!(cvr = NAG_ALLOC(n, Complex)) ||
!(indices = NAG_ALLOC(2 * n, size_t))
{
    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++)
#ifdef _WIN32
    scanf_s("%lf%lf", COMPLEX(&a[j * pda + i]));
#else
    scanf("%lf%lf", COMPLEX(&a[j * pda + i]));
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Read in the matrix B */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
#ifdef _WIN32
    scanf_s("%lf%lf", COMPLEX(&b[j * pdb + i]));
#else
    scanf("%lf%lf", COMPLEX(&b[j * pdb + i]));
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Read in the matrix C */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
#ifdef _WIN32
    scanf_s("%lf%lf", COMPLEX(&c[j * pdc + i]));
#else
    scanf("%lf%lf", COMPLEX(&c[j * pdc + i]));
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* nag_eigen_complex_gen_quad (f02jqc):
 * Solve the quadratic eigenvalue problem */
nag_eigen_complex_gen_quad(scal, jobvl, jobvr, sense, tol, n, a, pda, b,
                           pdb, c, pdc, alpha, beta, vl, pdvl, vr, pdvr, s,
                           bevl, bevr, &iwarn, &fail);

if (fail.code != NE_NOERROR) {
    printf("Error from nag_eigen_complex_gen_quad (f02jqc).\n%s\n",
           fail.message);
    exit_status = -1;
    goto END;
}
else if (iwarn != 0) {

```

```

printf("Warning from nag_eigen_complex_gen_quad (f02jqc).");
printf("  iwarn = %" NAG_IFMT "\n", iwarn);
}

/* Display eigenvalues */
for (j = 0; j < 2 * n; j++) {
  rbetaj = beta[j].re;
  if (rbetaj > 0.0) {
    e[j].re = alpha[j].re / rbetaj;
    e[j].im = alpha[j].im / rbetaj;
  } else {
    isinf = j + 1;
  }
}
if (isinf) {
  printf("Eigenvalue(%3" NAG_IFMT ") is infinite\n", isinf);
} else {

/* Sort eigenvalues by decscending absolute value and then by
 * descending real part. Sort requested eigenvectors correspondingly.
 */
nag_rank_sort((Pointer) e, 2*n, (ptrdiff_t) (sizeof(Complex)),
              compare, Nag_Descending, indices, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_rank_sort (m0ldsc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
/* nag_make_indices (m0lzac).
 * Inverts a permutation converting a rank vector to an
 * index vector or vice versa
 */
nag_make_indices(indices, 2*n, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_make_indices (m0lzac).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
/* nag_reorder_vector (m0lesc).
 * Reorders set of values of arbitrary data type into the
 * order specified by a set of indices
 */
nag_reorder_vector((Pointer) e, 2*n, sizeof(Complex),
                  (ptrdiff_t) (sizeof(Complex)), indices, &fail);
if (fail.code == NE_NOERROR && jobvl == Nag_LeftVecs) {
  for (i = 0; i < n; i++) {
    nag_reorder_vector((Pointer) &vl[i], 2*n, sizeof(Complex),
                      (ptrdiff_t) (n*sizeof(Complex)), indices, &fail);
  }
}
if (fail.code == NE_NOERROR && jobvr == Nag_RightVecs) {
  for (i = 0; i < n; i++) {
    nag_reorder_vector((Pointer) &vr[i], 2*n, sizeof(Complex),
                      (ptrdiff_t) (n*sizeof(Complex)), indices, &fail);
  }
}
if (fail.code != NE_NOERROR) {
  printf("Error from nag_reorder_vector (m0lesc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print eigenvalues as 1 by 2n matrix using
 * nag_gen_real_mat_print_comp (x04dbc).
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                              Nag_NonUnitDiag, 1, 2*n, e, 1,
                              Nag_BracketForm, "%7.4f", "Eigenvalues:",
                              Nag_NoLabels, 0, Nag_IntegerLabels, 0,
                              60, 0, 0, &fail);

```

```

}
printf("\n");

if (fail.code == NE_NOERROR && jobvr == Nag_RightVecs) {
    /* Print right eigenvectors using
     * nag_gen_complx_mat_print_comp (x04dbc).
     */
    fflush(stdout);
    nag_gen_complx_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                  Nag_NonUnitDiag, n, 2*n, vr, pdvr,
                                  Nag_BracketForm, "%7.4f",
                                  "Right Eigenvectors:", Nag_NoLabels,
                                  0, Nag_IntegerLabels, 0, 60, 0, 0, &fail);

    printf("\n");
}

if (fail.code == NE_NOERROR && jobvl == Nag_LeftVecs) {
    /* Print left eigenvectors using
     * nag_gen_complx_mat_print_comp (x04dbc).
     */
    fflush(stdout);
    nag_gen_complx_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                  Nag_NonUnitDiag, n, 2*n, vl, pdvl,
                                  Nag_BracketForm, "%7.4f",
                                  "Left Eigenvectors:", Nag_NoLabels,
                                  0, Nag_IntegerLabels, 0, 60, 0, 0, &fail);

    printf("\n");
}

if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

if (!cond_errors)
    goto END;

/* Display condition numbers and errors, as required */
if (sense == Nag_CondOnly || sense == Nag_CondBackErrLeft ||
    sense == Nag_CondBackErrRight || sense == Nag_CondBackErrBoth) {
    /* Print eigenvalue condition numbers as 1 by 2n matrix using
     * nag_gen_real_mat_print_comp (x04cbc).
     */
    fflush(stdout);
    nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                Nag_NonUnitDiag, 1, 2*n, s, 1,
                                "%17.2f", "Eigenvalue Condition numbers:",
                                Nag_NoLabels, 0, Nag_IntegerLabels, 0,
                                60, 0, 0, &fail);

    printf("\n");
}

if (sense == Nag_BackErrRight || sense == Nag_BackErrBoth ||
    sense == Nag_CondBackErrRight || sense == Nag_CondBackErrBoth) {
    if (fail.code == NE_NOERROR) {
        fflush(stdout);
        nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                    Nag_NonUnitDiag, 1, 2*n, bevr, 1,
                                    "%17.1e",
                                    "Backward errors for eigenvalues and "
                                    "right eigenvectors:",
                                    Nag_NoLabels, 0, Nag_IntegerLabels, 0,
                                    60, 0, 0, &fail);

        printf("\n");
    }
}

if (sense == Nag_BackErrLeft || sense == Nag_BackErrBoth ||
    sense == Nag_CondBackErrLeft || sense == Nag_CondBackErrBoth) {

```

```

    if (fail.code == NE_NOERROR) {
        fflush(stdout);
        nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                    Nag_NonUnitDiag, 1, 2*n, bevl, 1,
                                    "%17.1e",
                                    "Backward errors for eigenvalues and "
                                    "left eigenvectors:",
                                    Nag_NoLabels, 0, Nag_IntegerLabels, 0,
                                    60, 0, 0, &fail);

        printf("\n");
    }
}
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
          fail.message);
    exit_status = 2;
}
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(alpha);
NAG_FREE(beta);
NAG_FREE(e);
NAG_FREE(vl);
NAG_FREE(vr);
NAG_FREE(s);
NAG_FREE(bevr);
NAG_FREE(bevl);
NAG_FREE(cvr);
NAG_FREE(indices);

return (exit_status);
}
static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b)
{
    double a_re = *((const Complex *) a).re;
    double a_im = *((const Complex *) a).im;
    double b_re = *((const Complex *) b).re;
    double b_im = *((const Complex *) b).im;
    double diff;
    Integer x;

    diff = (a_re*a_re + a_im*a_im)-(b_re*b_re + b_im*b_im);
    if (fabs(diff) < 1000.0*x02ajc()) {
        if (a_re < b_re) {
            x = -1;
        } else if (a_re > b_re) {
            x = 1;
        } else {
            x = 0;
        }
    } else if (diff < 0.0) {
        x = -1;
    } else {
        x = 1;
    }
    return x;
}
}

```

## 10.2 Program Data

```

nag_eigen_complex_gen_quad (f02jqc) Example Program Data
 3 Nag_CondFanLinVanDooren Nag_CondBackErrBoth : n, scal, sense
Nag_LeftVecs Nag_RightVecs : jobvl, jobvr

0.0 2.0 0.0 4.0 0.0 4.0
0.0 6.0 0.0 2.0 0.0 2.0
0.0 6.0 0.0 4.0 0.0 2.0 : a

```

```

3.0 3.0    2.0 2.0    1.0 1.0
2.0 2.0    1.0 1.0    3.0 3.0
1.0 1.0    3.0 3.0    2.0 2.0           : b

1.0 0.0    1.0 0.0    2.0 0.0
2.0 0.0    3.0 0.0    1.0 0.0
3.0 0.0    1.0 0.0    2.0 0.0           : c

```

### 10.3 Program Results

nag\_eigen\_complex\_gen\_quad (f02jqc) Example Program Results

```

Eigenvalues:
           1           2           3
(-1.9256, 1.9256) ( 0.1053, 0.6975) (-0.6975,-0.1053)

           4           5           6
( 0.5729, 0.0496) (-0.0496,-0.5729) ( 0.3945,-0.3945)

Right Eigenvectors:
           1           2           3
(-0.2108, 0.0000) ( 0.3751,-0.1877) ( 0.3751, 0.1877)
( 0.7695, 0.0000) ( 0.5020,-0.2433) ( 0.5020, 0.2433)
(-0.6028,-0.0000) ( 0.7162, 0.0000) ( 0.7162, 0.0000)

           4           5           6
(-0.6593, 0.0424) (-0.6593,-0.0424) (-0.3478, 0.0000)
( 0.0302, 0.0197) ( 0.0302,-0.0197) ( 0.8277, 0.0000)
( 0.7498, 0.0000) ( 0.7498, 0.0000) (-0.4405,-0.0000)

Left Eigenvectors:
           1           2           3
( 0.1052,-0.0000) ( 0.7816, 0.0000) ( 0.7816, 0.0000)
( 0.7381, 0.0000) ( 0.5075,-0.1352) ( 0.5075, 0.1352)
(-0.6664, 0.0000) ( 0.3202,-0.1038) ( 0.3202, 0.1038)

           4           5           6
( 0.8079, 0.0000) ( 0.8079, 0.0000) ( 0.0358, 0.0000)
(-0.1124,-0.0314) (-0.1124, 0.0314) ( 0.7072, 0.0000)
(-0.5704, 0.0913) (-0.5704,-0.0913) (-0.7061,-0.0000)

```

---