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 \(j = 1, 2, \ldots, 2n\), 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

```c
#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 alphar[], double alphai[], 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


5 Arguments

1: **scal** – Nag_ScaleType
   
   *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
   
   *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
   
   *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
   
   *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.
Compute condition numbers for the eigenvalues and backward errors for the right eigenpairs.

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.

On entry: tol is used as the tolerance for making decisions on rank in the deflation procedure. If tol is zero on entry then \( \frac{n}{\max(||A||, ||B||, ||C||)} \times \text{machine precision} \) is used in place of tol, where \( ||Z|| \) is the Frobenius norm of the (scaled) matrix \( Z \) and \( \text{machine precision} \) 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.

On entry: the order of the matrices \( A, B \) and \( C \).

Constraint: \( n \geq 0 \).

Note: the dimension, dim, of the array a must be at least pda × n.

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

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.

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

Constraint: pda \( \geq n \).

Note: the dimension, dim, of the array b must be at least pdb × n.

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

On entry: the \( n \) by \( n \) matrix \( B \).

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

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

Constraint: pdb \( \geq n \).

Note: the dimension, dim, of the array c must be at least pdc × n.

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

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: \texttt{pdc} – Integer \textit{Input}

\textit{On entry:} the stride separating matrix row elements in the array \texttt{c}.

\textit{Constraint:} \texttt{pdc} \geq \texttt{n}.

13: \texttt{alphar}[2 \times \texttt{n}] – double \textit{Output}

\textit{On exit:} \texttt{alphar}[j - 1], for \( j = 1, 2, \ldots, 2n \), contains the real part of \( \alpha_j \) for the \( j \)th eigenvalue pair \((\alpha_j, \beta_j)\) of the quadratic eigenvalue problem.

14: \texttt{alpha}[2 \times \texttt{n}] – double \textit{Output}

\textit{On exit:} \texttt{alpha}[j - 1], for \( j = 1, 2, \ldots, 2n \), contains the imaginary part of \( \alpha_j \) for the \( j \)th eigenvalue pair \((\alpha_j, \beta_j)\) of the quadratic eigenvalue problem. If \texttt{alpha}[j - 1] is zero then the \( j \)th eigenvalue is real; if \texttt{alpha}[j - 1] is positive then the \( j \)th and \((j + 1)\)th eigenvalues are a complex conjugate pair, with \texttt{alpha}[j] negative.

15: \texttt{beta}[2 \times \texttt{n}] – double \textit{Output}

\textit{On exit:} \texttt{beta}[j - 1], for \( j = 1, 2, \ldots, 2n \), 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: \texttt{vl[dim]} – double \textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{vl} must be at least \( 2 \times \texttt{n} \) when \texttt{jobvl} = Nag_LeftVecs.

Where \texttt{VL}(i, j) appears in this document, it refers to the array element \texttt{vl}[(j - 1) \times \texttt{pdvl} + i - 1].

\textit{On exit:} if \texttt{jobvl} = Nag_LeftVecs, the left eigenvectors \( y_j \) are stored one after another in \texttt{vl}, in the same order as the corresponding eigenvalues. If the \( j \)th eigenvalue is real, then \( y_j = \texttt{VL}(:, j) \), the \( j \)th column of \texttt{VL}. If the \( j \)th and \((j + 1)\)th eigenvalues form a complex conjugate pair, then \( y_j = \texttt{VL}(:, j) + i \times \texttt{VL}(:, j + 1) \) and \( y_{j+1} = \texttt{VL}(:, j) - i \times \texttt{VL}(:, j + 1) \). Each eigenvector will be normalized with length unity and with the element of largest modulus real and positive.

If \texttt{jobvl} = Nag_NotLeftVecs, \texttt{vl} is not referenced and may be \texttt{NULL}.

17: \texttt{pdvl} – Integer \textit{Input}

\textit{On entry:} the stride separating matrix row elements in the array \texttt{vl}.

\textit{Constraint:} \texttt{pdvl} \geq \texttt{n}.

18: \texttt{vr[dim]} – double \textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{vr} must be at least \( 2 \times \texttt{n} \) when \texttt{jobvr} = Nag_RightVecs.

Where \texttt{VR}(i, j) appears in this document, it refers to the array element \texttt{vr}[(j - 1) \times \texttt{pdvr} + i - 1].

\textit{On exit:} if \texttt{jobvr} = Nag_RightVecs, the right eigenvectors \( x_j \) are stored one after another in \texttt{vr}, in the same order as the corresponding eigenvalues. If the \( j \)th eigenvalue is real, then \( x_j = \texttt{VR}(:, j) \), the \( j \)th column of \texttt{VR}. If the \( j \)th and \((j + 1)\)th eigenvalues form a complex conjugate pair, then \( x_j = \texttt{VR}(:, j) + i \times \texttt{VR}(:, j + 1) \) and \( x_{j+1} = \texttt{VR}(:, j) - i \times \texttt{VR}(:, j + 1) \). Each eigenvector will be normalized with length unity and with the element of largest modulus real and positive.

If \texttt{jobvr} = Nag_NotRightVecs, \texttt{vr} is not referenced and may be \texttt{NULL}.

19: \texttt{pdvr} – Integer \textit{Input}

\textit{On entry:} the stride separating matrix row elements in the array \texttt{vr}.

\textit{Constraint:} \texttt{pdvr} \geq \texttt{n}.
Note: the dimension, \( dim \), of the array \( s \) must be at least \( 2 \times n \) when \( sense = \text{Nag\_CondOnly}, \text{Nag\_CondBackErrLeft}, \text{Nag\_CondBackErrRight} \) or \( \text{Nag\_CondBackErrBoth} \).

Note: also: computing the condition numbers of the eigenvalues requires that both the left and right eigenvectors be computed.

On exit: if \( sense = \text{Nag\_CondOnly}, \text{Nag\_CondBackErrLeft}, \text{Nag\_CondBackErrRight} \) or \( \text{Nag\_CondBackErrBoth} \), \( s[j - 1] \) 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 (\text{nag\_real\_largest\_number} (X02ALC)).

If \( sense = \text{Nag\_NoCondBackErr}, \text{Nag\_CondOnly}, \text{Nag\_BackErrRight} \) or \( \text{Nag\_CondBackErrRight} \), \( s \) is not referenced and may be NULL.

Note: the dimension, \( dim \), of the array \( bevl \) must be at least \( 2 \times n \) when \( sense = \text{Nag\_BackErrLeft}, \text{Nag\_BackErrBoth}, \text{Nag\_CondBackErrLeft} \) or \( \text{Nag\_CondBackErrBoth} \).

On exit: if \( sense = \text{Nag\_BackErrLeft}, \text{Nag\_BackErrBoth}, \text{Nag\_CondBackErrLeft} \) or \( \text{Nag\_CondBackErrBoth} \), \( bevl[j - 1] \) contains the backward error estimate for the computed left eigenpair \((\lambda_j, y_j)\).

If \( sense = \text{Nag\_NoCondBackErr}, \text{Nag\_CondOnly}, \text{Nag\_BackErrRight} \) or \( \text{Nag\_CondBackErrRight} \), \( bevl \) is not referenced and may be NULL.

Note: the dimension, \( dim \), of the array \( bevr \) must be at least \( 2 \times n \) when \( sense = \text{Nag\_BackErrRight}, \text{Nag\_BackErrBoth}, \text{Nag\_CondBackErrRight} \) or \( \text{Nag\_CondBackErrBoth} \).

On exit: if \( sense = \text{Nag\_BackErrRight}, \text{Nag\_BackErrBoth}, \text{Nag\_CondBackErrRight} \) or \( \text{Nag\_CondBackErrBoth} \), \( bevr[j - 1] \) contains the backward error estimate for the computed right eigenpair \((\lambda_j, x_j)\).

If \( sense = \text{Nag\_NoCondBackErr}, \text{Nag\_CondOnly}, \text{Nag\_BackErrLeft} \) or \( \text{Nag\_CondBackErrLeft} \), \( bevr \) is not referenced and may be NULL.

On exit: \( iwarn \) will be positive if there are warnings, otherwise \( iwarn \) will be 0.

If \( \text{fail\_code} = \text{NE\_NOERROR} \) 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 \( \text{fail\_code} = \text{NE\_ITERATIONS\_QZ} \), \text{f08xac} has flagged that \( iwarn \) eigenvalues are invalid.

If \( \text{fail\_code} = \text{NE\_ITERATIONS\_QZ} \), \text{f08wac} has flagged that \( iwarn \) eigenvalues are invalid.

The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE
On entry, pda = ⟨value⟩ and n = ⟨value⟩.
Constraint: pda ≥ n.
On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ n.
On entry, pdc = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdc ≥ n.
On entry, pdvl = ⟨value⟩, n = ⟨value⟩ and jobvl = ⟨value⟩.
Constraint: when jobvl = Nag_LeftVecs, pdvl ≥ n.
On entry, pdvr = ⟨value⟩, n = ⟨value⟩ and jobvr = ⟨value⟩.
Constraint: when jobvr = Nag_RightVecs, pdvr ≥ n.

NE_BAD_PARAM
On entry, argument ⟨value⟩ had an illegal value.

NE_INT
On entry, n = ⟨value⟩.
Constraint: n ≥ 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 the Essential Introduction for further information.

NE_INVALID_VALUE
On entry, sense = ⟨value⟩ and jobvl = ⟨value⟩.
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 = ⟨value⟩ and jobvr = ⟨value⟩.
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_dggev (f08wac).
iwarn returns the value of info returned by nag_dggev (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.
The quadratic matrix polynomial is nonregular (singular).

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

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 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.

Further Comments
None.

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.

Program Text
/* nag_eigen_real_gen_quad (f02jcc) Example Program. */
* * Copyright 2014 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 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, *alphar = 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");
    fflush(stdout);

    /* 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, _countof(cscal), csense, _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, _countof(cjobvl), cjobvr, _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, double)) ||
        !(b = NAG_ALLOC(n*pdb, double)) ||
        !(c = NAG_ALLOC(n*pdc, double)) ||
        !(alphai = NAG_ALLOC(2*n, double)) ||
        !(alphar = 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++)
    #ifdef _WIN32
        scanf_s("%lf", &a[j*pda+i]);
    #else
        scanf("%lf", &a[j*pda+i]);
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n ] ");
    #else
        scanf("%*[\n ] ");
    #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", &b[j*pdb+i]);
    #else
        scanf("%lf", &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", &c[j*pdc+i]);
    #else
        scanf("%lf", &c[j*pdc+i]);
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n ] ");
    #else
        scanf("%*[\n ] ");
    #endif

/* nag_eigen_real_gen_quad (f02jcc): Solve the quadratic eigenvalue problem */
ag_eigen_real_gen_quad(scal, jobvl, jobvr, sense, tol, n, a, pda, b, pdb, c, pdc, alphar, alphai, 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", fail.message);
    exit_status = -1;
    goto END;
} else if (iwarn != 0) {
    printf("Warning from nag_eigen_real_gen_quad (f02jcc).\n", iwarn);
    }

/* Infinity */
inf = X02ALC;

/* Display eigenvalues */
for (j = 0; j < 2*n; j++)
    if (beta[j] >= 1.0) {
        er[j] = alphar[j]/beta[j];
        ei[j] = alphai[j]/beta[j];
        }
} else {
    tmp = inf*beta[j];
    if ((fabs(alphar[j])<tmp) && (fabs(alphai[j])<tmp)) {
        er[j] = alphar[j]/beta[j];
        ei[j] = alphai[j]/beta[j];
    } else {
        er[j] = inf;
        ei[j] = 0.0;
    }
    if (er[j]<inf) {
        printf("Eigenvalue(%3"NAG_IFMT") = (%11.4e, %11.4e)\n",j+1,er[j],ei[j]);
    } else {
        printf("Eigenvalue(%3"NAG_IFMT") is infinite\n",j+1);
    }
}
if (jobvr == Nag_RightVecs) {
    printf("\n");
    fflush(stdout);
    /* 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");
    fflush(stdout);
    /* 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%f
", 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("%2"NAG_IFMT" %11.4e\n", j+1, s[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 bevr.
     */
    nag_dmax_val(2*n, bevr, 1, &j, &bmax, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dmax_val (f16jnc).\n%f
", 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_CondBackErrLeft || sense==Nag_CondBackErrBoth ||
    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 = 2;
    goto END;
}

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);
}
Left eigenvectors (matrix VL)

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