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 \(j = 1, 2, \ldots, 2n\), 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_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 \(|B| \leq \sqrt{|A| \cdot |C|}\).

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

4 References


5 Arguments

1: \texttt{scal} – Nag\_ScaleType \hspace{1cm} \textit{Input}

\textit{On entry:} determines the form of scaling to be performed on \(A\), \(B\) and \(C\).

\texttt{scal} = Nag\_NoScale
No scaling.

\texttt{scal} = Nag\_CondFanLinVanDooren (the recommended value)
Fan, Lin and Van Dooren scaling if \(\frac{\|B\|}{\sqrt{\|A\|\cdot\|C\|}} < 10\) and no scaling otherwise where \(\|Z\|\) is the Frobenius norm of \(Z\).

\texttt{scal} = Nag\_FanLinVanDooren
Fan, Lin and Van Dooren scaling.

\texttt{scal} = Nag\_TropicalLargest
Tropical scaling with largest root.

\texttt{scal} = Nag\_TropicalSmallest
Tropical scaling with smallest root.

\textit{Constraint:} \texttt{scal} = Nag\_NoScale, Nag\_CondFanLinVanDooren, Nag\_FanLinVanDooren, Nag\_TropicalLargest or Nag\_TropicalSmallest.

2: \texttt{jobvl} – Nag\_LeftVecsType \hspace{1cm} \textit{Input}

\textit{On entry:} if \texttt{jobvl} = Nag\_NotLeftVecs, do not compute left eigenvectors.

If \texttt{jobvl} = Nag\_LeftVecs, compute the left eigenvectors.

I f  \texttt{sense} = Nag\_CondOnly, Nag\_BackErrLeft, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth, \texttt{jobvl} must be set to Nag\_LeftVecs.

\textit{Constraint:} \texttt{jobvl} = Nag\_NotLeftVecs or Nag\_LeftVecs.

3: \texttt{jobvr} – Nag\_RightVecsType \hspace{1cm} \textit{Input}

\textit{On entry:} if \texttt{jobvr} = Nag\_NotRightVecs, do not compute right eigenvectors.

If \texttt{jobvr} = Nag\_RightVecs, compute the right eigenvectors.

I f  \texttt{sense} = Nag\_CondOnly, Nag\_BackErrRight, Nag\_BackErrBoth, Nag\_CondBackErrLeft, Nag\_CondBackErrRight or Nag\_CondBackErrBoth, \texttt{jobvr} must be set to Nag\_RightVecs.

\textit{Constraint:} \texttt{jobvr} = Nag\_NotRightVecs or Nag\_RightVecs.

4: \texttt{sense} – Nag\_CondErrType \hspace{1cm} \textit{Input}

\textit{On entry:} determines whether, or not, condition numbers and backward errors are computed.

\texttt{sense} = Nag\_NoCondBackErr
Do not compute condition numbers, or backward errors.

\texttt{sense} = Nag\_CondOnly
Just compute condition numbers for the eigenvalues.

\texttt{sense} = Nag\_BackErrLeft
Just compute backward errors for the left eigenpairs.

\texttt{sense} = Nag\_BackErrRight
Just compute backward errors for the right eigenpairs.

\texttt{sense} = Nag\_BackErrBoth
Compute backward errors for the left and right eigenpairs.

\texttt{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 $n /C2 \max(A, \|B\|, \|C\|) /C2$ machine precision
   is used in place of tol, where $\|Z\|$ is the Frobenius norm of the (scaled) matrix $Z$ and 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] – Complex
   Input/Output
   Note: the dimension, dim, of the array a must be at least pda $\times$ n.
   The $(i, j)$th element of the matrix $A$ is stored in $a[(j - 1) \times pda + i - 1]$.
   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: pda $\geq n$.

9: b[dim] – Complex
   Input/Output
   Note: the dimension, dim, of the array b must be at least pdb $\times$ n.
   The $(i, j)$th element of the matrix $B$ is stored in $b[(j - 1) \times pdb + i - 1]$.
   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: pdb $\geq n$.

11: c[dim] – Complex
    Input/Output
    Note: the dimension, dim, of the array c must be at least pdc $\times$ n.
    The $(i, j)$th element of the matrix $C$ is stored in $c[(j - 1) \times pdc + i - 1]$.
    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
     \textit{Input}
     \textit{On entry:} the stride separating matrix row elements in the array \(e\).
     \textit{Constraint:} \(pdc \geq n\).

13:  alpha\([2 \times n]\) – Complex \(\textit{Output}\)
     \textit{On exit:} \(\alpha_{j-1}\), for \(j = 1, 2, \ldots, 2n\), contains the first part of the \(j\)th eigenvalue pair \((\alpha_j, \beta_j)\) of the quadratic eigenvalue problem.

14:  beta\([2 \times n]\) – Complex \(\textit{Output}\)
     \textit{On exit:} \(\beta_{j-1}\), 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. Although \(\beta\) is declared complex, it is actually real and non-negative. Infinite eigenvalues have \(\beta_j\) set to zero.

15:  vl\([\text{dim}]\) – Complex \(\textit{Output}\)
     \textit{Note:} the dimension, \(\text{dim}\), of the array \(vl\) must be at least \(2 \times n\) when \(\text{jobvl} = \text{Nag LeftVecs}\).

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

     \textit{On exit:} if \(\text{jobvl} = \text{Nag LeftVecs}\), 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 \(\text{jobvl} = \text{Nag NotLeftVecs}\), \(vl\) is not referenced and may be \texttt{NULL}.

16:  pdvl – Integer \(\textit{Input}\)
     \textit{On entry:} the stride separating matrix row elements in the array \(vl\).
     \textit{Constraint:} \(pdvl \geq n\).

17:  vr\([\text{dim}]\) – Complex \(\textit{Output}\)
     \textit{Note:} the dimension, \(\text{dim}\), of the array \(vr\) must be at least \(2 \times n\) when \(\text{jobvr} = \text{Nag RightVecs}\).

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

     \textit{On exit:} if \(\text{jobvr} = \text{Nag RightVecs}\), 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 \(\text{jobvr} = \text{Nag NotRightVecs}\), \(vr\) is not referenced and may be \texttt{NULL}.

18:  pdvr – Integer \(\textit{Input}\)
     \textit{On entry:} the stride separating matrix row elements in the array \(vr\).
     \textit{Constraint:} \(pdvr \geq n\).

19:  s\([\text{dim}]\) – double \(\textit{Output}\)
     \textit{Note:} the dimension, \(\text{dim}\), of the array \(s\) must be at least \(2 \times n\) when \(\text{sense} = \text{Nag CondOnly, Nag CondBackErrLeft, Nag CondBackErrRight or Nag CondBackErrBoth}\).

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

     \textit{On exit:} if \(\text{sense} = \text{Nag CondOnly, Nag CondBackErrLeft, Nag CondBackErrRight or Nag CondBackErrBoth}\), \(s[j - 1]\) 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 (\texttt{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` = 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 `fail` = NE_ITERATIONS_QZ, nag_zgges (f08xnc) has flagged that `iwarn` eigenvalues are invalid.

If `fail` = NE_ITERATIONS_QZ, nag_zggev (f08wnc) has flagged that `iwarn` eigenvalues are invalid.

23: **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.

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, \( pdc = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pdc \geq n \).

On entry, \( pdvl = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \) and \( jobvl = \langle \text{value} \rangle \).
Constraint: when \( jobvl = \text{Nag\_LeftVecs} \), \( pdvl \geq n \).

On entry, \( pdvr = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \) and \( jobvr = \langle \text{value} \rangle \).
Constraint: when \( jobvr = \text{Nag\_RightVecs} \), \( pdvr \geq n \).

**NE\_BAD\_PARAM**

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

**NE\_INT**

On entry, \( n = \langle \text{value} \rangle \).
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 the Essential Introduction for further information.

**NE\_INVALID\_VALUE**

On entry, \( sense = \langle \text{value} \rangle \) and \( jobvl = \langle \text{value} \rangle \).
Constraint: when \( jobvl = \text{Nag\_NotLeftVecs} \), \( sense = \text{Nag\_NoCondBackErr} \) or \( \text{Nag\_BackErrRight} \),
when \( jobvl = \text{Nag\_LeftVecs} \), \( sense = \text{Nag\_CondOnly} \), \( \text{Nag\_BackErrLeft} \), \( \text{Nag\_BackErrBoth} \),
\( \text{Nag\_CondBackErrLeft} \), \( \text{Nag\_CondBackErrRight} \) or \( \text{Nag\_CondBackErrBoth} \).

On entry, \( sense = \langle \text{value} \rangle \) and \( jobvr = \langle \text{value} \rangle \).
Constraint: when \( jobvr = \text{Nag\_NotRightVecs} \), \( sense = \text{Nag\_NoCondBackErr} \) or \( \text{Nag\_BackErrLeft} \),
when \( jobvr = \text{Nag\_RightVecs} \), \( sense = \text{Nag\_CondOnly} \), \( \text{Nag\_BackErrRight} \), \( \text{Nag\_BackErrBoth} \),
\( \text{Nag\_CondBackErrLeft} \), \( \text{Nag\_CondBackErrRight} \) or \( \text{Nag\_CondBackErrBoth} \).

**NE\_ITERATIONS\_QZ**

The \( QZ \) iteration failed in \text{nag\_zggev} (f08wnc).
\( iwarn \) returns the value of \( \text{info} \) returned by \text{nag\_zggev} (f08wnc). This failure is unlikely to happen, but if it does, please contact NAG.

The \( QZ \) iteration failed in \text{nag\_zgges} (f08xnc).
\( iwarn \) returns the value of \( \text{info} \) returned by \text{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 the Essential Introduction 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. */
* * Copyright 2014 Numerical Algorithms Group.
* * Mark 24, 2013.
*/

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

#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;

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

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

/* Complex scalar and array declarations */
Complex *a = 0, *alpha = 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];

/* Initialise 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("%*[^
"]");
#else
    scanf("%*[^
"]");
#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, 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)))
{
    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
typedef struct COMPLEX {
    double re, im;
} COMPLEX;

#include <stdio.h>
#define scanf_s scanf

#define _WIN32

#define _WIN32
#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

/* 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

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" fail.message);
    exit_status = -1;
    goto END;
}
else if (iwarn!=0)
{
    printf("Warning from nag_eigen_complex_gen_quad (f02jqc).\n"
    printf(" iwarn = %"NAG_IFMT"
    }

/* Infinity */
inf = X02ALC;

/* Display eigenvalues * /
for (j = 0; j < 2*n; j++)
{
    rbetaj = beta[j].re;
    if (rbetaj >= 1.0)
    {
        e[j].re = alpha[j].re/rbetaj;
        e[j].im = alpha[j].im/rbetaj;
    }
    else
    {
        tmp = inf*rbetaj;
        if ((fabs(alpha[j].re)<tmp) && (fabs(alpha[j].im)<tmp))
        {
            // Code for handling infinite eigenvalues
        }
    }

END:
e[j].re = alpha[j].re/rbeta[j];
e[j].im = alpha[j].im/rbeta[j];
}
else
{
e[j].re = inf;
e[j].im = 0.0;
}
if (fabs(e[j].re)<inf)
{
    printf("Eigenvalue(%3"NAG_IFMT") = (%11.4e, %11.4e)\n",j+1,
        COMPLEX(e[j]));
}
else
{
    printf("Eigenvalue(%3"NAG_IFMT") is infinite\n",j+1);
}

if (jobvr == Nag_RightVecs)
{
    printf("\n");
    printf("Right Eigenvectors\n");
    for (j = 0; j < 2*n; j += 3)
    {
        printf(" Eigenvector "%NAG_IFMT",j + 1);
        if (j < 2*n-1)
            printf(" Eigenvector "%NAG_IFMT",j + 2);
        if (j < 2*n-2)
            printf(" Eigenvector "%NAG_IFMT",j + 3);
        printf("\n");
        for (i = 0; i < n; i++)
        {
            printf(" (%11.4e,%11.4e)\n", COMPLEX(vr[j*pdvr+i]));
            if (j < 2*n-1)
                printf(" (%11.4e,%11.4e)\n", COMPLEX(vr[(j+1)*pdvr+i]));
            if (j < 2*n-2)
                printf(" (%11.4e,%11.4e)\n", COMPLEX(vr[(j+2)*pdvr+i]));
            printf("\n");
        }
    }
}

if (jobvl == Nag_LeftVecs)
{
    printf("\n");
    printf("Left Eigenvectors\n");
    for (j = 0; j < 2*n; j += 3)
    {
        printf(" Eigenvector "%NAG_IFMT",j + 1);
        if (j < 2*n-1)
            printf(" Eigenvector "%NAG_IFMT",j + 2);
        if (j < 2*n-2)
            printf(" Eigenvector "%NAG_IFMT",j + 3);
        printf("\n");
        for (i = 0; i < n; i++)
        {
            printf(" (%11.4e,%11.4e)\n", COMPLEX(vl[j*pdvl+i]));
            if (j < 2*n-1)
                printf(" (%11.4e,%11.4e)\n", COMPLEX(vl[(j+1)*pdvl+i]));
            if (j < 2*n-2)
                printf(" (%11.4e,%11.4e)\n", COMPLEX(vl[(j+2)*pdvl+i]));
            printf("\n");
        }
    }

    /* 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("%11.4e\n", s[j]);
}

if (sense==Nag_BackErrRight || sense==Nag_BackErrBoth ||
    sense==Nag_CondBackErrRight || sense==Nag_CondBackErrBoth)
{
    printf("\n");
    printf("Backward errors for eigenvalues and right eigenvectors\n");
    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)
{
    printf("\n");
    printf("Backward errors for eigenvalues and left eigenvectors\n");
    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(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);

return(exit_status);

10.2 Program Data

f02 – Eigenvalues and Eigenvectors
f02jqc

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 6.0
0.0 6.0
3.0 3.0
2.0 2.0
1.0 1.0
1.0 0.0
2.0 0.0
3.0 0.0 : a

0.0 4.0
0.0 2.0
0.0 4.0
2.0 2.0
1.0 1.0
3.0 3.0
1.0 3.0
3.0 0.0
1.0 0.0 : b

0.0 4.0
0.0 2.0
0.0 2.0
1.0 1.0
3.0 3.0
2.0 0.0
2.0 0.0
1.0 0.0
2.0 0.0 : c
10.3 Program Results

**nag_eigen_complex_gen_quad (f02jqc) Example Program Results**

Eigenvalue( 1) = (-1.9256e+00,  1.9256e+00)
Eigenvalue( 2) = (-6.9748e-01,  -1.0532e-01)
Eigenvalue( 3) = ( 1.0532e-01,   6.9748e-01)
Eigenvalue( 4) = (-4.9622e-02,  -5.7288e-01)
Eigenvalue( 5) = ( 5.7288e-01,  4.9622e-02)
Eigenvalue( 6) = ( 3.9455e-01,  -3.9455e-01)

**Right Eigenvectors**

<table>
<thead>
<tr>
<th>Eigenvector 1</th>
<th>Eigenvector 2</th>
<th>Eigenvector 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-2.1083e-01, 7.4532e-17)</td>
<td>( 3.7508e-01, 1.8772e-01)</td>
<td>( 3.7508e-01, -1.8772e-01)</td>
</tr>
<tr>
<td>( 7.6950e-01, 0.0000e+00)</td>
<td>( 5.0200e-01, 2.4329e-01)</td>
<td>( 5.0200e-01, -2.4329e-01)</td>
</tr>
<tr>
<td>(-6.0285e-01, 5.1306e-16)</td>
<td>( 7.1616e-01, 0.0000e+00)</td>
<td>( 7.1616e-01, 0.0000e+00)</td>
</tr>
</tbody>
</table>

**Eigenvector 4**

<table>
<thead>
<tr>
<th>Eigenvector 5</th>
<th>Eigenvector 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-6.5928e-01, -4.2432e-02)</td>
<td>(-6.5928e-01, 4.2432e-02)</td>
</tr>
<tr>
<td>( 3.0158e-02, 1.9723e-02)</td>
<td>( 3.0158e-02, 1.9723e-02)</td>
</tr>
<tr>
<td>( 7.4984e-01, 0.0000e+00)</td>
<td>( 7.4984e-01, 0.0000e+00)</td>
</tr>
</tbody>
</table>

**Left Eigenvectors**

<table>
<thead>
<tr>
<th>Eigenvector 1</th>
<th>Eigenvector 2</th>
<th>Eigenvector 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1.0520e-01, 9.0132e-17)</td>
<td>( 7.8162e-01, 0.0000e+00)</td>
<td>( 7.8162e-01, 0.0000e+00)</td>
</tr>
<tr>
<td>( 7.3813e-01, 0.0000e+00)</td>
<td>( 5.0745e-01, 1.3518e-01)</td>
<td>( 5.0745e-01, -1.3518e-01)</td>
</tr>
<tr>
<td>(-6.6640e-01, -2.7733e-17)</td>
<td>( 3.2017e-01, 1.0381e-01)</td>
<td>( 3.2017e-01, -1.0381e-01)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Eigenvector 4</th>
<th>Eigenvector 5</th>
<th>Eigenvector 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 8.0788e-01, 0.0000e+00)</td>
<td>( 8.0788e-01, 0.0000e+00)</td>
<td>( 3.5830e-02, 1.4065e-16)</td>
</tr>
<tr>
<td>(-1.1236e-01, 3.1416e-02)</td>
<td>(-1.1236e-01, -3.1416e-02)</td>
<td>( 7.0720e-01, 0.0000e+00)</td>
</tr>
<tr>
<td>(-5.7041e-01, 9.1343e-02)</td>
<td>(-5.7041e-01, 9.1343e-02)</td>
<td>(-7.0611e-01, -2.6757e-16)</td>
</tr>
</tbody>
</table>

**Eigenvalue Condition numbers**

3.0717e+00
6.6202e-01
6.6202e-01
2.3848e+00
2.3848e+00
1.7625e+00

**Backward errors for eigenvalues and right eigenvectors**

3.0437e-16
2.6464e-16
2.0389e-16
1.6246e-16
1.9837e-16
3.0446e-16

**Backward errors for eigenvalues and left eigenvectors**

2.5244e-16
2.9994e-16
1.3515e-16
1.6514e-16
3.1230e-16
3.5800e-16