NAG Library Function Document

nag_zggevx (f08wpc)

1 Purpose

nag_zggevx (f08wpc) computes for a pair of \( n \) by \( n \) complex nonsymmetric matrices \((A,B)\) the generalized eigenvalues and, optionally, the left and/or right generalized eigenvectors using the \( QZ \) algorithm. Optionally it also computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors, reciprocal condition numbers for the eigenvalues, and reciprocal condition numbers for the right eigenvectors.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_zggevx (Nag_OrderType order, Nag_BalanceType balanc,
                 Nag_LeftVecsType jobvl, Nag_RightVecsType jobvr, Nag_RCondType sense,
                 Integer n, Complex a[], Integer pda, Complex b[[], Integer pdb,
                 Complex alpha[], Complex beta[], Complex vl[], Integer pdvl,
                 Complex vr[], Integer pdvr, Integer *ilo, Integer *ihi, double lscale[],
                 double rscale[], double *abnrm, double *bbnrm, double rconde[],
                 double rcondv[], NagError *fail)
```

3 Description

A generalized eigenvalue for a pair of matrices \((A,B)\) is a scalar \( \lambda \) or a ratio \( \alpha/\beta = \lambda \), such that \( A - \lambda B \) is singular. It is usually represented as the pair \((\alpha,\beta)\), as there is a reasonable interpretation for \( \beta = 0 \), and even for both being zero.

The right generalized eigenvector \( v_j \) corresponding to the generalized eigenvalue \( \lambda_j \) of \((A,B)\) satisfies

\[
Av_j = \lambda_j Bv_j.
\]

The left generalized eigenvector \( u_j \) corresponding to the generalized eigenvalue \( \lambda_j \) of \((A,B)\) satisfies

\[
u_j^H A = \lambda_j u_j^H B,
\]

where \( u_j^H \) is the conjugate-transpose of \( u_j \).

All the eigenvalues and, if required, all the eigenvectors of the complex generalized eigenproblem \( Ax = \lambda Bx \), where \( A \) and \( B \) are complex, square matrices, are determined using the \( QZ \) algorithm. The complex \( QZ \) algorithm consists of three stages:

1. \( A \) is reduced to upper Hessenberg form (with real, non-negative subdiagonal elements) and at the same time \( B \) is reduced to upper triangular form.
2. \( A \) is further reduced to triangular form while the triangular form of \( B \) is maintained and the diagonal elements of \( B \) are made real and non-negative. This is the generalized Schur form of the pair \((A,B)\).

This function does not actually produce the eigenvalues \( \lambda_j \), but instead returns \( \alpha_j \) and \( \beta_j \) such that

\[
\lambda_j = \alpha_j/\beta_j, \quad j = 1,2,\ldots,n.
\]

The division by \( \beta_j \) becomes your responsibility, since \( \beta_j \) may be zero, indicating an infinite eigenvalue.

3. If the eigenvectors are required they are obtained from the triangular matrices and then transformed back into the original coordinate system.
For details of the balancing option, see Section 3 in nag_zggbal (f08wvc).

4 References

5 Arguments
1: order – Nag_OrderType
   
   On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: balanc – Nag_BalanceType
   
   On entry: specifies the balance option to be performed.

   balanc = Nag_NoBalancing
   Do not diagonally scale or permute.

   balanc = Nag_BalancePermute
   Permute only.

   balanc = Nag_BalanceScale
   Scale only.

   balanc = Nag_BalanceBoth
   Both permute and scale.

   Computed reciprocal condition numbers will be for the matrices after permuting and/or balancing. Permuting does not change condition numbers (in exact arithmetic), but balancing does. In the absence of other information, balanc = Nag_BalanceBoth is recommended.

   Constraint: balanc = Nag_NoBalancing, Nag_BalancePermute, Nag_BalanceScale or Nag_BalanceBoth.

3: jobvl – Nag_LeftVecsType
   
   On entry: if jobvl = Nag_NotLeftVecs, do not compute the left generalized eigenvectors.

   If jobvl = Nag_LeftVecs, compute the left generalized eigenvectors.

   Constraint: jobvl = Nag_NotLeftVecs or Nag_LeftVecs.

4: jobvr – Nag_RightVecsType
   
   On entry: if jobvr = Nag_NotRightVecs, do not compute the right generalized eigenvectors.

   If jobvr = Nag_RightVecs, compute the right generalized eigenvectors.

   Constraint: jobvr = Nag_NotRightVecs or Nag_RightVecs.
5: sense – Nag_RCondType

On entry: determines which reciprocal condition numbers are computed.

sense = Nag_NotRCond
None are computed.
sense = Nag_RCondEigVals
Computed for eigenvalues only.
sense = Nag_RCondEigVecs
Computed for eigenvectors only.
sense = Nag_RCondBoth
Computed for eigenvalues and eigenvectors.

Constraint: sense = Nag_NotRCond, Nag_RCondEigVals, Nag_RCondEigVecs or
Nag_RCondBoth.

6: n – Integer

On entry: n, the order of the matrices A and B.

Constraint: n ≥ 0.

7: a[dim] – Complex

Note: the dimension, dim, of the array a must be at least max(1, pda × n).

Where A(i, j) appears in this document, it refers to the array element

a[(j - 1) × pda + i - 1] when order = Nag_ColMajor;
a[(i - 1) × pda + j - 1] when order = Nag_RowMajor.

On entry: the matrix A in the pair (A, B).

On exit: a has been overwritten. If jobvl = Nag_LeftVecs or jobvr = Nag_RightVecs or both, then
A contains the first part of the Schur form of the ‘balanced’ versions of the input A and B.

8: pda – Integer

On entry: the stride separating row or column elements (depending on the value of order) in the
array a.

Constraint: pda ≥ max(1, n).

9: b[dim] – Complex

Note: the dimension, dim, of the array b must be at least max(1, pdb × n).

Where B(i, j) appears in this document, it refers to the array element

b[(j - 1) × pdb + i - 1] when order = Nag_ColMajor;
b[(i - 1) × pdb + j - 1] when order = Nag_RowMajor.

On entry: the matrix B in the pair (A, B).

On exit: b has been overwritten.

10: pdb – Integer

On entry: the stride separating row or column elements (depending on the value of order) in the
array b.

Constraint: pdb ≥ max(1, n).

11: alpha[n] – Complex

On exit: see the description of beta.
On exit: \( \alpha[j - 1]/\beta[j - 1] \), for \( j = 1, 2, \ldots, n \), will be the generalized eigenvalues.

Note: the quotients \( \alpha[j - 1]/\beta[j - 1] \) may easily overflow or underflow, and \( \beta[j - 1] \) may even be zero. Thus, you should avoid naively computing the ratio \( \alpha_j/\beta_j \). However, \( \max(|\alpha_j|) \) will always be less than and usually comparable with \( \|a\|_2 \) in magnitude, and \( \max(|\beta_j|) \) will always be less than and usually comparable with \( \|b\|_2 \).

On exit: if \( \text{jobvl} = \text{Nag Left Vectors} \), the left generalized eigenvectors \( u_j \) are stored one after another in the columns of \( vl \), in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have real part 
\[ \text{part } u_j = 1. \]
If \( \text{jobvl} = \text{Nag Not Left Vectors} \), \( vl \) is not referenced.

On entry: the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( vl \).

Constraints:
\[
\begin{align*}
\text{if } \text{jobvl} = \text{Nag Left Vectors}, & \quad \text{pdvl} \geq \max(1, n); \\
& \quad \text{otherwise } \text{pdvl} \geq 1.
\end{align*}
\]

On exit: the dimension, \( dim \), of the array \( vl \) must be at least
\[
\max(1, \text{pdvl} \times n) \text{ when } \text{jobvl} = \text{Nag Left Vectors}; \\
1 \text{ otherwise}.
\]

The \((i,j)\)th element of the matrix is stored in
\[
vl[(j - 1) \times \text{pdvl} + i - 1] \text{ when } \text{order} = \text{Nag Col Major}; \\
v[[(i - 1) \times \text{pdvl} + j - 1] \text{ when } \text{order} = \text{Nag Row Major}.
\]

On exit: if \( \text{jobvr} = \text{Nag Right Vectors} \), the right generalized eigenvectors \( v_j \) are stored one after another in the columns of \( vr \), in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have [real part] + [imag. part] = 1.

If \( \text{jobvr} = \text{Nag Not Right Vectors} \), \( vr \) is not referenced.

On entry: the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( vr \).

Constraints:
\[
\begin{align*}
\text{if } \text{jobvr} = \text{Nag Right Vectors}, & \quad \text{pdvr} \geq \max(1, n); \\
& \quad \text{otherwise } \text{pdvr} \geq 1.
\end{align*}
\]
On exit: \( \text{ilo} \) and \( \text{ihi} \) are integer values such that \( A(i,j) = 0 \) and \( B(i,j) = 0 \) if \( i > j \) and \( j = 1,2,\ldots,\text{ilo} - 1 \) or \( i = \text{ihi} + 1,\ldots,\text{n} \).

If \( \text{balanc} = \text{Nag\_NoBalancing} \) or \( \text{Nag\_BalanceScale} \), \( \text{ilo} = 1 \) and \( \text{ihi} = \text{n} \).

- **ilo** – Integer * Output
  On exit: details of the permutations and scaling factors applied to the left side of \( A \) and \( B \).
  If \( pl_j \) is the index of the row interchanged with row \( j \), and \( dl_j \) is the scaling factor applied to row \( j \), then:
  \[
  \text{lscale}[j-1] = pl_j, \text{ for } j = 1,2,\ldots,\text{ilo} - 1; \\
  \text{lscale} = dl_j, \text{ for } j = \text{ilo},\ldots,\text{ihi}; \\
  \text{lscale} = pl_j, \text{ for } j = \text{ihi} + 1,\ldots,\text{n}.
  \]
  The order in which the interchanges are made is \( \text{n} \) to \( \text{ihi} + 1 \), then 1 to \( \text{ilo} - 1 \).

- **rside[n]** – double Output
  On exit: details of the permutations and scaling factors applied to the right side of \( A \) and \( B \).
  If \( pr_j \) is the index of the column interchanged with column \( j \), and \( dr_j \) is the scaling factor applied to column \( j \), then:
  \[
  \text{rscale}[j-1] = pr_j, \text{ for } j = 1,2,\ldots,\text{ilo} - 1; \\
  \text{rscale} = dr_j, \text{ for } j = \text{ilo},\ldots,\text{ihi}; \\
  \text{rscale} = pr_j, \text{ for } j = \text{ihi} + 1,\ldots,\text{n}.
  \]
  The order in which the interchanges are made is \( \text{n} \) to \( \text{ihi} + 1 \), then 1 to \( \text{ilo} - 1 \).

- **abnorm** – double * Output
  On exit: the 1-norm of the balanced matrix \( A \).

- **bbnorm** – double * Output
  On exit: the 1-norm of the balanced matrix \( B \).

- **rconde[dim]** – double Output
  Note: the dimension, \( \text{dim} \), of the array \( \text{rconde} \) must be at least \( \max(1,\text{n}) \).
  On exit: if \( \text{sense} = \text{Nag\_RCondEigVals} \) or \( \text{Nag\_RCondBoth} \), the reciprocal condition numbers of the eigenvalues, stored in consecutive elements of the array.
  If \( \text{sense} = \text{Nag\_NotRCond} \) or \( \text{Nag\_RCondEigVecs} \), \( \text{rconde} \) is not referenced.

- **rcondv[dim]** – double Output
  Note: the dimension, \( \text{dim} \), of the array \( \text{rcondv} \) must be at least \( \max(1,\text{n}) \).
  On exit: if \( \text{sense} = \text{Nag\_RCondEigVecs} \) or \( \text{Nag\_RCondBoth} \), the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array.
  If \( \text{sense} = \text{Nag\_NotRCond} \) or \( \text{Nag\_RCondEigVals} \), \( \text{rcondv} \) is not referenced.

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

NE_EIGENVECTORS
A failure occurred in nag_dtgevc (f08ykc) while computing generalized eigenvectors.

NE_ENUM_INT_2
On entry, \( \text{jobvl} = \langle \text{value} \rangle \), \( \text{pdvl} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: if \( \text{jobvl} = \text{Nag\_LeftVecs} \), \( \text{pdvl} \geq \max(1, n) \);
otherwise \( \text{pdvl} \geq 1 \).
On entry, \( \text{jobvr} = \langle \text{value} \rangle \), \( \text{pdvr} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: if \( \text{jobvr} = \text{Nag\_RightVecs} \), \( \text{pdvr} \geq \max(1, n) \);
otherwise \( \text{pdvr} \geq 1 \).

NE_INT
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).
On entry, \( \text{pda} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} > 0 \).
On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).
On entry, \( \text{pdvl} = \langle \text{value} \rangle \).
Constraint: \( \text{pdvl} > 0 \).
On entry, \( \text{pdvr} = \langle \text{value} \rangle \).
Constraint: \( \text{pdvr} > 0 \).

NE_INT_2
On entry, \( \text{pda} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pda} \geq \max(1, n) \).
On entry, \( \text{pdb} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} \geq \max(1, n) \).

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_ITERATION_QZ
The QZ iteration failed. No eigenvectors have been calculated but \textbf{alpha} and \textbf{beta} should be correct from element \( \langle \text{value} \rangle \).
The QZ iteration failed with an unexpected error, please contact NAG.
7 Accuracy

The computed eigenvalues and eigenvectors are exact for a nearby matrices \((A + E)\) and \((B + F)\), where

\[
\| (E, F) \|_F = O(\epsilon) \| (A, B) \|_F,
\]

and \(\epsilon\) is the machine precision.

An approximate error bound on the chordal distance between the \(i\)th computed generalized eigenvalue \(w\) and the corresponding exact eigenvalue \(\lambda\) is

\[
\epsilon \times \| abnrm, bbnrm \|_2 / rconde[i - 1].
\]

An approximate error bound for the angle between the \(i\)th computed eigenvector \(u_j\) or \(v_j\) is given by

\[
\epsilon \times \| abnrm, bbnrm \|_2 / rcondv[i - 1].
\]

For further explanation of the reciprocal condition numbers \(rconde\) and \(rcondv\), see Section 4.11 of Anderson et al. (1999).

**Note:** interpretation of results obtained with the QZ algorithm often requires a clear understanding of the effects of small changes in the original data. These effects are reviewed in Wilkinson (1979), in relation to the significance of small values of \(\alpha_j\) and \(\beta_j\). It should be noted that if \(\alpha_j\) and \(\beta_j\) are both small for any \(j\), it may be that no reliance can be placed on any of the computed eigenvalues \(\lambda_k = \alpha_k / \beta_k\). You are recommended to study Wilkinson (1979) and, if in difficulty, to seek expert advice on determining the sensitivity of the eigenvalues to perturbations in the data.

8 Parallelism and Performance

\texttt{nag_zggevx (f08wpc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag_zggevx (f08wpc)} 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

The total number of floating-point operations is proportional to \(n^3\).

The real analogue of this function is \texttt{nag_dggevx (f08wbc)}.

10 Example

This example finds all the eigenvalues and right eigenvectors of the matrix pair \((A, B)\), where

\[
A = \begin{pmatrix}
-21.10 & 22.50i & 53.50 - 50.50i & -34.50 + 127.50i & 7.50 + 0.50i \\
-0.46 & -7.78i & -3.50 - 37.50i & -15.50 + 58.50i & -10.50 - 1.50i \\
4.30 & 5.50i & 39.70 - 17.10i & -68.50 + 12.50i & -7.50 - 3.50i \\
5.50 & 4.40i & 14.40 + 43.30i & -32.50 - 46.00i & -19.00 - 32.50i \\
\end{pmatrix}
\]

and
\[
B = \begin{pmatrix}
1.00 - 5.00i & 1.60 + 1.20i & -3.00 + 0.00i & 0.00 - 1.00i \\
0.80 - 0.60i & 3.00 - 5.00i & -4.00 + 3.00i & -2.40 - 3.20i \\
1.00 + 0.00i & 2.40 + 1.80i & -4.00 - 5.00i & 0.00 - 3.00i \\
0.00 + 1.00i & -1.80 + 2.40i & 0.00 - 4.00i & 4.00 - 5.00i
\end{pmatrix},
\]

together with estimates of the condition number and forward error bounds for each eigenvalue and eigenvector. The option to balance the matrix pair is used.

10.1 Program Text

/* nag_zggevx (f08wpc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagx02.h>

int main(void)
{
    /* Scalars */
    Complex z;
    double abnorm, abnrm, bbnrm, eps, small, tol;
    Integer i, ihi, ilo, j, n, pda, pdb, pdvl, pdvr;
    Integer exit_status = 0;
    /* Arrays */
    Complex *a = 0, *alpha = 0, *b = 0, *beta = 0;
    double *lscale = 0, *rconde = 0, *rcondv = 0, *rscale = 0;
    char nag_enum_arg[40];
    NagError fail;
    Nag_OrderType order;
    Nag_LeftVecsType jobvl;
    Nag_RightVecsType jobvr;
    Nag_RCondType sense;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J-1)*pda + I - 1]
    #define B(I, J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda + J - 1]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    printf("nag_zggevx (f08wpc) Example Program Results\n");

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

    /* */
```c
#else
    scanf("%"NAG_IFMT"%[\n]\", &n);
#endif
    if (n < 0)
    {
        printf("Invalid n\n");
        exit_status = 1;
        goto END;
    }
#endif
    scanf("%39s%[\n]\", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
    * Converts NAG enum member name to value
    */
    jobvl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#endif
    scanf("%39s%[\n]\", nag_enum_arg);
#else
    scanf("%39s%[\n]\", nag_enum_arg);
#endif
    jobvr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#endif
    scanf("%39s%[\n]\", nag_enum_arg);
#else
    scanf("%39s%[\n]\", nag_enum_arg);
#endif
    sense = (Nag_RCondType) nag_enum_name_to_value(nag_enum_arg);

    pda = n;
    pdb = n;
    pdvl = (jobvl==Nag_LeftVecs?n:1);
    pdvr = (jobvr==Nag_RightVecs?n:1);

    /* Allocate memory */
    if (!(a = NAG_ALLOC(n*n, Complex)) ||
        !(b = NAG_ALLOC(n*n, Complex)) ||
        !(alpha = NAG_ALLOC(n, Complex)) ||
        !(beta = NAG_ALLOC(n, Complex)) ||
        !(vl = NAG_ALLOC(pdvl*pdvl, Complex)) ||
        !(lr = NAG_ALLOC(pdvl*pdvl, Complex)) ||
        !(lscale = NAG_ALLOC(n, double)) ||
        !(rconde = NAG_ALLOC(n, double)) ||
        !(rcondv = NAG_ALLOC(n, double)) ||
        !(rscale = NAG_ALLOC(n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read in the matrices A and B */
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j)
            scanf("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j)
            scanf("( %lf , %lf )", &B(i, j).re, &B(i, j).im);
```
/* Solve the generalized eigenvalue problem using nag_zggevx (f08wpc). */

nag_zggevx(order, Nag_BalanceBoth, jobvl, jobvr, sense, n, a, pda, b, pdb,
  alpha, beta, vl, pdvl, vr, pdvr, &ilo, &ihi, lscale, rscale,
  &abnrm, &bbnrm, rconde, rcondv, &fail);

if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zggevx (f08wpc).\n", fail.message);
  exit_status = 1;
  goto END;
}

/* nag_real_safe_small_number (x02amc), nag_machine_precision (x02ajc) */

eps = nag_machine_precision;
small = nag_real_safe_small_number;
if (abnrm == 0.0)
  abnorm = ABS(bbnrm);
else if (bbnrm == 0.0)
  abnorm = ABS(abnrm);
else if (ABS(abnrm) >= ABS(bbnrm))
  abnorm = ABS(abnrm)*sqrt(1.0+(bbnrm/abnrm)*(bbnrm/abnrm));
else
  abnorm = ABS(bbnrm)*sqrt(1.0+(abnrm/bbnrm)*(abnrm/bbnrm));

tol = eps * abnorm;

/* Print out eigenvalues and associated condition number and bounds */

if (sense!=Nag_NotRCond)
  printf("R = Reciprocal condition number, E = Error bound\n\n");

if (sense==Nag_RCondEigVals || sense==Nag_RCondBoth)
  printf("15s\%s", "R", "E");

for (j = 0; j < n; ++j)
{
  /* Print out information on the jth eigenvalue */
  if (nag_complex_abs(alpha[j])*small >= nag_complex_abs(beta[j]))
  
    printf("%2NAG_FMT\n", j+1);
  printf("%9.4f, %9.4f", alpha[j].re, alpha[j].im, beta[j].re, beta[j].im);

  else
  
    z = nag_complex_divide(alpha[j], beta[j]);

    printf("%2NAG_FMT\n", j+1, z.re, z.im);

    if (sense==Nag_RCondEigVals || sense==Nag_RCondBoth)
      printf("%10.1e", rconde[j]);
    if (rconde[j] > 0.0)
      printf("%9.1e", tol/rconde[j]);
    else
      printf(" infinite");
  
    printf("\n");
}

/* Print out information on the eigenvectors as requested */

if (jobvl==Nag_LeftVecs) {
  printf("\n");
  /* Print left eigenvectors using nag_gen_complx_mat_print (x04dac). */
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NotUnitDiag, n,
if (jobvr == Nag_RightVecs && fail.code == NE_NOERROR) {
    printf("\n");
    /* Print rightt eigenvectors using nag_gen_complx_mat_print (x04dac). */
    nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, vr, pdvr, " Right eigenvectors (columns)", 0, &fail);
}

if (fail.code != NE_NOERROR)
    {
    printf("Error from nag_gen_complx_mat_print (x04dac). \n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

if (sense == Nag_RCondEigVecs || sense == Nag_RCondBoth)
    {
    printf("%2s","R");
    for (j = 0; j < n; ++j) printf(" %8.1e", rcondv[j]);
    printf("%8.1e", tol/rcondv[j]);
    for (j = 0; j < n; ++j)
    {
        if (rcondv[j] > 0.0)
            printf(" %8.1e", tol/rcondv[j]);
        else
            printf(" infinite");
    }
    printf("\n");
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(alpha);
NAG_FREE(beta);
NAG_FREE(vl);
NAG_FREE(vr);
NAG_FREE(lscale);
NAG_FREE(rconde);
NAG_FREE(rcondv);
NAG_FREE(rscale);
return exit_status;
}

10.2 Program Data

nag_zggevx (f08wpc) Example Program Data

4 : n

Nag_NotLeftVecs : jobvl
Nag_RightVecs : jobvr
Nag_RCondBoth : sense

(-21.10,-22.50) ( 53.50,-50.50) (-34.50,127.50) ( 7.50, 0.50)
(-0.46, -7.78) (-3.50,-37.50) (-15.50, 58.50) (-10.50, -1.50)
( 4.30, -5.50) ( 39.70,-17.10) (-68.50, 12.50) (-7.50, -3.50)
( 5.50, 4.40) ( 14.40, 43.30) (-32.50,-46.00) (-19.00,-32.50) : A

( 1.00, -5.00) ( 1.60, 1.20) (-3.00, 0.00) ( 0.00, -1.00)
( 0.80, -0.60) ( 3.00, -5.00) (-4.00, 3.00) (-2.40, -3.20)
( 1.00, 0.00) ( 2.40, 1.80) (-4.00, -5.00) ( 0.00, -3.00)
( 0.00, 1.00) ( -1.80, 2.40) ( 0.00, -4.00) ( 4.00, -5.00) : B
10.3 Program Results

nag_zggevx (f08wpc) Example Program Results

R = Reciprocal condition number, E = Error bound

<table>
<thead>
<tr>
<th>Eigenvalues</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (3.0000e+00, -9.0000e+00)</td>
<td>5.1e-01</td>
<td>3.1e-15</td>
</tr>
<tr>
<td>2 (2.0000e+00, -5.0000e+00)</td>
<td>3.8e-01</td>
<td>4.3e-15</td>
</tr>
<tr>
<td>3 (3.0000e+00, -1.0000e+00)</td>
<td>1.3e-01</td>
<td>1.2e-14</td>
</tr>
<tr>
<td>4 (4.0000e+00, -5.0000e+00)</td>
<td>6.2e-01</td>
<td>2.6e-15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Right eigenvectors (columns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th>E</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7e-02</td>
<td>6.6e-02</td>
<td>1.7e-01</td>
<td>3.5e-02</td>
</tr>
<tr>
<td>3.4e-14</td>
<td>2.4e-14</td>
<td>9.3e-15</td>
<td>4.6e-14</td>
</tr>
</tbody>
</table>