1 Purpose

nag_dggevx (f08wbc) computes for a pair of \( n \) by \( n \) real 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_dggevx (Nag_OrderType order, Nag_BalanceType balanc,
                 Nag_LeftVecsType jobvl, Nag_RightVecsType jobvr, Nag_RCondType sense,
                 Integer n, double a[], Integer pda, double b[], Integer pdb,
                 double alphar[], double alphai[], double beta[],
                 Integer pdvl, double vl[], Integer pdvr, double vr[],
                 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 eigenvector \( v_j \) corresponding to the eigenvalue \( \lambda_j \) of \( (A, B) \) satisfies

\[
Av_j = \lambda_j Bv_j.
\]

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

\[
u_j^HA = \lambda_j u_j^HB,
\]

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

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

1. \( A \) is reduced to upper Hessenberg form and at the same time \( B \) is reduced to upper triangular form.
2. \( A \) is further reduced to quasi-triangular form while the triangular form of \( B \) is maintained. This is the real generalized Schur form of the pair \( (A, B) \).
3. The quasi-triangular form of \( A \) is reduced to triangular form and the eigenvalues extracted. This function does not actually produce the eigenvalues \( \lambda_j \), but instead returns \( \alpha_j \) and \( \beta_j \) such that

\[
\lambda_j = \frac{\alpha_j}{\beta_j}, \quad j = 1, 2, \ldots, n.
\]

The division by \( \beta_j \) becomes your responsibility, since \( \beta_j \) may be zero, indicating an infinite eigenvalue. Pairs of complex eigenvalues occur with \( \alpha_j/\beta_j \) and \( \alpha_{j+1}/\beta_{j+1} \) complex conjugates, even though \( \alpha_j \) and \( \alpha_{j+1} \) are not conjugate.
4. 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_dggbal (f08whc).

4 References


5 Arguments

1: order – Nag_OrderType Input

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 Input

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 Input

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 Input

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] – double

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 real 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] – double

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: alphar[n] – double

On exit: the element alphar[j - 1] contains the real part of α_j.
On exit: the element $\alpha[i - 1]$ contains the imaginary part of $\alpha_j$.

On exit: $(\alpha[j - 1] + \alpha[j - 1] \times i)/\beta[j - 1]$, for $j = 1, 2, \ldots, n$, will be the generalized eigenvalues.

If $\alpha[j - 1]$ is zero, then the $j$th eigenvalue is real; if positive, then the $j$th and $(j + 1)$st eigenvalues are a complex conjugate pair, with $\alpha[j]$ negative.

Note: the quotients $\alpha[j - 1]/\beta[j - 1]$ and $\alpha[j - 1]/\beta[j - 1]$ may easily overflow or underflow, and $\beta[j - 1]$ may even be zero. Thus, you should avoid naively computing the ratio $\alpha/j/\beta$. 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 $||b||_2$.

Note: the dimension, $\dim$, of the array $\mathbf{v}\mathbf{l}$ must be at least

$max(1, pdvl \times n)$ when $jobvl = \text{Nag LeftVecs};$

1 otherwise.

The $(i, j)$th element of the matrix is stored in

$\mathbf{v}\mathbf{l}[(j - 1) \times pdvl + i - 1]$ when $order = \text{Nag ColMajor};$

$\mathbf{v}\mathbf{l}[(i - 1) \times pdvl + j - 1]$ when $order = \text{Nag RowMajor}.$

On exit: if $jobvl = \text{Nag LeftVecs}$, the left generalized eigenvectors $u_j$ are stored one after another in the columns of $\mathbf{v}\mathbf{l}$, in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have $|\text{real part}| + |\text{imag. part}| = 1.$

If $jobvl = \text{Nag NotLeftVecs}$, $\mathbf{v}\mathbf{l}$ is not referenced.

On entry: the stride separating row or column elements (depending on the value of $order$) in the array $\mathbf{v}\mathbf{l}$.

Constraints:

if $jobvl = \text{Nag LeftVecs}$, $pdvl \geq max(1, n);$  
otherwise $pdvl \geq 1.$

Note: the dimension, $\dim$, of the array $\mathbf{v}\mathbf{r}$ must be at least

$max(1, pdvr \times n)$ when $jobvr = \text{Nag RightVecs};$

1 otherwise.

The $(i, j)$th element of the matrix is stored in

$\mathbf{v}\mathbf{r}[(j - 1) \times pdvr + i - 1]$ when $order = \text{Nag ColMajor};$

$\mathbf{v}\mathbf{r}[(i - 1) \times pdvr + j - 1]$ when $order = \text{Nag RowMajor}.$

On exit: if $jobvr = \text{Nag RightVecs}$, the right generalized eigenvectors $v_j$ are stored one after another in the columns of $\mathbf{v}\mathbf{r}$, in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have $|\text{real part}| + |\text{imag. part}| = 1.$

If $jobvr = \text{Nag NotRightVecs}$, $\mathbf{v}\mathbf{r}$ is not referenced.

On entry: the stride separating row or column elements (depending on the value of $order$) in the array $\mathbf{v}\mathbf{r}$.
Constraints:

if jobvr = Nag_RightVecs, pdvr ≥ max(1, n);
otherwise pdvr ≥ 1.

18: ilo – Integer *  Output
19: ihi – Integer *  Output

On exit: ilo and ihi are integer values such that A(i, j) = 0 and B(i, j) = 0 if i > j and
j = 1, 2, ..., ilo - 1 or i = ihi + 1, ..., n.
If balanc = Nag_NoBalancing or Nag_BalanceScale, ilo = 1 and ihi = n.

20: lscale[n] – double  Output

On exit: details of the permutations and scaling factors applied to the left side of A and B.
If plj is the index of the row interchanged with row j, and dlj is the scaling factor applied to row
j, then:
lscale[j - 1] = plj, for j = 1, 2, ..., ilo - 1;
lscale = dlj, for j = ilo, ..., ihi;
lscale = plj, for j = ihi + 1, ..., n.
The order in which the interchanges are made is n to ihi + 1, then 1 to ilo - 1.

21: rscale[n] – double  Output

On exit: details of the permutations and scaling factors applied to the right side of A and B.
If prj is the index of the column interchanged with column j, and drj is the scaling factor applied
to column j, then:
rscale[j - 1] = prj, for j = 1, 2, ..., ilo - 1;
if rscale = drj, for j = ilo, ..., ihi;
if rscale = prj, for j = ihi + 1, ..., n.
The order in which the interchanges are made is n to ihi + 1, then 1 to ilo - 1.

22: abnrm – double *  Output

On exit: the 1-norm of the balanced matrix A.

23: bbnrm – double *  Output

On exit: the 1-norm of the balanced matrix B.

24: rconde[dim] – double  Output

Note: the dimension, dim, of the array rconde must be at least max(1, n).
On exit: if sense = Nag_RCondEigVals or Nag_RCondBoth, the reciprocal condition numbers of
the eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of
eigenvalues two consecutive elements of rconde are set to the same value. Thus rconde[j - 1],
rcondv[j - 1], and the jth columns of vl and vr all correspond to the jth eigenpair.
If sense = Nag_RCondEigVecs, rconde is not referenced.

25: rcondv[dim] – double  Output

Note: the dimension, dim, of the array rcondv must be at least max(1, n).
On exit: if sense = Nag_RCondEigVecs or Nag_RCondBoth, the estimated reciprocal condition numbers of the eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of rcondv are set to the same value.

If sense = Nag_RCondEigVals, rcondv is not referenced.

26: fail – NagError *  

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 ⟨value⟩ had an illegal value.

NE_EIGENVECTORS

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

NE_ENUM_INT_2

On entry, jobvl = ⟨value⟩, pdvl = ⟨value⟩ and n = ⟨value⟩.
Constraint: if jobvl = Nag_LeftVecs, pdvl ≥ max(1, n); otherwise pdvl ≥ 1.

On entry, jobvr = ⟨value⟩, pdvr = ⟨value⟩ and n = ⟨value⟩.
Constraint: if jobvr = Nag_RightVecs, pdvr ≥ max(1, n); otherwise pdvr ≥ 1.

NE_INT

On entry, n = ⟨value⟩.
Constraint: n ≥ 0.

On entry, pda = ⟨value⟩.
Constraint: pda > 0.

On entry, pdb = ⟨value⟩.
Constraint: pdb > 0.

On entry, pdvl = ⟨value⟩.
Constraint: pdvl > 0.

On entry, pdvr = ⟨value⟩.
Constraint: pdvr > 0.

NE_INT_2

On entry, pda = ⟨value⟩ and n = ⟨value⟩.
Constraint: pda ≥ max(1, n).

On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ 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 $\text{alphar}[j]$, $\text{alphai}[j]$ and $\text{beta}[j]$ should be correct from element <value>.

The QZ iteration failed with an unexpected error, 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.

### 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

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

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

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

For further explanation of the reciprocal condition numbers $\text{rconde}$ and $\text{rcondv}$, 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 $\lambda_i = \alpha_i / \beta_i$. 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

nag_dggevx (f08wbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dggevx (f08wbc) 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 complex analogue of this function is nag_zggevx (f08wpc).
10 Example

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

\[
A = \begin{pmatrix} 3.9 & 12.5 & -34.5 & -0.5 \\ 4.3 & 21.5 & -47.5 & 7.5 \\ 4.3 & 21.5 & -43.5 & 3.5 \\ 4.4 & 26.0 & -46.0 & 6.0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1.0 & 2.0 & -3.0 & 1.0 \\ 1.0 & 3.0 & -5.0 & 4.0 \\ 1.0 & 3.0 & -4.0 & 3.0 \\ 1.0 & 3.0 & -4.0 & 4.0 \end{pmatrix},
\]

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

```c
/* nag_dggevx (f08wbc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

#include <math.h>
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <naga02.h>

int main(void)
{
    /* Scalars */
    Complex eig, eigl, eigr;
    double abnorm, abnrm, bbnrm, eps, sign, small, tol;
    Integer i, ihi, ilo, j, k, n, pda, pdb, pdvl, pdvr;
    Integer exit_status = 0;
    /* Arrays */
    double *a = 0, *alphai = 0, *alphar = 0, *b = 0, *beta = 0;
    double *lscale = 0, *rconde = 0, *rcondv = 0, *rscale = 0;
    double *vl = 0, *vr = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    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]
    #define VL(I, J) vl[(J-1)*pdvl + I - 1]
    #define VR(I, J) vr[(J-1)*pdvr + I - 1]
    #define order Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda + J - 1]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    #define VL(I, J) vl[(I-1)*pdvl + J - 1]
    #define VR(I, J) vr[(I-1)*pdvr + J - 1]
    #define order Nag_RowMajor;
    #endif

    INIT_FAIL(fail);

    printf("nag_dggevx (f08wbc) Example Program Results\n");

    /* Skip heading in data file */

    /*...*/
```
```c
#ifdef _WIN32
    scanf_s("%*\n");
#else
    scanf("%*\n");
#endif
#ifdef _WIN32
    scanf_s("%NAG_IFMT%*\n", &n);
#else
    scanf("%NAG_IFMT%*\n", &n);
#endif
if (n < 0)
{
    printf("Invalid n\n");
    exit_status = 1;
    goto END;
}
#ifdef _WIN32
    scanf_s(" %39s%*\n", nag_enum_arg, _countof(nag_enum_arg));
#else
    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 _WIN32
#ifdef _WIN32
    scanf_s(" %39s%*\n", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf(" %39s%*\n", nag_enum_arg);
#endif
    jobvr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#endif _WIN32
#ifdef _WIN32
    scanf_s(" %39s%*\n", nag_enum_arg, _countof(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, double)) ||
        !(alphai = NAG_ALLOC(n, double)) ||
        !(alphar = NAG_ALLOC(n, double)) ||
        !(b = NAG_ALLOC(n*n, double)) ||
        !(beta = NAG_ALLOC(n, double)) ||
        !(lscale = NAG_ALLOC(n, double)) ||
        !(rconde = NAG_ALLOC(n, double)) ||
        !(rcondv = NAG_ALLOC(n, double)) ||
        !(rscale = NAG_ALLOC(n, double)) ||
        !(vl = NAG_ALLOC(pdvl*pdvl, double)) ||
        !(vr = NAG_ALLOC(pdvr*pdvr, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read in the matrices A and B */
for (i = 1; i <= n; ++i)
    #ifdef _WIN32
        for (j = 1; j <= n; ++j) scanf_s("%lf", &A(i, j));
    #else
        for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
    #endif
    #ifdef _WIN32
        scanf_s("%*\n");
    #else
        scanf("%*\n");
    #endif
```

/* Solve the generalized eigenvalue problem using nag_dggevx (f08wbc). */

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

if (fail.code != NE_NOERROR)
    {  
        printf("Error from nag_dggevx (f08wbc).\n%s\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 vectors and associated condition number and bounds. */
for (j = 0; j < n; ++j)
    {  
        printf("Eigenvalue %2"NAG_IFMT" = (%13.4e, %13.4e)\n", j+1, eig.re, eig.im);
        if (sense==Nag_RCondEigVals || sense==Nag_RCondBoth) {  
            printf("\n Reciprocal condition number = %10.1e, rconde[j]\n", rconde[j]);
            if ( rconde[j] > 0.0)  
                printf(" Error bound = %10.1e, tol/rconde[j]\n", tol/rconde[j]);
        else
            printf(" Error bound is infinite\n");
    }
printf("\n\n");
/* Normalize and print out information on the jth eigenvector(s) */
if (jobvl==Nag_LeftVecs) printf("%21s%8s", "Left Eigenvector", "");
if (jobvr==Nag_RightVecs) printf("%21s", "Right Eigenvector");
printf("%2"NAG_IFMT"
", j+1);
if (alphai[j] == 0.0)
for (i = 1; i <= n; ++i) {
  if (jobvl==Nag_LeftVecs)
    printf("%7s%13.4e%12s", "", VL(i, j+1)/VL(n, j+1), "");
  if (jobvr==Nag_RightVecs)
    printf("%7s%13.4e", "", VR(i, j+1)/VR(n, j+1));
  printf("\n");
} else {
  k = (alphai[j]>0.0?j+1:j);
  sign = (alphai[j]>0.0?1.0:-1.0);
  if (jobvl==Nag_LeftVecs) eigl = nag_complex(VL(n,k), VL(n,k+1));
  if (jobvr==Nag_RightVecs) eigr = nag_complex(VR(n,k), VR(n,k+1));
  for (i = 1; i <= n; ++i) {
    if (jobvl==Nag_LeftVecs) {
      eig = nag_complex_divide(nag_complex(VL(i,k), VL(i,k+1)),
      eigl);
      printf(" (%13.4e,%13.4e) ", eig.re, sign*eig.im);
    }
    if (jobvr==Nag_RightVecs) {
      eig = nag_complex_divide(nag_complex(VR(i,k), VR(i,k+1)),
      eigr);
      printf(" (%13.4e,%13.4e)", eig.re, sign*eig.im);
    }
    printf("\n");
  }
}
if (sense==Nag_RCondEigVecs || sense==Nag_RCondBoth) {
  printf("\n Reciprocal condition number = %10.1e
", rcondv[j]);
  if (rcondv[j] > 0.0)
    printf(" Error bound = %10.1e\n", tol/rcondv[j]);
  else
    printf(" Error bound is infinite\n\n");
}
END:
NAG_FREE(a);
NAG_FREE(alphai);
NAG_FREE(alphar);
NAG_FREE(b);
NAG_FREE(beta);
NAG_FREE(lscale);
NAG_FREE(rconde);
NAG_FREE(rcondv);
NAG_FREE(rscale);
NAG_FREE(vl);
NAG_FREE(vr);
return exit_status;
}

10.2 Program Data

nag_dggevx (f08wbc) Example Program Data

4 : n
  Nag_NotLeftVecs : jobvl
  Nag_RightVecs : jobvr
  Nag_RCondBoth : sense
### 10.3 Program Results

nag_dggevx (f08wbc) Example Program Results

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Reciprocal condition number</th>
<th>Error bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0000e+00</td>
<td>9.5e-02</td>
<td>2.5e-14</td>
</tr>
<tr>
<td>(3.0000e+00, 4.0000e+00)</td>
<td>1.7e-01</td>
<td>1.4e-14</td>
</tr>
<tr>
<td>(3.0000e+00, -4.0000e+00)</td>
<td>1.7e-01</td>
<td>1.4e-14</td>
</tr>
<tr>
<td>4.0000e+00</td>
<td>5.1e-01</td>
<td>4.6e-15</td>
</tr>
</tbody>
</table>

Matrix $A$

```
3.9 12.5 -34.5 -0.5
4.3 21.5 -47.5 7.5
4.3 21.5 -43.5 3.5
4.4 26.0 -46.0 6.0 : matrix A
```

Matrix $B$

```
1.0 2.0 -3.0 1.0
1.0 3.0 -5.0 4.0
1.0 3.0 -4.0 3.0
1.0 3.0 -4.0 4.0 : matrix B
```
Reciprocal condition number = 7.1e-02
Error bound = 3.3e-14