

# 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

```
#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

$$\lambda_j = \alpha_j / \beta_j, \quad j = 1, 2, \dots, n.$$

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

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia <http://www.netlib.org/lapack/lug>

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th Edition) Johns Hopkins University Press, Baltimore

Wilkinson J H (1979) Kronecker's canonical form and the *QZ* algorithm *Linear Algebra Appl.* **28** 285–303

## 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 *Input*  
*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 *Input*  
*On entry:*  $n$ , the order of the matrices  $A$  and  $B$ .  
*Constraint:*  $n \geq 0$ .
- 7: **a**[ $dim$ ] – Complex *Input/Output*  
**Note:** the dimension,  $dim$ , of the array **a** must be at least  $\max(1, pda \times n)$ .  
 Where  $\mathbf{A}(i, j)$  appears in this document, it refers to the array element  

$$\mathbf{a}[(j-1) \times pda + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor};$$

$$\mathbf{a}[(i-1) \times pda + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*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 *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **a**.  
*Constraint:*  $pda \geq \max(1, n)$ .
- 9: **b**[ $dim$ ] – Complex *Input/Output*  
**Note:** the dimension,  $dim$ , of the array **b** must be at least  $\max(1, pdb \times n)$ .  
 Where  $\mathbf{B}(i, j)$  appears in this document, it refers to the array element  

$$\mathbf{b}[(j-1) \times pdb + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor};$$

$$\mathbf{b}[(i-1) \times pdb + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*On entry:* the matrix  $B$  in the pair  $(A, B)$ .  
*On exit:* **b** has been overwritten.
- 10: **pdb** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **b**.  
*Constraint:*  $pdb \geq \max(1, n)$ .
- 11: **alpha**[ $n$ ] – Complex *Output*  
*On exit:* see the description of **beta**.

12: **beta[n]** – Complex *Output*

*On exit:*  $\mathbf{alpha}[j-1]/\mathbf{beta}[j-1]$ , for  $j = 1, 2, \dots, \mathbf{n}$ , will be the generalized eigenvalues.

**Note:** the quotients  $\mathbf{alpha}[j-1]/\mathbf{beta}[j-1]$  may easily overflow or underflow, and  $\mathbf{beta}[j-1]$  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  $\|B\|_2$ .

13: **vl[dim]** – Complex *Output*

**Note:** the dimension, *dim*, of the array **vl** must be at least

$\max(1, \mathbf{pdvl} \times \mathbf{n})$  when **jobvl** = Nag\_LeftVecs;  
1 otherwise.

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

$\mathbf{vl}[(j-1) \times \mathbf{pdvl} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{vl}[(i-1) \times \mathbf{pdvl} + j - 1]$  when **order** = Nag\_RowMajor.

*On exit:* if **jobvl** = Nag\_LeftVecs, 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  $|\text{real part}| + |\text{imag. part}| = 1$ .

If **jobvl** = Nag\_NotLeftVecs, **vl** is not referenced.

14: **pdvl** – Integer *Input*

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

*Constraints:*

if **jobvl** = Nag\_LeftVecs, **pdvl**  $\geq \max(1, \mathbf{n})$ ;  
otherwise **pdvl**  $\geq 1$ .

15: **vr[dim]** – Complex *Output*

**Note:** the dimension, *dim*, of the array **vr** must be at least

$\max(1, \mathbf{pdvr} \times \mathbf{n})$  when **jobvr** = Nag\_RightVecs;  
1 otherwise.

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

$\mathbf{vr}[(j-1) \times \mathbf{pdvr} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{vr}[(i-1) \times \mathbf{pdvr} + j - 1]$  when **order** = Nag\_RowMajor.

*On exit:* if **jobvr** = Nag\_RightVecs, 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  $|\text{real part}| + |\text{imag. part}| = 1$ .

If **jobvr** = Nag\_NotRightVecs, **vr** is not referenced.

16: **pdvr** – Integer *Input*

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

*Constraints:*

if **jobvr** = Nag\_RightVecs, **pdvr**  $\geq \max(1, \mathbf{n})$ ;  
otherwise **pdvr**  $\geq 1$ .

- 17: **ilo** – Integer \* *Output*
- 18: **ihi** – Integer \* *Output*
- On exit:* **ilo** and **ihi** are integer values such that  $\mathbf{A}(i, j) = 0$  and  $\mathbf{B}(i, j) = 0$  if  $i > j$  and  $j = 1, 2, \dots, \mathbf{ilo} - 1$  or  $i = \mathbf{ihi} + 1, \dots, \mathbf{n}$ .
- If **balanc** = Nag\_NoBalancing or Nag\_BalanceScale, **ilo** = 1 and **ihi** = **n**.
- 19: **lscale**[**n**] – double *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:
- $$\mathbf{lscale}[j - 1] = pl_j, \text{ for } j = 1, 2, \dots, \mathbf{ilo} - 1;$$
- $$\mathbf{lscale} = dl_j, \text{ for } j = \mathbf{ilo}, \dots, \mathbf{ihi};$$
- $$\mathbf{lscale} = pl_j, \text{ for } j = \mathbf{ihi} + 1, \dots, \mathbf{n}.$$
- The order in which the interchanges are made is **n** to **ihi** + 1, then 1 to **ilo** - 1.
- 20: **rscale**[**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:
- $$\mathbf{rscale}[j - 1] = pr_j, \text{ for } j = 1, 2, \dots, \mathbf{ilo} - 1;$$
- $$\text{if } \mathbf{rscale} = dr_j, \text{ for } j = \mathbf{ilo}, \dots, \mathbf{ihi};$$
- $$\text{if } \mathbf{rscale} = pr_j, \text{ for } j = \mathbf{ihi} + 1, \dots, \mathbf{n}.$$
- The order in which the interchanges are made is **n** to **ihi** + 1, then 1 to **ilo** - 1.
- 21: **abnrm** – double \* *Output*
- On exit:* the 1-norm of the balanced matrix *A*.
- 22: **bbnrm** – double \* *Output*
- On exit:* the 1-norm of the balanced matrix *B*.
- 23: **rconde**[*dim*] – double *Output*
- Note:** the dimension, *dim*, of the array **rconde** must be at least  $\max(1, \mathbf{n})$ .
- On exit:* if **sense** = Nag\_RCondEigVals or Nag\_RCondBoth, the reciprocal condition numbers of the eigenvalues, stored in consecutive elements of the array.
- If **sense** = Nag\_NotRCond or Nag\_RCondEigVecs, **rconde** is not referenced.
- 24: **rcondv**[*dim*] – double *Output*
- Note:** the dimension, *dim*, of the array **rcondv** must be at least  $\max(1, \mathbf{n})$ .
- On exit:* if **sense** = Nag\_RCondEigVecs or Nag\_RCondBoth, the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array.
- If **sense** = Nag\_NotRCond or Nag\_RCondEigVals, **rcondv** is not referenced.
- 25: **fail** – NagError \* *Input/Output*
- The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

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

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_EIGENVECTORS

A failure occurred in nag\_ztgevc (f08yxc) while computing generalized eigenvectors.

### NE\_ENUM\_INT\_2

On entry,  $\mathbf{jobvl} = \langle value \rangle$ ,  $\mathbf{pdvl} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{jobvr} = \langle value \rangle$ ,  $\mathbf{pdvr} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint: if  $\mathbf{jobvr} = \text{Nag\_RightVecs}$ ,  $\mathbf{pdvr} \geq \max(1, \mathbf{n})$ ;  
otherwise  $\mathbf{pdvr} \geq 1$ .

### NE\_INT

On entry,  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{n} \geq 0$ .

On entry,  $\mathbf{pda} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} > 0$ .

On entry,  $\mathbf{pdb} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} > 0$ .

On entry,  $\mathbf{pdvl} = \langle value \rangle$ .

Constraint:  $\mathbf{pdvl} > 0$ .

On entry,  $\mathbf{pdvr} = \langle value \rangle$ .

Constraint:  $\mathbf{pdvr} > 0$ .

### NE\_INT\_2

On entry,  $\mathbf{pda} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} \geq \max(1, \mathbf{n})$ .

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} \geq \max(1, \mathbf{n})$ .

### 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 How to Use the NAG Library and its Documentation for further information.

### NE\_ITERATION\_QZ

The  $QZ$  iteration failed. No eigenvectors have been calculated but  $\mathbf{alpha}$  and  $\mathbf{beta}$  should be correct from element  $\langle value \rangle$ .

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 How to Use the NAG Library and its Documentation for further information.

**7 Accuracy**

The computed eigenvalues and eigenvectors are exact for 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

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

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

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

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  $\lambda_i = \alpha_i / \beta_i$ . 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\_zggevx (f08wpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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 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.
*
* NAGPRODCODE Version.
*
* Copyright 2016 Numerical Algorithms Group.
*
* Mark 26, 2016.
*/

#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, abnorm, 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, *vl = 0, *vr = 0;
    double *lscale = 0, *rconde = 0, *rcondv = 0, *rscale = 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]
    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
}

```



```

    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, (unsigned)_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);
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif
    jobvr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, (unsigned)_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, Complex)) ||
        !(b = NAG_ALLOC(n * n, Complex)) ||
        !(alpha = NAG_ALLOC(n, Complex)) ||
        !(beta = NAG_ALLOC(n, Complex)) ||
        !(v1 = NAG_ALLOC(pdvl * pdvl, Complex)) ||
        !(vr = NAG_ALLOC(pdvr * pdvr, 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)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
            scanf_s("%*[\n]");
#else
            scanf("%*[\n]");
#endif
        for (i = 1; i <= n; ++i)
            for (j = 1; j <= n; ++j)
#ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
                scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);

```

```

#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif

/* 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,
           &abnorm, &bnorm, rconde, rcondv, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zggevx (f08wpc).\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 (abnorm == 0.0)
    abnorm = ABS(bnorm);
else if (bnorm == 0.0)
    abnorm = ABS(abnorm);
else if (ABS(abnorm) >= ABS(bnorm))
    abnorm = ABS(abnorm) * sqrt(1.0 + (bnorm / abnorm) * (bnorm / abnorm));
else
    abnorm = ABS(bnorm) * sqrt(1.0 + (abnorm / bnorm) * (abnorm / bnorm));

tol = eps * abnorm;

/* Print out eigenvalues and associated condition number and bounds */
if (sense != Nag_NotRCond)
    printf("\n R = Reciprocal condition number, E = Error bound\n\n");
printf("%22s", "Eigenvalues");
if (sense == Nag_RCondEigVals || sense == Nag_RCondBoth)
    printf("%15s%10s", "R", "E");
printf("\n");
for (j = 0; j < n; ++j) {
    /* Print out information on the j-th eigenvalue */
    if (nag_complex_abs(alpha[j]) * small >= nag_complex_abs(beta[j])) {
        printf("%2" NAG_IFMT " is numerically infinite or undetermined\n",
               j + 1);
        printf("    alpha = (%9.4f, %9.4f), beta = (%9.4f, %9.4f)\n",
               alpha[j].re, alpha[j].im, beta[j].re, beta[j].im);
    }
    else {
        z = nag_complex_divide(alpha[j], beta[j]);
        printf("%2" NAG_IFMT " (%13.4e, %13.4e)", 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). */
    fflush(stdout);
    nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                             n, vl, pdvl, "    Left eigenvectors (columns)",
                             0, &fail);
}
if (jobvr == Nag_RightVecs && fail.code == NE_NOERROR) {
    printf("\n");
}

```

```

/* Print rightt eigenvectors using nag_gen_complx_mat_print (x04dac). */
fflush(stdout);
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("\n%2s", "E");
    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(v1);
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

	Eigenvalues	R	E
1 (	3.0000e+00, -9.0000e+00)	5.1e-01	3.1e-15
2 (	2.0000e+00, -5.0000e+00)	3.8e-01	4.3e-15

```
3 ( 3.0000e+00, -1.0000e+00) 1.3e-01 1.2e-14
4 ( 4.0000e+00, -5.0000e+00) 6.2e-01 2.6e-15
```

```
Right eigenvectors (columns)
      1      2      3      4
1 -0.7396  0.6237  0.4880 -0.3660
  -0.2604  0.3763  0.5120  0.6340

2 -0.1496  0.0041  0.1395  0.0010
   0.0471 -0.0004  0.0233  0.0081

3 -0.0471  0.0392  0.1405  0.0122
  -0.1496  0.0237 -0.0167 -0.0211

4  0.1496 -0.0237  0.0167 -0.0986
  -0.0471  0.0392  0.1405 -0.0569

R 4.7e-02  6.6e-02  1.7e-01  3.5e-02
E 3.4e-14  2.4e-14  9.3e-15  4.6e-14
```

---