

# NAG Library Function Document

## nag\_zgeevx (f08npc)

### 1 Purpose

nag\_zgeevx (f08npc) computes the eigenvalues and, optionally, the left and/or right eigenvectors for an  $n$  by  $n$  complex nonsymmetric matrix  $A$ .

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_zgeevx (Nag_OrderType order, Nag_BalanceType balanc,
                 Nag_LeftVecsType jobvl, Nag_RightVecsType jobvr, Nag_RCondType sense,
                 Integer n, Complex a[], Integer pda, Complex w[], Complex vl[],
                 Integer pdvl, Complex vr[], Integer pdvr, Integer *ilo, Integer *ihi,
                 double scale[], double *abnrm, double rconde[], double rcondv[],
                 NagError *fail)
```

### 3 Description

The right eigenvector  $v_j$  of  $A$  satisfies

$$Av_j = \lambda_j v_j$$

where  $\lambda_j$  is the  $j$ th eigenvalue of  $A$ . The left eigenvector  $u_j$  of  $A$  satisfies

$$u_j^H A = \lambda_j u_j^H$$

where  $u_j^H$  denotes the conjugate transpose of  $u_j$ .

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation  $DAD^{-1}$ , where  $D$  is a diagonal matrix, with the aim of making its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see Section 4.8.1.2 of Anderson *et al.* (1999).

Following the optional balancing, the matrix  $A$  is first reduced to upper Hessenberg form by means of unitary similarity transformations, and the  $QR$  algorithm is then used to further reduce the matrix to upper triangular Schur form,  $T$ , from which the eigenvalues are computed. Optionally, the eigenvectors of  $T$  are also computed and backtransformed to those of  $A$ .

### 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 (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

## 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:* indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues.  
**balanc** = Nag\_NoBalancing  
Do not diagonally scale or permute.  
**balanc** = Nag\_BalancePermute  
Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale.  
**balanc** = Nag\_BalanceScale  
Diagonally scale the matrix, i.e., replace  $A$  by  $DAD^{-1}$ , where  $D$  is a diagonal matrix chosen to make the rows and columns of  $A$  more equal in norm. Do not permute.  
**balanc** = Nag\_BalanceBoth  
Both diagonally scale and permute  $A$ .  
Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.  
*Constraint:* **balanc** = Nag\_NoBalancing, Nag\_BalancePermute, Nag\_BalanceScale or Nag\_BalanceBoth.
- 3: **jobvl** – Nag\_LeftVecsType *Input*  
*On entry:* if **jobvl** = Nag\_NotLeftVecs, the left eigenvectors of  $A$  are not computed.  
If **jobvl** = Nag\_LeftVecs, the left eigenvectors of  $A$  are computed.  
If **sense** = Nag\_RCondEigVals or Nag\_RCondBoth, **jobvl** must be set to **jobvl** = Nag\_LeftVecs.  
*Constraint:* **jobvl** = Nag\_NotLeftVecs or Nag\_LeftVecs.
- 4: **jobvr** – Nag\_RightVecsType *Input*  
*On entry:* if **jobvr** = Nag\_NotRightVecs, the right eigenvectors of  $A$  are not computed.  
If **jobvr** = Nag\_RightVecs, the right eigenvectors of  $A$  are computed.  
If **sense** = Nag\_RCondEigVals or Nag\_RCondBoth, **jobvr** must be set to **jobvr** = Nag\_RightVecs.  
*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 right eigenvectors only.

**sense** = Nag\_RCondBoth

Computed for eigenvalues and right eigenvectors.

If **sense** = Nag\_RCondEigVals or Nag\_RCondBoth, both left and right eigenvectors must also be computed (**jobvl** = Nag\_LeftVecs and **jobvr** = Nag\_RightVecs).

*Constraint:* **sense** = Nag\_NotRCond, Nag\_RCondEigVals, Nag\_RCondEigVecs or Nag\_RCondBoth.

6: **n** – Integer *Input*

*On entry:*  $n$ , the order of the matrix  $A$ .

*Constraint:*  $n \geq 0$ .

7: **a**[ $dim$ ] – Complex *Input/Output*

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

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

$a[(j-1) \times pda + i - 1]$  when **order** = Nag\_ColMajor;  
 $a[(i-1) \times pda + j - 1]$  when **order** = Nag\_RowMajor.

*On entry:* the  $n$  by  $n$  matrix  $A$ .

*On exit:* **a** has been overwritten. If **jobvl** = Nag\_LeftVecs or **jobvr** = Nag\_RightVecs,  $A$  contains the Schur form of the balanced version of the matrix  $A$ .

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: **w**[ $dim$ ] – Complex *Output*

**Note:** the dimension,  $dim$ , of the array **w** must be at least  $\max(1, n)$ .

*On exit:* contains the computed eigenvalues.

10: **vl**[ $dim$ ] – Complex *Output*

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

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

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

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

*On exit:* if **jobvl** = Nag\_LeftVecs, the left eigenvectors  $u_j$  are stored one after another in **vl**, in the same order as their corresponding eigenvalues; that is  $u_j = \mathbf{VL}(i, j)$ , for  $i = 1, 2, \dots, n$ .

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

11: **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,  $pdvl \geq \max(1, n)$ ;  
 otherwise  $pdvl \geq 1$ .

12: **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.

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

**vr**[(*j* − 1) × **pdvr** + *i* − 1] when **order** = Nag\_ColMajor;  
**vr**[(*i* − 1) × **pdvr** + *j* − 1] when **order** = Nag\_RowMajor.

*On exit:* if **jobvr** = Nag\_RightVecs, the right eigenvectors  $v_j$  are stored one after another in **vr**, in the same order as their corresponding eigenvalues; that is  $v_j = \mathbf{VR}(i, j)$ , for  $i = 1, 2, \dots, \mathbf{n}$ .

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

13: **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** ≥ max(1, **n**);  
otherwise **pdvr** ≥ 1.

14: **ilo** – Integer \* Output

15: **ihi** – Integer \* Output

*On exit:* **ilo** and **ihi** are integer values determined when *A* was balanced. The balanced *A* has  $a_{ij} = 0$  if  $i > j$  and  $j = 1, 2, \dots, \mathbf{ilo} - 1$  or  $i = \mathbf{ihi} + 1, \dots, \mathbf{n}$ .

16: **scale**[*dim*] – double Output

**Note:** the dimension, *dim*, of the array **scale** must be at least max(1, **n**).

*On exit:* details of the permutations and scaling factors applied when balancing *A*.

If  $p_j$  is the index of the row and column interchanged with row and column *j*, and  $d_j$  is the scaling factor applied to row and column *j*, then

**scale**[*j* − 1] =  $p_j$ , for  $j = 1, 2, \dots, \mathbf{ilo} - 1$ ;  
**scale**[*j* − 1] =  $d_j$ , for  $j = \mathbf{ilo}, \dots, \mathbf{ihi}$ ;  
**scale**[*j* − 1] =  $p_j$ , 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.

17: **abnrm** – double \* Output

*On exit:* the 1-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

18: **rconde**[*dim*] – double Output

**Note:** the dimension, *dim*, of the array **rconde** must be at least max(1, **n**).

*On exit:* **rconde**[*j* − 1] is the reciprocal condition number of the *j*th eigenvalue.

19: **rcondv**[*dim*] – double Output

**Note:** the dimension, *dim*, of the array **rcondv** must be at least max(1, **n**).

*On exit:* **rcondv**[*j* − 1] is the reciprocal condition number of the *j*th right eigenvector.

20: **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 2.3.1.2 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\_CONVERGENCE

The  $QR$  algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1 to  $ilo - 1$  and  $\langle value \rangle$  to  $n$  of  $w$  contain eigenvalues which have converged.

### NE\_ENUM\_INT\_2

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

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

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

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

### NE\_INT

On entry,  $n = \langle value \rangle$ .

Constraint:  $n \geq 0$ .

On entry,  $pda = \langle value \rangle$ .

Constraint:  $pda > 0$ .

On entry,  $pdvl = \langle value \rangle$ .

Constraint:  $pdvl > 0$ .

On entry,  $pdvr = \langle value \rangle$ .

Constraint:  $pdvr > 0$ .

### NE\_INT\_2

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

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

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

### NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.

See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

## 7 Accuracy

The computed eigenvalues and eigenvectors are exact for a nearby matrix  $(A + E)$ , where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and  $\epsilon$  is the *machine precision*. See Section 4.8 of Anderson *et al.* (1999) for further details.

## 8 Parallelism and Performance

nag\_zgeevx (f08npc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_zgeevx (f08npc) 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

Each eigenvector is normalized to have Euclidean norm equal to unity and the element of largest absolute value real.

The total number of floating-point operations is proportional to  $n^3$ .

The real analogue of this function is nag\_dgeevx (f08nbc).

## 10 Example

This example finds all the eigenvalues and right eigenvectors of the matrix

$$A = \begin{pmatrix} -3.97 - 5.04i & -4.11 + 3.70i & -0.34 + 1.01i & 1.29 - 0.86i \\ 0.34 - 1.50i & 1.52 - 0.43i & 1.88 - 5.38i & 3.36 + 0.65i \\ 3.31 - 3.85i & 2.50 + 3.45i & 0.88 - 1.08i & 0.64 - 1.48i \\ -1.10 + 0.82i & 1.81 - 1.59i & 3.25 + 1.33i & 1.57 - 3.44i \end{pmatrix},$$

together with estimates of the condition number and forward error bounds for each eigenvalue and eigenvector. The option to balance the matrix is used. In order to compute the condition numbers of the eigenvalues, the left eigenvectors also have to be computed, but they are not printed out in this example.

### 10.1 Program Text

```
/* nag_zgeevx (f08npc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx02.h>
#include <naga02.h>

int main(void)
{
    /* Scalars */
```

```

double abnrm, eps, rcnd, tol;
Integer i, ihi, ilo, j, n, pda, pdvl, pdvr;
Integer exit_status = 0;
/* Arrays */
Complex *a = 0, *vl = 0, *vr = 0, *w = 0;
double *rconde = 0, *rcondv = 0, *scale = 0;

/* Nag Types */
NagError fail;
Nag_OrderType order;

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

    INIT_FAIL(fail);

    printf("nag_zgeevx (f08npc) Example Program Results\n");

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

    pda = n;
    pdvl = n;
    pdvr = n;
    /* Allocate memory */
    if (!(a = NAG_ALLOC(n * n, Complex)) ||
        !(vl = NAG_ALLOC(n * n, Complex)) ||
        !(vr = NAG_ALLOC(n * n, Complex)) ||
        !(w = NAG_ALLOC(n, Complex)) ||
        !(rconde = NAG_ALLOC(n, double)) ||
        !(rcondv = NAG_ALLOC(n, double)) || !(scale = NAG_ALLOC(n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read the matrix A from data file */
    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

    /* Solve the eigenvalue problem using nag_zgeevx (f08npc). */
    nag_zgeevx(order, Nag_BalanceBoth, Nag_LeftVecs, Nag_RightVecs,
        Nag_RCondBoth, n, a, pda, w, vl, pdvl, vr, pdvr, &ilo, &ihi,
        scale, &abnrm, rconde, rcondv, &fail);

```

```

if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgeevx (f08npc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute the machine precision */
eps = nag_machine_precision;
tol = eps * abnrm;

/* Normalize the eigenvectors */
for (j = 0; j < n; j++)
    for (i = n - 1; i >= 0; i--)
        VR(i, j) = nag_complex_divide(VR(i, j), VR(0, j));

/* Print the eigenvalues/vectors, associated condition number and bounds. */
for (j = 0; j < n; ++j) {
    /* Print information on j-th eigenvalue */
    printf("\n\nEigenvalue %3" NAG_IFMT "%14s= ", j + 1, "");
    if (w[j].im == 0.)
        printf("%12.4e\n", w[j].re);
    else
        printf("(%13.4e, %13.4e)\n", w[j].re, w[j].im);

    rcnd = rconde[j];
    printf("\nReciprocal condition number = %9.1e\n", rcnd);
    if (rcnd > 0.0)
        printf("Error bound = %9.1e\n", tol / rcnd);
    else
        printf("Error bound is infinite\n");

    /* Print information on j-th eigenvector */
    printf("\nEigenvector %2" NAG_IFMT "\n", j + 1);
    for (i = 0; i < n; ++i)
        printf("%30s(%13.4e, %13.4e)\n", "", VR(i, j).re, VR(i, j).im);

    rcnd = rcondv[j];
    printf("\nReciprocal condition number = %9.1e\n", rcnd);
    if (rcnd > 0.0)
        printf("Error bound = %9.1e\n", tol / rcnd);
    else
        printf("Error bound is infinite\n");
}

END:
    NAG_FREE(a);
    NAG_FREE(vl);
    NAG_FREE(vr);
    NAG_FREE(w);
    NAG_FREE(rconde);
    NAG_FREE(rcondv);
    NAG_FREE(scale);

    return exit_status;
}

#undef A
#undef VR

```

## 10.2 Program Data

nag\_zgeevx (f08npc) Example Program Data

```

4                                     : n

(-3.97, -5.04) (-4.11,  3.70) (-0.34,  1.01) ( 1.29, -0.86)
( 0.34, -1.50) ( 1.52, -0.43) ( 1.88, -5.38) ( 3.36,  0.65)
( 3.31, -3.85) ( 2.50,  3.45) ( 0.88, -1.08) ( 0.64, -1.48)
(-1.10,  0.82) ( 1.81, -1.59) ( 3.25,  1.33) ( 1.57, -3.44) : matrix A

```



### 10.3 Program Results

nag\_zggev (f08npc) Example Program Results

```

Eigenvalue    1                = (  -6.0004e+00,   -6.9998e+00)

Reciprocal condition number =    9.9e-01
Error bound           =    1.6e-15

Eigenvector   1
                (   1.0000e+00,    0.0000e+00)
                (  -2.0956e-02,    3.5899e-01)
                (   1.0349e-01,    3.6827e-01)
                (  -6.6390e-02,   -3.4361e-01)

Reciprocal condition number =    8.4e+00
Error bound           =    1.9e-16

Eigenvalue    2                = (  -5.0000e+00,    2.0060e+00)

Reciprocal condition number =    1.0e+00
Error bound           =    1.6e-15

Eigenvector   2
                (   1.0000e+00,   -0.0000e+00)
                (   1.1997e+00,   -6.3394e-01)
                (  -1.3192e+00,   -5.9122e-01)
                (  -1.3191e-01,    7.9036e-01)

Reciprocal condition number =    8.0e+00
Error bound           =    2.0e-16

Eigenvalue    3                = (   7.9982e+00,   -9.9637e-01)

Reciprocal condition number =    9.8e-01
Error bound           =    1.6e-15

Eigenvector   3
                (   1.0000e+00,    0.0000e+00)
                (  -1.1841e+00,   -1.8270e+00)
                (   7.4024e-01,   -1.7252e+00)
                (  -4.6684e-01,   -6.3560e-01)

Reciprocal condition number =    5.8e+00
Error bound           =    2.7e-16

Eigenvalue    4                = (   3.0023e+00,   -3.9998e+00)

Reciprocal condition number =    9.8e-01
Error bound           =    1.6e-15

Eigenvector   4
                (   1.0000e+00,   -0.0000e+00)
                (  -1.5749e+00,    3.9438e-01)
                (   1.5862e+00,    3.8955e-01)
                (  -9.5943e-01,    4.8012e+00)

Reciprocal condition number =    5.8e+00
Error bound           =    2.7e-16

```

---