

# NAG Library Function Document

## nag\_zgees (f08pnc)

### 1 Purpose

nag\_zgees (f08pnc) computes the eigenvalues, the Schur form  $T$ , and, optionally, the matrix of Schur vectors  $Z$  for an  $n$  by  $n$  complex nonsymmetric matrix  $A$ .

### 2 Specification

```
#include <nag.h>
#include <nagf08.h>

void nag_zgees (Nag_OrderType order, Nag_JobType jobvs,
               Nag_SortEigValsType sort,
               Nag_Boolean (*select)(Complex w),
               Integer n, Complex a[], Integer pda, Integer *sdim, Complex w[],
               Complex vs[], Integer pdvs, NagError *fail)
```

### 3 Description

The Schur factorization of  $A$  is given by

$$A = ZTZ^H,$$

where  $Z$ , the matrix of Schur vectors, is unitary and  $T$  is the Schur form. A complex matrix is in Schur form if it is upper triangular.

Optionally, nag\_zgees (f08pnc) also orders the eigenvalues on the diagonal of the Schur form so that selected eigenvalues are at the top left. The leading columns of  $Z$  form an orthonormal basis for the invariant subspace corresponding to the selected eigenvalues.

### 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: **jobvs** – Nag\_JobType *Input*

*On entry:* if **jobvs** = Nag\_DoNothing, Schur vectors are not computed.

If **jobvs** = Nag\_Schur, Schur vectors are computed.

*Constraint:* **jobvs** = Nag\_DoNothing or Nag\_Schur.

3: **sort** – Nag\_SortEigValsType *Input*

*On entry:* specifies whether or not to order the eigenvalues on the diagonal of the Schur form.

**sort** = Nag\_NoSortEigVals

Eigenvalues are not ordered.

**sort** = Nag\_SortEigVals

Eigenvalues are ordered (see **select**).

*Constraint:* **sort** = Nag\_NoSortEigVals or Nag\_SortEigVals.

4: **select** – function, supplied by the user *External Function*

If **sort** = Nag\_SortEigVals, **select** is used to select eigenvalues to sort to the top left of the Schur form.

If **sort** = Nag\_NoSortEigVals, **select** is not referenced and nag\_zgees (f08pnc) may be specified as NULLFN.

An eigenvalue  $w[j - 1]$  is selected if **select**( $w[j - 1]$ ) is Nag\_TRUE.

The specification of **select** is:

Nag\_Boolean select (Complex w)

1: **w** – Complex *Input*

*On entry:* the real and imaginary parts of the eigenvalue.

5: **n** – Integer *Input*

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

*Constraint:*  $n \geq 0$ .

6: **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** is overwritten by its Schur form  $T$ .

7: **pda** – Integer *Input*

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

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

8: **sdim** – Integer \* *Output*

*On exit:* if **sort** = Nag\_NoSortEigVals, **sdim** = 0.

If **sort** = Nag\_SortEigVals, **sdim** = number of eigenvalues for which **select** is Nag\_TRUE.

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, in the same order that they appear on the diagonal of the output Schur form  $T$ .

10: **vs**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvs} \times \mathbf{n})$  when **jobvs** = Nag\_Schur;  
1 otherwise.

The *i*th element of the *j*th vector is stored in

**vs**[(*j* – 1) × **pdvs** + *i* – 1] when **order** = Nag\_ColMajor;  
**vs**[(*i* – 1) × **pdvs** + *j* – 1] when **order** = Nag\_RowMajor.

*On exit:* if **jobvs** = Nag\_Schur, **vs** contains the unitary matrix  $Z$  of Schur vectors.

If **jobvs** = Nag\_DoNothing, **vs** is not referenced.

11: **pdvs** – Integer *Input*

*On entry:* the stride used in the array **vs**.

*Constraints:*

if **jobvs** = Nag\_Schur, **pdvs** ≥ max(1, **n**);  
otherwise **pdvs** ≥ 1.

12: **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 *<value>* had an illegal value.

### NE\_CONVERGENCE

The  $QR$  algorithm failed to compute all the eigenvalues.

### NE\_ENUM\_INT\_2

On entry, **jobvs** = *<value>*, **pdvs** = *<value>* and **n** = *<value>*.

Constraint: if **jobvs** = Nag\_Schur, **pdvs** ≥ max(1, **n**);  
otherwise **pdvs** ≥ 1.

### NE\_INT

On entry, **n** = *<value>*.

Constraint: **n** ≥ 0.

On entry, **pda** = *<value>*.

Constraint: **pda** > 0.

On entry, **pdvs** = *<value>*.

Constraint: **pdvs** > 0.

**NE\_INT\_2**

On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 Constraint: **pda**  $\geq$   $\max(1, \mathbf{n})$ .

**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 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.

**NE\_SCHUR\_REORDER**

The eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned).

**NE\_SCHUR\_REORDER\_SELECT**

After reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy **select** = Nag-TRUE. This could also be caused by underflow due to scaling.

**7 Accuracy**

The computed Schur factorization satisfies

$$A + E = ZTZ^H,$$

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\_zgees (f08pnc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_zgees (f08pnc) 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\_dgees (f08pac).

## 10 Example

This example finds the Schur factorization of the matrix

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

### 10.1 Program Text

```

/* nag_zgees (f08pnc) 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 <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Complex alpha, beta;
    double anorm, eps, norm;
    Integer i, j, n, pda, pdc, pdd, pdvs, sdim;
    Integer exit_status = 0;

    /* Arrays */
    Complex *a = 0, *c = 0, *d = 0, *vs = 0, *w = 0;

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

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

    INIT_FAIL(fail);

    printf("nag_zgees (f08pnc) Example Program Results\n\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
}

```

```

if (n < 0) {
    printf("Invalid n\n");
    exit_status = 1;
    return exit_status;
}

pda = n;
pdc = n;
pdd = n;
pdvs = n;
/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, Complex)) ||
    !(c = NAG_ALLOC(n * n, Complex)) ||
    !(d = NAG_ALLOC(n * n, Complex)) ||
    !(vs = NAG_ALLOC(n * n, Complex)) || !(w = NAG_ALLOC(n, Complex)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read in the matrix A */
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

/* Copy A to D: nag_zge_copy (f16tfc),
 * Complex valued general matrix copy.
 */
nag_zge_copy(order, Nag_NoTrans, n, n, a, pda, d, pdd, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* nag_zge_norm (f16uac): Find norm of matrix A for use later
 * in relative error test.
 */
nag_zge_norm(order, Nag_OneNorm, n, n, a, pda, &anorm, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_gen_complx_mat_print_comp (x04dbc): Print matrix A. */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                              n, a, pda, Nag_BracketForm, "%7.4f",
                              "Matrix A", Nag_IntegerLabels, 0,
                              Nag_IntegerLabels, 0, 80, 0, 0, &fail);

printf("\n");
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Find the Schur factorization of A using nag_zgees (f08pnc). */
nag_zgees(order, Nag_Schur, Nag_NoSortEigVals, NULLFN, n, a, pda, &sdim, w,
          vs, pdvs, &fail);

```

```

if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgees (f08pnc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Reconstruct A from Schur Factorization Z*T*ConjTrans(Z) where T is upper
 * triangular and stored in A. This can be done using the following steps:
 * i. C = Z*T (nag_zgemm, f16zac),
 * ii. D = D-C*ConjTrans(Z) (nag_zgemm, f16zac).
 */
alpha = nag_complex(1.0, 0.0);
beta = nag_complex(0.0, 0.0);
nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, alpha, vs, pdvs, a, pda,
          beta, c, pdc, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_zgemm (f16zac):
 * Compute D = A - C*Z^H.
 */
alpha = nag_complex(-1.0, 0.0);
beta = nag_complex(1.0, 0.0);
nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alpha, c, pdc, vs,
          pdvs, beta, d, pdd, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_zge_norm (f16uac): Find norm of difference matrix D and print
 * warning if it is too large relative to norm of A.
 */
nag_zge_norm(order, Nag_OneNorm, n, n, d, pdd, &norm, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Get the machine precision, using nag_machine_precision (x02ajc) */
eps = nag_machine_precision;
if (norm > pow(eps, 0.8) * MAX(anorm, 1.0)) {
    printf("||A-(Z*T*Z^H)||/||A|| is larger than expected.\n"
          "Schur factorization has failed.\n");
    exit_status = 1;
    goto END;
}

/* Print details on eigenvalues */
printf("Eigenvalues\n");
for (i = 0; i < n; i++)
    printf("%4" NAG_IFMT " (%7.4f,%7.4f)\n", i + 1, w[i].re, w[i].im);

END:
NAG_FREE(a);
NAG_FREE(c);
NAG_FREE(d);
NAG_FREE(vs);
NAG_FREE(w);

return exit_status;
}

```

## 10.2 Program Data

nag\_zgees (f08pnc) 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\_zgees (f08pnc) Example Program Results

```

Matrix A
      1          2          3          4
1 (-3.9700,-5.0400) (-4.1100, 3.7000) (-0.3400, 1.0100) ( 1.2900,-0.8600)
2 ( 0.3400,-1.5000) ( 1.5200,-0.4300) ( 1.8800,-5.3800) ( 3.3600, 0.6500)
3 ( 3.3100,-3.8500) ( 2.5000, 3.4500) ( 0.8800,-1.0800) ( 0.6400,-1.4800)
4 (-1.1000, 0.8200) ( 1.8100,-1.5900) ( 3.2500, 1.3300) ( 1.5700,-3.4400)

```

Eigenvalues

```

1 (-6.0004,-6.9998)
2 (-5.0000, 2.0060)
3 ( 7.9982,-0.9964)
4 ( 3.0023,-3.9998)

```

---