

# NAG Library Function Document

## nag\_zhseqr (f08psc)

### 1 Purpose

nag\_zhseqr (f08psc) computes all the eigenvalues and, optionally, the Schur factorization of a complex Hessenberg matrix or a complex general matrix which has been reduced to Hessenberg form.

### 2 Specification

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

void nag_zhseqr (Nag_OrderType order, Nag_JobType job,
                Nag_ComputeZType compz, Integer n, Integer ilo, Integer ihi,
                Complex h[], Integer pdh, Complex w[], Complex z[], Integer pdz,
                NagError *fail)
```

### 3 Description

nag\_zhseqr (f08psc) computes all the eigenvalues and, optionally, the Schur factorization of a complex upper Hessenberg matrix  $H$ :

$$H = ZTZ^H,$$

where  $T$  is an upper triangular matrix (the Schur form of  $H$ ), and  $Z$  is the unitary matrix whose columns are the Schur vectors  $z_i$ . The diagonal elements of  $T$  are the eigenvalues of  $H$ .

The function may also be used to compute the Schur factorization of a complex general matrix  $A$  which has been reduced to upper Hessenberg form  $H$ :

$$\begin{aligned} A &= QHQ^H, \text{ where } Q \text{ is unitary,} \\ &= (QZ)T(QZ)^H. \end{aligned}$$

In this case, after nag\_zgehrd (f08nsc) has been called to reduce  $A$  to Hessenberg form, nag\_zunghr (f08ntc) must be called to form  $Q$  explicitly;  $Q$  is then passed to nag\_zhseqr (f08psc), which must be called with **compz** = Nag\_UpdateZ.

The function can also take advantage of a previous call to nag\_zgebal (f08nvc) which may have balanced the original matrix before reducing it to Hessenberg form, so that the Hessenberg matrix  $H$  has the structure:

$$\begin{pmatrix} H_{11} & H_{12} & H_{13} \\ & H_{22} & H_{23} \\ & & H_{33} \end{pmatrix}$$

where  $H_{11}$  and  $H_{33}$  are upper triangular. If so, only the central diagonal block  $H_{22}$  (in rows and columns  $i_{10}$  to  $i_{hi}$ ) needs to be further reduced to Schur form (the blocks  $H_{12}$  and  $H_{23}$  are also affected). Therefore the values of  $i_{10}$  and  $i_{hi}$  can be supplied to nag\_zhseqr (f08psc) directly. Also, nag\_zgebak (f08nwc) must be called after this function to permute the Schur vectors of the balanced matrix to those of the original matrix. If nag\_zgebal (f08nvc) has not been called however, then  $i_{10}$  must be set to 1 and  $i_{hi}$  to  $n$ . Note that if the Schur factorization of  $A$  is required, nag\_zgebal (f08nvc) must **not** be called with **job** = Nag\_Scale or Nag\_DoBoth, because the balancing transformation is not unitary.

nag\_zhseqr (f08psc) uses a multishift form of the upper Hessenberg  $QR$  algorithm, due to Bai and Demmel (1989). The Schur vectors are normalized so that  $\|z_i\|_2 = 1$ , but are determined only to within a complex factor of absolute value 1.

## 4 References

Bai Z and Demmel J W (1989) On a block implementation of Hessenberg multishift  $QR$  iteration *Internat. J. High Speed Comput.* **1** 97–112

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: **job** – Nag\_JobType *Input*  
*On entry:* indicates whether eigenvalues only or the Schur form  $T$  is required.  
**job** = Nag\_EigVals  
 Eigenvalues only are required.  
**job** = Nag\_Schur  
 The Schur form  $T$  is required.  
*Constraint:* **job** = Nag\_EigVals or Nag\_Schur.
- 3: **compz** – Nag\_ComputeZType *Input*  
*On entry:* indicates whether the Schur vectors are to be computed.  
**compz** = Nag\_NotZ  
 No Schur vectors are computed (and the array  $\mathbf{z}$  is not referenced).  
**compz** = Nag\_UpdateZ  
 The Schur vectors of  $A$  are computed (and the array  $\mathbf{z}$  must contain the matrix  $Q$  on entry).  
**compz** = Nag\_InitZ  
 The Schur vectors of  $H$  are computed (and the array  $\mathbf{z}$  is initialized by the function).  
*Constraint:* **compz** = Nag\_NotZ, Nag\_UpdateZ or Nag\_InitZ.
- 4: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $H$ .  
*Constraint:*  $\mathbf{n} \geq 0$ .
- 5: **ilo** – Integer *Input*  
 6: **ihi** – Integer *Input*  
*On entry:* if the matrix  $A$  has been balanced by nag\_zgebal (f08nvc), then **ilo** and **ihi** must contain the values returned by that function. Otherwise, **ilo** must be set to 1 and **ihi** to  $\mathbf{n}$ .  
*Constraint:*  $\mathbf{ilo} \geq 1$  and  $\min(\mathbf{ilo}, \mathbf{n}) \leq \mathbf{ihi} \leq \mathbf{n}$ .
- 7: **h**[*dim*] – Complex *Input/Output*  
**Note:** the dimension, *dim*, of the array  $\mathbf{h}$  must be at least  $\max(1, \mathbf{pdh} \times \mathbf{n})$ .

Where  $\mathbf{H}(i, j)$  appears in this document, it refers to the array element

$$\begin{aligned} & \mathbf{h}[(j-1) \times \mathbf{pdh} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{h}[(i-1) \times \mathbf{pdh} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*On entry:* the  $n$  by  $n$  upper Hessenberg matrix  $H$ , as returned by nag\_zgehrd (f08nsc).

*On exit:* if  $\mathbf{job} = \text{Nag\_EigVals}$ , the array contains no useful information.

If  $\mathbf{job} = \text{Nag\_Schur}$ ,  $\mathbf{h}$  is overwritten by the upper triangular matrix  $T$  from the Schur decomposition (the Schur form) unless  $\mathbf{fail.code} = \text{NE\_CONVERGENCE}$ .

8: **pdh** – Integer *Input*

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

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

9: **w**[*dim*] – Complex *Output*

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

*On exit:* the computed eigenvalues, unless  $\mathbf{fail.code} = \text{NE\_CONVERGENCE}$  (in which case see Section 6). The eigenvalues are stored in the same order as on the diagonal of the Schur form  $T$  (if computed).

10: **z**[*dim*] – Complex *Input/Output*

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

$$\begin{aligned} & \max(1, \mathbf{pdz} \times \mathbf{n}) \text{ when } \mathbf{compz} = \text{Nag\_UpdateZ} \text{ or } \text{Nag\_InitZ}; \\ & 1 \text{ when } \mathbf{compz} = \text{Nag\_NotZ}. \end{aligned}$$

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

$$\begin{aligned} & \mathbf{z}[(j-1) \times \mathbf{pdz} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{z}[(i-1) \times \mathbf{pdz} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*On entry:* if  $\mathbf{compz} = \text{Nag\_UpdateZ}$ ,  $\mathbf{z}$  must contain the unitary matrix  $Q$  from the reduction to Hessenberg form.

If  $\mathbf{compz} = \text{Nag\_InitZ}$ ,  $\mathbf{z}$  need not be set.

*On exit:* if  $\mathbf{compz} = \text{Nag\_UpdateZ}$  or  $\text{Nag\_InitZ}$ ,  $\mathbf{z}$  contains the unitary matrix of the required Schur vectors, unless  $\mathbf{fail.code} = \text{NE\_CONVERGENCE}$ .

If  $\mathbf{compz} = \text{Nag\_NotZ}$ ,  $\mathbf{z}$  is not referenced.

11: **pdz** – Integer *Input*

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

*Constraints:*

$$\begin{aligned} & \text{if } \mathbf{compz} = \text{Nag\_UpdateZ} \text{ or } \text{Nag\_InitZ}, \mathbf{pdz} \geq \max(1, \mathbf{n}); \\ & \text{if } \mathbf{compz} = \text{Nag\_NotZ}, \mathbf{pdz} \geq 1. \end{aligned}$$

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  $\langle value \rangle$  had an illegal value.

### NE\_CONVERGENCE

The algorithm has failed to find all the eigenvalues after a total of  $30(\mathbf{ihi} - \mathbf{ilo} + 1)$  iterations.

### NE\_ENUM\_INT\_2

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

Constraint: if  $\mathbf{compz} = \text{Nag\_UpdateZ}$  or  $\text{Nag\_InitZ}$ ,  $\mathbf{pdz} \geq \max(1, \mathbf{n})$ ;  
if  $\mathbf{compz} = \text{Nag\_NotZ}$ ,  $\mathbf{pdz} \geq 1$ .

### NE\_INT

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

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

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

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

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

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

### NE\_INT\_2

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

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

### NE\_INT\_3

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

Constraint:  $\mathbf{ilo} \geq 1$  and  $\min(\mathbf{ilo}, \mathbf{n}) \leq \mathbf{ihi} \leq \mathbf{n}$ .

### 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.

## 7 Accuracy

The computed Schur factorization is the exact factorization of a nearby matrix  $(H + E)$ , where

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

and  $\epsilon$  is the *machine precision*.

If  $\lambda_i$  is an exact eigenvalue, and  $\tilde{\lambda}_i$  is the corresponding computed value, then

$$|\tilde{\lambda}_i - \lambda_i| \leq \frac{c(n)\epsilon\|H\|_2}{s_i},$$

where  $c(n)$  is a modestly increasing function of  $n$ , and  $s_i$  is the reciprocal condition number of  $\lambda_i$ . The condition numbers  $s_i$  may be computed by calling `nag_ztrsna` (f08qyc).

## 8 Parallelism and Performance

`nag_zhseqr` (f08psc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_zhseqr` (f08psc) 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 real floating-point operations depends on how rapidly the algorithm converges, but is typically about:

$25n^3$  if only eigenvalues are computed;

$35n^3$  if the Schur form is computed;

$70n^3$  if the full Schur factorization is computed.

The real analogue of this function is `nag_dhseqr` (f08pec).

## 10 Example

This example computes all the eigenvalues and the Schur factorization of the upper Hessenberg matrix  $H$ , where

$$H = \begin{pmatrix} -3.9700 - 5.0400i & -1.1318 - 2.5693i & -4.6027 - 0.1426i & -1.4249 + 1.7330i \\ -5.4797 + 0.0000i & 1.8585 - 1.5502i & 4.4145 - 0.7638i & -0.4805 - 1.1976i \\ 0.0000 + 0.0000i & 6.2673 + 0.0000i & -0.4504 - 0.0290i & -1.3467 + 1.6579i \\ 0.0000 + 0.0000i & 0.0000 + 0.0000i & -3.5000 + 0.0000i & 2.5619 - 3.3708i \end{pmatrix}.$$

See also Section 10 in `nag_zunghr` (f08ntc), which illustrates the use of this function to compute the Schur factorization of a general matrix.

### 10.1 Program Text

```
/* nag_zhseqr (f08psc) 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 <nagf08.h>
#include <nagx04.h>
#include <nagf16.h>
```

```

#include <nagx02.h>

int main(void)
{
    /* Scalars */
    Complex alpha, beta;
    double norm;
    Integer i, j, n, pdc, pdd, pdh, pdz, w_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Complex *c = 0, *d = 0, *h = 0, *w = 0, *z = 0;

#ifdef NAG_COLUMN_MAJOR
#define H(I, J) h[(J-1)*pdh + I - 1]
#define D(I, J) d[(J-1)*pdd + I - 1]
    order = Nag_ColMajor;
#else
#define H(I, J) h[(I-1)*pdh + J - 1]
#define D(I, J) d[(I-1)*pdd + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    printf("nag_zhseqr (f08psc) 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
#ifdef NAG_COLUMN_MAJOR
    pdc = n;
    pdd = n;
    pdh = n;
    pdz = n;
#else
    pdc = n;
    pdd = n;
    pdh = n;
    pdz = n;
#endif
    w_len = n;

    /* Allocate memory */
    if (!(c = NAG_ALLOC(n * n, Complex)) ||
        !(d = NAG_ALLOC(n * n, Complex)) ||
        !(h = NAG_ALLOC(n * n, Complex)) ||
        !(w = NAG_ALLOC(w_len, Complex)) || !(z = NAG_ALLOC(n * n, Complex)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read H from data file */
    for (i = 1; i <= n; ++i) {
        for (j = 1; j <= n; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &H(i, j).re, &H(i, j).im);
#else
            scanf(" ( %lf , %lf )", &H(i, j).re, &H(i, j).im);
#endif
    }
}

```

```

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

/* Copy H into D */
for (i = 1; i <= n; ++i) {
    for (j = 1; j <= n; ++j) {
        D(i, j).re = H(i, j).re;
        D(i, j).im = H(i, j).im;
    }
}

/* nag_gen_complx_mat_print_comp (x04dbc): Print matrix H */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                              n, h, pdh, Nag_BracketForm, "%7.4f",
                              "Matrix H", 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;
}

/* Calculate the eigenvalues and Schur factorization of H */
/* nag_zhseqr (f08psc).
 * Eigenvalues and Schur factorization of complex upper
 * Hessenberg matrix reduced from complex general matrix
 */
nag_zhseqr(order, Nag_Schur, Nag_InitZ, n, 1, n, h, pdh, w, z, pdz, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zhseqr (f08psc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_zgemm (f16zac): Compute A - Z* $T$ * $Z^H$  from the factorization of */
/* A and store in matrix D */
alpha.re = 1.0;
alpha.im = 0.0;
beta.re = 0.0;
beta.im = 0.0;
nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, alpha, z, pdz,
          h, pdh, 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;
}
alpha.re = -1.0;
beta.re = 1.0;
nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alpha, c, pdc,
          z, pdz, 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 matrix D and print warning if */
/* it is too large */
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;
}

```

```

if (norm > pow(x02ajc(), 0.8)) {
    printf("%s\n%s\n", " Norm of H-(Z*T*Z^H) is much greater than 0.",
           " Schur factorization has failed.");
}
else {
    printf(" Eigenvalues\n");
    for (i = 1; i <= n; ++i)
        printf("%7.4f,%7.4f ", w[i - 1].re, w[i - 1].im);
    printf("\n\n");
}

END:
    NAG_FREE(c);
    NAG_FREE(d);
    NAG_FREE(h);
    NAG_FREE(w);
    NAG_FREE(z);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_zhseqr (f08psc) Example Program Data
4
(-3.9700,-5.0400) (-1.1318,-2.5693) (-4.6027,-0.1426) (-1.4249, 1.7330)
(-5.4797, 0.0000) ( 1.8585,-1.5502) ( 4.4145,-0.7638) (-0.4805,-1.1976)
( 0.0000, 0.0000) ( 6.2673, 0.0000) (-0.4504,-0.0290) (-1.3467, 1.6579)
( 0.0000, 0.0000) ( 0.0000, 0.0000) (-3.5000, 0.0000) ( 2.5619,-3.3708)
:End of matrix H
:Value of N

```

## 10.3 Program Results

```

nag_zhseqr (f08psc) Example Program Results

Matrix H
      1          2          3          4
1 (-3.9700,-5.0400) (-1.1318,-2.5693) (-4.6027,-0.1426) (-1.4249, 1.7330)
2 (-5.4797, 0.0000) ( 1.8585,-1.5502) ( 4.4145,-0.7638) (-0.4805,-1.1976)
3 ( 0.0000, 0.0000) ( 6.2673, 0.0000) (-0.4504,-0.0290) (-1.3467, 1.6579)
4 ( 0.0000, 0.0000) ( 0.0000, 0.0000) (-3.5000, 0.0000) ( 2.5619,-3.3708)

Eigenvalues
(-6.0004,-6.9998) (-5.0000, 2.0060) ( 7.9982,-0.9964) ( 3.0023,-3.9998)

```

---