NAG C 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

```
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_{lo} 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_{lo} 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_{lo} 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_Schur** 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 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

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, as follows:

if **job** = **Nag_EigVals**, eigenvalues only are required;

if **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 as follows:

if **compz** = **Nag_NotZ**, no Schur vectors are computed (and the array **z** is not referenced);

if **compz** = **Nag_InitZ**, the Schur vectors of H are computed (and the array **z** is initialised by the routine);

if **compz** = **Nag_UpdateZ**, the Schur vectors of A are computed (and the array **z** must contain the matrix Q on entry).

Constraint: **compz** = **Nag_NotZ**, **Nag_InitZ** or **Nag_UpdateZ**.

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 **n**.

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

7: $\mathbf{h}[dim]$ – Complex *Input/Output*

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

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix H is stored in $\mathbf{h}[(j-1) \times \mathbf{pdh} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix H is stored in $\mathbf{h}[(i-1) \times \mathbf{pdh} + j - 1]$.

On entry: the n by n upper Hessenberg matrix H, as returned by nag_zgehrd (f08nsc).

On exit: if **job** = **Nag_EigVals**, the array contains no useful information. If **job** = **Nag_Schur**, H is overwritten by the upper triangular matrix T from the Schur decomposition (the Schur form) unless $\mathbf{fail} > 0$.

8: **pdh** – Integer *Input*

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

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

9: **w**[dim] – Complex *Output*

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

On exit: the computed eigenvalues, unless **fail** > 0 (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 **z** must be at least
$\max(1, \mathbf{pdz} \times \mathbf{n})$ when **compz** = **Nag_UpdateZ** or **Nag_InitZ**;
1 when **compz** = **Nag_NotZ**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix Z is stored in **z**$[(j-1) \times \mathbf{pdz} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix Z is stored in **z**$[(i-1) \times \mathbf{pdz} + j - 1]$.

On entry: if **compz** = **Nag_UpdateZ**, **z** must contain the unitary matrix Q from the reduction to Hessenberg form; if **compz** = **Nag_InitZ**, **z** need not be set.

On exit: if **compz** = **Nag_UpdateZ** or **Nag_InitZ**, **z** contains the unitary matrix of the required Schur vectors, unless **fail** > 0.

z is not referenced if **compz** = **Nag_NotZ**.

11: **pdz** – Integer *Input*

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

Constraints:

if **compz** = **Nag_UpdateZ** or **Nag_InitZ**, **pdz** $\geq \max(1, \mathbf{n})$;
if **compz** = **Nag_NotZ**, **pdz** ≥ 1.

12: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

NE_INT_2

On entry, **pdh** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: **pdh** $\geq \max(1, \mathbf{n})$.

NE_INT_3

On entry, **n** = ⟨*value*⟩, **ilo** = ⟨*value*⟩, **ihi** = ⟨*value*⟩.
Constraint: **ilo** ≥ 1 and $\min(\mathbf{ilo}, \mathbf{n}) \leq \mathbf{ihi} \leq \mathbf{n}$.

NE_ENUM_INT_2

On entry, **compz** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdz** = ⟨*value*⟩.
Constraint: if **compz** = **Nag_UpdateZ** or **Nag_InitZ**, **pdz** $\geq \max(1, \mathbf{n})$;
if **compz** = **Nag_NotZ**, **pdz** ≥ 1.

NE_CONVERGENCE

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ had an illegal value.

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 consult NAG for assistance.

7 Accuracy

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

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

and ϵ is the ***machine precision***.

If λ_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 λ_i. The condition numbers s_i may be computed by calling nag_ztrsna (f08qyc).

8 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).

9 Example

To compute 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 nag_zunghr (f08ntc), which illustrates the use of this function to compute the Schur factorization of a general matrix.

9.1 Program Text

```
/* nag_zhseqr (f08psc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer  i, j, n, pdh, pdz, w_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex *h=0, *w=0, *z=0;

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

  INIT_FAIL(fail);
  Vprintf("f08psc Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
  pdh = n;
  pdz = n;
#else
  pdh = n;
  pdz = n;
#endif
  w_len = n;

  /* Allocate memory */
  if ( !(h = NAG_ALLOC(n * n, Complex)) ||
       !(w = NAG_ALLOC(w_len, Complex)) ||
       !(z = NAG_ALLOC(n * n, Complex)) )
    {
      Vprintf("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)
        Vscanf(" ( %lf , %lf )", &H(i,j).re, &H(i,j).im);
    }
  Vscanf("%*[^\n] ");

  /* Calculate the eigenvalues and Schur factorization of H */
  f08psc(order, Nag_Schur, Nag_InitZ, n, 1, n, h, pdh,
         w, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
```

```
      Vprintf("Error from f08psc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  Vprintf(" Eigenvalues\n");
  for (i = 1; i <= n; ++i)
    Vprintf("(%7.4f,%7.4f) ", w[i-1].re, w[i-1].im);
  Vprintf("\n\n");

  /* Print Schur form */
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         h, pdh, Nag_BracketForm, "%7.4f", "Schur form",
         Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
         0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
      /* Print Schur vectors */
  Vprintf("\n");
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         z, pdz, Nag_BracketForm, "%7.4f",
         "Schur vectors of H", Nag_IntegerLabels, 0,
         Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (h) NAG_FREE(h);
  if (w) NAG_FREE(w);
  if (z) NAG_FREE(z);

  return exit_status;
}
```

9.2 Program Data

```
f08psc Example Program Data
  4                                                              :Value of N
 (-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
```

9.3 Program Results

```
f08psc Example Program Results

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

 Schur form
                  1                  2                  3                  4
 1  (-6.0004,-6.9998)  (-0.2080, 0.4719)  (-0.4829, 0.1768)  ( 0.1301, 0.9052)
 2  ( 0.0000, 0.0000)  (-5.0000, 2.0060)  (-0.6653, 0.2814)  ( 0.0038, 0.2639)
 3  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 7.9982,-0.9964)  ( 0.2004, 1.0595)
 4  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 3.0023,-3.9998)

 Schur vectors of H
                  1                  2                  3                  4
 1  ( 0.8457, 0.0000)  ( 0.1380, 0.3602)  (-0.2677,-0.1091)  (-0.2213,-0.0582)
 2  ( 0.2824,-0.3304)  (-0.4612, 0.2075)  ( 0.6846, 0.0000)  ( 0.2927, 0.0320)
 3  ( 0.0748, 0.2800)  ( 0.7239, 0.0000)  ( 0.5924,-0.0189)  (-0.0229, 0.2005)
 4  ( 0.0670, 0.0860)  ( 0.2169, 0.1560)  (-0.2745, 0.1454)  ( 0.9057, 0.0000)
```