NAG C Library Function Document

nag_dhseqr (f08pec)

1 Purpose

nag_dhseqr (f08pec) computes all the eigenvalues, and optionally the Schur factorization, of a real Hessenberg matrix or a real general matrix which has been reduced to Hessenberg form.

2 Specification

```
void nag_dhseqr (Nag_OrderType order, Nag_JobType job, Nag_ComputeZType compz,
     Integer n, Integer ilo, Integer ihi, double h[], Integer pdh, double wr[],
     double wi[], double z[], Integer pdz, NagError *fail)
```

3 Description

nag_dhseqr (f08pec) computes all the eigenvalues, and optionally the Schur factorization, of a real upper Hessenberg matrix H:

$$H = ZTZ^T,$$

where T is an upper quasi-triangular matrix (the Schur form of H), and Z is the orthogonal matrix whose columns are the Schur vectors z_i. See Section 8 for details of the structure of T.

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

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

In this case, after nag_dgehrd (f08nec) has been called to reduce A to Hessenberg form, nag_dorghr (f08nfc) must be called to form Q explicitly; Q is then passed to nag_dhseqr (f08pec), which must be called with **compz** = **Nag_UpdateZ**.

The function can also take advantage of a previous call to nag_dgebal (f08nhc) 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_dhseqr (f08pec) directly. Also, nag_dgebak (f08njc) must be called after this function to permute the Schur vectors of the balanced matrix to those of the original matrix. If nag_dgebal (f08nhc) 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_dgebal (f08nhc) must **not** be called with **job** = **Nag_Schur** or **Nag_DoBoth**, because the balancing transformation is not orthogonal.

nag_dhseqr (f08pec) 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 factor ± 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: **n** ≥ 0.

5: **ilo** – Integer *Input*
6: **ihi** – Integer *Input*

On entry: if the matrix A has been balanced by nag_dgebal (f08nhc), then **ilo** and **ihi** must contain the values returned by that function. Otherwise, **ilo** must be set to 1 and **ihi** to **n**.

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

7: **h**[dim] – double *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_dgehrd (f08nec).

On exit: if **job** = **Nag_EigVals**, then the array contains no useful information. If **job** = **Nag_Schur**, then H is overwritten by the upper quasi-triangular matrix T from the Schur decomposition (the Schur form) unless **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: **wr**[dim] – double *Output*
10: **wi**[dim] – double *Output*

Note: the dimensions, dim, of the arrays **wr** and **wi** must each be at least $\max(1, \mathbf{n})$.

On exit: the real and imaginary parts, respectively, of the computed eigenvalues, unless **fail** > 0 (in which case see Section 6). Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having positive imaginary part first. The eigenvalues are stored in the same order as on the diagonal of the Schur form T (if computed); see Section 8 for details.

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

On entry: if **compz** = **Nag_UpdateZ**, **z** must contain the orthogonal 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 orthogonal matrix of the required Schur vectors, unless **fail** > 0.

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

12: **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.

13: **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_dtrsna (f08qlc).

8 Further Comments

The total number of floating-point operations depends on how rapidly the algorithm converges, but is typically about:

$7n^3$ if only eigenvalues are computed;

$10n^3$ if the Schur form is computed;

$20n^3$ if the full Schur factorization is computed.

The Schur form T has the following structure (referred to as **canonical** Schur form).

If all the computed eigenvalues are real, T is upper triangular, and the diagonal elements of T are the eigenvalues; $\mathbf{wr}[i] = t_{ii}$, for $i = 1, 2, \ldots, n$ and $\mathbf{wi}[i] = 0.0$.

If some of the computed eigenvalues form complex conjugate pairs, then T has 2 by 2 diagonal blocks. Each diagonal block has the form

$$\begin{pmatrix} t_{ii} & t_{i,i+1} \\ t_{i+1,i} & t_{i+1,i+1} \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \gamma & \alpha \end{pmatrix}$$

where $\beta\gamma < 0$. The corresponding eigenvalues are $\alpha \pm \sqrt{\beta\gamma}$; $\mathbf{wr}[i] = \mathbf{wr}[i+1] = \alpha$; $\mathbf{wi}[i] = +\sqrt{|\beta\gamma|}$; $\mathbf{wi}[i+1] = -\mathbf{wi}[i]$.

The complex analogue of this function is nag_zhseqr (f08psc).

9 Example

To compute all the eigenvalues and the Schur factorization of the upper Hessenberg matrix H, where

$$H = \begin{pmatrix} 0.3500 & -0.1160 & -0.3886 & -0.2942 \\ -0.5140 & 0.1225 & 0.1004 & 0.1126 \\ 0.0000 & 0.6443 & -0.1357 & -0.0977 \\ 0.0000 & 0.0000 & 0.4262 & 0.1632 \end{pmatrix}.$$

See also the example for nag_dorghr (f08nfc), which illustrates the use of this function to compute the Schur factorization of a general matrix.

9.1 Program Text

```
/* nag_dhseqr (f08pec) 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, wi_len, wr_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double *h=0, *wi=0, *wr=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("f08pec 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
  wr_len = n;
  wi_len = n;

  /* Allocate memory */
  if ( !(h = NAG_ALLOC(n * n, double)) ||
```

```
       !(wi = NAG_ALLOC(wi_len, double)) ||
       !(wr = NAG_ALLOC(wr_len, double)) ||
       !(z = NAG_ALLOC(n * n, double)) )
    {
      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", &H(i,j));
    }
  Vscanf("%*[^\n] ");

  /* Calculate the eigenvalues and Schur factorization of H */
  f08pec(order, Nag_Schur, Nag_InitZ, n, 1, n, h, pdh, wr,
         wi, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08pec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  Vprintf(" Eigenvalues\n");
  for (i = 1; i <= n; ++i)
    Vprintf(" (%8.4f,%8.4f)", wr[i-1], wi[i-1]);
  Vprintf("\n");

  /* Print Schur form */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         h, pdh, "Schur form", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
      /* Print Schur vectors */
  Vprintf("\n");
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         z, pdz, "Schur vectors of H", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (h) NAG_FREE(h);
  if (wi) NAG_FREE(wi);
  if (wr) NAG_FREE(wr);
  if (z) NAG_FREE(z);

  return exit_status;
}
```

9.2 Program Data

```
f08pec Example Program Data
  4                                    :Value of N
  0.3500  -0.1160  -0.3886  -0.2942
 -0.5140   0.1225   0.1004   0.1126
  0.0000   0.6443  -0.1357  -0.0977
  0.0000   0.0000   0.4262   0.1632   :End of matrix H
```

9.3 Program Results

```
f08pec Example Program Results

 Eigenvalues
 (  0.7995,  0.0000) ( -0.0994,  0.4008) ( -0.0994, -0.4008) ( -0.1007,  0.0000)
 Schur form
          1        2        3        4
 1   0.7995   0.0061  -0.1144  -0.0335
 2   0.0000  -0.0994  -0.6483  -0.2026
 3   0.0000   0.2477  -0.0994  -0.3474
 4   0.0000   0.0000   0.0000  -0.1007

 Schur vectors of H
          1        2        3        4
 1  -0.6551  -0.3450  -0.1036   0.6641
 2   0.5972  -0.1706   0.5246   0.5823
 3   0.3845  -0.7143  -0.5789  -0.0821
 4   0.2576   0.5845  -0.6156   0.4616
```