NAG 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

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

```
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:

$$H = ZTZ^{\mathrm{T}},$$

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 9 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:

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

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_Scale 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 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 **z** is not referenced).

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

compz = Nag_InitZ
The Schur vectors of H are computed (and the array **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: **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 $\max(1, \mathbf{pdh} \times \mathbf{n})$.

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

$\mathbf{h}[(j-1) \times \mathbf{pdh} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{h}[(i-1) \times \mathbf{pdh} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by n upper Hessenberg matrix H, as returned by nag_dgehrd (f08nec).

On exit: if **job** = Nag_EigVals, the array contains no useful information.

If **job** = Nag_Schur, **h** is overwritten by the upper quasi-triangular matrix T from the Schur decomposition (the Schur form) unless **fail.code** = NE_CONVERGENCE.

8: **pdh** – Integer *Input*

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

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

9: **wr**[*dim*] – double *Output*
10: **wi**[*dim*] – double *Output*

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

On exit: the real and imaginary parts, respectively, of the computed eigenvalues, unless **fail.code** = NE_CONVERGENCE (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 9 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.

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

$\mathbf{z}[(j-1) \times \mathbf{pdz} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{z}[(i-1) \times \mathbf{pdz} + j - 1]$ when **order** = Nag_RowMajor.

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.code** = NE_CONVERGENCE.

If **compz** = Nag_NotZ, **z** is not referenced.

12: **pdz** – Integer *Input*

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

Constraints:

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

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

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **pdh** = ⟨*value*⟩.
Constraint: $\mathbf{pdh} > 0$.

On entry, **pdz** = ⟨*value*⟩.
Constraint: $\mathbf{pdz} > 0$.

NE_INT_2

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

NE_INT_3

On entry, **n** = ⟨*value*⟩, **ilo** = ⟨*value*⟩ and **ihi** = ⟨*value*⟩.
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 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 Parallelism and Performance

nag_dhseqr (f08pec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dhseqr (f08pec) 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 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; $\mathbf{wr}[i-1] = t_{ii}$, for $i = 1, 2, \ldots, n$, and $\mathbf{wi}[i-1] = 0.0$.

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 $\alpha \pm \sqrt{\beta\gamma}$; $\mathbf{wr}[i-1] = \mathbf{wr}[i] = \alpha$; $\mathbf{wi}[i-1] = +\sqrt{|\beta\gamma|}$; $\mathbf{wi}[i] = -\mathbf{wi}[i-1]$.

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

10 Example

This example computes 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 Section 10 in nag_dorghr (f08nfc), which illustrates the use of this function to compute the Schur factorization of a general matrix.

10.1 Program Text

```
/* nag_dhseqr (f08pec) 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 <nagf16.h>
#include <nagx02.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double alpha, beta, norm;
  Integer i, j, n, pdc, pdd, pdh, pdz, wi_len, wr_len;
  Integer exit_status = 0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double *c = 0, *d = 0, *h = 0, *wi = 0, *wr = 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_dhseqr (f08pec) 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
  wr_len = n;
  wi_len = n;

  /* Allocate memory */
```

```
  if (!(c = NAG_ALLOC(n * n, double)) ||
      !(d = NAG_ALLOC(n * n, double)) ||
      !(h = NAG_ALLOC(n * n, double)) ||
      !(wi = NAG_ALLOC(wi_len, double)) ||
      !(wr = NAG_ALLOC(wr_len, double)) || !(z = NAG_ALLOC(n * n, double)))
  {
    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", &H(i, j));
#else
      scanf("%lf", &H(i, j));
#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) = H(i, j);
  }

  /* nag_gen_real_mat_print (x04cac): Print Matrix H. */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                         h, pdh, "Matrix A", 0, &fail);
  printf("\n");
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Calculate the eigenvalues and Schur factorization of H */
  /* nag_dhseqr (f08pec).
   * Eigenvalues and Schur factorization of real upper
   * Hessenberg matrix reduced from real general matrix
   */
  nag_dhseqr(order, Nag_Schur, Nag_InitZ, n, 1, n, h, pdh, wr,
             wi, z, pdz, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dhseqr (f08pec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_dgemm (f16yac): Compute H - Z*T*Z^T from the factorization of */
  /* H and store in matrix D */
  alpha = 1.0;
  beta = 0.0;
  nag_dgemm(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_dgemm (f16yac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  alpha = -1.0;
  beta = 1.0;
  nag_dgemm(order, Nag_NoTrans, Nag_Trans, n, n, n, alpha, c, pdc, z,
            pdz, beta, d, pdd, &fail);
```

```
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgemm (f16yac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_dge_norm (f16rac): Find norm of matrix D and print warning if */
  /* it is too large */
  nag_dge_norm(order, Nag_OneNorm, n, n, d, pdd, &norm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  if (norm > pow(x02ajc(), 0.8)) {
    printf("\n%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(" (%8.4f,%8.4f)", wr[i - 1], wi[i - 1]);
    printf("\n");
  }

END:
  NAG_FREE(c);
  NAG_FREE(d);
  NAG_FREE(h);
  NAG_FREE(wi);
  NAG_FREE(wr);
  NAG_FREE(z);

  return exit_status;
}
```

10.2 Program Data

```
nag_dhseqr (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
```

10.3 Program Results

```
nag_dhseqr (f08pec) Example Program Results

 Matrix A
          1       2       3       4
 1   0.3500 -0.1160 -0.3886 -0.2942
 2  -0.5140  0.1225  0.1004  0.1126
 3   0.0000  0.6443 -0.1357 -0.0977
 4   0.0000  0.0000  0.4262  0.1632

 Eigenvalues
 (  0.7995,  0.0000) ( -0.0994,  0.4008) ( -0.0994, -0.4008) ( -0.1007,  0.0000)
```