NAG Library Function Document

nag_zhsein (f08pxc)

1 Purpose

nag_zhsein (f08pxc) computes selected left and/or right eigenvectors of a complex upper Hessenberg matrix corresponding to specified eigenvalues, by inverse iteration.

2 Specification

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

void nag_zhsein (Nag_OrderType order, Nag_SideType side,
                 Nag_EigValsSourceType eig_source, Nag_InitVeenumtype initv,
                 const Nag_Boolean select[], Integer n, const Complex h[], Integer pdh,
                 Complex w[], Complex vl[], Integer pdvl, Complex vr[], Integer pdvr,
                 Integer mm, Integer *m, Integer ifaill[], Integer ifailr[],
                 NagError *fail)
```

3 Description

nag_zhsein (f08pxc) computes left and/or right eigenvectors of a complex upper Hessenberg matrix $H$, corresponding to selected eigenvalues.

The right eigenvector $x$, and the left eigenvector $y$, corresponding to an eigenvalue $\lambda$, are defined by:

$$ Hx = \lambda x \quad \text{and} \quad y^H H = \lambda y^H \quad \text{(or} \quad H^H y = \bar{\lambda} y). $$

The eigenvectors are computed by inverse iteration. They are scaled so that $\max(|\Re(x_i)| + |\Im(x_i)|) = 1$.

If $H$ has been formed by reduction of a complex general matrix $A$ to upper Hessenberg form, then the eigenvectors of $H$ may be transformed to eigenvectors of $A$ by a call to nag_zunmhr (f08nuc).

4 References


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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2:  **side** – Nag_SideType

   *Input*

   On entry: indicates whether left and/or right eigenvectors are to be computed.

   side = Nag_RightSide
   Only right eigenvectors are computed.

   side = Nag_LeftSide
   Only left eigenvectors are computed.
side = Nag BothSides
    Both left and right eigenvectors are computed.

Constraint: side = Nag RightSide, Nag LeftSide or Nag BothSides.

3:   eig_source – Nag EigValsSourceType  \(\text{Input}\)
    On entry: indicates whether the eigenvalues of \(H\) (stored in \(w\)) were found using nag zhseqr (f08psc).

\(\text{eig_source} = \text{Nag HSEQRSource}\)
    The eigenvalues of \(H\) were found using nag zhseqr (f08psc); thus if \(H\) has any zero subdiagonal elements (and so is block triangular), then the \(j\)th eigenvalue can be assumed to be an eigenvalue of the block containing the \(j\)th row/column. This property allows the function to perform inverse iteration on just one diagonal block.

\(\text{eig_source} = \text{Nag NotKnown}\)
    No such assumption is made and the function performs inverse iteration using the whole matrix.

Constraint: eig_source = Nag HSEQRSource or Nag NotKnown.

4:   initv – Nag InitVeenumtype  \(\text{Input}\)
    On entry: indicates whether you are supplying initial estimates for the selected eigenvectors.

\(\text{initv} = \text{Nag NoVec}\)
    No initial estimates are supplied.

\(\text{initv} = \text{Nag UserVec}\)
    Initial estimates are supplied in \(vl\) and/or \(vr\).

Constraint: initv = Nag NoVec or Nag UserVec.

5:   select[dim] – const Nag Boolean  \(\text{Input}\)
    Note: the dimension, \(dim\), of the array select must be at least \(\max(1,n)\).
    On entry: specifies which eigenvectors are to be computed. To select the eigenvector corresponding to the eigenvalue \(w[j-1]\), \(\text{select}[j-1]\) must be set to Nag_TRUE.

6:   n – Integer  \(\text{Input}\)
    On entry: \(n\), the order of the matrix \(H\).
    Constraint: \(n \geq 0\).

7:   h[dim] – const Complex  \(\text{Input}\)
    Note: the dimension, \(dim\), of the array \(h\) must be at least \(\max(1,\text{pdh} \times n)\).
    The \((i,j)\)th element of the matrix \(H\) is stored in
    \[ h[(j-1) \times \text{pdh} + i - 1] \text{ when order = Nag ColMajor} \]
    \[ h[(i-1) \times \text{pdh} + j - 1] \text{ when order = Nag RowMajor} \]
    On entry: the \(n\) by \(n\) upper Hessenberg matrix \(H\).

8:   pdh – Integer  \(\text{Input}\)
    On entry: the stride separating row or column elements (depending on the value of order) in the array \(h\).
    Constraint: \(\text{pdh} \geq \max(1,n)\).

9:   w[dim] – Complex  \(\text{Input/Output}\)
    Note: the dimension, \(dim\), of the array \(w\) must be at least \(\max(1,n)\).
On entry: the eigenvalues of the matrix $H$. If `eig_source` = Nag_HSEQRSource, the array must be exactly as returned by nag_zhseqr (f08psc).

On exit: the real parts of some elements of $w$ may be modified, as close eigenvalues are perturbed slightly in searching for independent eigenvectors.

10: $vl[dim]$ – Complex 

**Input/Output**

Note: the dimension, $dim$, of the array $vl$ must be at least

$$\max(1, pdvl \times mm)$$ when $side$ = Nag_LeftSide or Nag_BothSides and  
$order = Nag_ColMajor$;

$$\max(1, n \times pdvl)$$ when $side$ = Nag_LeftSide or Nag_BothSides and  
$order = Nag_RowMajor$;

1 when $side$ = Nag_RightSide.

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

$$vl[(j - 1) \times pdvl + i - 1]$$ when $order = Nag_ColMajor$;

$$vl[(i - 1) \times pdvl + j - 1]$$ when $order = Nag_RowMajor$.

On entry: if `initv` = Nag_UserVec and $side$ = Nag_LeftSide or Nag_BothSides, $vl$ must contain starting vectors for inverse iteration for the left eigenvectors. Each starting vector must be stored in the same row or column as will be used to store the corresponding eigenvector (see below).

If `initv` = Nag_NoVec, $vl$ need not be set.

On exit: if $side$ = Nag_LeftSide or Nag_BothSides, $vl$ contains the computed left eigenvectors (as specified by `select`). The eigenvectors are stored consecutively in the rows or columns of the array (depending on the value of $order$), in the same order as their eigenvalues.

If $side$ = Nag_RightSide, $vl$ is not referenced.

11: $pdvl$ – Integer 

**Input**

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

**Constraints:**

- if $order = Nag_ColMajor$,
- if $side = Nag_LeftSide$ or Nag_BothSides, $pdvl \geq n$;
- if $side = Nag_RightSide$, $pdvl \geq 1$;
- if $order = Nag_RowMajor$,
- if $side = Nag_LeftSide$ or Nag_BothSides, $pdvl \geq \max(1, mm)$;
- if $side = Nag_RightSide$, $pdvl \geq 1$.

12: $vr[dim]$ – Complex 

**Input/Output**

Note: the dimension, $dim$, of the array $vr$ must be at least

$$\max(1, pdvr \times mm)$$ when $side$ = Nag_RightSide or Nag_BothSides and  
$order = Nag_ColMajor$;

$$\max(1, n \times pdvr)$$ when $side$ = Nag_RightSide or Nag_BothSides and  
$order = Nag_RowMajor$;

1 when $side$ = Nag_LeftSide.

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

$$vr[(j - 1) \times pdvr + i - 1]$$ when $order = Nag_ColMajor$;

$$vr[(i - 1) \times pdvr + j - 1]$$ when $order = Nag_RowMajor$. 

Mark 25
On entry: if **initv** = Nag_UserVec and **side** = Nag_RightSide or Nag_BothSides, **vr** must contain starting vectors for inverse iteration for the right eigenvectors. Each starting vector must be stored in the same row or column as will be used to store the corresponding eigenvector (see below).

If **initv** = Nag_NoVec, **vr** need not be set.

On exit: if **side** = Nag_RightSide or Nag_BothSides, **vr** contains the computed right eigenvectors (as specified by **select**). The eigenvectors are stored consecutively in the rows or columns of the array (depending on the value of **order**), in the same order as their eigenvalues.

If **side** = Nag_LeftSide, **vr** is not referenced.

---

13: **pdvr** – Integer  

**Input**  

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

**Constraints:**

- If **order** = Nag_ColMajor,
  - if **side** = Nag_RightSide or Nag_BothSides, **pdvr** ≥ **n**;
  - if **side** = Nag_LeftSide, **pdvr** ≥ 1.;

- If **order** = Nag_RowMajor,
  - if **side** = Nag_RightSide or Nag_BothSides, **pdvr** ≥ max(1, **mm**);
  - if **side** = Nag_LeftSide, **pdvr** ≥ 1.

14: **mm** – Integer  

**Input**  

On entry: the number of columns in the arrays **vl** and/or **vr** if **order** = Nag_ColMajor or the number of rows in the arrays if **order** = Nag_RowMajor. The actual number of rows or columns required, **requiredowcol**, is obtained by counting 1 for each selected real eigenvector and 2 for each selected complex eigenvector (see **select**); 0 ≤ **requiredowcol** ≤ **n**.

**Constraint:** **mm** ≥ **requiredowcol**.

15: **m** – Integer*  

**Output**  

On exit: **requiredowcol**, the number of selected eigenvectors.

16: **ifail**[**dim**] – Integer  

**Output**  

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

- max(1, **mm**) when **side** = Nag_LeftSide or Nag_BothSides;
- 1 when **side** = Nag_RightSide.

**On exit:** if **side** = Nag_LeftSide or Nag_BothSides, then **ifail**[i − 1] = 0 if the selected left eigenvector converged and **ifail**[i − 1] = j ≥ 0 if the eigenvector stored in the i-th row or column of **vl** (corresponding to the j-th eigenvalue) failed to converge.

If **side** = Nag_RightSide, **ifail** is not referenced.

17: **ifailr**[**dim**] – Integer  

**Output**  

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

- max(1, **mm**) when **side** = Nag_RightSide or Nag_BothSides;
- 1 when **side** = Nag_LeftSide.

**On exit:** if **side** = Nag_RightSide or Nag_BothSides, then **ifailr**[i − 1] = 0 if the selected right eigenvector converged and **ifailr**[i − 1] = j ≥ 0 if the eigenvector stored in the i-th column of **vr** (corresponding to the j-th eigenvalue) failed to converge.
If side = Nag_LeftSide, ifailr is not referenced.

18: fail – NagError *

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**

On entry, argument \langle value \rangle had an illegal value.

**NE_CONVERGENCE**

\langle value \rangle eigenvectors (as indicated by arguments ifail and/or ifailr) failed to converge. The corresponding columns of vl and/or vr contain no useful information.

**NE_ENUM_INT_2**

On entry, side = \langle value \rangle, pdvl = \langle value \rangle, m = \langle value \rangle.

Constraint: if side = Nag_LeftSide or Nag_BothSides, pdvl ≥ max(1, m); if side = Nag_RightSide, pdvl ≥ 1.

On entry, side = \langle value \rangle, pdvl = \langle value \rangle, mm = \langle value \rangle.

Constraint: if side = Nag_LeftSide or Nag_BothSides, pdvl ≥ max(1, mm); if side = Nag_RightSide, pdvl ≥ 1.

On entry, side = \langle value \rangle, pdvl = \langle value \rangle and n = \langle value \rangle.

Constraint: if side = Nag_LeftSide or Nag_BothSides, pdvl ≥ n; if side = Nag_RightSide, pdvl ≥ 1.

On entry, side = \langle value \rangle, pdvr = \langle value \rangle, m = \langle value \rangle.

Constraint: if side = Nag_RightSide or Nag_BothSides, pdvr ≥ max(1, m); if side = Nag_LeftSide, pdvr ≥ 1.

On entry, side = \langle value \rangle, pdvr = \langle value \rangle, mm = \langle value \rangle.

Constraint: if side = Nag_RightSide or Nag_BothSides, pdvr ≥ max(1, mm); if side = Nag_LeftSide, pdvr ≥ 1.

On entry, side = \langle value \rangle, pdvr = \langle value \rangle and n = \langle value \rangle.

Constraint: if side = Nag_RightSide or Nag_BothSides, pdvr ≥ n; if side = Nag_LeftSide, pdvr ≥ 1.

**NE_INT**

On entry, mm = \langle value \rangle.

Constraint: mm ≥ required, owcol, where required, owcol is the number of selected eigenvectors.

On entry, n = \langle value \rangle.

Constraint: n ≥ 0.

On entry, pdh = \langle value \rangle.

Constraint: pdh > 0.

On entry, pdvl = \langle value \rangle.

Constraint: pdvl > 0.

On entry, pdvr = \langle value \rangle.

Constraint: pdvr > 0.
NE_INT_2
On entry, \( \text{pdh} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdh} \geq \max(1, n) \).

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 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy
Each computed right eigenvector \( x_i \) is the exact eigenvector of a nearby matrix \( A + E_i \), such that \( \| E_i \| = O(\varepsilon) \| A \| \). Hence the residual is small:
\[
\| A x_i - \lambda_i x_i \| = O(\varepsilon) \| A \| .
\]
However, eigenvectors corresponding to close or coincident eigenvalues may not accurately span the relevant subspaces.
Similar remarks apply to computed left eigenvectors.

8 Parallelism and Performance
\text{nag_zhsein (f08pxc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
\text{nag_zhsein (f08pxc)} 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 real analogue of this function is \text{nag_dhsein (f08pkc)}.

10 Example
See Section 10 in \text{nag_zunmhr (f08nuc)}. 