NAG Library Function Document

nag_ztrevc (f08qxc)

1 Purpose

nag_ztrevc (f08qxc) computes selected left and/or right eigenvectors of a complex upper triangular matrix.

2 Specification

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

```
void nag_ztrevc (Nag_OrderType order, Nag_SideType side,
     Nag_HowManyType how_many, const Nag_Boolean select[], Integer n,
     Complex t[], Integer pdt, Complex vl[], Integer pdvl, Complex vr[],
     Integer pdvr, Integer mm, Integer *m, NagError *fail)
```

3 Description

nag_ztrevc (f08qxc) computes left and/or right eigenvectors of a complex upper triangular matrix T. Such a matrix arises from the Schur factorization of a complex general matrix, as computed by nag_zhseqr (f08psc), for example.

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

$$Tx = \lambda x \quad \text{and} \quad y^{\mathrm{H}}T = \lambda y^{\mathrm{H}} \big(\text{or } T^{\mathrm{H}}y = \bar{\lambda}y\big).$$

The function can compute the eigenvectors corresponding to selected eigenvalues, or it can compute all the eigenvectors. In the latter case the eigenvectors may optionally be pre-multiplied by an input matrix Q. Normally Q is a unitary matrix from the Schur factorization of a matrix A as $A = QTQ^{\mathrm{H}}$; if x is a (left or right) eigenvector of T, then Qx is an eigenvector of A.

The eigenvectors are computed by forward or backward substitution. They are scaled so that $\max(|\mathrm{Re}(x_i)| + |\mathrm{Im}\, x_i|) = 1$.

4 References

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 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: **how_many** – Nag_HowManyType *Input*

On entry: indicates how many eigenvectors are to be computed.

how_many = Nag_ComputeAll
All eigenvectors (as specified by **side**) are computed.

how_many = Nag_BackTransform
All eigenvectors (as specified by **side**) are computed and then pre-multiplied by the matrix Q (which is overwritten).

how_many = Nag_ComputeSelected
Selected eigenvectors (as specified by **side** and **select**) are computed.

Constraint: **how_many** = Nag_ComputeAll, Nag_BackTransform or Nag_ComputeSelected.

4: **select**[*dim*] – const Nag_Boolean *Input*

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

n when **how_many** = Nag_ComputeSelected;
otherwise **select** may be **NULL**.

On entry: specifies which eigenvectors are to be computed if **how_many** = Nag_ComputeSelected. To obtain the eigenvector corresponding to the eigenvalue λ_j, **select**$[j-1]$ must be set Nag_TRUE.

If **how_many** = Nag_ComputeAll or Nag_BackTransform, **select** is not referenced and may be **NULL**.

5: **n** – Integer *Input*

On entry: n, the order of the matrix T.

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

6: **t**[*dim*] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **t** must be at least $\mathbf{pdt} \times \mathbf{n}$.

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

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

On entry: the n by n upper triangular matrix T, as returned by nag_zhseqr (f08psc).

On exit: is used as internal workspace prior to being restored and hence is unchanged.

7: **pdt** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdt} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdt} \geq \mathbf{n}$.

8: **vl**[dim] – Complex *Input/Output*

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

pdvl $\times$ **mm** when **side** $=$ Nag_LeftSide or Nag_BothSides and **order** $=$ Nag_ColMajor;
n $\times$ **pdvl** when **side** $=$ Nag_LeftSide or Nag_BothSides and **order** $=$ Nag_RowMajor;
otherwise **vl** may be **NULL**.

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 **how_many** $=$ Nag_BackTransform and **side** $=$ Nag_LeftSide or Nag_BothSides, **vl** must contain an n by n matrix Q (usually the matrix of Schur vectors returned by nag_zhseqr (f08psc)).

If **how_many** $=$ Nag_ComputeAll or Nag_ComputeSelected, **vl** need not be set.

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

If **side** $=$ Nag_RightSide, **vl** is not referenced and may be **NULL**.

9: **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, **vl** may be **NULL**.;

if **order** $=$ Nag_RowMajor,

if **side** $=$ Nag_LeftSide or Nag_BothSides, **pdvl** $\geq$ **mm**;
if **side** $=$ Nag_RightSide, **vl** may be **NULL**..

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

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

pdvr $\times$ **mm** when **side** $=$ Nag_RightSide or Nag_BothSides and **order** $=$ Nag_ColMajor;
n $\times$ **pdvr** when **side** $=$ Nag_RightSide or Nag_BothSides and **order** $=$ Nag_RowMajor;
otherwise **vr** may be **NULL**.

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.

On entry: if **how_many** $=$ Nag_BackTransform and **side** $=$ Nag_RightSide or Nag_BothSides, **vr** must contain an n by n matrix Q (usually the matrix of Schur vectors returned by nag_zhseqr (f08psc)).

If **how_many** $=$ Nag_ComputeAll or Nag_ComputeSelected, **vr** need not be set.

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

If **side** $=$ Nag_LeftSide, **vr** is not referenced and may be **NULL**.

11: **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** $\geq$ **n**;
if **side** = Nag_LeftSide, **vr** may be **NULL**.;

if **order** = Nag_RowMajor,

if **side** = Nag_RightSide or Nag_BothSides, **pdvr** $\geq$ **mm**;
if **side** = Nag_LeftSide, **vr** may be **NULL**..

12: **mm** – Integer *Input*

On entry: the number of rows or columns (depending on the value of **order**) in the arrays **vl** and/or **vr**. The precise number of rows or columns required, $required_rowcol$, is n if **how_many** = Nag_ComputeAll or Nag_BackTransform; if **how_many** = Nag_ComputeSelected, $required_rowcol$ is the number of selected eigenvectors (see **select**), in which case $0 \leq required_rowcol \leq n$.

Constraints:

if **how_many** = Nag_ComputeAll or Nag_BackTransform, **mm** $\geq$ **n**;
otherwise **mm** $\geq required_rowcol$.

13: **m** – Integer * *Output*

On exit: $required_rowcol$, the number of selected eigenvectors. If **how_many** = Nag_ComputeAll or Nag_BackTransform, **m** is set to n.

14: **fail** – NagError * *Input/Output*

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_ENUM_INT

On entry, **side** = $\langle value\rangle$ and **mm** = $\langle value\rangle$.
Constraint: **mm** > 0.

NE_ENUM_INT_2

On entry, **how_many** = $\langle value\rangle$, **mm** = $\langle value\rangle$ and **n** = $\langle value\rangle$.
Constraint: if **how_many** = Nag_ComputeAll or Nag_BackTransform, **mm** $\geq$ **n**;
otherwise **mm** $\geq required_rowcol$.

On entry, **side** = $\langle value\rangle$, **pdvl** = $\langle value\rangle$, **mm** = $\langle value\rangle$.
Constraint: if **side** = Nag_LeftSide or Nag_BothSides, **pdvl** $\geq$ **mm**.

On entry, **side** = $\langle value\rangle$, **pdvl** = $\langle value\rangle$ and **n** = $\langle value\rangle$.
Constraint: if **side** = Nag_LeftSide or Nag_BothSides, **pdvl** $\geq$ **n**.

On entry, **side** $= \langle value \rangle$, **pdvr** $= \langle value \rangle$, **mm** $= \langle value \rangle$.
Constraint: if **side** = Nag_RightSide or Nag_BothSides, **pdvr** $\geq$ **mm**.

On entry, **side** $= \langle value \rangle$, **pdvr** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **side** = Nag_RightSide or Nag_BothSides, **pdvr** $\geq$ **n**.

NE_INT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** > 0.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **pdt** $= \langle value \rangle$.
Constraint: **pdt** > 0.

On entry, **pdvl** $= \langle value \rangle$.
Constraint: **pdvl** > 0.

On entry, **pdvr** $= \langle value \rangle$.
Constraint: **pdvr** > 0.

NE_INT_2

On entry, **pdt** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdt** $\geq \max(1,$ **n**$)$.

On entry, **pdt** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdt** $\geq$ **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

If x_i is an exact right eigenvector, and $\tilde{x}_i$ is the corresponding computed eigenvector, then the angle $\theta(\tilde{x}_i, x_i)$ between them is bounded as follows:

$$\theta(\tilde{x}_i, x_i) \leq \frac{c(n)\epsilon\|T\|_2}{sep_i}$$

where sep_i is the reciprocal condition number of x_i.

The condition number sep_i may be computed by calling nag_ztrsna (f08qyc).

8 Parallelism and Performance

nag_ztrevc (f08qxc) is not threaded by NAG in any implementation.

nag_ztrevc (f08qxc) 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 nag_dtrevc (f08qkc).

10 Example

See Section 10 in nag_zgebal (f08nvc).