NAG Library Function Document

nag_dtrevc (f08qkc)

1 Purpose

nag_dtrevc (f08qkc) computes selected left and/or right eigenvectors of a real upper quasi-triangular matrix.

2 Specification

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

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

3 Description

nag_dtrevc (f08qkc) computes left and/or right eigenvectors of a real upper quasi-triangular matrix $T$ in canonical Schur form. Such a matrix arises from the Schur factorization of a real general matrix, as computed by nag_dhseqr (f08pec), for example.

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

$Tx = \lambda x$ and $y^H T = \lambda y^H$ (or $T^T y = \lambda y$).

Note that even though $T$ is real, $\lambda$, $x$, and $y$ may be complex. If $x$ is an eigenvector corresponding to a complex eigenvalue $\lambda$, then the complex conjugate vector $\bar{x}$ is the eigenvector corresponding to the complex conjugate eigenvalue $\bar{\lambda}$.

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 an orthogonal matrix from the Schur factorization of a matrix $A$ as $A = QTQ^T$; 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, for a real eigenvector $x$, $\max(|x_i|) = 1$, and for a complex eigenvector, $\max(|\text{Re}(x_i)| + |\text{Im}(x_i)|) = 1$.

4 References


5 Arguments

1:  order – Nag_OrderType  
    
    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] – Nag_Boolean

*Input/Output*

*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 real eigenvector corresponding to the real eigenvalue $\lambda_j$, `select[j - 1]` must be set Nag_TRUE. To select the complex eigenvector corresponding to a complex conjugate pair of eigenvalues $\lambda_j$ and $\lambda_{j+1}$, `select[j - 1]` and/or `select[j]` must be set Nag_TRUE; the eigenvector corresponding to the first eigenvalue in the pair is computed.

*On exit:* if a complex eigenvector was selected as specified above, then `select[j - 1]` is set to Nag_TRUE and `select[j]` to Nag_FALSE.

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:* $n \geq 0$.

6: t[dim] – const double

*Input*

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

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

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

*On entry:* the $n$ by $n$ upper quasi-triangular matrix $T$ in canonical Schur form, as returned by nag_dhseqr (f08pec).
7: pdt – Integer  

*Input*

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

Constraint: pdt \( \geq \max(1, n) \).

8: vl[dim] – double  

*Input/Output*

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

\[
\text{pdvl} \times \text{mm when side} = \text{Nag\_LeftSide or Nag\_BothSides and order} = \text{Nag\_ColMajor;}
\]

\[
n \times \text{pdvl when side} = \text{Nag\_LeftSide or Nag\_BothSides and order} = \text{Nag\_RowMajor;}
\]

otherwise vl may be NULL.

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

\[
\text{vl}[(j - 1) \times \text{pdvl} + i - 1] \text{ when order} = \text{Nag\_ColMajor;}
\]

\[
\text{vl}[(i - 1) \times \text{pdvl} + j - 1] \text{ when order} = \text{Nag\_RowMajor.}
\]

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\_dhseqr (f08pec)).

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 of the array, in the same order as their eigenvalues. Corresponding to each real eigenvalue is a real eigenvector, occupying one row or column. Corresponding to each complex conjugate pair of eigenvalues, is a complex eigenvector occupying two rows or columns; the first row or column holds the real part and the second row or column holds the imaginary part.

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 \(\geq \text{mm}\);

if side = Nag\_RightSide, vl may be NULL..

10: vr[dim] – double  

*Input/Output*

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

\[
\text{pdvr} \times \text{mm when side} = \text{Nag\_RightSide or Nag\_BothSides and order} = \text{Nag\_ColMajor;}
\]

\[
n \times \text{pdvr when side} = \text{Nag\_RightSide or Nag\_BothSides and order} = \text{Nag\_RowMajor;}
\]

otherwise vr may be NULL.

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

\[
\text{vr}[(j - 1) \times \text{pdvr} + i - 1] \text{ when order} = \text{Nag\_ColMajor;}
\]

\[
\text{vr}[(i - 1) \times \text{pdvr} + j - 1] \text{ when order} = \text{Nag\_RowMajor.}
\]

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\_dhseqr (f08pec)).

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 of the array, in the same order as their eigenvalues. Corresponding to each real eigenvalue is a real eigenvector, occupying one row or column. Corresponding to each complex conjugate pair of eigenvalues, is a complex eigenvector occupying two rows or columns; the first row or column holds the real part and the second row or column holds the imaginary part.

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

11: pdvr – Integer

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, vr may be NULL;
if order = Nag_RowMajor,
  if side = Nag_RightSide or Nag_BothSides, pdvr ≥ mm;
  if side = Nag_LeftSide, vr may be NULL.

12: mm – Integer

On entry: the number of rows or columns in the arrays vl and/or vr. The precise number of rows or columns required (depending on the value of order), required_rowcol, is n if how_many = Nag_ComputeAll or Nag_BackTransform; if how_many = Nag_ComputeSelected, required_rowcol is obtained by counting 1 for each selected real eigenvector and 2 for each selected complex eigenvector (see select), in which case 0 ≤ required_rowcol ≤ n.

Constraints:

if how_many = Nag_ComputeAll or Nag_BackTransform, mm ≥ n;
otherwise mm ≥ required_rowcol.

13: m – Integer *

On exit: required_rowcol, the number of rows or columns of vl and/or vr actually used to store the computed eigenvectors. If how_many = Nag_ComputeAll or Nag_BackTransform, m is set to n.

14: 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 ⟨value⟩ had an illegal value.

**NE_ENUM_INT_2**

On entry, how_many = ⟨value⟩, mm = ⟨value⟩ and n = ⟨value⟩. Constraint: if how_many = Nag_ComputeAll or Nag_BackTransform, mm ≥ n; otherwise mm ≥ 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 ≥ mm.

On entry, side = \langle value \rangle, pdvl = \langle value \rangle and n = \langle value \rangle.
Constraint: if side = Nag_LEFT or Nag_BothSides, pdvl ≥ n.

On entry, side = \langle value \rangle, pdvr = \langle value \rangle, mm = \langle value \rangle.
Constraint: if side = Nag_RIGHT or Nag_BothSides, pdvr ≥ mm.

On entry, side = \langle value \rangle, pdvr = \langle value \rangle and n = \langle value \rangle.
Constraint: if side = Nag_RIGHT or Nag_BothSides, pdvr ≥ n.

NE_INT

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 ≥ 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

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:

\[
\theta(\tilde{x}_i, x_i) \leq \frac{c(n)\|T\|_2}{\text{sep}_i}
\]

where \( \text{sep}_i \) is the reciprocal condition number of \( x_i \).

The condition number \( \text{sep}_i \) may be computed by calling nag_dtrsna (f08qlc).

8 Parallelism and Performance

nag_dtrevc (f08qkc) is not threaded by NAG in any implementation.

nag_dtrevc (f08qkc) 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
For a description of canonical Schur form, see the document for nag_dhseqr (f08pec).
The complex analogue of this function is nag_ztrevc (f08qxc).

10 Example
See Section 10 in nag_dgebal (f08nhe).