NAG Library Function Document

nag_dtgevc (f08ykc)

1 Purpose

nag_dtgevc (f08ykc) computes some or all of the right and/or left generalized eigenvectors of a pair of real matrices $(A, B)$ which are in generalized real Schur form.

2 Specification

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

void nag_dtgevc (Nag_OrderType order, Nag_SideType side, Nag_HowManyType how_many, const Nag_Boolean select[], Integer n, const double a[], Integer pda, const double b[], Integer pdb, double vl[], Integer pdvl, double vr[], Integer pdvr, Integer mm, Integer *m, NagError *fail)
```

3 Description

nag_dtgevc (f08ykc) computes some or all of the right and/or left generalized eigenvectors of the matrix pair $(A, B)$ which is assumed to be in generalized upper Schur form. If the matrix pair $(A, B)$ is not in the generalized upper Schur form, then nag_dhgeqz (f08xec) should be called before invoking nag_dtgevc (f08ykc).

The right generalized eigenvector $x$ and the left generalized eigenvector $y$ of $(A, B)$ corresponding to a generalized eigenvalue $\lambda$ are defined by

$$(A - \lambda B)x = 0$$

and

$$y^H(A - \lambda B) = 0.$$ 

If a generalized eigenvalue is determined as 0/0, which is due to zero diagonal elements at the same locations in both $A$ and $B$, a unit vector is returned as the corresponding eigenvector.

Note that the generalized eigenvalues are computed using nag_dhgeqz (f08xec) but nag_dtgevc (f08ykc) does not explicitly require the generalized eigenvalues to compute eigenvectors. The ordering of the eigenvectors is based on the ordering of the eigenvalues as computed by nag_dtgevc (f08ykc).

If all eigenvectors are requested, the function may either return the matrices $X$ and/or $Y$ of right or left eigenvectors of $(A, B)$, or the products $ZX$ and/or $QY$, where $Z$ and $Q$ are two matrices supplied by you. Usually, $Q$ and $Z$ are chosen as the orthogonal matrices returned by nag_dhgeqz (f08xec). Equivalently, $Q$ and $Z$ are the left and right Schur vectors of the matrix pair supplied to nag_dhgeqz (f08xec). In that case, $QY$ and $ZX$ are the left and right generalized eigenvectors, respectively, of the matrix pair supplied to nag_dhgeqz (f08xec).

A must be block upper triangular; with 1 by 1 and 2 by 2 diagonal blocks. Corresponding to each 2 by 2 diagonal block is a complex conjugate pair of eigenvalues and eigenvectors; only one eigenvector of the pair is computed, namely the one corresponding to the eigenvalue with positive imaginary part. Each 1 by 1 block gives a real generalized eigenvalue and a corresponding eigenvector.
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

   On entry: specifies the required sets of generalized eigenvectors.

      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_BothSides, Nag_LeftSide or Nag_RightSide.

3: how many – Nag_HowManyType

   On entry: specifies further details of the required generalized eigenvectors.

      how many = Nag_ComputeAll
      All right and/or left eigenvectors are computed.

      how many = Nag_BackTransform
      All right and/or left eigenvectors are computed; they are backtransformed using the input matrices supplied in arrays vr and/or vl.

      how many = Nag_ComputeSelected
      Selected right and/or left eigenvectors, defined by the array select, are computed.

   Constraint: how many = Nag_ComputeAll, Nag_BackTransform or Nag_ComputeSelected.

4: select[dim] – const Nag_Boolean

   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 the eigenvectors to be computed if how many = Nag_ComputeSelected. To select the generalized eigenvector corresponding to the jth generalized eigenvalue, the jth element of select should be set to Nag_TRUE; if the eigenvalue corresponds to a complex conjugate pair,
then real and imaginary parts of eigenvectors corresponding to the complex conjugate eigenvalue pair will be computed.

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

Constraint: if how_many = Nag_ComputeSelected, select[j] = Nag_TRUE or Nag_FALSE, for j = 0, 1, \ldots, n - 1.

5: \textbf{n} – Integer \hspace{1cm} \textit{Input}

On entry: n, the order of the matrices A and B.

Constraint: \textbf{n} \geq 0.

6: \textbf{a}[\textit{dim}] – const double \hspace{1cm} \textit{Input}

Note: the dimension, \textit{dim}, of the array \textbf{a} must be at least \textbf{pda} \times \textbf{n}.

The (i,j)th element of the matrix A is stored in
\begin{align*}
\textbf{a}[(j - 1) \times \textbf{pda} + i - 1] & \text{ when order} = \text{Nag\_ColMajor;} \\
\textbf{a}[(i - 1) \times \textbf{pda} + j - 1] & \text{ when order} = \text{Nag\_RowMajor}.
\end{align*}

On entry: the matrix pair (A, B) must be in the generalized Schur form. Usually, this is the matrix A returned by nag_dhgeqz (f08xec).

7: \textbf{pda} – Integer \hspace{1cm} \textit{Input}

On entry: the stride separating row or column elements (depending on the value of order) in the array \textbf{a}.

Constraint: \textbf{pda} \geq \text{max}(1, \textbf{n}).

8: \textbf{b}[\textit{dim}] – const double \hspace{1cm} \textit{Input}

Note: the dimension, \textit{dim}, of the array \textbf{b} must be at least \textbf{pdb} \times \textbf{n}.

The (i,j)th element of the matrix B is stored in
\begin{align*}
\textbf{b}[(j - 1) \times \textbf{pdb} + i - 1] & \text{ when order} = \text{Nag\_ColMajor;} \\
\textbf{b}[(i - 1) \times \textbf{pdb} + j - 1] & \text{ when order} = \text{Nag\_RowMajor}.
\end{align*}

On entry: the matrix pair (A, B) must be in the generalized Schur form. If A has a 2 by 2 diagonal block then the corresponding 2 by 2 block of B must be diagonal with positive elements. Usually, this is the matrix B returned by nag_dhgeqz (f08xec).

9: \textbf{pdb} – Integer \hspace{1cm} \textit{Input}

On entry: the stride separating row or column elements (depending on the value of order) in the array \textbf{b}.

Constraint: \textbf{pdb} \geq \text{max}(1, \textbf{n}).

10: \textbf{vl}[\textit{dim}] – double \hspace{1cm} \textit{Input/Output}

Note: the dimension, \textit{dim}, of the array \textbf{vl} must be at least
\begin{align*}
\textbf{pdvl} \times \textbf{mm} & \text{ when side} = \text{Nag\_LeftSide} \text{ or Nag\_BothSides and order} = \text{Nag\_ColMajor;} \\
\textbf{n} \times \textbf{pdvl} & \text{ when side} = \text{Nag\_LeftSide} \text{ or Nag\_BothSides and order} = \text{Nag\_RowMajor;} \\
\text{otherwise \textbf{vl}} & \text{ may be NULL}.
\end{align*}

The (i,j)th element of the matrix is stored in
\begin{align*}
\textbf{vl}[(j - 1) \times \textbf{pdvl} + i - 1] & \text{ when order} = \text{Nag\_ColMajor;} \\
\textbf{vl}[(i - 1) \times \textbf{pdvl} + j - 1] & \text{ when order} = \text{Nag\_RowMajor}.
\end{align*}
On entry: if how_many = Nag_BackTransform and side = Nag_LeftSide or Nag_BothSides, vl must be initialized to an \( n \times n \) matrix \( Q \). Usually, this is the orthogonal matrix \( Q \) of left Schur vectors returned by nag_dhgeqz (f08xec).

On exit: if side = Nag_LeftSide or Nag_BothSides, vl contains:

- if how_many = Nag_ComputeAll, the matrix \( V \) of left eigenvectors of \((A, B)\);
- if how_many = Nag_BackTransform, the matrix \( QY \);
- if how_many = Nag_ComputeSelected, the left eigenvectors of \((A, B)\) specified by select, stored consecutively in the rows or columns (depending on the value of order) of the array vl, in the same order as their corresponding eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive rows or columns, the first holding the real part, and the second the imaginary part.

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

11: pdvl – Integer

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 \( \geq \min(n, m) \);
  - if side = Nag_RightSide, vl may be NULL.

12: \( vr[\text{dim}] \) – double

Note: the dimension, \text{dim}, of the array vr must be at least

- \( \min(n, m) \times \min(n, m) \) when side = Nag_RightSide or Nag_BothSides and order = Nag_ColMajor;
- \( n \times \text{pdvr} \) 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

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

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

On entry: if how_many = Nag_BackTransform and side = Nag_RightSide or Nag_BothSides, vr must be initialized to an \( n \times n \) matrix \( Z \). Usually, this is the orthogonal matrix \( Z \) of right Schur vectors returned by nag_dhgeqz (f08xec).

On exit: if side = Nag_RightSide or Nag_BothSides, vr contains:

- if how_many = Nag_ComputeAll, the matrix \( X \) of right eigenvectors of \((A, B)\);
- if how_many = Nag_BackTransform, the matrix \( ZX \);
- if how_many = Nag_ComputeSelected, the right eigenvectors of \((A, B)\) specified by select, stored consecutively in the rows or columns (depending on the value of order) of the array vr, in the same order as their corresponding eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive rows or columns, the first holding the real part, and the second the imaginary part.

If side = Nag_LeftSide, vr is not referenced and may be NULL.
13: \texttt{pdvr} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{vr}.

\textit{Constraints:}

if \texttt{order} = \texttt{Nag\_ColMajor},
  if \texttt{side} = \texttt{Nag\_RightSide} or \texttt{Nag\_BothSides}, \texttt{pdvr} $\geq$ \texttt{n};
  if \texttt{side} = \texttt{Nag\_LeftSide}, \texttt{vr} may be \texttt{NULL};

if \texttt{order} = \texttt{Nag\_RowMajor},
  if \texttt{side} = \texttt{Nag\_RightSide} or \texttt{Nag\_BothSides}, \texttt{pdvr} $\geq$ \texttt{mm};
  if \texttt{side} = \texttt{Nag\_LeftSide}, \texttt{vr} may be \texttt{NULL}.

14: \texttt{mm} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the number of columns in the arrays \texttt{vl} and/or \texttt{vr}.

\textit{Constraints:}

if \texttt{how\_many} = \texttt{Nag\_ComputeAll} or \texttt{Nag\_BackTransform}, \texttt{mm} $\geq$ \texttt{n};
if \texttt{how\_many} = \texttt{Nag\_ComputeSelected}, \texttt{mm} must not be less than the number of requested eigenvectors.

15: \texttt{m} – Integer * \hspace{1cm} \textit{Output}

\textit{On exit:} the number of columns in the arrays \texttt{vl} and/or \texttt{vr} actually used to store the eigenvectors.

If \texttt{how\_many} = \texttt{Nag\_ComputeAll} or \texttt{Nag\_BackTransform}, \texttt{m} is set to \texttt{n}. Each selected real eigenvector occupies one row or column and each selected complex eigenvector occupies two rows or columns.

16: \texttt{fail} – NagError * 

\textit{Input/Output}

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

\section{Error Indicators and Warnings}

\texttt{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\texttt{NE\_BAD\_PARAM}

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

\texttt{NE\_CONSTRAINT}

On entry, \texttt{how\_many} = \langle\texttt{value}\rangle and \texttt{select}[j] = \langle\texttt{value}\rangle.

Constraint: if \texttt{how\_many} = \texttt{Nag\_ComputeSelected}, \texttt{select}[j] = \texttt{Nag\_TRUE} or \texttt{Nag\_FALSE}, for $j = 0, 1, \ldots, n - 1$.

\texttt{NE\_ENUM\_INT\_2}

On entry, \texttt{how\_many} = \langle\texttt{value}\rangle, \texttt{n} = \langle\texttt{value}\rangle and \texttt{mm} = \langle\texttt{value}\rangle.

Constraint: if \texttt{how\_many} = \texttt{Nag\_ComputeAll} or \texttt{Nag\_BackTransform}, \texttt{mm} $\geq$ \texttt{n};
if \texttt{how\_many} = \texttt{Nag\_ComputeSelected}, \texttt{mm} must not be less than the number of requested eigenvectors.

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

Constraint: if \texttt{side} = \texttt{Nag\_LeftSide} or \texttt{Nag\_BothSides}, \texttt{pdvl} $\geq$ \texttt{mm}.

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

Constraint: if \texttt{side} = \texttt{Nag\_LeftSide} or \texttt{Nag\_BothSides}, \texttt{pdvl} $\geq$ \texttt{n}. 

Mark 25
On entry, \( \text{side} = \langle \text{value} \rangle \), \( \text{pdvr} = \langle \text{value} \rangle \), \( \text{mm} = \langle \text{value} \rangle \).
Constraint: if \( \text{side} = \text{Nag\_RightSide} \) or \( \text{Nag\_BothSides} \), \( \text{pdvr} \geq \text{mm} \).

On entry, \( \text{side} = \langle \text{value} \rangle \), \( \text{pdvr} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: if \( \text{side} = \text{Nag\_RightSide} \) or \( \text{Nag\_BothSides} \), \( \text{pdvr} \geq \text{n} \).

**NE\_INT**

On entry, \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{n} \geq 0 \).

On entry, \( \text{pda} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdvl} = \langle \text{value} \rangle \).
Constraint: \( \text{pdvl} > 0 \).

On entry, \( \text{pdvr} = \langle \text{value} \rangle \).
Constraint: \( \text{pdvr} > 0 \).

**NE\_INT\_2**

On entry, \( \text{pda} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} \geq \max(1, \text{n}) \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} \geq \max(1, \text{n}) \).

**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.

**NE\_NOT\_COMPLEX**

The 2 by 2 block \( \langle \langle \text{value} \rangle : \langle \text{value} \rangle + 1 \rangle \) does not have complex eigenvalues.

7  Accuracy

It is beyond the scope of this manual to summarise the accuracy of the solution of the generalized eigenvalue problem. Interested readers should consult Section 4.11 of the LAPACK Users’ Guide (see Anderson et al. (1999)) and Chapter 6 of Stewart and Sun (1990).

8  Parallelism and Performance

\textit{nag\_dtgevc (f08ykc)} is not threaded by NAG in any implementation.

\textit{nag\_dtgevc (f08ykc)} 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

nag_dtgevc (f08ykc) is the sixth step in the solution of the real generalized eigenvalue problem and is called after nag_dhgeqz (f08xec).

The complex analogue of this function is nag_ztgevc (f08yxc).

10 Example

This example computes the $\alpha$ and $\beta$ arguments, which defines the generalized eigenvalues and the corresponding left and right eigenvectors, of the matrix pair $(A, B)$ given by

\[
A = \begin{pmatrix}
1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\
2.0 & 4.0 & 8.0 & 16.0 & 32.0 \\
3.0 & 9.0 & 27.0 & 81.0 & 243.0 \\
4.0 & 16.0 & 64.0 & 256.0 & 1024.0 \\
5.0 & 25.0 & 125.0 & 625.0 & 3125.0 \\
\end{pmatrix}
\[
\text{and}
B = \begin{pmatrix}
1.0 & 2.0 & 3.0 & 4.0 & 5.0 \\
1.0 & 4.0 & 9.0 & 16.0 & 25.0 \\
1.0 & 8.0 & 27.0 & 64.0 & 125.0 \\
1.0 & 16.0 & 81.0 & 256.0 & 625.0 \\
1.0 & 32.0 & 243.0 & 1024.0 & 3125.0 \\
\end{pmatrix}.
\]

To compute generalized eigenvalues, it is required to call five functions: nag_dggbal (f08whc) to balance the matrix, nag_dgeqrf (f08aec) to perform the $QR$ factorization of $B$, nag_dormqr (f08agc) to apply $Q$ to $A$, nag_dgghrd (f08wec) to reduce the matrix pair to the generalized Hessenberg form and nag_dhgeqz (f08xec) to compute the eigenvalues via the $QZ$ algorithm.

The computation of generalized eigenvectors is done by calling nag_dtgevc (f08ykc) to compute the eigenvectors of the balanced matrix pair. The function nag_dggbak (f08wjc) is called to backward transform the eigenvectors to the user-supplied matrix pair. If both left and right eigenvectors are required then nag_dggbak (f08wjc) must be called twice.

10.1 Program Text

/* nag_dtgevc (f08ykc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group.
 * * Mark 7, 2001.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf06.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

static Integer normalize_vectors(Nag_OrderType order, Integer n, double qz[], double alphai[], const char* title);

int main(void)
{
  /* Scalars */
  Integer i, icols, ihi, ilo, irows, j, m, n, pda, pdb, pdq, pdz;
  Integer exit_status = 0;
  Nag_Boolean ileft, iright;

  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double *a = 0, *alphai = 0, *alphar = 0, *beta = 0;
  double *lscale = 0, *q = 0, *r = 0, *tau = 0, *z = 0;

  #ifdef NAG_COLUMN_MAJOR
  #define A(I, J) a[(J-1)*pda +I-1 ]
  #define B(I, J) b[(J-1)*pdb +I-1 ]
  #define Q(I, J) q[(J-1)*pdb +I-1 ]
  #define Z(I, J) z[(J-1)*pdz +I-1 ]
  order = Nag_ColMajor;
  #else
  #define A(I, J) a[(J-1)*pda +I -1 ]
  #define B(I, J) b[(J-1)*pdb +I -1 ]
  #define Q(I, J) q[(J-1)*pdb +I -1 ]
  #define Z(I, J) z[(J-1)*pdz +I -1 ]
  order = Nag_RowMajor;
  #endif

  /* Scalars */
  ...
  
  /* Arrays */
  ...
  
  /* Main program */
  ...
  
  /* Normalization */
  ...
  
  /* Call functions */
  ...
  
  /* Outputs */
  ...
  
  #endif

Mark 25
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
#define Q(I, J) q[(I-1)*pdq + J - 1]
#define Z(I, J) z[(I-1)*pdz + J - 1]
    order = Nag_RowMajor;
#endif

INIT_FAIL(fail);

printf("nag_dtgevc (f08ykc) Example Program Results\n\n");

/* ileft is Nag_TRUE if left eigenvectors are required */
/* iright is Nag_TRUE if right eigenvectors are required */
ileft = Nag_TRUE;
iright = Nag_TRUE;

/* 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
pda = n;
pdb = n;
pdq = n;
pdz = n;

/* Allocate memory */
if (0
    !(a = NAG_ALLOC(n * n, double)) ||
    !(b = NAG_ALLOC(n * n, double)) ||
    !(q = NAG_ALLOC(n * n, double)) ||
    !(z = NAG_ALLOC(n * n, double)) ||
    !(alphai = NAG_ALLOC(n, double)) ||
    !(alphar = NAG_ALLOC(n, double)) ||
    !(beta = NAG_ALLOC(n, double)) ||
    !(lscale = NAG_ALLOC(n, double)) ||
    !(rscale = NAG_ALLOC(n, double)) ||
    !(tau = NAG_ALLOC(n, double))
    
    printf("Allocation failure\n");
exit_status = -1;
goto END;
)

/* READ matrix A from data file */
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s("%lf", &A(i, j));
#else
        scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
        scanf_s("%*[\n"]
#else
        scanf("%*[\n"]
#endif
/* READ matrix B from data file */
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s("%lf", &B(i, j));
#else
        scanf("%*[\n"]
#endif
/* Balance the real general matrix pair (A, B) using nag_dggbal (F08WHC). */

#define _WIN32

#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Print matrices A and B after balancing using
 * nag_gen_real_mat_print (X04CAC). */

fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a, pda, "Matrix A after balancing", 0, &fail);
if (fail.code == NE_NOERROR) {
    fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b, pdb, "Matrix B after balancing", 0, &fail);
}

/* Reduce B to triangular form using QR and multiplying both sides by Q’T */
irows = ihi + 1 - ilo;
icols = n + 1 - ilo;

/* QR factorization of real general rectangular matrix */
nag_dgeqrf(order, irows, icols, &B(ilo, ilo), pdb, tau, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgeqrf (F08AEC).\n%s\n", fail.message);
    exit_status = 3;
    goto END;
}

/* Apply the Q to matrix A - nag_dormqr (F08AGC) */

nag_dormqr(order, Nag_LeftSide, Nag_Trans, irows, icols, &B(ilo, ilo), pdb, tau, &A(ilo, ilo), pda, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dormqr (F08AGC).\n%s\n", fail.message);
    exit_status = 4;
    goto END;
}

/* Initialize Q (if left eigenvectors are required) */
if (ileft) {
    /* Q = I. */
    nag_dge_load(order, n, n, 0.0, 1.0, q, pdq, &fail);
    /* Copy B to Q using nag_dge_copy (F16QFC). */
    nag_dge_copy(order, Nag_NoTrans, icols-1, icols-1, &B(ilo+1, ilo), pdb, &Q(ilo+1, ilo), pdq, &fail);
    /* nag_dorgqr (F08AFC). */
    * Form all or part of orthogonal Q from QR factorization
    * determined by nag_dgeqrf (F08AEC) or nag_dgeqpf (F08BEC) */
nag_dorgqr(order, irows, irows, irows, &Q(ilo, ilo), pdq, tau, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dorgqr (f08afc).\n%s\n", fail.message);
    exit_status = 5;
    goto END;
}

/* Initialize Z (if right eigenvectors are required) */
if (iright) {
    /* Z=I */
    nag_dge_load(order, n, n, 0.0, 1.0, z, pdz, &fail);
}

/* Compute the generalized Hessenberg form of (A,B) */
/* nag_dgghrd (f08wec).
* Orthogonal reduction of a pair of real general matrices
* to generalized upper Hessenberg form */

nag_dgghrd(order, Nag_UpdateSchur, Nag_UpdateZ, n, ilo, ihi, a, pda,
            b, pdb, q, pdq, z, pdz, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgghrd (f08wec).\n%s\n", fail.message);
    exit_status = 6;
    goto END;
}

/* Matrix A in generalized Hessenberg form */
/* flush(stdout); 

nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a,
    pda, "Matrix A in Hessenberg form", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 7;
    goto END;
}

/* Matrix B in generalized Hessenberg form */
/* flush(stdout); 

nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b,
    pdb, "Matrix B in Hessenberg form", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dhgeqz (f08xec).
* Eigenvalues and generalized Schur factorization of real
* generalized upper Hessenberg form reduced from a pair of
* real general matrices. */

nag_dhgeqz(order, Nag_Schur, Nag_AccumulateQ, Nag_AccumulateZ, n, ilo, ihi,
        a, pda, b, pdb, alphar, alphai, beta, q, pdq, z, pdz, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dhgeqz (f08xec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the generalized eigenvalue parameters */
printf("\n Generalized eigenvalues\n");
for (i = 0; i < n; ++i) {
    if (beta[i] != 0.0) {
        printf(" %4i\t\%7.3f,%7.3f\n", i+1, alphar[i]/beta[i],
                alphai[i]/beta[i]);
    } else
else
/* Compute left and right generalized eigenvectors * of the balanced matrix - nag_dtgevc (f08ykc). */

nag_dtgevc(order, Nag_BothSides, Nag_BackTransform, NULL, n, a, pda,
b, pdb, q, pdq, z, pdz, n, &m, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_dtgevc (f08ykc).\n\n", fail.message);
  exit_status = 1;
  goto END;
}
if (iright) {
  /* Compute right eigenvectors of the original matrix pair *
   * supplied on nag_dggbal (f08whc) using nag_dggbak (f08wjc).
   */
  nag_dggbak(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, lscale,
             rscale, n, z, pdz, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dggbak (f08wjc).\n\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Normalize and print the right eigenvectors */
  exit_status = normalize_vectors(order, n, z, alphai, "Right eigenvectors");
  printf("\n");
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(q);
NAG_FREE(z);
NAG_FREE(alphai);
NAG_FREE(alphar);
NAG_FREE(beta);
NAG_FREE(lscale);
NAG_FREE(rscale);
NAG_FREE(tau);

return exit_status;

static Integer normalize_vectors(Nag_OrderType order, Integer n, double qz[],
                                 double alphai[], const char* title)
{
  /* Real eigenvectors are scaled so that the maximum value of elements is 1.0; *
   * each complex eigenvector z[] is normalized so that the element of largest *
   * magnitude is scaled to be (1.0,0.0). */

double a, b, u, v, r, ri;
Integer colinc, rowinc, i, j, k, indqz, errors=0;
NagError fail;
INIT_FAIL(fail);
if (order==Nag_ColMajor) {
    rowinc = 1;
    colinc = n;
} else {
    rowinc = n;
    colinc = 1;
}

indqz = 0;
for (j=0; j<n; j++) {
    if (alphai[j]>=0.0) {
        if (alphai[j]==0.0) {
            /* Find element of eigenvector with largest absolute value using
             * nag_damax_val (f16jqc).
             */
            nag_damax_val(n, &qz[indqz], rowinc, &k, &r, &fail);
            if (fail.code != NE_NOERROR) {
                printf("Error from nag_damax_val (f16jqc).\n%s\n", fail.message);
                errors = 1;
                goto END;
            }
            r = qz[indqz+k];
            for (i=0; i<n*rowinc; i+=rowinc) {
                qz[indqz+i] = qz[indqz+i]/r;
            }
        } else {
            /* norm of j-th complex eigenvector using nag_dge_norm (f16rac),
             * stored as two arrays of length n.
             */
            k = 0;
            r = -1.0;
            for (i=0; i<n*rowinc;i+=rowinc) {
                ri = abs(qz[indqz+i])+abs(qz[indqz+colinc+i]);
                if (ri>r) {
                    k = i;
                    r = ri;
                }
            }
            a = qz[indqz+k];
            b = qz[indqz+colinc+k];
            r = a*a + b*b;
            for (i=0; i<n*rowinc; i+=rowinc) {
                u = qz[indqz+i];
                v = qz[indqz+colinc+i];
                qz[indqz+i] = (u*a + v*b)/r;
                qz[indqz+colinc+i] = (v*a - u*b)/r;
            }
            indqz += colinc;
            indqz += colinc;
        }
    }
}

/* Print the normalized eigenvectors using
 * nag_gen_real_mat_print (x04cac)
 */
fflush(stdout);

nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                        qz, n, title, 0, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
        errors = 1;
    }
END:
return errors;
}
10.2 Program Data

nag_dtgevc (f08ykc) Example Program Data

<table>
<thead>
<tr>
<th>Value of N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>2.00</td>
</tr>
<tr>
<td>3.00</td>
</tr>
<tr>
<td>4.00</td>
</tr>
<tr>
<td>5.00</td>
</tr>
</tbody>
</table>

:End of matrix A

<table>
<thead>
<tr>
<th>End of matrix B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_dtgevc (f08ykc) Example Program Results

Matrix A after balancing

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.1000</td>
<td>0.1000</td>
</tr>
<tr>
<td>2</td>
<td>2.0000</td>
<td>4.0000</td>
<td>0.8000</td>
<td>1.6000</td>
</tr>
<tr>
<td>3</td>
<td>0.3000</td>
<td>0.9000</td>
<td>0.2700</td>
<td>0.8100</td>
</tr>
<tr>
<td>4</td>
<td>0.4000</td>
<td>1.6000</td>
<td>0.6400</td>
<td>2.5600</td>
</tr>
<tr>
<td>5</td>
<td>0.5000</td>
<td>2.5000</td>
<td>1.2500</td>
<td>6.2500</td>
</tr>
</tbody>
</table>

Matrix B after balancing

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>2.0000</td>
<td>0.3000</td>
<td>0.4000</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>4.0000</td>
<td>0.9000</td>
<td>1.6000</td>
</tr>
<tr>
<td>3</td>
<td>0.1000</td>
<td>0.8000</td>
<td>0.2700</td>
<td>0.6400</td>
</tr>
<tr>
<td>4</td>
<td>0.1000</td>
<td>1.6000</td>
<td>0.8100</td>
<td>2.5600</td>
</tr>
<tr>
<td>5</td>
<td>0.1000</td>
<td>3.2000</td>
<td>2.4300</td>
<td>10.2400</td>
</tr>
</tbody>
</table>

Matrix A in Hessenberg form

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.1898</td>
<td>-0.3181</td>
<td>2.0547</td>
<td>4.7371</td>
</tr>
<tr>
<td>2</td>
<td>-0.8395</td>
<td>-0.0426</td>
<td>1.7132</td>
<td>7.5194</td>
</tr>
<tr>
<td>3</td>
<td>0.0000</td>
<td>-0.2846</td>
<td>-1.0101</td>
<td>-7.5927</td>
</tr>
<tr>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0376</td>
<td>1.4070</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.3813</td>
<td>-0.9937</td>
</tr>
</tbody>
</table>

Matrix B in Hessenberg form

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.4248</td>
<td>-0.3476</td>
<td>2.1175</td>
<td>5.5813</td>
</tr>
<tr>
<td>2</td>
<td>0.0000</td>
<td>-0.0782</td>
<td>0.1189</td>
<td>8.0940</td>
</tr>
<tr>
<td>3</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0021</td>
<td>-10.9356</td>
</tr>
<tr>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.5820</td>
<td>-0.0730</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.5321</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Generalized eigenvalues

| 1  | ( -2.437, 0.000) |
| 2  | ( -0.607, 0.795) |
| 3  | ( -0.607, -0.795) |
| 4  | ( 1.000, 0.000) |
| 5  | ( -0.410, 0.000) |

Right eigenvectors

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.4655</td>
<td>1.0000</td>
<td>0.0000</td>
<td>-0.5469</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>-0.7945</td>
<td>-0.5235</td>
<td>1.0000</td>
</tr>
<tr>
<td>3</td>
<td>-0.9428</td>
<td>0.2277</td>
<td>0.2509</td>
<td>-0.7383</td>
</tr>
<tr>
<td>4</td>
<td>0.4126</td>
<td>-0.0300</td>
<td>-0.0700</td>
<td>0.1953</td>
</tr>
<tr>
<td>5</td>
<td>-0.0662</td>
<td>0.0014</td>
<td>0.0102</td>
<td>-0.0273</td>
</tr>
</tbody>
</table>

Left eigenvectors

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>-0.5106</td>
<td>1.0000</td>
<td>0.0000</td>
<td>-0.5469</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>-0.7945</td>
<td>-0.5235</td>
<td>1.0000</td>
</tr>
<tr>
<td>3</td>
<td>-0.7350</td>
<td>0.2277</td>
<td>0.2509</td>
<td>-0.7383</td>
</tr>
<tr>
<td>4</td>
<td>0.2343</td>
<td>-0.0300</td>
<td>-0.0700</td>
<td>0.1953</td>
</tr>
<tr>
<td>5</td>
<td>-0.0261</td>
<td>0.0014</td>
<td>0.0102</td>
<td>-0.0273</td>
</tr>
</tbody>
</table>