NAG C 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

```
void nag_dtgevc (Nag_OrderType order, Nag_SideType side, Nag_HowManyType how_many,
      const 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 λ 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 the user. 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

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Moler C B and Stewart G W (1973) An algorithm for generalized matrix eigenproblems *SIAM J. Numer. Anal.* **10** 241–256

Stewart G W and Sun J-G (1990) *Matrix Perturbation Theory* Academic Press, London

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **side** – Nag_SideType *Input*

On entry: specifies the required sets of generalized eigenvectors:

if **side** = **Nag_RightSide**, only right eigenvectors are computed;

if **side** = **Nag_LeftSide**, only left eigenvectors are computed;

if **side** = **Nag_BothSides**, both left and right eigenvectors are computed.

Constraint: **side** = **Nag_BothSides**, **Nag_LeftSide** or **Nag_RightSide**.

3: **how_many** – Nag_HowManyType *Input*

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

if **how_many** = **Nag_ComputeAll**, all right and/or left eigenvectors are computed;

if **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**;

if **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 Boolean *Input*

Note: the dimension, dim, of the array **select** must be at least $\max(1, \mathbf{n})$ when **how_many** = **Nag_ComputeSelected** and at least 1 otherwise.

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 **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.

Constraint: **select**[j] = **TRUE** or **FALSE** for $j = 0, 1, \ldots, n-1$.

5: **n** – Integer *Input*

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

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

6: **a**[dim] – const double *Input*

Note: the dimension, dim, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix A is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix A is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

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: **pda** – Integer *Input*

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

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

8: **b**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **b** must be at least $\max(1, \mathbf{pdb} \times \mathbf{n})$.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix B is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix B is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.

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: **pdb** – Integer *Input*

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

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

10: **vl**[*dim*] – double *Input/Output*

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

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

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

1 when **side** = **Nag_RightSide**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix is stored in $\mathbf{vl}[(j-1) \times \mathbf{pdvl} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix is stored in $\mathbf{vl}[(i-1) \times \mathbf{pdvl} + j - 1]$.

On entry: if **how_many** = **Nag_BackTransform** and **side** = **Nag_LeftSide** or **Nag_BothSides**, **vl** must be initialised to an n by 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 Y 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.

11: **pdvl** – Integer *Input*

On entry: the stride separating matrix 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** $\geq \max(1, \mathbf{n})$;

if **side** = **Nag_RightSide**, **pdvl** ≥ 1;

if **order** = **Nag_RowMajor**,
if **side** = **Nag_LeftSide** or **Nag_BothSides**, **pdvl** $\geq \max(1, \mathbf{mm})$;
if **side** = **Nag_RightSide**, **pdvl** ≥ 1.

12: **vr**[dim] – double *Input/Output*

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

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

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

1 when **side** = **Nag_LeftSide**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix is stored in **vr**$[(j-1) \times \mathbf{pdvr} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix is stored in **vr**$[(i-1) \times \mathbf{pdvr} + j - 1]$.

On entry: if **how_many** = **Nag_BackTransform** and **side** = **Nag_RightSide** or **Nag_BothSides**, **vr** must be initialised to an n by 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.

13: **pdvr** – Integer *Input*

On entry: the stride separating matrix 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** $\geq \max(1, \mathbf{n})$;
if **side** = **Nag_LeftSide**, **pdvr** ≥ 1;

if **order** = **Nag_RowMajor**,
if **side** = **Nag_RightSide** or **Nag_BothSides**, **pdvr** $\geq \max(1, \mathbf{mm})$;
if **side** = **Nag_LeftSide**, **pdvr** ≥ 1.

14: **mm** – Integer *Input*

On entry: the number of columns in the arrays **vl** and/or **vr**.

Constraints:

if **how_many** = **Nag_ComputeAll** or **Nag_BackTransform**, **mm** $\geq$ **n**;
if **how_many** = **Nag_ComputeSelected**, **mm** must not be less than the number of requested eigenvectors.

15: **m** – Integer * *Output*

On exit: the number of columns in the arrays **vl** and/or **vr** actually used to store the eigenvectors. If **how_many** = **Nag_ComputeAll** or **Nag_BackTransform**, **m** is set to **n**. Each selected real eigenvector occupies one row or column and each selected complex eigenvector occupies two rows or columns.

16: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

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

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

NE_INT_2

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

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

NE_ENUM_INT_2

On entry, **side** $= \langle value \rangle$, **n** $= \langle value \rangle$, **pdvl** $= \langle value \rangle$.
Constraint: if **side** = **Nag_LeftSide** or **Nag_BothSides**, **pdvl** $\geq \max(1, \mathbf{n})$;
if **side** = **Nag_RightSide**, **pdvl** ≥ 1.

On entry, **side** $= \langle value \rangle$, **n** $= \langle value \rangle$, **pdvr** $= \langle value \rangle$.
Constraint: if **side** = **Nag_RightSide** or **Nag_BothSides**, **pdvr** $\geq \max(1, \mathbf{n})$;
if **side** = **Nag_LeftSide**, **pdvr** ≥ 1.

On entry, **how_many** $= \langle value \rangle$, **n** $= \langle value \rangle$, **mm** $= \langle value \rangle$.
Constraint: if **how_many** = **Nag_ComputeAll** or **Nag_BackTransform**, **mm** $\geq$ **n**;
if **how_many** = **Nag_ComputeSelected**, **mm** must not be less than the number of requested eigenvectors.

On entry, **side** $= \langle value \rangle$, **mm** $= \langle value \rangle$, **pdvl** $= \langle value \rangle$.
Constraint: if **side** = **Nag_LeftSide** or **Nag_BothSides**, **pdvl** $\geq \max(1, \mathbf{mm})$;
if **side** = **Nag_RightSide**, **pdvl** ≥ 1.

On entry, **side** $= \langle value \rangle$, **mm** $= \langle value \rangle$, **pdvr** $= \langle value \rangle$.
Constraint: if **side** = **Nag_RightSide** or **Nag_BothSides**, **pdvr** $\geq \max(1, \mathbf{mm})$;
if **side** = **Nag_LeftSide**, **pdvr** ≥ 1.

NE_CONSTRAINT

General constraint: **select**$[j]$ = **TRUE** or **FALSE** for $j = 0, \ldots, n-1$.

NE_NOT_COMPLEX

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ had an illegal value.

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 consult NAG for assistance.

7 Accuracy

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

8 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).

9 Example

The example program computes the α and β parameters, which defines the generalized eigenvalues and the corresponding left and right eigenvectors, of the matrix pair (A, B) given by

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

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.

9.1 Program Text

```
/* nag_dtgevc (f08ykc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
```

```
#include <nagf08.h>
#include <nagx04.h>

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

  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double  *a=0, *alphai=0, *alphar=0, *b=0, *beta=0, *lscale=0;
  double  *q=0, *rscale=0, *tau=0, *z=0;
  Boolean *select=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)*pdq + I - 1]
#define Z(I,J) z[(J-1)*pdz + I - 1]
  order = Nag_ColMajor;
#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);
  Vprintf("f08ykc Example Program Results\n\n");

  /* ILEFT  is TRUE if left  eigenvectors are required */
  /* IRIGHT is TRUE if right eigenvectors are required */
  ileft = TRUE;
  iright = TRUE;

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");

  Vscanf("%ld %*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdb = n;
  pdq = n;
  pdz = n;
#else
  pda = n;
  pdb = n;
  pdq = n;
  pdz = n;
#endif
  alpha_len = n;
  beta_len = n;
  scale_len = n;
  tau_len = n;
  select_len = n;

  /* Allocate memory */
  if (
      !(a = NAG_ALLOC(n * n, double)) ||
      !(alphai = NAG_ALLOC(alpha_len, double)) ||
      !(alphar = NAG_ALLOC(alpha_len, double)) ||
      !(b = NAG_ALLOC(n * n, double)) ||
      !(beta = NAG_ALLOC(beta_len, double)) ||
      !(lscale = NAG_ALLOC(scale_len, double)) ||
      !(rscale = NAG_ALLOC(scale_len, double)) ||
      !(q = NAG_ALLOC(n * n, double)) ||
```

```
      !(tau = NAG_ALLOC(tau_len, double)) ||
      !(z = NAG_ALLOC(n * n, double)) ||
      !(select = NAG_ALLOC(select_len, Boolean)) )
    {
      Vprintf("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)
        Vscanf("%lf", &A(i,j));
    }
  Vscanf("%*[^\n] ");

  /* READ matrix B from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        Vscanf("%lf", &B(i,j));
    }
  Vscanf("%*[^\n] ");

  /* Balance matrix pair (A,B) */
  f08whc(order, Nag_DoBoth, n, a, pda, b, pdb, &ilo, &ihi, lscale,
         rscale, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08whc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Matrix A after balancing */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a, pda,
         "Matrix A after balancing", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Matrix B after balancing */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b, pdb,
         "Matrix B after balancing", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  Vprintf("\n");

  /* Reduce B to triangular form using QR */
  irows = ihi + 1 - ilo;
  icols = n + 1 - ilo;
  f08aec(order, irows, icols, &B(ilo, ilo), pdb, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08aec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Apply the orthogonal transformation to matrix A */
  f08agc(order, Nag_LeftSide, Nag_Trans, irows, icols, irows,
         &B(ilo, ilo), pdb, tau, &A(ilo, ilo), pda, &fail);
  if (fail.code != NE_NOERROR)
    {
```

```
      Vprintf("Error from f08agc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Initialize Q (if left eigenvectors are required) */
  if (ileft)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= n; ++j)
            Q(i,j) = 0.0;
          Q(i,i) = 1.0;
        }
      for (i = ilo+1; i <= ilo+irows-1; ++i)
        {
          for (j = ilo; j <= MIN(i,ilo+irows-2); ++j)
            Q(i,j) = B(i,j);
        }
      f08afc(order, irows, irows, irows, &Q(ilo, ilo), pdq, tau,
             &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08afc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
    }

  /* Initialize Z (if right eigenvectors are required) */
  if (iright)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= n; ++j)
            Z(i,j) = 0.0;
          Z(i,i) = 1.0;
        }
    }

  /* Compute the generalized Hessenberg form of (A,B) */
  f08wec(order, Nag_UpdateSchur, Nag_UpdateZ, n, ilo, ihi, a, pda,
         b, pdb, q, pdq, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08wec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Matrix A in generalized Hessenberg form */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a, pda,
         "Matrix A in Hessenberg form", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  Vprintf("\n");

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

```
/* Compute the generalized Schur form */
/* The Schur form also gives parameters */
/* required to compute generalized eigenvalues */
f08xec(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)
  {
    Vprintf("Error from f08xec.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Print the generalized eigenvalue parameters */
Vprintf("\n Generalized eigenvalues\n");
for (i = 1; i <= n; ++i)
  {
    if (beta[i-1] != 0.0)
      {
        Vprintf(" %4ld     (%7.3f,%7.3f)\n", i,
                alphar[i-1]/beta[i-1], alphai[i-1]/beta[i-1]);
      }
    else
      Vprintf(" %4ldEigenvalue is infinite\n", i);
  }
Vprintf("\n");

/* Compute left and right generalized eigenvectors */
/* of the balanced matrix */
f08ykc(order, Nag_BothSides, Nag_BackTransform, select, n, a, pda,
       b, pdb, q, pdq, z, pdz, n, &m, &fail);
if (fail.code != NE_NOERROR)
  {
    Vprintf("Error from f08ykc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
if (iright)
  {

    /* Compute right eigenvectors of the original matrix */
    f08wjc(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, lscale,
           rscale, n, z, pdz, &fail);
    if (fail.code != NE_NOERROR)
      {
        Vprintf("Error from f08wjc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }

    /* Print the right eigenvectors */
    x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, z, pdz,
           "Right eigenvectors", 0, &fail);
    if (fail.code != NE_NOERROR)
      {
        Vprintf("Error from x04cac.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
    Vprintf("\n");
  }

/* Compute left eigenvectors of the original matrix */
if (ileft)
  {
    f08wjc(order, Nag_DoBoth, Nag_LeftSide, n, ilo, ihi, lscale,
           rscale, n, q, pdq, &fail);
    if (fail.code != NE_NOERROR)
      {
        Vprintf("Error from f08wjc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
```

```
        }

      /* Print the left eigenvectors */
      x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, q, pdq,
             "Left eigenvectors", 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from x04cac.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
    }
 END:
  if (a) NAG_FREE(a);
  if (alphai) NAG_FREE(alphai);
  if (alphar) NAG_FREE(alphar);
  if (b) NAG_FREE(b);
  if (beta) NAG_FREE(beta);
  if (lscale) NAG_FREE(lscale);
  if (q) NAG_FREE(q);
  if (rscale) NAG_FREE(rscale);
  if (tau) NAG_FREE(tau);
  if (z) NAG_FREE(z);
  if (select) NAG_FREE(select);

  return exit_status;
}
```

9.2 Program Data

```
f08ykc Example Program Data
   5                                                           :Value of N
1.00        1.00        1.00        1.00        1.00
2.00        4.00        8.00       16.00       32.00
3.00        9.00       27.00       81.00      243.00
4.00       16.00       64.00      256.00     1024.00
5.00       25.00      125.00      625.00     3125.00          :End of matrix A
1.00        2.00        3.00        4.00        5.00
1.00        4.00        9.00       16.00       25.00
1.00        8.00       27.00       64.00      125.00
1.00       16.00       81.00      256.00      625.00
1.00       32.00      243.00     1024.00     3125.00          :End of matrix B
```

9.3 Program Results

```
f08ykc Example Program Results

 Matrix A after balancing
             1           2           3           4           5
 1      1.0000      1.0000      0.1000      0.1000      0.1000
 2      2.0000      4.0000      0.8000      1.6000      3.2000
 3      0.3000      0.9000      0.2700      0.8100      2.4300
 4      0.4000      1.6000      0.6400      2.5600     10.2400
 5      0.5000      2.5000      1.2500      6.2500     31.2500
 Matrix B after balancing
             1           2           3           4           5
 1      1.0000      2.0000      0.3000      0.4000      0.5000
 2      1.0000      4.0000      0.9000      1.6000      2.5000
 3      0.1000      0.8000      0.2700      0.6400      1.2500
 4      0.1000      1.6000      0.8100      2.5600      6.2500
 5      0.1000      3.2000      2.4300     10.2400     31.2500

 Matrix A in Hessenberg form
             1           2           3           4           5
 1     -2.1898     -0.3181      2.0547      4.7371     -4.6249
 2     -0.8395     -0.0426      1.7132      7.5194    -17.1850
 3      0.0000     -0.2846     -1.0101     -7.5927     26.4499
 4      0.0000      0.0000      0.0376      1.4070     -3.3643
 5      0.0000      0.0000      0.0000      0.3813     -0.9937
```

```
Matrix B in Hessenberg form
            1          2          3          4          5
1     -1.4248    -0.3476     2.1175     5.5813    -3.9269
2      0.0000    -0.0782     0.1189     8.0940   -15.2928
3      0.0000     0.0000     1.0021   -10.9356    26.5971
4      0.0000     0.0000     0.0000     0.5820    -0.0730
5      0.0000     0.0000     0.0000     0.0000     0.5321

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
              1             2             3             4             5
1   -4.9374e-02   -2.0772e-01    2.5702e-02   -7.4074e-02   -6.9466e-02
2    1.0606e-01    1.7848e-01    8.8325e-02    1.3545e-01    1.3605e-01
3   -1.0000e-01   -5.3742e-02   -4.6258e-02   -1.0000e-01   -1.0000e-01
4    4.3761e-02    8.0277e-03    1.3765e-02    2.6455e-02    3.1879e-02
5   -7.0192e-03   -5.5974e-04   -2.0807e-03   -3.7037e-03   -3.5534e-03

Left eigenvectors
              1             2             3             4             5
1   -6.9466e-02   -2.0922e-01   -5.2678e-03   -7.4074e-02    4.9374e-02
2    1.3605e-01    1.6346e-01    1.1371e-01    1.3545e-01   -1.0606e-01
3   -1.0000e-01   -4.6314e-02   -5.3686e-02   -1.0000e-01    1.0000e-01
4    3.1879e-02    5.9054e-03    1.4799e-02    2.6455e-02   -4.3761e-02
5   -3.5534e-03   -2.4617e-04   -2.1404e-03   -3.7037e-03    7.0192e-03
```