NAG C Library Function Document

nag_ztgevc (f08yxc)

1 Purpose

nag_ztgevc (f08yxc) computes some or all of the right and/or left generalized eigenvectors of a pair of complex upper triangular matrices (A, B).

2 Specification

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

3 Description

nag_ztgevc (f08yxc) computes some or all of the right and/or left generalized eigenvectors of the matrix pair (A, B) which is assumed to be in upper triangular form. If the matrix pair (A, B) is not upper triangular then the function nag_zhgeqz (f08xsc) should be called before invoking nag_ztgevc (f08yxc).

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_zhgeqz (f08xsc) but nag_ztgevc (f08yxc) 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_ztgevc (f08yxc).

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 unitary matrices returned by nag_zhgeqz (f08xsc). Equivalently, Q and Z are the left and right Schur vectors of the matrix pair supplied to nag_zhgeqz (f08xsc). In that case, QY and ZX are the left and right generalized eigenvectors, respectively, of the matrix pair supplied to nag_zhgeqz (f08xsc).

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**.

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 Complex *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 A must be in upper triangular form. Usually, this is the matrix A returned by nag_zhgeqz (f08xsc).

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

8: **b**[dim] – const Complex *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 B must be in upper triangular form with non-negative real diagonal elements. Usually, this is the matrix B returned by nag_zhgeqz (f08xsc)

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

10: **vl**[dim] – Complex *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 unitary matrix Q of left Schur vectors returned by nag_zhgeqz (f08xsc).

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.

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**, $\mathbf{pdvl} \geq \max(1, \mathbf{n})$;
  if **side** = **Nag_RightSide**, $\mathbf{pdvl} \geq 1$;

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

12: **vr**[dim] – Complex *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**$[(j-1) \times$ **pdvr** $+ i - 1]$ and if **order** $=$ **Nag_RowMajor**, the (i,j)th element of the matrix is stored in **vr**$[(i-1) \times$ **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 unitary 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.

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 eigenvector occupies one column.

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_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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_ztgevc (f08yxc) is the sixth step in the solution of the complex generalized eigenvalue problem and is usually called after nag_zhgeqz (f08xsc).

The real analogue of this function is nag_dtgevc (f08ykc).

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

$$A = \begin{pmatrix} 1.0 + 3.0i & 1.0 + 4.0i & 1.0 + 5.0i & 1.0 + 6.0i \\ 2.0 + 2.0i & 4.0 + 3.0i & 8.0 + 4.0i & 16.0 + 5.0i \\ 3.0 + 1.0i & 9.0 + 2.0i & 27.0 + 3.0i & 81.0 + 4.0i \\ 4.0 + 0.0i & 16.0 + 1.0i & 64.0 + 2.0i & 256.0 + 3.0i \end{pmatrix}$$

$$B = \begin{pmatrix} 1.0 + 0.0i & 2.0 + 1.0i & 3.0 + 2.0i & 4.0 + 3.0i \\ 1.0 + 1.0i & 4.0 + 2.0i & 9.0 + 3.0i & 16.0 + 4.0i \\ 1.0 + 2.0i & 8.0 + 3.0i & 27.0 + 4.0i & 64.0 + 5.0i \\ 1.0 + 3.0i & 16.0 + 4.0i & 81.0 + 5.0i & 256.0 + 6.0i \end{pmatrix}.$$

To compute generalized eigenvalues, it is required to call five functions: nag_zggbal (f08wvc) to balance the matrix, nag_zgeqrf (f08asc) to perform the QR factorization on B, nag_zunmqr (f08auc) to apply Q to A, nag_zgghrd (f08wsc) to reduce the matrix pair to the generalized Hessenberg form and nag_zhgeqz (f08xsc) to compute the eigenvalues via the QZ algorithm.

The computation of generalized eigenvectors is done by calling nag_ztgevc (f08yxc) to compute the eigenvectors of the balanced matrix pair. The function nag_zggbak (f08wwc) is called to backward transform the eigenvectors to the user-supplied matrix pair. If both left and right eigenvectors are required then nag_zggbak (f08wwc) must be called twice.

9.1 Program Text

```
/* nag_ztgevc (f08yxc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.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;
  Complex  e;
  Boolean  ileft, iright;

  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex *a=0, *alpha=0, *b=0, *beta=0, *q=0, *tau=0, *z=0;
  double  *lscale=0, *rscale=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("f08yxc 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, Complex)) ||
      !(alpha = NAG_ALLOC(alpha_len, Complex)) ||
      !(b = NAG_ALLOC(n * n, Complex)) ||
      !(beta = NAG_ALLOC(beta_len, Complex)) ||
      !(lscale = NAG_ALLOC(scale_len, double)) ||
      !(rscale = NAG_ALLOC(scale_len, double)) ||
      !(q = NAG_ALLOC(n * n, Complex)) ||
      !(tau = NAG_ALLOC(tau_len, Complex)) ||
      !(z = NAG_ALLOC(n * n, Complex)) ||
      !(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 , %lf )", &A(i,j).re, &A(i,j).im);
    }
  Vscanf("%*[^\n] ");

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

  /* Balance matrix pair (A,B) */
  f08wvc(order, Nag_DoBoth, n, a, pda, b, pdb, &ilo, &ihi, lscale,
         rscale, &fail);
  if (fail.code != NE_NOERROR)
```

```
    {
      Vprintf("Error from f08wvc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Matrix A after balancing */
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a, pda,
         Nag_BracketForm, "%7.4f", "Matrix A after balancing",
         Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Matrix B after balancing */
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b, pdb,
         Nag_BracketForm, "%7.4f", "Matrix B after balancing",
         Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\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;
  f08asc(order, irows, icols, &B(ilo, ilo), pdb, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08asc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Apply the orthogonal transformation to matrix A */
  f08auc(order, Nag_LeftSide, Nag_ConjTrans, irows, icols, irows,
         &B(ilo, ilo), pdb, tau, &A(ilo, ilo), pda, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08auc.\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).re = 0.0;
              Q(i,j).im = 0.0;
            }
          Q(i,i).re = 1.0;
        }
      for (i = ilo+1; i <= ilo+irows-1; ++i)
        {
          for (j = ilo; j <= MIN(i,ilo+irows-2); ++j)
            {
              Q(i,j).re = B(i,j).re;
              Q(i,j).im = B(i,j).im;
            }
        }
      f08atc(order, irows, irows, irows, &Q(ilo, ilo), pdq, tau,
             &fail);
```

```
    if (fail.code != NE_NOERROR)
      {
        Vprintf("Error from f08atc.\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).re = 0.0;
            Z(i,j).im = 0.0;
          }
        Z(i,i).re = 1.0;
      }
  }

/* Compute the generalized Hessenberg form of (A,B) */
f08wsc(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 f08wsc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Matrix A in generalized Hessenberg form */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a, pda,
       Nag_BracketForm, "%7.3f", "Matrix A in Hessenberg form",
       Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
  {
    Vprintf("Error from x04dbc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
Vprintf("\n");

/* Matrix B in generalized Hessenberg form */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b, pdb,
       Nag_BracketForm, "%7.3f", "Matrix B in Hessenberg form",
       Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
  {
    Vprintf("Error from x04dbc.\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 */
f08xsc(order, Nag_Schur, Nag_AccumulateQ, Nag_AccumulateZ, n, ilo, ihi, a,
       pda, b, pdb, alpha, beta, q, pdq, z, pdz, &fail);
if (fail.code != NE_NOERROR)
  {
    Vprintf("Error from f08xsc.\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].re != 0.0 || beta[i-1].im != 0.0)
        {
          e = a02cdc(alpha[i - 1], beta[i - 1]);
          Vprintf(" %4ld     (%7.3f,%7.3f)\n", i, e.re, e.im);
        }
      else
        Vprintf(" %4ldEigenvalue is infinite\n", i);
    }
  Vprintf("\n");

  /* Compute left and right generalized eigenvectors */
  /* of the balanced matrix */
  f08yxc(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 f08yxc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  if (iright)
    {

      /* Compute right eigenvectors of the original matrix */
      f08wwc(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, lscale,
             rscale, n, z, pdz, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08wwc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Print the right eigenvectors */
      x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, z, pdz,
             Nag_BracketForm, "%7.4f", "Right eigenvectors",
             Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from x04dbc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      Vprintf("\n");
    }

  /* Compute left eigenvectors of the original matrix */
  if (ileft)
    {
      f08wwc(order, Nag_DoBoth, Nag_LeftSide, n, ilo, ihi, lscale,
             rscale, n, q, pdq, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08wwc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Print the left eigenvectors */
      x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, q, pdq,
             Nag_BracketForm, "%7.4f", "Left eigenvectors",
             Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from x04dbc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
    }
 END:
  if (a) NAG_FREE(a);
```

```
  if (alpha) NAG_FREE(alpha);
  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

```
f08yxc Example Program Data
   4                                                                   :Value of N
(  1.00, 3.00)  (  1.00, 4.00)  (  1.00, 5.00)  (  1.00, 6.00)
(  2.00, 2.00)  (  4.00, 3.00)  (  8.00, 4.00)  ( 16.00, 5.00)
(  3.00, 1.00)  (  9.00, 2.00)  ( 27.00, 3.00)  ( 81.00, 4.00)
(  4.00, 0.00)  ( 16.00, 1.00)  ( 64.00, 2.00)  (256.00, 3.00)  :End of matrix A
(  1.00, 0.00)  (  2.00, 1.00)  (  3.00, 2.00)  (  4.00, 3.00)
(  1.00, 1.00)  (  4.00, 2.00)  (  9.00, 3.00)  ( 16.00, 4.00)
(  1.00, 2.00)  (  8.00, 3.00)  ( 27.00, 4.00)  ( 64.00, 5.00)
(  1.00, 3.00)  ( 16.00, 4.00)  ( 81.00, 5.00)  (256.00, 6.00)  :End of matrix B
```

9.3 Program Results

```
f08yxc Example Program Results

 Matrix A after balancing
                   1                  2                  3                  4
 1  ( 1.0000, 3.0000)  ( 1.0000, 4.0000)  ( 0.1000, 0.5000)  ( 0.1000, 0.6000)
 2  ( 2.0000, 2.0000)  ( 4.0000, 3.0000)  ( 0.8000, 0.4000)  ( 1.6000, 0.5000)
 3  ( 0.3000, 0.1000)  ( 0.9000, 0.2000)  ( 0.2700, 0.0300)  ( 0.8100, 0.0400)
 4  ( 0.4000, 0.0000)  ( 1.6000, 0.1000)  ( 0.6400, 0.0200)  ( 2.5600, 0.0300)
 Matrix B after balancing
                   1                  2                  3                  4
 1  ( 1.0000, 0.0000)  ( 2.0000, 1.0000)  ( 0.3000, 0.2000)  ( 0.4000, 0.3000)
 2  ( 1.0000, 1.0000)  ( 4.0000, 2.0000)  ( 0.9000, 0.3000)  ( 1.6000, 0.4000)
 3  ( 0.1000, 0.2000)  ( 0.8000, 0.3000)  ( 0.2700, 0.0400)  ( 0.6400, 0.0500)
 4  ( 0.1000, 0.3000)  ( 1.6000, 0.4000)  ( 0.8100, 0.0500)  ( 2.5600, 0.0600)

 Matrix A in Hessenberg form
                   1                  2                  3                  4
 1  ( -2.868, -1.595)  ( -0.809, -0.328)  ( -4.900, -0.987)  ( -0.048,  1.163)
 2  ( -2.672,  0.595)  ( -0.790,  0.049)  ( -4.955, -0.163)  ( -0.439, -0.574)
 3  (  0.000,  0.000)  ( -0.098, -0.011)  ( -1.168, -0.137)  ( -1.756, -0.205)
 4  (  0.000,  0.000)  (  0.000,  0.000)  (  0.087,  0.004)  (  0.032,  0.001)

 Matrix B in Hessenberg form
                   1                  2                  3                  4
 1  ( -1.775,  0.000)  ( -0.721,  0.043)  ( -5.021,  1.190)  ( -0.145,  0.726)
 2  (  0.000,  0.000)  ( -0.218,  0.035)  ( -2.541, -0.146)  ( -0.823, -0.418)
 3  (  0.000,  0.000)  (  0.000,  0.000)  ( -1.396, -0.163)  ( -1.747, -0.204)
 4  (  0.000,  0.000)  (  0.000,  0.000)  (  0.000,  0.000)  ( -0.100, -0.004)

 Generalized eigenvalues
    1     ( -0.635,  1.653)
    2     (  0.493,  0.910)
    3     (  0.674, -0.050)
    4     (  0.458, -0.843)

 Right eigenvectors
                   1                  2                  3                  4
 1  ( 0.0870,-0.1955)  ( 0.0550, 0.0318)  (-0.5392,-0.2697)  ( 0.0467,-0.0597)
 2  (-0.1298, 0.1446)  (-0.1060,-0.0705)  ( 0.6027, 0.1760)  (-0.0801, 0.0956)
 3  ( 0.0480,-0.0520)  ( 0.0639, 0.0361)  (-0.0726,-0.0274)  ( 0.0562,-0.0438)
 4  (-0.0069, 0.0091)  (-0.0139,-0.0030)  (-0.0042,-0.0007)  (-0.0129, 0.0042)
```

```
Left eigenvectors
                   1                   2                   3                   4
1  (-0.2725,-0.1776)  ( 0.0474, 0.0490)  (-0.1146,-0.1935)  ( 0.0765,-0.0082)
2  ( 0.2762, 0.0441)  (-0.1435,-0.0529)  ( 0.3578, 0.2103)  (-0.1643, 0.0183)
3  (-0.0954,-0.0046)  ( 0.0864, 0.0136)  (-0.0677,-0.0323)  ( 0.0952,-0.0048)
4  ( 0.0128,-0.0019)  (-0.0164, 0.0031)  ( 0.0094, 0.0034)  (-0.0179,-0.0045)
```