

# NAG 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

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

void nag_ztgevc (Nag_OrderType order, Nag_SideType side,
                Nag_HowManyType how_many, const Nag_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  $\lambda$  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\_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 you. 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 Arguments

- 1: **order** – Nag\_OrderType *Input*  
*On entry:* the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag\_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.  
*Constraint:* **order** = Nag\_RowMajor or Nag\_ColMajor.
- 2: **side** – Nag\_SideType *Input*  
*On entry:* 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 *Input*  
*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**<sub>[dim]</sub> – const Nag\_Boolean *Input*  
**Note:** the dimension, *dim*, of the array **select** must be at least **n** when **how\_many** = Nag\_ComputeSelected; otherwise **select** may be **NULL**.  
*On entry:* specifies the eigenvectors to be computed if **how\_many** = Nag\_ComputeSelected. To select the generalized eigenvector corresponding to the *j*th generalized eigenvalue, the *j*th element of **select** should be set to Nag\_TRUE.  
*Constraint:* if **how\_many** = Nag\_ComputeSelected, **select**<sub>[*j*]</sub> = Nag\_TRUE or Nag\_FALSE, for *j* = 0, 1, ..., *n* – 1.
- 5: **n** – Integer *Input*  
*On entry:* *n*, the order of the matrices *A* and *B*.  
*Constraint:* **n** ≥ 0.

- 6: **a**[*dim*] – const Complex *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least **pda** × **n**.  
The (*i*, *j*)th element of the matrix *A* is stored in  

$$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor};$$

$$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*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 row or column elements (depending on the value of **order**) in the array **a**.  
*Constraint:* **pda** ≥ max(1, **n**).
- 8: **b**[*dim*] – const Complex *Input*  
**Note:** the dimension, *dim*, of the array **b** must be at least **pdb** × **n**.  
The (*i*, *j*)th element of the matrix *B* is stored in  

$$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor};$$

$$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*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 row or column elements (depending on the value of **order**) in the array **b**.  
*Constraint:* **pdb** ≥ max(1, **n**).
- 10: **vl**[*dim*] – Complex *Input/Output*  
**Note:** the dimension, *dim*, of the array **vl** must be at least  

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

$$\mathbf{n} \times \mathbf{pdvl} \text{ when } \mathbf{side} = \text{Nag\_LeftSide} \text{ or } \text{Nag\_BothSides} \text{ and } \mathbf{order} = \text{Nag\_RowMajor};$$
otherwise **vl** may be **NULL**.  
The *i*th element of the *j*th vector is stored in  

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

$$\mathbf{vl}[(i-1) \times \mathbf{pdvl} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*On entry:* if **how\_many** = Nag\_BackTransform and **side** = Nag\_LeftSide or Nag\_BothSides, **vl** must be initialized 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 used in the array **vl**.

*Constraints:*

if **order** = Nag\_ColMajor,  
     if **side** = Nag\_LeftSide or Nag\_BothSides, **pdvl**  $\geq$  **n**;  
     if **side** = Nag\_RightSide, **vl** may be **NULL**.;  
 if **order** = Nag\_RowMajor,  
     if **side** = Nag\_LeftSide or Nag\_BothSides, **pdvl**  $\geq$  **mm**;  
     if **side** = Nag\_RightSide, **vl** may be **NULL**.

12: **vr**[*dim*] – Complex

*Input/Output*

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

**pdvr**  $\times$  **mm** when **side** = Nag\_RightSide or Nag\_BothSides and **order** = Nag\_ColMajor;  
**n**  $\times$  **pdvr** when **side** = Nag\_RightSide or Nag\_BothSides and **order** = Nag\_RowMajor;  
 otherwise **vr** may be **NULL**.

The *i*th element of the *j*th vector is stored in

**vr**[(*j* – 1)  $\times$  **pdvr** + *i* – 1] when **order** = Nag\_ColMajor;  
**vr**[(*i* – 1)  $\times$  **pdvr** + *j* – 1] when **order** = Nag\_RowMajor.

*On entry:* if **how\_many** = Nag\_BackTransform and **side** = Nag\_RightSide or Nag\_BothSides, **vr** must be initialized 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 used in the array **vr**.

*Constraints:*

if **order** = Nag\_ColMajor,  
     if **side** = Nag\_RightSide or Nag\_BothSides, **pdvr**  $\geq$  **n**;  
     if **side** = Nag\_LeftSide, **vr** may be **NULL**.;  
 if **order** = Nag\_RowMajor,  
     if **side** = Nag\_RightSide or Nag\_BothSides, **pdvr**  $\geq$  **mm**;  
     if **side** = Nag\_LeftSide, **vr** may be **NULL**.

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 \*

Input/Output

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

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_CONSTRAINT

On entry, **how\_many** =  $\langle value \rangle$  and **select**[ $j$ ] =  $\langle value \rangle$ .

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

### NE\_ENUM\_INT\_2

On entry, **how\_many** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$  and **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** =  $\langle value \rangle$ , **pdvl** =  $\langle value \rangle$ , **mm** =  $\langle value \rangle$ .

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

On entry, **side** =  $\langle value \rangle$ , **pdvl** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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

On entry, **side** =  $\langle value \rangle$ , **pdvr** =  $\langle value \rangle$ , **mm** =  $\langle value \rangle$ .

Constraint: if **side** = Nag\_RightSide or Nag\_BothSides, **pdvr**  $\geq$  **mm**.

On entry, **side** =  $\langle value \rangle$ , **pdvr** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: if **side** = Nag\_RightSide or Nag\_BothSides, **pdvr**  $\geq$  **n**.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq$  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** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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

On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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

**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**

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**

nag\_ztgevc (f08yxc) is not threaded by NAG in any implementation.

nag\_ztgevc (f08yxc) 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\_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).

**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

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

and

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

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 of  $B$ , 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.

## 10.1 Program Text

```

/* nag_ztgevc (f08yxc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf06.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>
#include <naga02.h>

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

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

    NagError     fail;
    Nag_OrderType order;
    /* Arrays */
    Complex      *a = 0, *alpha = 0, *b = 0, *beta = 0, *q = 0, *tau = 0;
    Complex      *z = 0;
    double       *lscale = 0, *rscale = 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);

    printf("nag_ztgevc (f08yxc) Example Program Results\n\n");

    /* ileft  is true if left  eigenvectors are required;
     *  iright is true if right eigenvectors are required.
     */
    ileft = Nag_TRUE;
    iright = Nag_TRUE;
    zero = nag_complex(0.0,0.0);
    one = nag_complex(1.0,0.0);

    /* Skip heading in data file and read matrix size.*/
#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 (
        !(a      = NAG_ALLOC(n * n, Complex)) ||
        !(b      = NAG_ALLOC(n * n, Complex)) ||
        !(q      = NAG_ALLOC(n * n, Complex)) ||
        !(z      = NAG_ALLOC(n * n, Complex)) ||
        !(alpha  = NAG_ALLOC(n, Complex)) ||
        !(beta   = NAG_ALLOC(n, Complex)) ||
        !(tau    = NAG_ALLOC(n, Complex)) ||
        !(lscale = NAG_ALLOC(n, double)) ||
        !(rscale = 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 , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
    #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 , %lf )", &B(i, j).re, &B(i, j).im);
#else
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
    #endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    /* Balance pair (A,B) of complex general matrices using
     * nag_zggbal (f08wvc).
     */
    nag_zggbal(order, Nag_DoBoth, n, a, pda, b, pdb, &ilo, &ihi, lscale,
               rscale, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_zggbal (f08wvc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print complex general matrices A and B after balancing using
     * nag_gen_complex_mat_print_comp (x04dbc).

```



```

*/
fflush(stdout);
nag_gen_complx_mat_print_comp(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) {
    fflush(stdout);
    nag_gen_complx_mat_print_comp(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) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}
printf("\n");

/* Reduce B to triangular form using QR and premultiply A by Q^H. */
irows = ihi + 1 - ilo;
icols = n + 1 - ilo;
/* nag_zgeqrf (f08asc).
 * QR factorization of complex general rectangular matrix B.
 */
nag_zgeqrf(order, irows, icols, &B(ilo, ilo), pdb, tau, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgeqrf (f08asc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Apply the orthogonal transformation Q^H to matrix A using
 * nag_zunmqr (f08auc).
 */
nag_zunmqr(order, Nag_LeftSide, Nag_ConjTrans, irows, icols, irows,
           &B(ilo, ilo), pdb, tau, &A(ilo, ilo), pda, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zunmqr (f08auc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Initialize Q (if left eigenvectors are required) */
if (ileft) {
    /* Q = I */
    nag_zge_load(order, n, n, zero, one, q, pdq, &fail);
    /* Q = B using nag_zge_copy (f16tfc). */
    nag_zge_copy(order, Nag_NoTrans, irows-1, irows-1, &B(ilo+1,ilo), pdb,
                &Q(ilo+1,ilo), pdq, &fail);
    /* Form Q from QR factorization using nag_zungqr (f08atc). */
    nag_zungqr(order, irows, irows, irows, &Q(ilo, ilo), pdq, tau, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_zungqr (f08atc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}

if (iright) {
    /* Z = I. */
    nag_zge_load(order, n, n, zero, one, z, pdz, &fail);
}

/* Compute the generalized Hessenberg form of (A,B) by Unitary reduction
 * using nag_zgghrd (f08wsc).
 */

```

```

nag_zgghrd(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_zgghrd (f08wsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print generalized Hessenberg form of (A,B) using
 * nag_gen_complx_mat_print_comp (x04dbc).
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(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) {
    printf("\n");

    fflush(stdout);
    nag_gen_complx_mat_print_comp(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) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Compute the generalized Schur form - nag_zhgeqz (f08xsc).
 * Eigenvalues and generalized Schur factorization of
 * complex generalized upper Hessenberg form reduced from a
 * pair of complex general matrices
 */
nag_zhgeqz(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) {
    printf("Error from nag_zhgeqz (f08xsc).\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].re != 0.0 || beta[i].im != 0.0) {
        /* nag_complex_divide (a02cdc) - Quotient of two complex numbers. */
        e = nag_complex_divide(alpha[i], beta[i]);
        printf(" %4"NAG_IFMT" (%7.3f,%7.3f)\n", i+1, e.re, e.im);
    }
    else
        printf(" %4"NAG_IFMT"Eigenvalue is infinite\n", i+1);
}
printf("\n");

/* nag_ztgevc (f08yxc).
 * Left and right eigenvectors of a pair of complex upper
 * triangular matrices
 */
nag_ztgevc(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_ztgevc (f08yxc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
if (iright) {

```

```

/* nag_zggbak (f08wwc).
 * Transform eigenvectors of a pair of complex balanced
 * matrices to those of original matrix pair supplied to
 * nag_zggbal (f08wvc)
 */
nag_zggbak(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, lscale,
           rscale, n, z, pdz, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zggbak (f08wwc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Normalize and print the right eigenvectors */
exit_status = normalize_vectors(order, n, z, "Right eigenvectors");
printf("\n");
}

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

    /* Normalize and print the left eigenvectors */
    exit_status = normalize_vectors(order, n, q, "Left eigenvectors");
}
END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(q);
NAG_FREE(z);
NAG_FREE(alpha);
NAG_FREE(beta);
NAG_FREE(tau);
NAG_FREE(lscale);
NAG_FREE(rscale);

return exit_status;
}

static Integer normalize_vectors(Nag_OrderType order, Integer n, Complex qz[],
                               const char* title)
{
    /* Each complex eigenvector z[] is normalized so that the element of largest
     * magnitude is scaled to be (1.0,0.0).
     */

    double          r;
    Integer          colinc, rowinc, j, k, indqz, errors=0;
    Complex          alpha, beta, x[1];
    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++) {
        /* Find element of eigenvector with largest absolute value using

```

```

    * nag_zamax_val (f16jsc).
    */
nag_zamax_val(n, &qz[indqz], rowinc, &k, &r, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zamax_val (f16jac).\n%s\n", fail.message);
    errors = 1;
    goto END;
}

/* Use nag_complex_divide (a02cdc) to form reciprocal of qz[indqz+k]. */
beta = nag_complex_divide(nag_complex(1.0,0.0),qz[indqz+k]);

/* nag_zaxpby (f16gcc) performs y := alpha*x + beta*y;
 * here to make largest element (1,0).
 */
alpha = nag_complex(0.0,0.0);
nag_zaxpby(n, alpha, x, 1, beta, &qz[indqz], rowinc, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zaxpby (f16gcc).\n%s\n", fail.message);
    errors = 2;
    goto END;
}
indqz += colinc;
}
/* Print the normalized eigenvectors using
 * nag_gen_complx_mat_print_comp (x04dbc)
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                              n, n, qz, n, Nag_BracketForm, "%7.4f",
                              title, Nag_IntegerLabels, 0,
                              Nag_IntegerLabels, 0, 80, 0, 0,
                              &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    errors = 3;
}
END:
return errors;
}

```

## 10.2 Program Data

```

nag_ztgevc (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

```

## 10.3 Program Results

```

nag_ztgevc (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 ( -2.868, -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 ( -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 ( 0.458, -0.843)
3 ( 0.674, -0.050)
4 ( 0.493, 0.910)

```

Right eigenvectors

```

1 ( 1.0000, 0.0000) (-0.6076, 0.0203) (-0.9448,-0.1716) (-0.4982, 0.0310)
2 (-0.8639,-0.2796) ( 1.0000, 0.0000) ( 1.0000, 0.0000) ( 1.0000, 0.0000)
3 ( 0.3132, 0.1060) (-0.5587,-0.1192) (-0.1233,-0.0094) (-0.5749, 0.0421)
4 (-0.0518,-0.0122) ( 0.0924, 0.0578) (-0.0067, 0.0009) ( 0.1040,-0.0406)

```

Left eigenvectors

```

1 ( 1.0000, 0.0000) (-0.4651,-0.0020) (-0.4744,-0.2622) (-0.4018,-0.1933)
2 (-0.7857, 0.3499) ( 1.0000, 0.0000) ( 1.0000, 0.0000) ( 1.0000, 0.0000)
3 ( 0.2535,-0.1483) (-0.5755,-0.0349) (-0.1801, 0.0156) (-0.5610, 0.1120)
4 (-0.0297, 0.0264) ( 0.1048, 0.0389) ( 0.0237,-0.0044) ( 0.0937,-0.0562)

```

---