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

```c
#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


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

5:  
   n – Integer
       
   On entry: n, the order of the matrices A and B.
   
   Constraint: n \geq 0.
6: \( \mathbf{a}[\text{dim}] \) – const Complex

**Input**

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

The \((i, j)\)th element of the matrix \( \mathbf{A} \) is stored in

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

\[
\mathbf{a}[(i - 1) \times \text{pda} + j - 1] \quad \text{when order} = \text{Nag\_RowMajor}.
\]

**On entry:** the matrix \( \mathbf{A} \) must be in upper triangular form. Usually, this is the matrix \( \mathbf{A} \) returned by \text{nag\_zhgeqz} (f08xsc).

7: \( \text{pda} \) – Integer

**Input**

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

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

8: \( \mathbf{b}[\text{dim}] \) – const Complex

**Input**

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

The \((i, j)\)th element of the matrix \( \mathbf{B} \) is stored in

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

\[
\mathbf{b}[(i - 1) \times \text{pdb} + j - 1] \quad \text{when order} = \text{Nag\_RowMajor}.
\]

**On entry:** the matrix \( \mathbf{B} \) must be in upper triangular form with non-negative real diagonal elements. Usually, this is the matrix \( \mathbf{B} \) returned by \text{nag\_zhgeqz} (f08xsc).

9: \( \text{pdb} \) – Integer

**Input**

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

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

10: \( \mathbf{vl}[\text{dim}] \) – Complex

**Input/Output**

**Note:** the dimension, \( \text{dim} \), of the array \( \mathbf{vl} \) must be at least

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

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

otherwise \( \mathbf{vl} \) may be \text{NULL}.

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

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

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

**On entry:** if \text{how\_many} = \text{Nag\_BackTransform} and \text{side} = \text{Nag\_LeftSide} or \text{Nag\_BothSides}, \( \mathbf{vl} \) must be initialized to an \( n \times n \) matrix \( Q \). Usually, this is the unitary matrix \( Q \) of left Schur vectors returned by \text{nag\_zhgeqz} (f08xsc).

**On exit:** if \text{side} = \text{Nag\_LeftSide} or \text{Nag\_BothSides}, \( \mathbf{vl} \) contains:

- if \text{how\_many} = \text{Nag\_ComputeAll}, the matrix \( Y \) of left eigenvectors of \( (A, B) \);
- if \text{how\_many} = \text{Nag\_BackTransform}, the matrix \( QY \);
- if \text{how\_many} = \text{Nag\_ComputeSelected}, the left eigenvectors of \( (A, B) \) specified by \text{select}, stored consecutively in the rows or columns (depending on the value of \text{order}) of the array \( \mathbf{vl} \), in the same order as their corresponding eigenvalues.

11: \( \text{pdvl} \) – Integer

**Input**

**On entry:** the stride used in the array \( \mathbf{vl} \).
Constraints:

if order = Nag_ColMajor,
    if side = Nag_LeftSide or Nag_BothSides, pdvl ≥ n;
    if side = Nag_RightSide, vl may be NULL.
if order = Nag_RowMajor,
    if side = Nag_LeftSide or Nag_BothSides, pdvl ≥ mm;
    if side = Nag_RightSide, vl may be NULL.

12: \( \text{vr} \left[ \text{dim} \right] \) – Complex \hspace{1cm} \text{Input/Output}

Note: the dimension, \( \text{dim} \), of the array \( \text{vr} \) must be at least
\( \text{pdvl} \times \text{mm} \) when side = Nag_RightSide or Nag_BothSides and order = Nag_ColMajor;
\( n \times \text{pdvl} \) when side = Nag_RightSide or Nag_BothSides and order = Nag_RowMajor;
otherwise \( \text{vr} \) may be NULL.

The \( i \)-th element of the \( j \)-th vector is stored in
\( \text{vr} \left[ \left( j - 1 \right) \times \text{pdvl} + i - 1 \right] \) when order = Nag_ColMajor;
\( \text{vr} \left[ \left( i - 1 \right) \times \text{pdvl} + j - 1 \right] \) when order = Nag_RowMajor.

On entry: if how_many = Nag_BackTransform and side = Nag_RightSide or Nag_BothSides, \( \text{vr} \) 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, \( \text{vr} \) 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 \( \text{vr} \), in the same order as their corresponding eigenvalues.

13: \( \text{pdvr} \) – Integer \hspace{1cm} \text{Input}

On entry: the stride used in the array \( \text{vr} \).

Constraints:

if order = Nag_ColMajor,
    if side = Nag_RightSide or Nag_BothSides, pdvr ≥ n;
    if side = Nag_LeftSide, \( \text{vl} \) may be NULL.
if order = Nag_RowMajor,
    if side = Nag_RightSide or Nag_BothSides, pdvr ≥ mm;
    if side = Nag_LeftSide, \( \text{vl} \) may be NULL.

14: \( \text{mm} \) – Integer \hspace{1cm} \text{Input}

On entry: the number of columns in the arrays \( \text{vl} \) and/or \( \text{vr} \).

Constraints:

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

15: \( \text{m} \) – Integer \* \hspace{1cm} \text{Output}

On exit: the number of columns in the arrays \( \text{vl} \) and/or \( \text{vr} \) actually used to store the eigenvectors.
If how_many = Nag_ComputeAll or Nag_BackTransform, \( \text{m} \) is set to \( n \). Each selected eigenvector occupies one column.
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 \( j = 0, 1, \ldots, n - 1 \).

**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, n).

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

\texttt{f08yxc} is not threaded by NAG in any implementation.

\texttt{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

\texttt{f08yxc} is the sixth step in the solution of the complex generalized eigenvalue problem and is usually called after \texttt{f08xsc}.

The real analogue of this function is \texttt{f08yc}. The real analogue of this function is \texttt{f08yc}.

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: \texttt{f08wvc} to balance the matrix, \texttt{f08asc} to perform the $QR$ factorization of $B$, \texttt{f08auc} to apply $Q$ to $A$, \texttt{f08wsc} to reduce the matrix pair to the generalized Hessenberg form and \texttt{f08xsc} to compute the eigenvalues via the $QZ$ algorithm.

The computation of generalized eigenvectors is done by calling \texttt{f08yxc} to compute the eigenvectors of the balanced matrix pair. The function \texttt{f08wvc} 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;

    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

/* 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)) ||
  !(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

/* 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

/* 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\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_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, 
n, a, pda, NagBracketForm, "%7.4f",
"Matrix A after balancing",
NagIntegerLabels, 0, NagIntegerLabels, 0, 80, 
0, 0, &fail);

if (fail.code == NE_NOERROR) {
  fflush(stdout);
  nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, 
n, b, pdb, NagBracketForm, "%7.4f",
"Matrix B after balancing",
NagIntegerLabels, 0, NagIntegerLabels, 0, 80, 
0, 0, &fail);
}

if (fail.code != NE_NOERROR) {
  printf("Error from nag_gen_complex_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, icols, 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)
", 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(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");

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\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\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\n", fail.message);
    errors = 3;
} END: return errors;

10.2 Program Data

nag_ztgevc (f08yxc) Example Program Data

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.00, 3.000</td>
<td>1.00, 4.000</td>
<td>1.00, 5.000</td>
<td>1.00, 6.000</td>
</tr>
<tr>
<td>2</td>
<td>2.00, 2.000</td>
<td>4.00, 3.000</td>
<td>8.00, 4.000</td>
<td>16.00, 5.000</td>
</tr>
<tr>
<td>3</td>
<td>3.00, 1.000</td>
<td>9.00, 2.000</td>
<td>27.00, 3.000</td>
<td>81.00, 4.000</td>
</tr>
<tr>
<td>4</td>
<td>4.00, 0.000</td>
<td>16.00, 1.000</td>
<td>64.00, 2.000</td>
<td>256.00, 3.000</td>
</tr>
</tbody>
</table>

:Value of N

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00, 0.000</td>
<td>2.00, 1.000</td>
<td>3.00, 2.000</td>
<td>4.00, 3.000</td>
</tr>
<tr>
<td>2</td>
<td>1.00, 0.000</td>
<td>4.00, 2.000</td>
<td>9.00, 3.000</td>
<td>16.00, 4.000</td>
</tr>
<tr>
<td>3</td>
<td>1.00, 2.000</td>
<td>8.00, 3.000</td>
<td>27.00, 4.000</td>
<td>64.00, 5.000</td>
</tr>
<tr>
<td>4</td>
<td>1.00, 3.000</td>
<td>16.00, 4.000</td>
<td>81.00, 5.000</td>
<td>256.00, 6.000</td>
</tr>
</tbody>
</table>

:End of matrix A

10.3 Program Results

nag_ztgevc (f08yxc) Example Program Results

Matrix A after balancing

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000, 3.0000</td>
<td>1.0000, 4.0000</td>
<td>0.1000, 0.5000</td>
<td>0.1000, 0.6000</td>
</tr>
<tr>
<td>2</td>
<td>2.0000, 2.0000</td>
<td>4.0000, 3.0000</td>
<td>0.8000, 0.4000</td>
<td>1.6000, 0.5000</td>
</tr>
<tr>
<td>3</td>
<td>0.3000, 0.1000</td>
<td>0.9000, 0.2000</td>
<td>0.2700, 0.0300</td>
<td>0.8100, 0.0400</td>
</tr>
<tr>
<td>4</td>
<td>0.4000, 0.0000</td>
<td>1.6000, 0.1000</td>
<td>0.6400, 0.0200</td>
<td>2.5600, 0.0300</td>
</tr>
</tbody>
</table>

Matrix B after balancing

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000, 0.0000</td>
<td>2.0000, 1.0000</td>
<td>0.3000, 0.2000</td>
<td>0.4000, 0.3000</td>
</tr>
<tr>
<td>2</td>
<td>1.0000, 1.0000</td>
<td>4.0000, 2.0000</td>
<td>0.9000, 0.3000</td>
<td>1.6000, 0.4000</td>
</tr>
</tbody>
</table>

Mark 25
Matrix A in Hessenberg form

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-2.868, -1.595)</td>
<td>(-0.809, -0.328)</td>
<td>(-4.900, -0.987)</td>
<td>(-0.048, 1.163)</td>
</tr>
<tr>
<td>2</td>
<td>(-2.672, 0.595)</td>
<td>(-0.790, -0.328)</td>
<td>(-4.955, -0.163)</td>
<td>(-0.439, -0.574)</td>
</tr>
<tr>
<td>3</td>
<td>(0.000, 0.000)</td>
<td>(-0.098, -0.011)</td>
<td>(-1.168, -0.137)</td>
<td>(-1.756, -0.205)</td>
</tr>
<tr>
<td>4</td>
<td>(0.000, 0.000)</td>
<td>(0.000, 0.000)</td>
<td>(0.087, 0.004)</td>
<td>(0.032, 0.001)</td>
</tr>
</tbody>
</table>

Matrix B in Hessenberg form

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-1.775, 0.000)</td>
<td>(-0.721, 0.043)</td>
<td>(-5.021, 1.190)</td>
<td>(-0.145, 0.726)</td>
</tr>
<tr>
<td>2</td>
<td>(0.000, 0.000)</td>
<td>(-0.218, 0.035)</td>
<td>(-2.541, -0.146)</td>
<td>(-0.823, -0.418)</td>
</tr>
<tr>
<td>3</td>
<td>(0.000, 0.000)</td>
<td>(0.000, 0.000)</td>
<td>(-1.396, -0.163)</td>
<td>(-1.747, -0.204)</td>
</tr>
<tr>
<td>4</td>
<td>(0.000, 0.000)</td>
<td>(0.000, 0.000)</td>
<td>(0.000, 0.000)</td>
<td>(-0.100, -0.004)</td>
</tr>
</tbody>
</table>

Generalized eigenvalues

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

Right eigenvectors

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1.0000, 0.0000)</td>
<td>(-0.6076, 0.0203)</td>
<td>(-0.9448, -0.1716)</td>
<td>(-0.4982, 0.0310)</td>
</tr>
<tr>
<td>2</td>
<td>(-0.8639, -0.2796)</td>
<td>(1.0000, 0.0000)</td>
<td>(1.0000, 0.0000)</td>
<td>(1.0000, 0.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(0.3132, 0.1060)</td>
<td>(-0.5587, -0.1192)</td>
<td>(-0.1233, -0.0094)</td>
<td>(-0.5749, 0.0421)</td>
</tr>
<tr>
<td>4</td>
<td>(-0.0518, -0.0122)</td>
<td>(0.0924, 0.0578)</td>
<td>(-0.0067, 0.0009)</td>
<td>(0.1040, -0.0406)</td>
</tr>
</tbody>
</table>

Left eigenvectors

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1.0000, 0.0000)</td>
<td>(-0.4651, -0.0020)</td>
<td>(-0.4744, -0.2622)</td>
<td>(-0.4018, -0.1933)</td>
</tr>
<tr>
<td>2</td>
<td>(-0.7857, 0.3499)</td>
<td>(1.0000, 0.0000)</td>
<td>(1.0000, 0.0000)</td>
<td>(1.0000, 0.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(0.2535, -0.1483)</td>
<td>(-0.5755, -0.0349)</td>
<td>(-0.1801, 0.0156)</td>
<td>(-0.5610, 0.1120)</td>
</tr>
<tr>
<td>4</td>
<td>(-0.0297, 0.0264)</td>
<td>(0.1048, 0.0389)</td>
<td>(0.0237, -0.0044)</td>
<td>(0.0937, -0.0562)</td>
</tr>
</tbody>
</table>