NAG Library Function Document

nag_zggesx (f08xpc)

1 Purpose

nag_zggesx (f08xpc) computes the generalized eigenvalues, the generalized Schur form (S,T) and, optionally, the left and/or right generalized Schur vectors for a pair of n by n complex nonsymmetric matrices (A,B).

Estimates of condition numbers for selected generalized eigenvalue clusters and Schur vectors are also computed.

2 Specification

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

```
void nag_zggesx (Nag_OrderType order, Nag_LeftVecsType jobvsl,
     Nag_RightVecsType jobvsr, Nag_SortEigValsType sort,
     Nag_Boolean (*selctg)(Complex a, Complex b),
     Nag_RCondType sense, Integer n, Complex a[], Integer pda, Complex b[],
     Integer pdb, Integer *sdim, Complex alpha[], Complex beta[],
     Complex vsl[], Integer pdvsl, Complex vsr[], Integer pdvsr,
     double rconde[], double rcondv[], NagError *fail)
```

3 Description

The generalized Schur factorization for a pair of complex matrices (A,B) is given by

$$A = QSZ^{\mathrm{H}}, \quad B = QTZ^{\mathrm{H}},$$

where Q and Z are unitary, T and S are upper triangular. The generalized eigenvalues, λ, of (A,B) are computed from the diagonals of T and S and satisfy

$$Az = \lambda Bz,$$

where z is the corresponding generalized eigenvector. λ is actually returned as the pair (α,β) such that

$$\lambda = \alpha/\beta$$

since β, or even both α and β can be zero. The columns of Q and Z are the left and right generalized Schur vectors of (A,B).

Optionally, nag_zggesx (f08xpc) can order the generalized eigenvalues on the diagonals of (S,T) so that selected eigenvalues are at the top left. The leading columns of Q and Z then form an orthonormal basis for the corresponding eigenspaces, the deflating subspaces.

nag_zggesx (f08xpc) computes T to have real non-negative diagonal entries. The generalized Schur factorization, before reordering, is computed by the QZ algorithm.

The reciprocals of the condition estimates, the reciprocal values of the left and right projection norms, are returned in **rconde**[0] and **rconde**[1] respectively, for the selected generalized eigenvalues, together with reciprocal condition estimates for the corresponding left and right deflating subspaces, in **rcondv**[0] and **rcondv**[1]. See Section 4.11 of Anderson *et al.* (1999) for further information.

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 http://www.netlib.org/lapack/lug

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

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: **jobvsl** – Nag_LeftVecsType *Input*

On entry: if **jobvsl** = Nag_NotLeftVecs, do not compute the left Schur vectors.

If **jobvsl** = Nag_LeftVecs, compute the left Schur vectors.

Constraint: **jobvsl** = Nag_NotLeftVecs or Nag_LeftVecs.

3: **jobvsr** – Nag_RightVecsType *Input*

On entry: if **jobvsr** = Nag_NotRightVecs, do not compute the right Schur vectors.

If **jobvsr** = Nag_RightVecs, compute the right Schur vectors.

Constraint: **jobvsr** = Nag_NotRightVecs or Nag_RightVecs.

4: **sort** – Nag_SortEigValsType *Input*

On entry: specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form.

sort = Nag_NoSortEigVals
Eigenvalues are not ordered.

sort = Nag_SortEigVals
Eigenvalues are ordered (see **selctg**).

Constraint: **sort** = Nag_NoSortEigVals or Nag_SortEigVals.

5: **selctg** – function, supplied by the user *External Function*

If **sort** = Nag_SortEigVals, **selctg** is used to select generalized eigenvalues to the top left of the generalized Schur form.

If **sort** = Nag_NoSortEigVals, **selctg** is not referenced by nag_zggesx (f08xpc), and may be specified as NULLFN.

The specification of **selctg** is:

```
Nag_Boolean selctg (Complex a, Complex b)
```

1: **a** – Complex *Input*
2: **b** – Complex *Input*

On entry: an eigenvalue $\mathbf{a}[j-1]/\mathbf{b}[j-1]$ is selected if $\mathbf{selctg}(\mathbf{a}[j-1], \mathbf{b}[j-1])$ is Nag_TRUE.

Note that in the ill-conditioned case, a selected generalized eigenvalue may no longer satisfy **selctg**($\mathbf{a}[j-1], \mathbf{b}[j-1]$) = Nag_TRUE after ordering. **fail.code** = NE_SCHUR_REORDER_SELECT in this case.

6: **sense** – Nag_RCondType *Input*

On entry: determines which reciprocal condition numbers are computed.

sense = Nag_NotRCond
None are computed.

sense = Nag_RCondEigVals
Computed for average of selected eigenvalues only.

sense = Nag_RCondEigVecs
Computed for selected deflating subspaces only.

sense = Nag_RCondBoth
Computed for both.

If **sense** = Nag_RCondEigVals, Nag_RCondEigVecs or Nag_RCondBoth, **sort** = Nag_SortEigVals.

Constraint: **sense** = Nag_NotRCond, Nag_RCondEigVals, Nag_RCondEigVecs or Nag_RCondBoth.

7: **n** – Integer *Input*

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

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

8: **a**[*dim*] – Complex *Input/Output*

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

The (i, j)th element of the matrix A is stored in

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the first of the pair of matrices, A.

On exit: **a** has been overwritten by its generalized Schur form S.

9: **pda** – Integer *Input*

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

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

10: **b**[*dim*] – Complex *Input/Output*

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

The (i, j)th element of the matrix B is stored in

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the second of the pair of matrices, B.

On exit: **b** has been overwritten by its generalized Schur form T.

11: **pdb** – Integer *Input*

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

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

12: **sdim** – Integer * *Output*

On exit: if $\mathbf{sort} = \text{Nag\_NoSortEigVals}$, $\mathbf{sdim} = 0$.

If $\mathbf{sort} = \text{Nag\_SortEigVals}$, $\mathbf{sdim} =$ number of eigenvalues (after sorting) for which **selctg** is Nag_TRUE.

13: **alpha**[**n**] – Complex *Output*

On exit: see the description of **beta**.

14: **beta**[**n**] – Complex *Output*

On exit: $\mathbf{alpha}[j-1]/\mathbf{beta}[j-1]$, for $j = 1, 2, \ldots, \mathbf{n}$, will be the generalized eigenvalues. $\mathbf{alpha}[j-1]$ and $\mathbf{beta}[j-1], j = 1, 2, \ldots, \mathbf{n}$ are the diagonals of the complex Schur form (S, T). $\mathbf{beta}[j-1]$ will be non-negative real.

Note: the quotients $\mathbf{alpha}[j-1]/\mathbf{beta}[j-1]$ may easily overflow or underflow, and $\mathbf{beta}[j-1]$ may even be zero. Thus, you should avoid naively computing the ratio α/β. However, **alpha** will always be less than and usually comparable with $\|\mathbf{a}\|$ in magnitude, and **beta** will always be less than and usually comparable with $\|\mathbf{b}\|$.

15: **vsl**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvsl} \times \mathbf{n})$ when $\mathbf{jobvsl} = \text{Nag\_LeftVecs}$;
1 otherwise.

The ith element of the jth vector is stored in

$\mathbf{vsl}[(j-1) \times \mathbf{pdvsl} + i - 1]$ when $\mathbf{order} = \text{Nag\_ColMajor}$;
$\mathbf{vsl}[(i-1) \times \mathbf{pdvsl} + j - 1]$ when $\mathbf{order} = \text{Nag\_RowMajor}$.

On exit: if $\mathbf{jobvsl} = \text{Nag\_LeftVecs}$, **vsl** will contain the left Schur vectors, Q.

If $\mathbf{jobvsl} = \text{Nag\_NotLeftVecs}$, **vsl** is not referenced.

16: **pdvsl** – Integer *Input*

On entry: the stride used in the array **vsl**.

Constraints:

if $\mathbf{jobvsl} = \text{Nag\_LeftVecs}$, $\mathbf{pdvsl} \geq \max(1, \mathbf{n})$;
otherwise $\mathbf{pdvsl} \geq 1$.

17: **vsr**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvsr} \times \mathbf{n})$ when $\mathbf{jobvsr} = \text{Nag\_RightVecs}$;
1 otherwise.

The ith element of the jth vector is stored in

$\mathbf{vsr}[(j-1) \times \mathbf{pdvsr} + i - 1]$ when $\mathbf{order} = \text{Nag\_ColMajor}$;
$\mathbf{vsr}[(i-1) \times \mathbf{pdvsr} + j - 1]$ when $\mathbf{order} = \text{Nag\_RowMajor}$.

On exit: if $\mathbf{jobvsr} = \text{Nag\_RightVecs}$, **vsr** will contain the right Schur vectors, Z.

If $\mathbf{jobvsr} = \text{Nag\_NotRightVecs}$, **vsr** is not referenced.

18: **pdvsr** – Integer *Input*

On entry: the stride used in the array **vsr**.

Constraints:

if **jobvsr** $=$ Nag_RightVecs, **pdvsr** $\geq \max(1, \mathbf{n})$;
otherwise **pdvsr** ≥ 1.

19: **rconde**[**2**] – double *Output*

On exit: if **sense** $=$ Nag_RCondEigVals or Nag_RCondBoth, **rconde**[0] and **rconde**[1] contain the reciprocal condition numbers for the average of the selected eigenvalues.

If **sense** $=$ Nag_NotRCond or Nag_RCondEigVecs, **rconde** is not referenced.

20: **rcondv**[**2**] – double *Output*

On exit: if **sense** $=$ Nag_RCondEigVecs or Nag_RCondBoth, **rcondv**[0] and **rcondv**[1] contain the reciprocal condition numbers for the selected deflating subspaces.

if **sense** $=$ Nag_NotRCond or Nag_RCondEigVals, **rcondv** is not referenced.

21: **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_ENUM_INT_2

On entry, **jobvsl** $= \langle$*value*$\rangle$, **pdvsl** $= \langle$*value*$\rangle$ and **n** $= \langle$*value*$\rangle$.
Constraint: if **jobvsl** $=$ Nag_LeftVecs, **pdvsl** $\geq \max(1, \mathbf{n})$;
otherwise **pdvsl** ≥ 1.

On entry, **jobvsr** $= \langle$*value*$\rangle$, **pdvsr** $= \langle$*value*$\rangle$ and **n** $= \langle$*value*$\rangle$.
Constraint: if **jobvsr** $=$ Nag_RightVecs, **pdvsr** $\geq \max(1, \mathbf{n})$;
otherwise **pdvsr** ≥ 1.

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, **pdvsl** $= \langle$*value*$\rangle$.
Constraint: **pdvsl** > 0.

On entry, **pdvsr** $= \langle$*value*$\rangle$.
Constraint: **pdvsr** > 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_ITERATION_QZ

The QZ iteration failed. (A, B) are not in Schur form, but **alpha**$[j]$ and **beta**$[j]$ should be correct from element $\langle value \rangle$.

The QZ iteration failed with an unexpected error, please contact NAG.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_SCHUR_REORDER

The eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned).

NE_SCHUR_REORDER_SELECT

After reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the generalized Schur form no longer satisfy **selctg** = Nag_TRUE. This could also be caused by underflow due to scaling.

7 Accuracy

The computed generalized Schur factorization satisfies

$$A + E = QSZ^{\mathrm{T}}, \quad B + F = QTZ^{\mathrm{T}},$$

where

$$\|(E, F)\|_F = O(\epsilon)\|(A, B)\|_F$$

and ϵ is the ***machine precision***. See Section 4.11 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_zggesx (f08xpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zggesx (f08xpc) 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

The total number of floating-point operations is proportional to n^3.

The real analogue of this function is nag_dggesx (f08xbc).

10 Example

This example finds the generalized Schur factorization of the matrix pair (A, B), where

$$A = \begin{pmatrix} -21.10 - 22.50i & 53.50 - 50.50i & -34.50 + 127.50i & 7.50 + \ 0.50i \\ -0.46 - \ 7.78i & -3.50 - 37.50i & -15.50 + \ 58.50i & -10.50 - \ 1.50i \\ 4.30 - \ 5.50i & 39.70 - 17.10i & -68.50 + \ 12.50i & -7.50 - \ 3.50i \\ 5.50 + \ 4.40i & 14.40 + 43.30i & -32.50 - \ 46.00i & -19.00 - 32.50i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 1.00 - 5.00i & 1.60 + 1.20i & -3.00 + 0.00i & 0.00 - 1.00i \\ 0.80 - 0.60i & 3.00 - 5.00i & -4.00 + 3.00i & -2.40 - 3.20i \\ 1.00 + 0.00i & 2.40 + 1.80i & -4.00 - 5.00i & 0.00 - 3.00i \\ 0.00 + 1.00i & -1.80 + 2.40i & 0.00 - 4.00i & 4.00 - 5.00i \end{pmatrix},$$

such that the eigenvalues of (A, B) for which $|\lambda| < 6$ correspond to the top left diagonal elements of the generalized Schur form, (S, T). Estimates of the condition numbers for the selected eigenvalue cluster and corresponding deflating subspaces are also returned.

10.1 Program Text

```
/* nag_zggesx (f08xpc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 25, 2014.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif
  static Nag_Boolean NAG_CALL selctg(const Complex a, const Complex b);
#ifdef __cplusplus
}
#endif

int main(void)
{

  /* Scalars */
  Complex              alph, bet, z;
  double               abnorm, norma, normb, normd, norme, eps, tol;
  Integer              i, j, n, sdim, pda, pdb, pdc, pdd, pde, pdvsl, pdvsr;
  Integer              exit_status = 0;

  /* Arrays */
  Complex              *a = 0, *alpha = 0, *b = 0, *beta = 0, *c = 0, *d = 0;
  Complex              *e = 0, *vsl = 0, *vsr = 0;
  double               rconde[2], rcondv[2];
  char                 nag_enum_arg[40];
  /* Nag Types */
  NagError            fail;
  Nag_OrderType       order;
  Nag_LeftVecsType    jobvsl;
  Nag_RightVecsType   jobvsr;
  Nag_SortEigValsType sort = Nag_SortEigVals;
  Nag_RCondType       sense;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + 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]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zggesx (f08xpc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif
  if (n < 0)
    {
      printf("Invalid n\n");
      exit_status = 1;
      return exit_status;
    }
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n]", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  jobvsl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n]", nag_enum_arg);
#endif
  jobvsr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n]", nag_enum_arg);
#endif
  sense = (Nag_RCondType) nag_enum_name_to_value(nag_enum_arg);

  pdvsl = (jobvsl==Nag_LeftVecs?n:1);
  pdvsr = (jobvsr==Nag_RightVecs?n:1);
  pda = n;
  pdb = n;
  pdc = n;
  pdd = n;
  pde = n;
  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, Complex)) ||
      !(b = NAG_ALLOC(n * n, Complex)) ||
      !(c = NAG_ALLOC(n * n, Complex)) ||
      !(d = NAG_ALLOC(n * n, Complex)) ||
      !(e = NAG_ALLOC(n * n, Complex)) ||
      !(alpha = NAG_ALLOC(n, Complex)) ||
      !(beta = NAG_ALLOC(n, Complex)) ||
      !(vsl = NAG_ALLOC(pdvsl*pdvsl, Complex)) ||
      !(vsr = NAG_ALLOC(pdvsr*pdvsr, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read in the matrices A and B */
  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
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  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
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Copy matrices A and B to matrices D and E using nag_zge_copy (f16tfc),
   * Complex valued general matrix copy.
   * The copies will be used as comparison against reconstructed matrices.
   */
  nag_zge_copy(order, Nag_NoTrans, n, n, a, pda, d, pdd, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  nag_zge_copy(order, Nag_NoTrans, n, n, b, pdb, e, pde, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_zge_norm (f16uac): Find norms of input matrices A and B. */
  nag_zge_norm(order, Nag_FrobeniusNorm, n, n, a, pda, &norma, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  nag_zge_norm(order, Nag_FrobeniusNorm, n, n, b, pdb, &normb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_gen_complx_mat_print_comp (x04dbc): Print matrices A and B. */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                n, a, pda, Nag_BracketForm, "%6.2f",
                                "Matrix A", Nag_IntegerLabels, 0,
                                Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  printf("\n");
  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;
    }
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                n, b, pdb, Nag_BracketForm, "%6.2f",
                                "Matrix B", Nag_IntegerLabels, 0,
                                Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  printf("\n");
  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;
    }

  /* Find the generalized Schur form using nag_zggesx (f08xpc). */
  nag_zggesx(order, jobvsl, jobvsr, sort, selctg, sense, n, a, pda, b, pdb,
             &sdim, alpha, beta, vsl, pdvsl, vsr, pdvsr, rconde, rcondv, &fail);

  if (fail.code != NE_NOERROR && fail.code != NE_SCHUR_REORDER_SELECT)
    {
      printf("Error from nag_zggesx (f08xpc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Check generalized Schur Form by reconstruction of Schur vectors are
   * available.
   */
  if (jobvsl==Nag_NotLeftVecs || jobvsr==Nag_NotRightVecs)
    {
      /* Cannot check factorization by reconstruction Schur vectors. */
      goto END;
    }

  /* Reconstruct A as Q*S*Z^H and subtract from original (D) using the steps
   * C = Q (Q in vsl) using nag_zge_copy (f16tfc).
   * C = C*S (S in a, upper triangular) using nag_ztrmm (f16zfc).
   * D = D - C*Z^H (Z in vsr) using nag_zgemm (f16zac).
   */
  nag_zge_copy(order, Nag_NoTrans, n, n, vsl, pdvsl, c, pdc, &fail);
  alph = nag_complex(1.0,0.0);
  /* nag_ztrmm (f16zfc)  Triangular complex matrix-matrix multiply. */
  nag_ztrmm(order, Nag_RightSide, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n,
            n, alph, a, pda, c, pdc, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztrmm (f16zfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  alph = nag_complex(-1.0,0.0);
  bet = nag_complex(1.0,0.0);
  nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alph, c, pdc, vsr,
            pdvsr, bet, d, pdd, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Reconstruct B as Q*T*Z^H and subtract from original (E) using the steps
   * Q = Q*T (Q in vsl, T in b, upper triangular) using nag_ztrmm (f16zfc).
   * E = E - Q*Z^H (Z in vsr) using nag_zgemm (f16zac).
   */
  alph = nag_complex(1.0,0.0);
  /* nag_ztrmm (f16zfc)  Triangular complex matrix-matrix multiply. */
  nag_ztrmm(order, Nag_RightSide, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n,
            n, alph, b, pdb, vsl, pdvsl, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztrmm (f16zfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  alph = nag_complex(-1.0,0.0);
  bet = nag_complex(1.0,0.0);
  nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alph, vsl, pdvsl, vsr,
            pdvsr, bet, e, pde, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_zge_norm (f16uac): Find norms of difference matrices D and E. */
  nag_zge_norm(order, Nag_FrobeniusNorm, n, n, d, pdd, &normd, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  nag_zge_norm(order, Nag_FrobeniusNorm, n, n, e, pde, &norme, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Get the machine precision, using nag_machine_precision (x02ajc) */
  eps = nag_machine_precision;
  if (MAX(normd,norme) > pow(eps,0.8)*MAX(norma,normb))
    {
      printf("The norm of the error in the reconstructed matrices is greater "
             "than expected.\nThe Schur factorization has failed.\n");
      exit_status = 1;
      goto END;
    }

  /* Print details on eigenvalues */
  printf("Number of sorted eigenvalues = %4"NAG_IFMT"\n\n", sdim);
  if (fail.code == NE_SCHUR_REORDER_SELECT) {
    printf("*** Note that rounding errors mean that leading eigenvalues in the"
           " generalized\n    Schur form no longer satisfy selctg = Nag_TRUE"
           "\n\n");
  } else {
    printf("The selected eigenvalues are:\n");
    for (i=0;i<sdim;i++) {
      if (beta[i].re != 0.0 || beta[i].im != 0.0) {
        z = nag_complex_divide(alpha[i], beta[i]);
        printf("%3"NAG_IFMT" (%13.4e, %13.4e)\n", i+1, z.re, z.im);
      }
      else
        printf("%3"NAG_IFMT" Eigenvalue is infinite\n", i + 1);
    }
  }

  abnorm = sqrt(pow(norma, 2) + pow(normb, 2));
  tol = eps*abnorm;

  if (sense==Nag_RCondEigVals || sense==Nag_RCondBoth) {
    /* Print out the reciprocal condition number and error bound */
    printf("\n");
    printf("For the selected eigenvalues,\nthe reciprocals of projection "
           "norms onto the deflating subspaces are\n");
    printf(" for left  subspace, rcond = %10.1e\n for right subspace, rcond = "
           "%10.1e\n\n", rconde[0], rconde[1]);
    printf(" asymptotic error bound    = %10.1e\n\n", tol / rconde[0]);
  }
  if (sense==Nag_RCondEigVecs || sense==Nag_RCondBoth) {
    /* Print out the reciprocal condition numbers and error bound. */
     printf("For the left and right deflating subspaces,\n");
    printf("reciprocal condition numbers are:\n");
    printf(" for left  subspace, rcond = %10.1e\n for right subspace, rcond = "
           "%10.1e\n\n", rcondv[0], rcondv[1]);
    printf(" approximate error bound   = %10.1e\n", tol / rcondv[1]);
  }

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(c);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(alpha);
  NAG_FREE(beta);
  NAG_FREE(vsl);
  NAG_FREE(vsr);

  return exit_status;
}

static Nag_Boolean NAG_CALL selctg(const Complex a, const Complex b)
{
  /* Boolean function selctg for use with nag_zggesx (f08xpc)
   * Returns the value Nag_TRUE if the absolute value of the eigenvalue
   * a/b < 6.0
   */

  return (nag_complex_abs(a) < 6.0*nag_complex_abs(b) ? Nag_TRUE : Nag_FALSE);
}
```

10.2 Program Data

```
nag_zggesx (f08xpc) Example Program Data

     4                                                          : n

   Nag_LeftVecs                                                 : jobvsl
   Nag_RightVecs                                                : jobvsr
   Nag_RCondBoth                                                : sense

  (-21.10,-22.50) ( 53.50,-50.50) (-34.50,127.50) (  7.50,  0.50)
  ( -0.46, -7.78) ( -3.50,-37.50) (-15.50, 58.50) (-10.50, -1.50)
  (  4.30, -5.50) ( 39.70,-17.10) (-68.50, 12.50) ( -7.50, -3.50)
  (  5.50,  4.40) ( 14.40, 43.30) (-32.50,-46.00) (-19.00,-32.50) : A

  (  1.00, -5.00) (  1.60,  1.20) ( -3.00,  0.00) (  0.00, -1.00)
  (  0.80, -0.60) (  3.00, -5.00) ( -4.00,  3.00) ( -2.40, -3.20)
  (  1.00,  0.00) (  2.40,  1.80) ( -4.00, -5.00) (  0.00, -3.00)
  (  0.00,  1.00) ( -1.80,  2.40) (  0.00, -4.00) (  4.00, -5.00) : B
```

10.3 Program Results

```
nag_zggesx (f08xpc) Example Program Results

 Matrix A
                1                 2                 3                 4
 1  (-21.10,-22.50)  ( 53.50,-50.50)  (-34.50,127.50)  (  7.50,  0.50)
 2  ( -0.46, -7.78)  ( -3.50,-37.50)  (-15.50, 58.50)  (-10.50, -1.50)
 3  (  4.30, -5.50)  ( 39.70,-17.10)  (-68.50, 12.50)  ( -7.50, -3.50)
 4  (  5.50,  4.40)  ( 14.40, 43.30)  (-32.50,-46.00)  (-19.00,-32.50)

 Matrix B
                1                 2                 3                 4
 1  (  1.00, -5.00)  (  1.60,  1.20)  ( -3.00,  0.00)  (  0.00, -1.00)
 2  (  0.80, -0.60)  (  3.00, -5.00)  ( -4.00,  3.00)  ( -2.40, -3.20)
 3  (  1.00,  0.00)  (  2.40,  1.80)  ( -4.00, -5.00)  (  0.00, -3.00)
 4  (  0.00,  1.00)  ( -1.80,  2.40)  (  0.00, -4.00)  (  4.00, -5.00)

Number of sorted eigenvalues =    2

The selected eigenvalues are:
  1 (   2.0000e+00,   -5.0000e+00)
  2 (   3.0000e+00,   -1.0000e+00)

For the selected eigenvalues,
the reciprocals of projection norms onto the deflating subspaces are
 for left  subspace, rcond =    1.2e-01
 for right subspace, rcond =    1.6e-01

 asymptotic error bound    =    1.9e-13

For the left and right deflating subspaces,
reciprocal condition numbers are:
 for left  subspace, rcond =    4.8e-01
 for right subspace, rcond =    4.7e-01

 approximate error bound   =    4.9e-14
```