NAG Library Function Document

nag_zgeesx (f08ppc)

1 Purpose

nag_zgeesx (f08ppc) computes the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z for an n by n complex nonsymmetric matrix A.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zgeesx (Nag_OrderType order, Nag_JobType jobvs,
     Nag_SortEigValsType sort,
     Nag_Boolean (*select)(Complex w),
     Nag_RCondType sense, Integer n, Complex a[], Integer pda, Integer *sdim,
     Complex w[], Complex vs[], Integer pdvs, double *rconde, double *rcondv,
     NagError *fail)
```

3 Description

The Schur factorization of A is given by

$$A = ZTZ^{\mathrm{H}},$$

where Z, the matrix of Schur vectors, is unitary and T is the Schur form. A complex matrix is in Schur form if it is upper triangular.

Optionally, nag_zgeesx (f08ppc) also orders the eigenvalues on the diagonal of the Schur form so that selected eigenvalues are at the top left; computes a reciprocal condition number for the average of the selected eigenvalues (**rconde**); and computes a reciprocal condition number for the right invariant subspace corresponding to the selected eigenvalues (**rcondv**). The leading columns of Z form an orthonormal basis for this invariant subspace.

For further explanation of the reciprocal condition numbers **rconde** and **rcondv**, see Section 4.8 of Anderson *et al.* (1999) (where these quantities are called s and sep respectively).

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: **jobvs** – Nag_JobType *Input*

On entry: if **jobvs** = Nag_DoNothing, Schur vectors are not computed.

If **jobvs** = Nag_Schur, Schur vectors are computed.

Constraint: **jobvs** = Nag_DoNothing or Nag_Schur.

3: **sort** – Nag_SortEigValsType *Input*

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

sort = Nag_NoSortEigVals
Eigenvalues are not ordered.

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

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

4: **select** – function, supplied by the user *External Function*

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

If **sort** = Nag_NoSortEigVals, **select** is not referenced and nag_zgeesx (f08ppc) may be specified as NULLFN.

An eigenvalue $\mathbf{w}[j-1]$ is selected if $\mathbf{select}(\mathbf{w}[j-1])$ is Nag_TRUE.

The specification of **select** is:

```
Nag_Boolean select (Complex w)
```

1: **w** – Complex *Input*

On entry: the real and imaginary parts of the eigenvalue.

5: **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 right invariant subspace 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.

6: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

7: $\mathbf{a}[dim]$ – 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

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

On entry: the n by n matrix A.

On exit: **a** is overwritten by its Schur form T.

8: **pda** – Integer *Input*

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

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

9: **sdim** – Integer * *Output*

On exit: if **sort** = Nag_NoSortEigVals, **sdim** $= 0$.

If **sort** = Nag_SortEigVals, **sdim** = number of eigenvalues for which **select** is Nag_TRUE.

10: **w**[*dim*] – Complex *Output*

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

On exit: contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

11: **vs**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvs} \times \mathbf{n})$ when **jobvs** = Nag_Schur;
1 otherwise.

The ith element of the jth vector is stored in

vs$[(j-1) \times$ **pdvs** $+ i - 1]$ when **order** = Nag_ColMajor;
vs$[(i-1) \times$ **pdvs** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: if **jobvs** = Nag_Schur, **vs** contains the unitary matrix Z of Schur vectors.

If **jobvs** = Nag_DoNothing, **vs** is not referenced.

12: **pdvs** – Integer *Input*

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

Constraints:

if **jobvs** = Nag_Schur, **pdvs** $\geq \max(1, \mathbf{n})$;
otherwise **pdvs** ≥ 1.

13: **rconde** – double * *Output*

On exit: if **sense** = Nag_RCondEigVals or Nag_RCondBoth, contains the reciprocal condition number for the average of the selected eigenvalues.

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

14: **rcondv** – double * *Output*

On exit: if **sense** = Nag_RCondEigVecs or Nag_RCondBoth, **rcondv** contains the reciprocal condition number for the selected right invariant subspace.

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

15: **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 ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

The QR algorithm failed to compute all the eigenvalues.

NE_ENUM_INT_2

On entry, **jobvs** $=$ ⟨*value*⟩, **pdvs** $=$ ⟨*value*⟩ and **n** $=$ ⟨*value*⟩.
Constraint: if **jobvs** $=$ Nag_Schur, **pdvs** $\geq \max(1, \mathbf{n})$;
otherwise **pdvs** ≥ 1.

NE_INT

On entry, **n** $=$ ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **pda** $=$ ⟨*value*⟩.
Constraint: **pda** > 0.

On entry, **pdvs** $=$ ⟨*value*⟩.
Constraint: **pdvs** > 0.

NE_INT_2

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

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.

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 Schur form no longer satisfy **select** $=$ Nag_TRUE. This could also be caused by underflow due to scaling.

7 Accuracy

The computed Schur factorization satisfies

$$A + E = ZTZ^{\mathrm{H}},$$

where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ is the ***machine precision***. See Section 4.8 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_zgeesx (f08ppc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgeesx (f08ppc) 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_dgeesx (f08pbc).

10 Example

This example finds the Schur factorization of the matrix

$$A = \begin{pmatrix} -3.97 - 5.04i & -4.11 + 3.70i & -0.34 + 1.01i & 1.29 - 0.86i \\ 0.34 - 1.50i & 1.52 - 0.43i & 1.88 - 5.38i & 3.36 + 0.65i \\ 3.31 - 3.85i & 2.50 + 3.45i & 0.88 - 1.08i & 0.64 - 1.48i \\ -1.10 + 0.82i & 1.81 - 1.59i & 3.25 + 1.33i & 1.57 - 3.44i \end{pmatrix},$$

such that the eigenvalues of A with positive real part of are the top left diagonal elements of the Schur form, T. Estimates of the condition numbers for the selected eigenvalue cluster and corresponding invariant subspace are also returned.

10.1 Program Text

```
/* nag_zgeesx (f08ppc) 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 select_fun(const Complex w);
#ifdef __cplusplus
}
#endif

int main(void)
{

  /* Scalars */
  Complex       alpha, beta;
  double        anorm, eps, norm, rconde, rcondv;
  Integer       i, j, n, pda, pdc, pdd, pdvs, sdim;
  Integer       exit_status = 0;

  /* Arrays */
  Complex       *a = 0, *c = 0, *d = 0, *vs = 0, *w = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

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

  INIT_FAIL(fail);

  printf("nag_zgeesx (f08ppc) 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;
    }

  pda = n;
  pdc = n;
  pdd = n;
  pdvs = n;
  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, Complex)) ||
      !(c = NAG_ALLOC(n * n, Complex)) ||
      !(d = NAG_ALLOC(n * n, Complex)) ||
      !(vs = NAG_ALLOC(n * n, Complex)) ||
      !(w = NAG_ALLOC(n, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read in the matrix A */
  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

  /* Copy A to D: nag_zge_copy (f16tfc),
   * Complex valued general matrix copy.
   */
  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_norm (f16uac): Find norm of matrix A for use later
   * in relative error test.
   */
  nag_zge_norm(order, Nag_OneNorm, n, n, a, pda, &anorm, &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 matrix A */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                n, a, pda, Nag_BracketForm, "%7.4f",
                                "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;
    }

  /* Find the Schur factorization of A using nag_zgeesx (f08ppc). */
  nag_zgeesx(order, Nag_Schur, Nag_SortEigVals, select_fun, Nag_RCondBoth,
             n, a, pda, &sdim, w, vs, pdvs, &rconde, &rcondv, &fail);

  if (fail.code != NE_NOERROR && fail.code != NE_SCHUR_REORDER_SELECT)
    {
      printf("Error from nag_zgeesx (f08ppc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Reconstruct A from Schur Factorization Z*T*ConjTrans(Z) where T is upper
   * triangular and stored in A. This can be done using the following steps:
   * i.  C = Z*T (nag_zgemm, f16zac),
   * ii. D = D-C*ConjTrans(Z) (nag_zgemm, f16zac).
   */
  alpha = nag_complex(1.0,0.0);
  beta = nag_complex(0.0,0.0);
  nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, alpha, vs, pdvs, a, pda,
            beta, c, pdc, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);
      exit_status = 1;
```

```
      goto END;
    }

  /* nag_zgemm (f16zac):
   * Compute D = A - C*Z^H.
   */
  alpha = nag_complex(-1.0,0.0);
  beta = nag_complex(1.0,0.0);
  nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alpha, c, pdc,
            vs, pdvs, beta, d, pdd, &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 norm of difference matrix D and print
   * warning if it is too large relative to norm of A.
   */
  nag_zge_norm(order, Nag_OneNorm, n, n, d, pdd, &norm, &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 (norm > pow(eps,0.8)*MAX(anorm,1.0))
    {
      printf("||A-(Z*T*Z^H)||/||A|| is larger than expected.\n"
             "Schur factorization has failed.\n");
      exit_status = 1;
      goto END;
    }

  /* Print details on eigenvalues */
  printf("Number of eigenvalues for which select is true = %4"NAG_IFMT"\n\n",
         sdim);
  if (fail.code == NE_SCHUR_REORDER_SELECT) {
    printf(" ** Note that rounding errors mean that leading eigenvalues in the"
           " Schur form\n    no longer satisfy select(lambda) = Nag_TRUE\n\n");
  } else {
    printf("The selected eigenvalues are:\n");
    for (i=0;i<sdim;i++)
      printf("%3"NAG_IFMT" (%13.4e, %13.4e)\n", i+1, w[i].re, w[i].im);
  }

  /* Print out the reciprocal condition numbers */
  printf("\nReciprocal of projection norm onto the invariant subspace\n");
  printf("%26sfor the selected eigenvalues rconde = %8.1e\n\n", "", rconde);
  printf("Reciprocal condition number for the invariant subspace rcondv = "
         "%8.1e\n\n", rcondv);

  /* Compute the approximate asymptotic error bound on the average absolute
   * error of the selected eigenvalues given by  eps*norm(A)/rconde.
   */
  printf("Approximate asymptotic error bound for selected eigenvalues   = "
         "%8.1e\n\n", eps * anorm / rconde);

  /* Compute an approximate asymptotic bound on the maximum angular error in
   * the computed invariant subspace given by  eps*norm(A)/rcondv
   */
  printf("Approximate asymptotic error bound for the invariant subspace = "
         "%8.1e\n", eps * anorm / rcondv);

 END:
  NAG_FREE(a);
  NAG_FREE(c);
```

```
  NAG_FREE(d);
  NAG_FREE(vs);
  NAG_FREE(w);

  return exit_status;
}

static Nag_Boolean NAG_CALL select_fun(const Complex w)
{
  /* Boolean function select for use with nag_zgeesx (f08ppc)
   * Returns the value Nag_TRUE if the real part of the eigenvalue w
   * is positive.
   */

  return (w.re>0.0 ? Nag_TRUE : Nag_FALSE);
}
```

10.2 Program Data

```
nag_zgeesx (f08ppc) Example Program Data

  4                                                            : n

(-3.97, -5.04)  (-4.11,  3.70)  (-0.34,  1.01)  ( 1.29, -0.86)
( 0.34, -1.50)  ( 1.52, -0.43)  ( 1.88, -5.38)  ( 3.36,  0.65)
( 3.31, -3.85)  ( 2.50,  3.45)  ( 0.88, -1.08)  ( 0.64, -1.48)
(-1.10,  0.82)  ( 1.81, -1.59)  ( 3.25,  1.33)  ( 1.57, -3.44) : matrix A
```

10.3 Program Results

```
nag_zgeesx (f08ppc) Example Program Results

 Matrix A
                    1                   2                   3                   4
 1  (-3.9700,-5.0400)  (-4.1100, 3.7000)  (-0.3400, 1.0100)  ( 1.2900,-0.8600)
 2  ( 0.3400,-1.5000)  ( 1.5200,-0.4300)  ( 1.8800,-5.3800)  ( 3.3600, 0.6500)
 3  ( 3.3100,-3.8500)  ( 2.5000, 3.4500)  ( 0.8800,-1.0800)  ( 0.6400,-1.4800)
 4  (-1.1000, 0.8200)  ( 1.8100,-1.5900)  ( 3.2500, 1.3300)  ( 1.5700,-3.4400)

Number of eigenvalues for which select is true =    2

The selected eigenvalues are:
  1 (   7.9982e+00,   -9.9637e-01)
  2 (   3.0023e+00,   -3.9998e+00)

Reciprocal of projection norm onto the invariant subspace
                          for the selected eigenvalues rconde =  9.9e-01

Reciprocal condition number for the invariant subspace rcondv =  8.4e+00

Approximate asymptotic error bound for selected eigenvalues   =  1.6e-15

Approximate asymptotic error bound for the invariant subspace =  1.9e-16
```