NAG Library Function Document

nag_ztgexc (f08ytc)

1 Purpose

nag_ztgexc (f08ytc) reorders the generalized Schur factorization of a complex matrix pair in generalized Schur form.

2 Specification

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

```
void nag_ztgexc (Nag_OrderType order, Nag_Boolean wantq, Nag_Boolean wantz,
     Integer n, Complex a[], Integer pda, Complex b[], Integer pdb,
     Complex q[], Integer pdq, Complex z[], Integer pdz, Integer ifst,
     Integer *ilst, NagError *fail)
```

3 Description

nag_ztgexc (f08ytc) reorders the generalized complex n by n matrix pair (S, T) in generalized Schur form, so that the diagonal element of (S, T) with row index i_1 is moved to row i_2, using a unitary equivalence transformation. That is, S and T are factorized as

$$S = \hat{Q}\hat{S}\hat{Z}^{\mathrm{H}}, \quad T = \hat{Q}\hat{T}\hat{Z}^{\mathrm{H}},$$

where $\left(\hat{S}, \hat{T}\right)$ are also in generalized Schur form.

The pair (S, T) are in generalized Schur form if S and T are upper triangular as returned, for example, by nag_zgges (f08xnc), or nag_zhgeqz (f08xsc) with **job** $=$ Nag_Schur.

If S and T are the result of a generalized Schur factorization of a matrix pair (A, B)

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

then, optionally, the matrices Q and Z can be updated as $Q\hat{Q}$ and $Z\hat{Z}$.

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

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: **wantq** – Nag_Boolean *Input*

On entry: if **wantq** $=$ Nag_TRUE, update the left transformation matrix Q.

If **wantq** $=$ Nag_FALSE, do not update Q.

3: **wantz** – Nag_Boolean *Input*

On entry: if **wantz** $=$ Nag_TRUE, update the right transformation matrix Z.

If **wantz** $=$ Nag_FALSE, do not update Z.

4: **n** – Integer *Input*

On entry: n, the order of the matrices S and T.

Constraint: **n** ≥ 0.

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

On entry: the matrix S in the pair (S, T).

On exit: the updated matrix $\hat{S}$.

6: **pda** – Integer *Input*

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

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

7: **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

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

On entry: the matrix T, in the pair (S, T).

On exit: the updated matrix $\hat{T}$

8: **pdb** – Integer *Input*

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

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

9: **q**[*dim*] – Complex *Input/Output*

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

$\max(1, \mathbf{pdq} \times \mathbf{n})$ when **wantq** $=$ Nag_TRUE;
1 otherwise.

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

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

On entry: if **wantq** $=$ Nag_TRUE, the unitary matrix Q.

On exit: if **wantq** $=$ Nag_TRUE, the updated matrix $Q\hat{Q}$.

If **wantq** $=$ Nag_FALSE, **q** is not referenced.

10: **pdq** – Integer *Input*

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

Constraints:

if **wantq** = Nag_TRUE, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 1.

11: **z**$[dim]$ – Complex *Input/Output*

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

$\max(1, \mathbf{pdz} \times \mathbf{n})$ when **wantz** = Nag_TRUE;
1 otherwise.

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

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

On entry: if **wantz** = Nag_TRUE, the unitary matrix Z.

On exit: if **wantz** = Nag_TRUE, the updated matrix $Z\hat{Z}$.

If **wantz** = Nag_FALSE, **z** is not referenced.

12: **pdz** – Integer *Input*

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

Constraints:

if **wantz** = Nag_TRUE, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

13: **ifst** – Integer *Input*
14: **ilst** – Integer * *Input/Output*

On entry: the indices i_1 and i_2 that specify the reordering of the diagonal elements of (S, T). The element with row index **ifst** is moved to row **ilst**, by a sequence of swapping between adjacent diagonal elements.

On exit: **ilst** points to the row in its final position.

Constraint: $1 \leq \mathbf{ifst} \leq \mathbf{n}$ and $1 \leq \mathbf{ilst} \leq \mathbf{n}$.

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.

NE_BAD_PARAM

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

NE_CONSTRAINT

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

On entry, **wantz** $= \langle value \rangle$, **pdz** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **wantz** $=$ Nag_TRUE, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 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, **pdq** $= \langle value \rangle$.
Constraint: **pdq** > 0.

On entry, **pdz** $= \langle value \rangle$.
Constraint: **pdz** > 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_INT_3

On entry, **ifst** $= \langle value \rangle$, **ilst** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: $1 \leq$ **ifst** $\leq$ **n** and $1 \leq$ **ilst** $\leq$ **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.

NE_SCHUR

The transformed matrix pair would be too far from generalized Schur form; the problem is ill-conditioned. (S, T) may have been partially reordered, and **ilst** points to the first row of the current position of the block being moved.

7 Accuracy

The computed generalized Schur form is nearly the exact generalized Schur form for nearby matrices $(S + E)$ and $(T + F)$, where

$$\|E\|_2 = O\,\epsilon\|S\|_2 \quad \text{and} \quad \|F\|_2 = O\,\epsilon\|T\|_2,$$

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

8 Parallelism and Performance

Not applicable.

9 Further Comments

The real analogue of this function is nag_dtgexc (f08yfc).

10 Example

This example exchanges rows 4 and 1 of the matrix pair (S, T), where

$$S = \begin{pmatrix} 4.0+4.0i & 1.0+1.0i & 1.0+1.0i & 2.0-1.0i \\ 0 & 2.0+1.0i & 1.0+1.0i & 1.0+1.0i \\ 0 & 0 & 2.0-1.0i & 1.0+1.0i \\ 0 & 0 & 0 & 6.0-2.0i \end{pmatrix}$$

and

$$T = \begin{pmatrix} 2.0 & 1.0+1.0i & 1.0+1.0i & 3.0-1.0i \\ 0 & 1.0 & 2.0+1.0i & 1.0+1.0i \\ 0 & 0 & 1.0 & 1.0+1.0i \\ 0 & 0 & 0 & 2.0 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_ztgexc (f08ytc) Example Program.
 *
 * Copyright 2011 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf08.h>

int main(void)
{
  /* Scalars */
  Integer            i, ifst, ilst, j, n, pdq, pds, pdt, pdz;
  Integer            exit_status = 0;

  /* Arrays */
  Complex            *q = 0, *s = 0, *t = 0, *z = 0;
  char               nag_enum_arg[40];

  NagError           fail;
  Nag_OrderType      order;
  Nag_Boolean        wantq, wantz;
  Nag_MatrixType     upmat = Nag_UpperMatrix;
  Nag_DiagType       diag   = Nag_NonUnitDiag;
  Nag_LabelType      intlab = Nag_IntegerLabels;
  Nag_ComplexFormType brac   = Nag_BracketForm;

#ifdef NAG_COLUMN_MAJOR
#define S(I, J) s[(J-1)*pds + I - 1]
#define T(I, J) t[(J-1)*pdt + I - 1]
  order = Nag_ColMajor;
#else
#define S(I, J) s[(I-1)*pds + J - 1]
#define T(I, J) t[(I-1)*pdt + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_ztgexc (f08ytc) Example Program Results\n\n");

  /* Skip heading in data file */
```

```
  scanf("%*[^\n]");
  scanf("%ld%*[^\n]", &n);
  if (n < 0)
    {
      printf("Invalid n\n");
      exit_status = 1;
      goto END;
    }
  scanf(" %39s%*[^\n]", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  wantq = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
  scanf(" %39s%*[^\n]", nag_enum_arg);
  wantz = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);

  pds = n;
  pdt = n;
  pdq = (wantq?n:1);
  pdz = (wantz?n:1);

  /* Allocate memory */
  if (
      !(s = NAG_ALLOC(n*n, Complex)) ||
      !(t = NAG_ALLOC(n*n, Complex)) ||
      !(q = NAG_ALLOC(pdq*pdq, Complex)) ||
      !(z = NAG_ALLOC(pdz*pdz, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read S and T from data file */
  for (i = 1; i <= n; ++i)
      for (j = 1; j <= n; ++j)
        scanf(" ( %lf , %lf )", &S(i, j).re, &S(i, j).im);
  scanf("%*[^\n]");
  for (i = 1; i <= n; ++i)
      for (j = 1; j <= n; ++j)
        scanf(" ( %lf , %lf )", &T(i, j).re, &T(i, j).im);
  scanf("%*[^\n]");

  /* Read the row indices */
  scanf("%ld%ld%*[^\n]", &ifst, &ilst);

  /* Reorder the S and T */
  nag_ztgexc(order, wantq, wantz, n, s, pds, t, pdt, q, pdq, z, pdz, ifst,
             &ilst, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztgexc (f08ytc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print reordered generalized Schur form */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, upmat, diag, n, n, s, pds, brac,
                                "%7.4f", "Reordered Schur matrix S",
                                intlab, NULL, intlab, NULL, 80, 0, NULL,
                                &fail);
  if (fail.code != NE_NOERROR) goto PRERR;
  printf("\n");
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, upmat, diag, n, n, t, pdt, brac,
                                "%7.4f", "Reordered Schur matrix T",
                                intlab, NULL, intlab, NULL, 80, 0, NULL,
                                &fail);
 PRERR:
  if (fail.code != NE_NOERROR)
```

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

 END:
  NAG_FREE(q);
  NAG_FREE(s);
  NAG_FREE(t);
  NAG_FREE(z);

  return exit_status;
}
```

10.2 Program Data

```
nag_ztgexc (f08ytc) Example Program Data

  4                                                  : n

  Nag_FALSE                                          : wantq
  Nag_FALSE                                          : wantz

 ( 4.0, 4.0) ( 1.0, 1.0) ( 1.0, 1.0) ( 2.0,-1.0)
 ( 0.0, 0.0) ( 2.0, 1.0) ( 1.0, 1.0) ( 1.0, 1.0)
 ( 0.0, 0.0) ( 0.0, 0.0) ( 2.0,-1.0) ( 1.0, 1.0)
 ( 0.0, 0.0) ( 0.0, 0.0) ( 0.0, 0.0) ( 6.0,-2.0)   : matrix S

 ( 2.0, 0.0) ( 1.0, 1.0) ( 1.0, 1.0) ( 3.0,-1.0)
 ( 0.0, 0.0) ( 1.0, 0.0) ( 2.0, 1.0) ( 1.0, 1.0)
 ( 0.0, 0.0) ( 0.0, 0.0) ( 1.0, 0.0) ( 1.0, 1.0)
 ( 0.0, 0.0) ( 0.0, 0.0) ( 0.0, 0.0) ( 2.0, 0.0)   : matrix T

  1          4                                       : ifst and ilst
```

10.3 Program Results

```
nag_ztgexc (f08ytc) Example Program Results

 Reordered Schur matrix S
                    1                   2                   3                   4
 1  ( 3.7081, 3.7081)  (-2.0834,-0.5688)  ( 2.6374, 1.0772)  ( 0.2845, 0.7991)
 2                     ( 1.6097, 1.5656)  (-0.0634, 1.9234)  (-0.0301, 0.9720)
 3                                        ( 4.7029,-2.1187)  ( 1.1379,-3.1199)
 4                                                           ( 2.3085,-1.8289)

 Reordered Schur matrix T
                    1                   2                   3                   4
 1  ( 2.2249, 0.7416)  (-1.1631, 1.5347)  ( 2.2608, 2.0851)  ( 1.1094,-0.3205)
 2                     ( 0.3308, 0.9482)  ( 0.3919, 1.8172)  (-0.6305, 1.6053)
 3                                        ( 1.6227,-0.1653)  ( 0.9966,-0.9074)
 4                                                           ( 0.1199,-1.0343)
```