NAG C Library Function Document

nag_dtrexc (f08qfc)

1 Purpose

nag_dtrexc (f08qfc) reorders the Schur factorization of a real general matrix.

2 Specification

```
void nag_dtrexc (Nag_OrderType order, Nag_ComputeQType compq, Integer n,
     double t[], Integer pdt, double q[], Integer pdq, Integer *ifst, Integer *ilst,
     NagError *fail)
```

3 Description

nag_dtrexc (f08qfc) reorders the Schur factorization of a real general matrix $A = QTQ^T$, so that the diagonal element or block of T with row index **ifst** is moved to row **ilst**.

The reordered Schur form $\tilde{T}$ is computed by an orthogonal similarity transformation: $\tilde{T} = Z^T TZ$. Optionally the updated matrix $\tilde{Q}$ of Schur vectors is computed as $\tilde{Q} = QZ$, giving $A = \tilde{Q}\tilde{T}\tilde{Q}^T$.

4 References

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

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **compq** – Nag_ComputeQType *Input*

On entry: indicates whether the matrix Q of Schur vectors is to be updated, as follows:

if **compq** = **Nag_UpdateSchur**, the matrix Q of Schur vectors is updated;

if **compq** = **Nag_NotQ**, no Schur vectors are updated.

Constraint: **compq** = **Nag_UpdateSchur** or **Nag_NotQ**.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **t**[dim] – double *Input/Output*

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

If **order** = **Nag_ColMajor**, the (i,j)th element of the matrix T is stored in $\mathbf{t}[(j-1) \times \mathbf{pdt} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i,j)th element of the matrix T is stored in $\mathbf{t}[(i-1) \times \mathbf{pdt} + j - 1]$.

On entry: the n by n upper quasi-triangular matrix T in canonical Schur form, as returned by nag_dhseqr (f08pec).

On exit: T is overwritten by the updated matrix $\tilde{T}$. See also Section 8.

5: **pdt** – Integer *Input*

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

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

6: **q**[dim] – double *Input/Output*

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

$\max(1, \mathbf{pdq} \times \mathbf{n})$ when **compq** $=$ **Nag_UpdateSchur**;

1 when **compq** $=$ **Nag_NotQ**.

If **order** $=$ **Nag_ColMajor**, the (i,j)th element of the matrix Q is stored in **q**$[(j-1) \times \mathbf{pdq} + i - 1]$ and if **order** $=$ **Nag_RowMajor**, the (i,j)th element of the matrix Q is stored in **q**$[(i-1) \times \mathbf{pdq} + j - 1]$.

On entry: if **compq** $=$ **Nag_UpdateSchur**, **q** must contain the n by n orthogonal matrix Q of Schur vectors.

On exit: if **compq** $=$ **Nag_UpdateSchur**, **q** contains the updated matrix of Schur vectors.

q is not referenced if **compq** $=$ **Nag_NotQ**.

7: **pdq** – Integer *Input*

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

Constraints:

if **compq** $=$ **Nag_UpdateSchur**, **pdq** $\geq \max(1, \mathbf{n})$;
if **compq** $=$ **Nag_NotQ**, **pdq** ≥ 1.

8: **ifst** – Integer * *Input/Output*
9: **ilst** – Integer * *Input/Output*

On entry: **ifst** and **ilst** must specify the reordering of the diagonal elements or blocks of T. The element or block with row index **ifst** is moved to row **ilst** by a sequence of exchanges between adjacent elements or blocks.

On exit: if **ifst** pointed to the second row of a 2 by 2 block on entry, it is changed to point to the first row. **ilst** always points to the first row of the block in its final position (which may differ from its input value by ± 1).

Constraint: $1 \leq$ **ifst** $\leq$ **n** and $1 \leq$ **ilst** $\leq$ **n**.

10: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = $\langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **pdt** $= \langle value \rangle$.
Constraint: **pdt** > 0.

On entry, **pdq** $= \langle value \rangle$.
Constraint: **pdq** > 0.

NE_INT_2

On entry, **pdt** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdt** $\geq \max(1, \mathbf{n})$.

NE_INT_3

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

NE_ENUM_INT_2

On entry, **compq** $= \langle value \rangle$, **n** $= \langle value \rangle$, **pdq** $= \langle value \rangle$.
Constraint: if **compq** = **Nag_UpdateSchur**, **pdq** $\geq \max(1, \mathbf{n})$;
if **compq** = **Nag_NotQ**, **pdq** ≥ 1.

NE_EXCHANGE

Two adjacent diagonal elements or blocks could not be successfully exchanged. This error can only occur if the exchange involves at least one 2 by 2 block; it implies that the problem is very ill-conditioned, and that the eigenvalues of the two blocks are very close. On exit, T may have been partially reordered, and **ilst** points to the first row of the current position of the block being moved; Q (if requested) is updated consistently with T.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 consult NAG for assistance.

7 Accuracy

The computed matrix $\tilde{T}$ is exactly similar to a matrix $T + E$, where

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

and ϵ is the ***machine precision***.

Note that if a 2 by 2 diagonal block is involved in the re-ordering, its off-diagonal elements are in general changed; the diagonal elements and the eigenvalues of the block are unchanged unless the block is sufficiently ill-conditioned, in which case they may be noticeably altered. It is possible for a 2 by 2 block to break into two 1 by 1 blocks, that is, for a pair of complex eigenvalues to become purely real. The values of real eigenvalues however are never changed by the re-ordering.

8 Further Comments

The total number of floating-point operations is approximately $6nr$ if **compq** = **Nag_NotQ**, and $12nr$ if **compq** = **Nag_UpdateSchur**, where $r = |\mathbf{ifst} - \mathbf{ilst}|$.

The input matrix T must be in canonical Schur form, as is the output matrix $\tilde{T}$. This has the following structure.

If all the computed eigenvalues are real, T is upper triangular and its diagonal elements are the eigenvalues.

If some of the computed eigenvalues form complex conjugate pairs, then T has 2 by 2 diagonal blocks. Each diagonal block has the form

$$\begin{pmatrix} t_{ii} & t_{i,i+1} \\ t_{i+1,i} & t_{i+1,i+1} \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \gamma & \alpha \end{pmatrix}$$

where $\beta\gamma < 0$. The corresponding eigenvalues are $\alpha \pm \sqrt{\beta\gamma}$.

The complex analogue of this function is nag_ztrexc (f08qtc).

9 Example

To reorder the Schur factorization of the matrix T so that the 2 by 2 block with row index 2 is moved to row 1, where

$$T = \begin{pmatrix} 0.80 & -0.11 & 0.01 & 0.03 \\ 0.00 & -0.10 & 0.25 & 0.35 \\ 0.00 & -0.65 & -0.10 & 0.20 \\ 0.00 & 0.00 & 0.00 & -0.10 \end{pmatrix}.$$

9.1 Program Text

```
/* nag_dtrexc (f08qfc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer  i, ifst, ilst, j, n, pdq, pdt;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double *q=0, *t=0;

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

  INIT_FAIL(fail);
  Vprintf("f08qfc Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
  pdq = 1;
  pdt = n;
#else
  pdq = 1;
  pdt = n;
#endif

  /* Allocate memory */
```

```
  if ( !(q = NAG_ALLOC(1 * 1, double)) ||
       !(t = NAG_ALLOC(n * n, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read T from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        Vscanf("%lf", &T(i,j));
    }
  Vscanf("%*[^\n] ");
  Vscanf("%ld%ld%*[^\n] ", &ifst, &ilst);

  /* Reorder the Schur factorization T */
  f08qfc(order, Nag_NotQ, n, t, pdt, q, pdq, &ifst, &ilst, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08qfc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print reordered Schur form */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         t, pdt, "Reordered Schur form", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (q) NAG_FREE(q);
  if (t) NAG_FREE(t);

  return exit_status;
}
```

9.2 Program Data

```
f08qfc Example Program Data
  4                               :Value of N
  0.80  -0.11   0.01   0.03
  0.00  -0.10   0.25   0.35
  0.00  -0.65  -0.10   0.20
  0.00   0.00   0.00  -0.10   :End of matrix T
  2  1                          :Values of IFST and ILST
```

9.3 Program Results

```
f08qfc Example Program Results

 Reordered Schur form
          1         2         3         4
 1  -0.1000   -0.6463    0.0874    0.2010
 2   0.2514   -0.1000    0.0927    0.3505
 3   0.0000    0.0000    0.8000   -0.0117
 4   0.0000    0.0000    0.0000   -0.1000
```