NAG Library Function Document

nag_dtrexc (f08qfc)

1 Purpose

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

2 Specification

```
#include <nag.h>
#include <nagf08.h>
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 $A = QTQ^{\mathrm{T}}$, 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: $\tilde{T} = Z^{\mathrm{T}}TZ$. Optionally the updated matrix $\tilde{Q}$ of Schur vectors is computed as $\tilde{Q} = QZ$, giving $A = \tilde{Q}\tilde{T}\tilde{Q}^{\mathrm{T}}$.

4 References

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: **compq** – Nag_ComputeQType *Input*

On entry: indicates whether the matrix Q of Schur vectors is to be updated.

compq = Nag_UpdateSchur
  The matrix Q of Schur vectors is updated.

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 $\max(1, \textbf{pdt} \times \textbf{n})$.

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

$\textbf{t}[(j-1) \times \textbf{pdt} + i - 1]$ when **order** = Nag_ColMajor;
$\textbf{t}[(i-1) \times \textbf{pdt} + j - 1]$ when **order** = Nag_RowMajor.

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 9.

5: **pdt** – Integer *Input*

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

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

6: **q**$[dim]$ – double *Input/Output*

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

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

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

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

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.

If **compq** = Nag_NotQ, **q** is not referenced.

7: **pdq** – Integer *Input*

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

Constraints:

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

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

10: **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_ENUM_INT_2

On entry, **compq** = ⟨*value*⟩, **pdq** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
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_INT

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

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

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

NE_INT_2

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

NE_INT_3

On entry, **n** = ⟨*value*⟩, **ifst** = ⟨*value*⟩ and **ilst** = ⟨*value*⟩.
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.

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.

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 reordering, 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, i.e., for a pair of complex eigenvalues to become purely real. The values of real eigenvalues however are never changed by the reordering.

8 Parallelism and Performance

nag_dtrexc (f08qfc) is not threaded by NAG in any implementation.

nag_dtrexc (f08qfc) 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 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).

10 Example

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

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

10.1 Program Text

```
/* nag_dtrexc (f08qfc) Example Program.
 *
 * Copyright 2014 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_LOAD_FP
  /* The following line is needed to force the Microsoft linker
     to load floating point support */
  float         force_loading_of_ms_float_support = 0;
#endif /* NAG_LOAD_FP */

#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);

  printf("nag_dtrexc (f08qfc) 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
#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)))
    {
      printf("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)
#ifdef _WIN32
        scanf_s("%lf", &T(i, j));
#else
        scanf("%lf", &T(i, j));
#endif
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &ifst, &ilst);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &ifst, &ilst);
#endif

  /* Reorder the Schur factorization T */
  /* nag_dtrexc (f08qfc).
   * Reorder Schur factorization of real matrix using
   * orthogonal similarity transformation
   */
  nag_dtrexc(order, Nag_NotQ, n, t, pdt, q, pdq, &ifst, &ilst, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtrexc (f08qfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print reordered Schur form */
  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                         t, pdt, "Reordered Schur form", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(q);
  NAG_FREE(t);

  return exit_status;
}
```

10.2 Program Data

```
nag_dtrexc (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
```

10.3 Program Results

```
nag_dtrexc (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
```