NAG Library Function Document

nag_dorgtr (f08ffc)

1 Purpose

nag_dorgtr (f08ffc) generates the real orthogonal matrix Q, which was determined by nag_dsytrd (f08fec) when reducing a symmetric matrix to tridiagonal form.

2 Specification

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

```
void nag_dorgtr (Nag_OrderType order, Nag_UploType uplo, Integer n,
     double a[], Integer pda, const double tau[], NagError *fail)
```

3 Description

nag_dorgtr (f08ffc) is intended to be used after a call to nag_dsytrd (f08fec), which reduces a real symmetric matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation: $A = QTQ^{\mathrm{T}}$. nag_dsytrd (f08fec) represents the orthogonal matrix Q as a product of $n-1$ elementary reflectors.

This function may be used to generate Q explicitly as a square matrix.

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: **uplo** – Nag_UploType *Input*

On entry: this **must** be the same argument **uplo** as supplied to nag_dsytrd (f08fec).

Constraint: **uplo** = Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **a**[*dim*] – double *Input/Output*

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

On entry: details of the vectors which define the elementary reflectors, as returned by nag_dsytrd (f08fec).

On exit: the n by n orthogonal matrix Q.

If **order** $=$ Nag_ColMajor, the (i,j)th element of the matrix is stored in $\mathbf{a}[(j-1)\times\mathbf{pda}+i-1]$.

If **order** $=$ Nag_RowMajor, the (i,j)th element of the matrix is stored in $\mathbf{a}[(i-1)\times\mathbf{pda}+j-1]$.

5: **pda** – Integer *Input*

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

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

6: **tau**[dim] – const double *Input*

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

On entry: further details of the elementary reflectors, as returned by nag_dsytrd (f08fec).

7: **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_INT

On entry, $\mathbf{n}=\langle\mathit{value}\rangle$.
Constraint: $\mathbf{n}\geq0$.

On entry, $\mathbf{pda}=\langle\mathit{value}\rangle$.
Constraint: $\mathbf{pda}>0$.

NE_INT_2

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

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.

7 Accuracy

The computed matrix Q differs from an exactly orthogonal matrix by a matrix E such that

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

where ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_dorgtr (f08ffc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dorgtr (f08ffc) 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 $\frac{4}{3}n^3$.

The complex analogue of this function is nag_zungtr (f08ftc).

10 Example

This example computes all the eigenvalues and eigenvectors of the matrix A, where

$$A = \begin{pmatrix} 2.07 & 3.87 & 4.20 & -1.15 \\ 3.87 & -0.21 & 1.87 & 0.63 \\ 4.20 & 1.87 & 1.15 & 2.06 \\ -1.15 & 0.63 & 2.06 & -1.81 \end{pmatrix}.$$

Here A is symmetric and must first be reduced to tridiagonal form by nag_dsytrd (f08fec). The program then calls nag_dorgtr (f08ffc) to form Q, and passes this matrix to nag_dsteqr (f08jec) which computes the eigenvalues and eigenvectors of A.

10.1 Program Text

```
/* nag_dorgtr (f08ffc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer       i, j, n, pda, pdz, d_len, e_len, tau_len;
  Integer       exit_status = 0;
  NagError      fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
```

```
  char          nag_enum_arg[40];
  double        *a = 0, *d = 0, *e = 0, *tau = 0, *z = 0;

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

  INIT_FAIL(fail);

  printf("nag_dorgtr (f08ffc) 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

  pda = n;
  pdz = n;
  tau_len = n-1;
  d_len = n;
  e_len = n-1;
  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, double)) ||
      !(d = NAG_ALLOC(d_len, double)) ||
      !(e = NAG_ALLOC(e_len, double)) ||
      !(tau = NAG_ALLOC(tau_len, double)) ||
      !(z = NAG_ALLOC(n * n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A from data file */
#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
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
```

```
      scanf("%*[^\n] ");
#endif
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  /* Reduce A to tridiagonal form T = (Q**T)*A*Q */
  /* nag_dsytrd (f08fec).
   * Orthogonal reduction of real symmetric matrix to
   * symmetric tridiagonal form
   */
  nag_dsytrd(order, uplo, n, a, pda, d, e, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsytrd (f08fec).\n%s\n", fail.message);
      exit_status = 1;
    }

  /* Copy A into Z using  nag_dtr_copy (f16qec). */
  nag_dtr_copy(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n, a, pda, z, pdz,
               &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from dtr_copy.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Form Q explicitly, storing the result in z using  nag_dorgtr (f08ffc). */
  nag_dorgtr(order, uplo, n, z, pdz, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dorgtr (f08ffc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Calculate all the eigenvalues and eigenvectors of matrix A */
  nag_dsteqr(order, Nag_UpdateZ, n, d, e, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsteqr (f08jec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Normalize the eigenvectors */
  for(j=1; j<=n; j++)
    {
      for(i=n; i>=1; i--)
        {
          Z(i, j) = Z(i, j) / Z(1,j);
        }
    }

  /* Print eigenvalues and eigenvectors */
```

```
  printf("Eigenvalues\n");
  for (i = 1; i <= n; ++i)
    printf("%8.4f%s", d[i-1], i%8 == 0?"\n":" ");
  printf("\n\n");

  /* 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,
                         z, pdz, "Eigenvectors", 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(a);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(tau);
  NAG_FREE(z);

  return exit_status;
}
```

10.2 Program Data

```
nag_dorgtr (f08ffc) Example Program Data
  4                           :Value of N
  Nag_Lower                   :Value of UPLO
  2.07
  3.87  -0.21
  4.20   1.87   1.15
 -1.15   0.63   2.06  -1.81   :End of matrix A
```

10.3 Program Results

```
nag_dorgtr (f08ffc) Example Program Results

Eigenvalues
 -5.0034  -1.9987   0.2013   8.0008

 Eigenvectors
             1          2          3          4
 1      1.0000     1.0000     1.0000     1.0000
 2     -0.6148    -3.4333     0.4489     0.6668
 3     -0.8378     1.7553    -1.3572     0.8248
 4      1.0219    -1.6052    -1.8213     0.0988
```