NAG Library Function Document

nag_dopmtr (f08ggc)

1 Purpose

nag_dopmtr (f08ggc) multiplies an arbitrary real matrix C by the real orthogonal matrix Q which was determined by nag_dsptrd (f08gec) when reducing a real symmetric matrix to tridiagonal form.

2 Specification

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

```
void nag_dopmtr (Nag_OrderType order, Nag_SideType side, Nag_UploType uplo,
     Nag_TransType trans, Integer m, Integer n, double ap[],
     const double tau[], double c[], Integer pdc, NagError *fail)
```

3 Description

nag_dopmtr (f08ggc) is intended to be used after a call to nag_dsptrd (f08gec), which reduces a real symmetric matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation: $A = QTQ^{\mathrm{T}}$. nag_dsptrd (f08gec) represents the orthogonal matrix Q as a product of elementary reflectors.

This function may be used to form one of the matrix products

$$QC, Q^{\mathrm{T}}C, CQ \text{ or } CQ^{\mathrm{T}},$$

overwriting the result on C (which may be any real rectangular matrix).

A common application of this function is to transform a matrix Z of eigenvectors of T to the matrix QZ of eigenvectors of A.

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: **side** – Nag_SideType *Input*

On entry: indicates how Q or Q^{T} is to be applied to C.

side = Nag_LeftSide
Q or Q^{T} is applied to C from the left.

side = Nag_RightSide
Q or Q^{T} is applied to C from the right.

Constraint: **side** = Nag_LeftSide or Nag_RightSide.

3: **uplo** – Nag_UploType *Input*

On entry: this **must** be the same argument **uplo** as supplied to nag_dsptrd (f08gec).

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

4: **trans** – Nag_TransType *Input*

On entry: indicates whether Q or Q^{T} is to be applied to C.

trans = Nag_NoTrans
Q is applied to C.

trans = Nag_Trans
Q^{T} is applied to C.

Constraint: **trans** = Nag_NoTrans or Nag_Trans.

5: **m** – Integer *Input*

On entry: m, the number of rows of the matrix C; m is also the order of Q if **side** = Nag_LeftSide.

Constraint: **m** ≥ 0.

6: **n** – Integer *Input*

On entry: n, the number of columns of the matrix C; n is also the order of Q if **side** = Nag_RightSide.

Constraint: **n** ≥ 0.

7: **ap**[*dim*] – double *Input/Output*

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

$\max(1, \mathbf{m} \times (\mathbf{m}+1)/2)$ when **side** = Nag_LeftSide;
$\max(1, \mathbf{n} \times (\mathbf{n}+1)/2)$ when **side** = Nag_RightSide.

On entry: details of the vectors which define the elementary reflectors, as returned by nag_dsptrd (f08gec).

On exit: is used as internal workspace prior to being restored and hence is unchanged.

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

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

$\max(1, \mathbf{m}-1)$ when **side** = Nag_LeftSide;
$\max(1, \mathbf{n}-1)$ when **side** = Nag_RightSide.

On entry: further details of the elementary reflectors, as returned by nag_dsptrd (f08gec).

9: **c**[*dim*] – double *Input/Output*

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

$\max(1, \mathbf{pdc} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdc})$ when **order** = Nag_RowMajor.

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

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

On entry: the m by n matrix C.

On exit: **c** is overwritten by QC or $Q^{\mathrm{T}}C$ or CQ or CQ^{T} as specified by **side** and **trans**.

10: **pdc** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdc** $\geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, **pdc** $\geq \max(1, \mathbf{n})$.

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

NE_INT

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

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

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

NE_INT_2

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

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

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.

7 Accuracy

The computed result differs from the exact result by a matrix E such that

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

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

8 Parallelism and Performance

nag_dopmtr (f08ggc) is not threaded by NAG in any implementation.

nag_dopmtr (f08ggc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of floating-point operations is approximately $2m^2n$ if **side** = Nag_LeftSide and $2mn^2$ if **side** = Nag_RightSide.

The complex analogue of this function is nag_zupmtr (f08guc).

10 Example

This example computes the two smallest eigenvalues, and the associated eigenvectors, of the matrix A, where

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

using packed storage. Here A is symmetric and must first be reduced to tridiagonal form T by nag_dsptrd (f08gec). The program then calls nag_dstebz (f08jjc) to compute the requested eigenvalues and nag_dstein (f08jkc) to compute the associated eigenvectors of T. Finally nag_dopmtr (f08ggc) is called to transform the eigenvectors to those of A.

10.1 Program Text

```
/* nag_dopmtr (f08ggc) 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       ap_len, i, j, m, n, nsplit, pdz, d_len, e_len;
  Integer       tau_len;
  Integer       exit_status = 0;
  double        vl = 0.0, vu = 0.0;
  NagError      fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
  char          nag_enum_arg[40];
  Integer       *iblock = 0, *ifailv = 0, *isplit = 0;
  double        *ap = 0, *d = 0, *e = 0, *tau = 0, *w = 0, *z = 0;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J * (J - 1) / 2 + I - 1]
#define A_LOWER(I, J) ap[(2 * n - J) * (J - 1) / 2 + I - 1]
#define Z(I, J) z[(J - 1) * pdz + I - 1]
  order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I * (I - 1) / 2 + J - 1]
#define A_UPPER(I, J) ap[(2 * n - I) * (I - 1) / 2 + J - 1]
#define Z(I, J) z[(I - 1) * pdz + J - 1]
  order = Nag_RowMajor;
#endif
```

```
  INIT_FAIL(fail);

  printf("nag_dopmtr (f08ggc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%*[^\n] ", &n);
  pdz = n;

  ap_len = n*(n+1)/2;
  tau_len = n-1;
  d_len = n;
  e_len = n-1;
  /* Allocate memory */
  if (!(ap = NAG_ALLOC(ap_len, double)) ||
      !(d = NAG_ALLOC(d_len, double)) ||
      !(e = NAG_ALLOC(e_len, double)) ||
      !(iblock = NAG_ALLOC(n, Integer)) ||
      !(ifailv = NAG_ALLOC(n, Integer)) ||
      !(isplit = NAG_ALLOC(n, Integer)) ||
      !(w = NAG_ALLOC(n, 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 */
  scanf("%39s%*[^\n] ", nag_enum_arg);
  /* 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)
            scanf("%lf", &A_UPPER(i, j));
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            scanf("%lf", &A_LOWER(i, j));
        }
      scanf("%*[^\n] ");
    }

  /* Reduce A to tridiagonal form T = (Q**T)*A*Q */
  /* nag_dsptrd (f08gec).
   * Orthogonal reduction of real symmetric matrix to
   * symmetric tridiagonal form, packed storage
   */
  nag_dsptrd(order, uplo, n, ap, d, e, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsptrd (f08gec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Calculate the two smallest eigenvalues of T (same as A) */
  /* nag_dstebz (f08jjc).
   * Selected eigenvalues of real symmetric tridiagonal matrix
   * by bisection
```

```
  */
 nag_dstebz(Nag_Indices, Nag_ByBlock, n, vl, vu, 1, 2, 0.0,
            d, e, &m, &nsplit, w, iblock, isplit, &fail);
 if (fail.code != NE_NOERROR)
   {
     printf("Error from nag_dstebz (f08jjc).\n%s\n", fail.message);
     exit_status = 1;
     goto END;
   }

 /* Print eigenvalues */
 printf("Eigenvalues\n");
 for (i = 0; i < m; ++i)
   printf("%8.4f%s", w[i], (i+1)%8 == 0?"\n":" ");
 printf("\n\n");
 /* Calculate the eigenvectors of T storing the result in Z */
 /* nag_dstein (f08jkc).
  * Selected eigenvectors of real symmetric tridiagonal
  * matrix by inverse iteration, storing eigenvectors in real
  * array
  */
 nag_dstein(order, n, d, e, m, w, iblock, isplit, z, pdz, ifailv,
            &fail);
 if (fail.code != NE_NOERROR)
   {
     printf("Error from nag_dstein (f08jkc).\n%s\n", fail.message);
     exit_status = 1;
     goto END;
   }
 /* Calculate all the eigenvectors of A = Q*(eigenvectors of T) */
 /* nag_dopmtr (f08ggc).
  * Apply orthogonal transformation determined by nag_dsptrd
  * (f08gec)
  */
 nag_dopmtr(order, Nag_LeftSide, uplo, Nag_NoTrans, n, m, ap,
            tau, z, pdz, &fail);
 if (fail.code != NE_NOERROR)
   {
     printf("Error from nag_dopmtr (f08ggc).\n%s\n", fail.message);
     exit_status = 1;
     goto END;
   }
 /* Normalize the eigenvectors */
 for(j=1; j<=m; j++)
   {
     for(i=n; i>=1; i--)
       {
         Z(i, j) = Z(i, j) / Z(1,j);
       }
   }
 /* Print eigenvectors */
 /* 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, m, 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(ap);
 NAG_FREE(d);
 NAG_FREE(e);
 NAG_FREE(iblock);
 NAG_FREE(ifailv);
 NAG_FREE(isplit);
```

```
  NAG_FREE(tau);
  NAG_FREE(w);
  NAG_FREE(z);

  return exit_status;
}
```

10.2 Program Data

```
nag_dopmtr (f08ggc) Example Program Data
  4                            :Value of n
  Nag_Upper                    :Value of uplo
  2.07   3.87   4.20  -1.15
        -0.21   1.87   0.63
                1.15   2.06
                      -1.81    :End of matrix A
```

10.3 Program Results

```
nag_dopmtr (f08ggc) Example Program Results

Eigenvalues
 -5.0034  -1.9987

 Eigenvectors
            1          2
 1      1.0000     1.0000
 2     -0.6148    -3.4333
 3     -0.8378     1.7553
 4      1.0219    -1.6052
```