NAG Library Function Document

nag_dormtr (f08fgc)

1 Purpose
nag_dormtr (f08fgc) multiplies an arbitrary real matrix C by the real orthogonal matrix Q which was determined by nag_dsytrd (f08fec) when reducing a real symmetric matrix to tridiagonal form.

2 Specification
#include <nag.h>
#include <nagf08.h>

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

3 Description
nag_dormtr (f08fgc) 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: 
\[ A = QTQ^T \]. nag_dsytrd (f08fec) represents the orthogonal matrix Q as a product of elementary reflectors.

This function may be used to form one of the matrix products 
\[ QC, Q^TC, CQ \text{ or } CQ^T, \]
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

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

Constraint: uplo = Nag_Upper or Nag_Lower.

4: trans – Nag_TransType  
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  
On entry: \( m \), the number of rows of the matrix \( C \); \( m \) is also the order of \( Q \) if side = Nag_LeftSide.

Constraint: \( m \geq 0 \).

6: n – Integer  
On entry: \( n \), the number of columns of the matrix \( C \); \( n \) is also the order of \( Q \) if side = Nag_RightSide.

Constraint: \( n \geq 0 \).

7: a[dim] – const double  
Note: the dimension, \( dim \), of the array \( a \) must be at least
    \[ \max(1, pda \times m) \text{ when } \text{side} = \text{NagLeftSide}; \]
    \[ \max(1, pda \times n) \text{ when } \text{side} = \text{NagRightSide}. \]

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

8: pda – Integer  
On entry: the stride separating row or column elements (depending on the value of order) of the matrix \( A \) in the array \( a \).

Constraints:
    if \( \text{side} = \text{NagLeftSide}, pda \geq \max(1, m); \)
    if \( \text{side} = \text{NagRightSide}, pda \geq \max(1, n). \)

9: tau[dim] – const double  
Note: the dimension, \( dim \), of the array \( tau \) must be at least
    \[ \max(1, m - 1) \text{ when } \text{side} = \text{NagLeftSide}; \]
    \[ \max(1, n - 1) \text{ when } \text{side} = \text{NagRightSide}. \]

On entry: further details of the elementary reflectors, as returned by nag_dsytrd (f08fec).
10:  \( \mathbf{c}[\text{dim}] \) – double

*Input/Output*

**Note:** the dimension, \( \text{dim} \), of the array \( \mathbf{c} \) must be at least

\[
\max(1, p\text{dc} \times n) \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor};
\]

\[
\max(1, m \times p\text{dc}) \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

The \((i,j)\)th element of the matrix \( \mathbf{C} \) is stored in

\[
\mathbf{c}[\hspace{1em}(j - 1) \times p\text{dc} \times i - 1] \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor};
\]

\[
\mathbf{c}[\hspace{1em}(i - 1) \times p\text{dc} \times j - 1] \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

**On entry:** the \( m \) by \( n \) matrix \( \mathbf{C} \).

**On exit:** \( \mathbf{c} \) is overwritten by \( \mathbf{Q} \mathbf{C} \) or \( \mathbf{Q}^\mathsf{T} \mathbf{C} \) or \( \mathbf{C} \mathbf{Q} \) or \( \mathbf{C} \mathbf{Q}^\mathsf{T} \) as specified by \( \text{side} \) and \( \text{trans} \).

11:  \( p\text{dc} \) – Integer

*Input*

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

**Constraints:**

- if \( \text{order} = \text{Nag\_ColMajor}, \ p\text{dc} \geq \max(1, m) \);
- if \( \text{order} = \text{Nag\_RowMajor}, \ p\text{dc} \geq \max(1, n) \).

12:  \( \text{fail} \) – 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 \( \langle \text{value} \rangle \) had an illegal value.

**NE\_ENUM\_INT\_3**

On entry, \( \text{side} = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \) and \( p\text{da} = \langle \text{value} \rangle \).

Constraint: if \( \text{side} = \text{Nag\_LeftSide}, \ p\text{da} \geq \max(1, m) \);

if \( \text{side} = \text{Nag\_RightSide}, \ p\text{da} \geq \max(1, n) \).

**NE\_INT**

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 0 \).

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( p\text{da} = \langle \text{value} \rangle \).

Constraint: \( p\text{da} > 0 \).

On entry, \( p\text{dc} = \langle \text{value} \rangle \).

Constraint: \( p\text{dc} > 0 \).

**NE\_INT\_2**

On entry, \( p\text{dc} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).

Constraint: \( p\text{dc} \geq \max(1, m) \).
On entry, $pdc = \langle\text{value}\rangle$ and $n = \langle\text{value}\rangle$.
Constraint: $pdc \geq \max(1, n)$.

**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 result differs from the exact result by a matrix $E$ such that

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

where $\epsilon$ is the *machine precision*.

8  Parallelism and Performance

`nag_dormtr (f08fgc)` is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_dormtr (f08fgc)` 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 $2m^2n$ if `side = Nag_LeftSide` and $2mn^2$ if `side = Nag_RightSide`.

The complex analogue of this function is `nag_zunmtr (f08fuc)`.

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

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

Here $A$ is symmetric and must first be reduced to tridiagonal form $T$ by `nag_dsytrd (f08fec)`. 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_dormtr (f08fgc)` is called to transform the eigenvectors to those of $A$. 
10.1 Program Text

/* nag_dormtr (f08fgc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
*/

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

int main(void)
{
    /* Scalars */
    Integer i, j, m, n, nsplit, pda, pdz, d_len, e_len, 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 *a = 0, *d = 0, *e = 0, *tau = 0, *w = 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_dormtr (f08fgc) 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)) |
        !(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)))
    {
        /* Further error handling */
    }
}

Mark 25
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
    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
}
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", 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)
{
/* Print eigenvalues */
printf("Eigenvalues\n\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 */
/* 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_dormtr (f08fgc). * Apply orthogonal transformation determined by nag_dsytrd * (f08fec) */
nag_dormtr(order, Nag_LeftSide, uplo, Nag_NoTrans, n, m, a, pda, tau, z, pdz, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dormtr (f08fgc).\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(a);
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_dormtr (f08fgc) 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_dormtr (f08fgc) Example Program Results

Eigenvalues
-5.0034 -1.9987

Eigenvectors

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>-0.6148</td>
<td>-3.4333</td>
</tr>
<tr>
<td>3</td>
<td>-0.8378</td>
<td>1.7553</td>
</tr>
<tr>
<td>4</td>
<td>1.0219</td>
<td>-1.6052</td>
</tr>
</tbody>
</table>