NAG C 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

```c
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 Parameters

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry:} the \texttt{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 \texttt{order = Nag_RowMajor}. See Section 2.2.1.4 of the Essential Introduction for a more detailed
explanation of the use of this parameter.

\textit{Constraint:} \texttt{order = Nag_RowMajor} or \texttt{Nag_ColMajor}.

2: \textbf{side} – Nag_SideType \hspace{1cm} \textit{Input}

\textit{On entry:} indicates how \( Q \) or \( Q^T \) is to be applied to \( C \) as follows:
\begin{align*}
\text{if} \texttt{side = Nag_LeftSide, } Q \text{ or } Q^T \text{ is applied to } C \text{ from the left;} \\
\text{if} \texttt{side = Nag_RightSide, } Q \text{ or } Q^T \text{ is applied to } C \text{ from the right.}
\end{align*}

\textit{Constraint:} \texttt{side = Nag_LeftSide} or \texttt{Nag_RightSide}.

3: \textbf{uplo} – Nag_UploType \hspace{1cm} \textit{Input}

\textit{On entry:} this \texttt{must} be the same parameter \texttt{uplo} as supplied to nag_dsytrd (f08fec).

\textit{Constraint:} \texttt{uplo = Nag_Upper} or \texttt{Nag_Lower}.

[NP3645/7]
trans (Nag_TransType) is an Input parameter.

On entry: indicates whether \(Q\) or \(Q^T\) is to be applied to \(C\) as follows:

- if \(\text{trans} = \text{Nag}_\text{NoTrans}\), \(Q\) is applied to \(C\);
- if \(\text{trans} = \text{Nag}_\text{Trans}\), \(Q^T\) is applied to \(C\).

Constraint: \(\text{trans} = \text{Nag}_\text{NoTrans}\) or \(\text{Nag}_\text{Trans}\).

\[\text{m} - \text{Integer}\]

Input

On entry: \(m\), the number of rows of the matrix \(C\); \(m\) is also the order of \(Q\) if \(\text{side} = \text{Nag}_\text{LeftSide}\).

Constraint: \(m \geq 0\).

\[\text{n} - \text{Integer}\]

Input

On entry: \(n\), the number of columns of the matrix \(C\); \(n\) is also the order of \(Q\) if \(\text{side} = \text{Nag}_\text{RightSide}\).

Constraint: \(n \geq 0\).

\[\text{a}[\text{dim}] - \text{double}\]

Input/Output

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

\[\max(1, \text{pda} \times \text{m})\] when \(\text{side} = \text{Nag}_\text{LeftSide}\);

\[\max(1, \text{pda} \times \text{n})\] when \(\text{side} = \text{Nag}_\text{RightSide}\).

If \(\text{order} = \text{Nag}_\text{ColMajor}\), the \((i, j)\)th element of the matrix \(A\) is stored in \(\text{a}[(j - 1) \times \text{pda} + i - 1]\) and if \(\text{order} = \text{Nag}_\text{RowMajor}\), the \((i, j)\)th element of the matrix \(A\) is stored in \(\text{a}[(i - 1) \times \text{pda} + j - 1]\).

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

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

\[\text{pda} - \text{Integer}\]

Input

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

Constraints:

- if \(\text{side} = \text{Nag}_\text{LeftSide}\), \(\text{pda} \geq \max(1, \text{m})\);
- if \(\text{side} = \text{Nag}_\text{RightSide}\), \(\text{pda} \geq \max(1, \text{n})\).

\[\text{tau}[\text{dim}] - \text{const double}\]

Input

Note: the dimension, \(\text{dim}\), of the array \(\text{tau}\) must be at least \(\max(1, \text{m} - 1)\) when \(\text{side} = \text{Nag}_\text{LeftSide}\) and at least \(\max(1, \text{n} - 1)\) when \(\text{side} = \text{Nag}_\text{RightSide}\).

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

\[\text{c}[\text{dim}] - \text{double}\]

Input/Output

Note: the dimension, \(\text{dim}\), of the array \(\text{c}\) must be at least \(\max(1, \text{pdc} \times \text{n})\) when \(\text{order} = \text{Nag}_\text{ColMajor}\) and at least \(\max(1, \text{pdc} \times \text{m})\) when \(\text{order} = \text{Nag}_\text{RowMajor}\).

If \(\text{order} = \text{Nag}_\text{ColMajor}\), the \((i, j)\)th element of the matrix \(C\) is stored in \(\text{c}[(j - 1) \times \text{pdc} + i - 1]\) and if \(\text{order} = \text{Nag}_\text{RowMajor}\), the \((i, j)\)th element of the matrix \(C\) is stored in \(\text{c}[(i - 1) \times \text{pdc} + j - 1]\).

On entry: the \(m\) by \(n\) matrix \(C\).

On exit: \(\text{c}\) is overwritten by \(QC\) or \(Q^TC\) or \(CQ\) or \(CQT\) as specified by \(\text{side}\) and \(\text{trans}\).
11: **pdc** – Integer

    Input

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

    **Constraints**:
    - if **order** = Nag_ColMajor, **pdc** ≥ max(1, **m**);  
    - if **order** = Nag_RowMajor, **pdc** ≥ max(1, **n**).

12: **fail** – NagError *  

    Output

    The NAG error parameter (see the Essential Introduction).

### 6 Error Indicators and Warnings

**NE_INT**

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

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

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

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

**NE_INT_2**

    On entry, **pdc** = ⟨value⟩, **m** = ⟨value⟩.
    Constraint: **pdc** ≥ max(1, **m**).

    On entry, **pdc** = ⟨value⟩, **n** = ⟨value⟩.
    Constraint: **pdc** ≥ max(1, **n**).

**NE_ENUM_INT_3**

    On entry, **side** = ⟨value⟩, **m** = ⟨value⟩, **n** = ⟨value⟩, **pda** = ⟨value⟩.
    Constraint: if **side** = Nag_LeftSide, **pda** ≥ max(1, **m**);  
    if **side** = Nag_RightSide, **pda** ≥ max(1, **n**).

**NE_ALLOC_FAIL**

    Memory allocation failed.

**NE_BAD_PARAM**

    On entry, parameter ⟨value⟩ 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 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 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).

9 Example

To compute 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 $A$.

9.1 Program Text

```c
/* nag_dormtr (f08fgc) Example Program. */
* Copyright 2001 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 uplo_char[2];
    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]
    order = Nag_ColMajor;
    #else
    #define A(I,J) a[(I-1)*pda+J-1]
    order = Nag_RowMajor;
    #endif
    INIT_FAIL(fail);
    Vprintf("f08fgc Example Program Results\n\n");
    /* Skip heading in data file */
    Vscanf("%*\n");
    Vscanf("%*\n", &n);
    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)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A from data file */
Vscanf("' %1s ' %*[^
]\n", uplo_char);
if (*((unsigned char *)uplo_char == 'L'))
    uplo = Nag_Lower;
else if (*((unsigned char *)uplo_char == 'U'))
    uplo = Nag_Upper;
else
    {
        Vprintf("Unrecognised character for Nag_UploType type\n");
        exit_status = -1;
        goto END;
    }
if (uplo == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = i; j <= n; ++j)
                Vscanf("%lf", &A(i,j));
        }
    }
else
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = 1; j <= i; ++j)
                Vscanf("%lf", &A(i,j));
        }
    }

/*@ Reduce A to tridiagonal form T = (Q**T)*A*Q */
f08fec(order, uplo, n, a, pda, d, e, tau, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08fec.\n", fail.message);
        exit_status = 1;
        goto END;
    }

/*@ Calculate the two smallest eigenvalues of T (same as A) */
f08jjc(Nag_Indices, Nag_ByBlock, n, vl, vu, 1, 2, 0.0,
        d, e, &m, &nsplit, w, iblock, isplit, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08jjc.\n", fail.message);
        exit_status = 1;
        goto END;
    }

/*@ Print eigenvalues */
Vprintf("Eigenvalues\n");
for (i = 0; i < m; ++i)
Vprintf("%8.4f\n", w[i], (i+1)%8==0 ?"\n":" ");
Vprintf("\n\n");

/* Calculate the eigenvectors of T storing the result in Z */
f08jkc(order, n, d, e, m, w, iblock, isplit, z, pdz, ifailv, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f08jkc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Calculate all the eigenvectors of A = Q*(eigenvectors of T) */
f08fgc(order, Nag_LeftSide, uplo, Nag_NoTrans, n, m, a, pda, tau, z, pdz, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f08fgc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print eigenvectors */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m, z, pdz, "Eigenvectors", 0, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from x04cac.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

END:
if (a) NAG_FREE(a);
if (d) NAG_FREE(d);
if (e) NAG_FREE(e);
if (iblock) NAG_FREE(iblock);
if (ifailv) NAG_FREE(ifailv);
if (isplit) NAG_FREE(isplit);
if (tau) NAG_FREE(tau);
if (w) NAG_FREE(w);
if (z) NAG_FREE(z);
return exit_status;

9.2 Program Data

f08fgc Example Program Data

4 :Value of N
'L' :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

9.3 Program Results

f08fgc Example Program Results

Eigenvalues
-5.0034 -1.9987

Eigenectors
  1      2
1  0.5658 -0.2328
2 -0.3478  0.7994
3 -0.4740 -0.4087
4  0.5781  0.3737