

NAG C Library Function Document

nag_zunmtr (f08fuc)

1 Purpose

nag_zunmtr (f08fuc) multiplies an arbitrary complex matrix C by the complex unitary matrix Q which was determined by nag_zhetrd (f08fsc) when reducing a complex Hermitian matrix to tridiagonal form.

2 Specification

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

3 Description

nag_zunmtr (f08fuc) is intended to be used after a call to nag_zhetrd (f08fsc), which reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $A = QTQ^H$. nag_zhetrd (f08fsc) represents the unitary matrix Q as a product of elementary reflectors.

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

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

overwriting the result on C (which may be any complex 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 Parameters

- 1: **order** – Nag_OrderType *Input*
On entry: the **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 **order** = **Nag_RowMajor**. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.
Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.
- 2: **side** – Nag_SideType *Input*
On entry: indicates how Q or Q^H is to be applied to C as follows:
 - if **side** = **Nag_LeftSide**, Q or Q^H is applied to C from the left;
 - if **side** = **Nag_RightSide**, Q or Q^H 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 parameter **uplo** as supplied to nag_zhetrd (f08fsc).
Constraint: **uplo** = **Nag_Upper** or **Nag_Lower**.

- 4: **trans** – Nag_TransType Input
On entry: indicates whether Q or Q^H is to be applied to C as follows:
 if **trans** = **Nag_NoTrans**, Q is applied to C ;
 if **trans** = **Nag_ConjTrans**, Q^H is applied to C .
Constraint: **trans** = **Nag_NoTrans** or **Nag_ConjTrans**.
- 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: $m \geq 0$.
- 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: $n \geq 0$.
- 7: **a**[dim] – Complex Input/Output
Note: the dimension, dim , of the array **a** must be at least
 $\max(1, pda \times m)$ when **side** = **Nag_LeftSide**;
 $\max(1, pda \times n)$ when **side** = **Nag_RightSide**.
 If **order** = **Nag_ColMajor**, the (i, j) th element of the matrix A is stored in **a**[($j - 1$) $\times$ **pda** + $i - 1$] and
 if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix A is stored in **a**[($i - 1$) $\times$ **pda** + $j - 1$].
On entry: details of the vectors which define the elementary reflectors, as returned by nag_zhetrd (f08fsc).
On exit: used as internal workspace prior to being restored and hence is unchanged.
- 8: **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**.
Constraints:
 if **side** = **Nag_LeftSide**, **pda** $\geq \max(1, m)$;
 if **side** = **Nag_RightSide**, **pda** $\geq \max(1, n)$.
- 9: **tau**[dim] – const Complex Input
Note: the dimension, dim , of the array **tau** must be at least $\max(1, m - 1)$ when **side** = **Nag_LeftSide** and at least $\max(1, n - 1)$ when **side** = **Nag_RightSide**.
On entry: further details of the elementary reflectors, as returned by nag_zhetrd (f08fsc).
- 10: **c**[dim] – Complex Input/Output
Note: the dimension, dim , of the array **c** must be at least $\max(1, pdc \times n)$ when **order** = **Nag_ColMajor** and at least $\max(1, pdc \times m)$ when **order** = **Nag_RowMajor**.
 If **order** = **Nag_ColMajor**, the (i, j) th element of the matrix C is stored in **c**[($j - 1$) $\times$ **pdc** + $i - 1$] and
 if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix C is stored in **c**[($i - 1$) $\times$ **pdc** + $j - 1$].
On entry: the m by n matrix C .
On exit: **c** is overwritten by QC or $Q^H C$ or CQ or CQ^H as specified by **side** and **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** $\geq \max(1, \mathbf{m})$;
 if **order** = **Nag_RowMajor**, **pdc** $\geq \max(1, \mathbf{n})$.
- 12: **fail** – NagError * *Output*
 The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **m** = $\langle value \rangle$.
 Constraint: **m** ≥ 0 .

On entry, **n** = $\langle value \rangle$.
 Constraint: **n** ≥ 0 .

On entry, **pda** = $\langle value \rangle$.
 Constraint: **pda** > 0 .

On entry, **pdc** = $\langle value \rangle$.
 Constraint: **pdc** > 0 .

NE_INT_2

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

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

NE_ENUM_INT_3

On entry, **side** = $\langle value \rangle$, **m** = $\langle value \rangle$, **n** = $\langle value \rangle$, **pda** = $\langle value \rangle$.
 Constraint: if **side** = **Nag_LeftSide**, **pda** $\geq \max(1, \mathbf{m})$;
 if **side** = **Nag_RightSide**, **pda** $\geq \max(1, \mathbf{n})$.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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 real floating-point operations is approximately $8m^2n$ if **side** = **Nag_LeftSide** and $8mn^2$ if **side** = **Nag_RightSide**.

The real analogue of this function is nag_dormtr (f08fgc).

9 Example

To compute the two smallest eigenvalues, and the associated eigenvectors, of the matrix A , where

$$A = \begin{pmatrix} -2.28 + 0.00i & 1.78 - 2.03i & 2.26 + 0.10i & -0.12 + 2.53i \\ 1.78 + 2.03i & -1.12 + 0.00i & 0.01 + 0.43i & -1.07 + 0.86i \\ 2.26 - 0.10i & 0.01 - 0.43i & -0.37 + 0.00i & 2.31 - 0.92i \\ -0.12 - 2.53i & -1.07 - 0.86i & 2.31 + 0.92i & -0.73 + 0.00i \end{pmatrix}.$$

Here A is Hermitian and must first be reduced to tridiagonal form T by nag_zhetrd (f08fsc). The program then calls nag_dstebz (f08jjc) to compute the requested eigenvalues and nag_zstein (f08jxc) to compute the associated eigenvectors of T . Finally nag_zunmtr (f08fuc) is called to transform the eigenvectors to those of A .

9.1 Program Text

```
/* nag_zunmtr (f08fuc) 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 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;
    Complex *a=0, *tau=0, *z=0;
    double *d=0, *e=0, *w=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("f08fuc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^\\n] ");
    Vscanf("%ld%*[^\\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, Complex)) ||
      !(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(n-1, Complex)) ||
      !(z = NAG_ALLOC(n * n, Complex)) )
{
    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(" ( %1f , %1f )", &A(i,j).re, &A(i,j).im);
        Vscanf("%*[^\\n] ");
    }
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %1f , %1f )", &A(i,j).re, &A(i,j).im);
        Vscanf("%*[^\\n] ");
    }
}

/* Reduce A to tridiagonal form T = (Q**H)*A*Q */
f08fsc(order, uplo, n, a, pda, d, e, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08fsc.\\n%s\\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%s\\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print eigenvalues */
Vprintf("Eigenvalues\\n");

```

```

for (i = 0; i < m; ++i)
    Vprintf("%8.4f%s", w[i], (i+1)%8==0 ?"\n":" ");
Vprintf("\n\n");
/* Calculate the eigenvectors of T storing the result in Z */
f08jxc(order, n, d, e, m, w, iblock, isplit, z, pdz, ifailv,
    &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08jxc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Calculate all the eigenvectors of A = Q*(eigenvectors of T) */
f08fuc(order, Nag_LeftSide, uplo, Nag_NoTrans, n, m, a, pda,
    tau, z, pdz, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08fuc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print eigenvectors */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
    z, pdz, Nag_BracketForm, "%7.4f", "Eigenvectors",
    Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0,
    0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04dbc.\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

f08fuc Example Program Data

```

4                                     :Value of N
'L'                                 :Value of UPLO
(-2.28, 0.00)
( 1.78, 2.03) (-1.12, 0.00)
( 2.26,-0.10) ( 0.01,-0.43) (-0.37, 0.00)
(-0.12,-2.53) (-1.07,-0.86) ( 2.31, 0.92) (-0.73, 0.00) :End of matrix A

```

9.3 Program Results

f08fuc Example Program Results

Eigenvalues

```

-6.0002          -3.0030

```

Eigenvectors

```

          1          2
1 ( 0.7299, 0.0000) (-0.2595, 0.0000)
2 (-0.1663,-0.2061) ( 0.5969, 0.4214)
3 (-0.4165,-0.1417) (-0.2965,-0.1507)
4 ( 0.1743, 0.4162) ( 0.3482, 0.4085)

```