NAG 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

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

```
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: $A = QTQ^{\mathrm{H}}$. 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

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

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 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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^{H} is to be applied to C.

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

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 argument **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.

trans = Nag_NoTrans
Q is applied to C.

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: $\mathbf{m} \geq 0$.

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: $\mathbf{n} \geq 0$.

7: **a**[dim] – const Complex *Input*

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

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

On entry: details of the vectors which define the elementary reflectors, as returned by nag_zhetrd (f08fsc).

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, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **side** = Nag_RightSide, $\mathbf{pda} \geq \max(1, \mathbf{n})$.

9: **tau**[dim] – const Complex *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_zhetrd (f08fsc).

10: **c**[dim] – Complex *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 $Q^{H}C$ or CQ or CQ^{H} as specified by **side** and **trans**.

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

12: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_3

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

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: $\mathbf{m} \geq 0$.

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **pda** = ⟨*value*⟩.
Constraint: $\mathbf{pda} > 0$.

On entry, **pdc** = ⟨*value*⟩.
Constraint: $\mathbf{pdc} > 0$.

NE_INT_2

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

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

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation 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 ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_zunmtr (f08fuc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zunmtr (f08fuc) 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 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).

10 Example

This example computes 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.

10.1 Program Text

```
/* nag_zunmtr (f08fuc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer i, j, m, n, nsplit, pda, pdz, d_len, e_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;
  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]
#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_zunmtr (f08fuc) 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;

  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)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read A from data file */
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, (unsigned)_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 , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#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 , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  /* Reduce A to tridiagonal form T = (Q^H)*A*Q */
  /* nag_zhetrd (f08fsc).
   * Unitary reduction of complex Hermitian matrix to real
   * symmetric tridiagonal form
   */
  nag_zhetrd(order, uplo, n, a, pda, d, e, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zhetrd (f08fsc).\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_zstein (f08jxc).
   * Selected eigenvectors of real symmetric tridiagonal
   * matrix by inverse iteration, storing eigenvectors in
   * complex array
   */
  nag_zstein(order, n, d, e, m, w, iblock, isplit, z, pdz, ifailv, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zstein (f08jxc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Calculate all the eigenvectors of A = Q*(eigenvectors of T) */
  /* nag_zunmtr (f08fuc).
   * Apply unitary transformation matrix determined by
   * nag_zhetrd (f08fsc)
   */
  nag_zunmtr(order, Nag_LeftSide, uplo, Nag_NoTrans, n, m, a, pda,
             tau, z, pdz, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zunmtr (f08fuc).\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) = nag_complex_divide(Z(i, j), Z(1, j));
    }
  }
  /* Print eigenvectors */
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(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) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\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_zunmtr (f08fuc) Example Program Data
  4                                                             :Value of n
  Nag_Upper                                                     :Value of uplo
 (-2.28, 0.00) ( 1.78,-2.03) ( 2.26, 0.10) (-0.12, 2.53)
               (-1.12, 0.00) ( 0.01, 0.43) (-1.07, 0.86)
                             (-0.37, 0.00) ( 2.31,-0.92)
                                           (-0.73, 0.00)   :End of matrix A
```

10.3 Program Results

```
nag_zunmtr (f08fuc) Example Program Results

Eigenvalues
 -6.0002           -3.0030

 Eigenvectors
                    1                  2
 1  ( 1.0000, 0.0000)  ( 1.0000,-0.0000)
 2  (-0.2278,-0.2824)  (-2.2999,-1.6237)
 3  (-0.5706,-0.1941)  ( 1.1424, 0.5807)
 4  ( 0.2388, 0.5702)  (-1.3415,-1.5739)
```