NAG Library Function Document

nag_zunmbr (f08kuc)

1 Purpose

nag_zunmbr (f08kuc) multiplies an arbitrary complex m by n matrix C by one of the complex unitary matrices Q or P which were determined by nag_zgebrd (f08ksc) when reducing a complex matrix to bidiagonal form.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_zunmbr (Nag_OrderType order, Nag_VectType vect, Nag_SideType side,
    Nag_TransType trans, Integer m, Integer n, Integer k, const Complex a[],
    Integer pda, const Complex tau[], Complex c[], Integer pdc,
    NagError *fail)
```

3 Description

nag_zunmbr (f08kuc) is intended to be used after a call to nag_zgebrd (f08ksc), which reduces a complex rectangular matrix A to real bidiagonal form B by a unitary transformation: \( A = QBP^H \). nag_zgebrd (f08ksc) represents the matrices Q and P\(^H\) as products of elementary reflectors.

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

\[ QC, Q^HC, CQ, CQ^H, PC, P^HC, CP \text{ or } CP^H, \]

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

4 References


5 Arguments

**Note:** in the descriptions below, \( r \) denotes the order of \( Q \) or \( P^H \); if `side` = Nag_LeftSide, \( r = m \) and if `side` = Nag_RightSide, \( r = n \).

1: \( \text{order} \) – 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: \( \text{vect} \) – Nag_VectType
   
   **Input**
   
   On entry: indicates whether \( Q \) or \( Q^H \) or \( P \) or \( P^H \) is to be applied to \( C \).
   
   `vect` = Nag_ApplyQ
   
   \( Q \) or \( Q^H \) is applied to \( C \).
vect = Nag_ApplyP
    P or \( P^H \) is applied to \( C \).

Constraint: vect = Nag_ApplyQ or Nag_ApplyP.

3: side – Nag_SideType
   On entry: indicates how \( Q \) or \( Q^H \) or \( P \) or \( P^H \) is to be applied to \( C \).
   side = Nag_LeftSide
       \( Q \) or \( Q^H \) or \( P \) or \( P^H \) is applied to \( C \) from the left.
   side = Nag_RightSide
       \( Q \) or \( Q^H \) or \( P \) or \( P^H \) is applied to \( C \) from the right.

Constraint: side = Nag_LeftSide or Nag_RightSide.

4: trans – Nag_TransType
   On entry: indicates whether \( Q \) or \( P \) or \( Q^H \) or \( P^H \) is to be applied to \( C \).
   trans = Nag_NoTrans
       \( Q \) or \( P \) is applied to \( C \).
   trans = Nag_ConjTrans
       \( Q^H \) or \( P^H \) is applied to \( C \).

Constraint: trans = Nag_NoTrans or Nag_ConjTrans.

5: m – Integer
   On entry: \( m \), the number of rows of the matrix \( C \).
   Constraint: \( m \geq 0 \).

6: n – Integer
   On entry: \( n \), the number of columns of the matrix \( C \).
   Constraint: \( n \geq 0 \).

7: k – Integer
   On entry: if vect = Nag_ApplyQ, the number of columns in the original matrix \( A \).
   If vect = Nag_ApplyP, the number of rows in the original matrix \( A \).
   Constraint: \( k \geq 0 \).

8: a[dim] – const Complex
   Note: the dimension, dim, of the array a must be at least
   \[
   \max(1, \text{pda} \times \min(r, k)) \quad \text{when vect = Nag_ApplyQ and order = Nag_ColMajor;}
   \]
   \[
   \max(1, r \times \text{pda}) \quad \text{when vect = Nag_ApplyQ and order = Nag_RowMajor;}
   \]
   \[
   \max(1, \text{pda} \times r) \quad \text{when vect = Nag_ApplyP and order = Nag_ColMajor;}
   \]
   \[
   \max(1, \min(r, k) \times \text{pda}) \quad \text{when vect = Nag_ApplyP and order = Nag_RowMajor.}
   \]
   On entry: details of the vectors which define the elementary reflectors, as returned by nag_zgebrd (f08ksc).

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

if order = Nag_ColMajor,
    if vect = Nag_ApplyQ, pda ≥ max(1, r);  
    if vect = Nag_ApplyP, pda ≥ max(1, min(r, k)).
if order = Nag_RowMajor,
    if vect = Nag_ApplyQ, pda ≥ max(1, min(r, k));  
    if vect = Nag_ApplyP, pda ≥ max(1, r).

10: tau[dim] – const Complex
    Input
    Note: the dimension, dim, of the array tau must be at least max(1, min(r, k)).
    On entry: further details of the elementary reflectors, as returned by nag_zgebrd (f08ksc) in its argument tauq if vect = Nag_ApplyQ, or in its argument taup if vect = Nag_ApplyP.

11: c[dim] – Complex
    Input/Output
    Note: the dimension, dim, of the array c must be at least
    max(1, pdc x n) when order = Nag_ColMajor;
    max(1, m x pdc) when order = Nag_RowMajor.
    The (i, j)th element of the matrix C is stored in
    c[(j - 1) x pdc + i - 1] when order = Nag_ColMajor;
    c[(i - 1) x pdc + j - 1] when order = Nag_RowMajor.
    On entry: the matrix C.
    On exit: c is overwritten by QC or QH C or CQ or CH Q or PC or PH C or CP or CH P as specified by vect, side and trans.

12: 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 ≥ max(1, m);
    if order = Nag_RowMajor, pdc ≥ max(1, n).

13: 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 ⟨value⟩ had an illegal value.

NE_ENUM_INT_2
    On entry, vect = ⟨value⟩, pda = ⟨value⟩, k = ⟨value⟩.
    Constraint: if vect = Nag_ApplyQ, pda ≥ max(1, min(r, k));  
    if vect = Nag_ApplyP, pda ≥ max(1, r).
On entry, \( \mathbf{vect} = \langle value \rangle \), \( \mathbf{pda} = \langle value \rangle \) and \( k = \langle value \rangle \).
Constraint: if \( \mathbf{vect} = \text{Nag\_ApplyQ} \), \( \mathbf{pda} \geq \max(1, r) \);
if \( \mathbf{vect} = \text{Nag\_ApplyP} \), \( \mathbf{pda} \geq \max(1, \min(r, k)) \).

**NE_INT**
On entry, \( k = \langle value \rangle \).
Constraint: \( k \geq 0 \).

On entry, \( m = \langle value \rangle \).
Constraint: \( m \geq 0 \).

On entry, \( n = \langle value \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( \mathbf{pda} = \langle value \rangle \).
Constraint: \( \mathbf{pda} > 0 \).

On entry, \( \mathbf{pdc} = \langle value \rangle \).
Constraint: \( \mathbf{pdc} > 0 \).

**NE_INT_2**
On entry, \( \mathbf{pdc} = \langle value \rangle \) and \( m = \langle value \rangle \).
Constraint: \( \mathbf{pdc} \geq \max(1, m) \).

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

**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
\( \text{nag\_zunmbr (f08kuc)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag\_zunmbr (f08kuc)} \) 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

\[ \begin{align*}
\text{if side} &= \text{Nag\_Left\_Side} \quad \text{and} \quad m \geq k, \quad 8nk(2m - k); \\
\text{if side} &= \text{Nag\_Right\_Side} \quad \text{and} \quad n \geq k, \quad 8nk(2n - k); \\
\text{if side} &= \text{Nag\_Left\_Side} \quad \text{and} \quad m < k, \quad 8m^2n; \\
\text{if side} &= \text{Nag\_Right\_Side} \quad \text{and} \quad n < k, \quad 8mn^2;
\end{align*} \]

where \( k \) is the value of the argument \( k \).

The real analogue of this function is \text{nag\_dormbr (f08kgc)}.

10  Example

For this function two examples are presented. Both illustrate how the reduction to bidiagonal form of a
matrix \( A \) may be preceded by a \( QR \) or \( LQ \) factorization of \( A \).

In the first example, \( m > n \), and

\[ A = \begin{pmatrix}
0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\
-0.98 + 1.98i & -1.20 - 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\
0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\
-0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\
0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\
1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i
\end{pmatrix}. \]

The function first performs a \( QR \) factorization of \( A \) as \( A = QbR \) and then reduces the factor \( R \) to
bidiagonal form \( B : R = QbBP^H \). Finally it forms \( Qa \) and calls \text{nag\_zunmbr (f08kuc)} to form \( Q = QaQb \).

In the second example, \( m < n \), and

\[ A = \begin{pmatrix}
0.28 - 0.36i & 0.50 - 0.86i & -0.77 - 0.48i & 1.58 + 0.66i \\
-0.50 - 1.10i & -1.21 + 0.76i & -0.32 - 0.24i & -0.27 - 1.15i \\
0.36 - 0.51i & -0.07 + 1.33i & -0.75 + 0.47i & -0.08 + 1.01i
\end{pmatrix}. \]

The function first performs an \( LQ \) factorization of \( A \) as \( A = LP^H \) and then reduces the factor \( L \) to
bidiagonal form \( B : L = QBP^H \). Finally it forms \( P^H \) and calls \text{nag\_zunmbr (f08kuc)} to form \( P^H = P^H \).

10.1  Program Text

/* nag\_zunmbr (f08kuc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 7, 2001. */

#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, ic, j, m, n, pda, pdph, pdu;
    Integer d\_len, e\_len, tau\_len, tauq\_len, taup\_len;
    Integer exit\_status = 0;
    NagError fail;
    Nag\_OrderType order;
    /* Arrays */
}
Complex  *a = 0, *ph = 0, *tau = 0, *taup = 0, *tauq = 0, *u = 0;
double  *d = 0, *e = 0;
#endif
#define A(I, J) a[(J-1)*pda + I - 1]
#define U(I, J) u[(J-1)*pdu + I - 1]
#define PH(I, J) ph[(J-1)*pdph + I - 1]
order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define U(I, J) u[(I-1)*pdu + J - 1]
#define PH(I, J) ph[(I-1)*pdph + J - 1]
order = Nag_RowMajor;
#endif
INIT_FAIL(fail);
printf("nag_zunmbr (f08kuc) Example Program Results\n");
#endif
/* Skip heading in data file */
#endif _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
for (ic = 1; ic <= 2; ++ic)
{
#endif _WIN32
scanf_s("%NAG_IFMT"%NAG_IFMT"%*[\n] ", &m, &n);
#else
scanf("%NAG_IFMT"%NAG_IFMT"%*[\n] ", &m, &n);
#endif
#endif NAG_COLUMN_MAJOR
pda = m;
pdph = n;
pdu = m;
#else
pda = n;
pdph = n;
pdu = m;
#endif
tau_len = n;
taup_len = n;
tauq_len = n;
d_len = n;
e_len = n - 1;

/* Allocate memory */
if (!((a = NAG_ALLOC(m * n, Complex)) ||
    (ph = NAG_ALLOC(n * n, Complex)) ||
    (!tau = NAG_ALLOC(tau_len, Complex)) ||
    (!tauq = NAG_ALLOC(tauq_len, Complex)) ||
    (!u = NAG_ALLOC(m * m, Complex)) ||
    (!d = NAG_ALLOC(d_len, double)) ||
    (!e = NAG_ALLOC(e_len, double))))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto ENDL;
}

/* Read A from data file */
for (i = 1; i <= m; ++i)
{
    for (j = 1; 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
}
```c
#endif
}
#else
    scanf("%*[\n] ");
#endif
if (m >= n)
{
    /* Compute the QR factorization of A */
    /* nag_zgeqrf (f08asc). */
    /* QR factorization of complex general rectangular matrix */
    nag_zgeqrf(order, m, n, a, pda, tau, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zgeqrf (f08asc).\n%s\n", fail.message);
        exit_status = 1;
        goto ENDL;
    }
    /* Copy A to U */
    for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= n; ++j)
        {
            U(i, j).re = A(i, j).re;
            U(i, j).im = A(i, j).im;
        }
    }
    /* Form Q explicitly, storing the result in U */
    /* nag_zungqr (f08atc). */
    /* Form all or part of unitary Q from QR factorization */
    /* determined by nag_zgeqrf (f08asc) or nag_zgeqpf (f08bsc) */
    nag_zungqr(order, m, n, n, u, pdu, tau, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zungqr (f08atc).\n%s\n", fail.message);
        exit_status = 1;
        goto ENDL;
    }
    /* Copy R to PH (used as workspace) */
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
        {
            PH(i, j).re = A(i, j).re;
            PH(i, j).im = A(i, j).im;
        }
    }
    /* Set the strictly lower triangular part of R to zero */
    for (i = 2; i <= n; ++i)
    {
        for (j = 1; j <= MIN(i - 1, n - 1); ++j)
        {
            PH(i, j).re = 0.0;
            PH(i, j).im = 0.0;
        }
    }
    /* Bidiagonalize R */
    /* nag_zgebrd (f08ksc). */
    /* Unitary reduction of complex general rectangular matrix */
    /* to bidiagonal form */
    nag_zgebrd(order, n, n, ph, pdph, d, e, tauq, taup, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zgebrd (f08ksc).\n%s\n", fail.message);
    }
}
```

Mark 25  

f08kuc.7
exit_status = 1;
goto ENDL;
}

/* Update Q, storing the result in U */
/* nag_zunmbr (f08kuc). */
/* Apply unitary transformations from reduction to */
/* bidiagonal form determined by nag_zgebrd (f08ksc) */
nag_zunmbr(order, Nag_ApplyQ, Nag_RightSide, Nag_NoTrans,
           m, n, n, ph, pdph, tauq, u, pdu, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zunmbr (f08kuc).\n%s\n", fail.message);
    exit_status = 1;
goto ENDL;
}

/* Print bidiagonal form and matrix Q */
printf("\nExample 1: bidiagonal matrix B\nDiagonal\n");
for (i = 1; i <= n; ++i)
    printf("%8.4f%s", d[i-1], i%8 == 0?"\n":" ");
printf("\nSuper-diagonal\n");
for (i = 1; i <= n - 1; ++i)
    printf("%8.4f%s", e[i-1], i%8 == 0?"\n":" ");
printf("\n\n");

/* nag_gen_complex_mat_print_comp (x04dbc). */
/* Print complex general matrix (comprehensive) */
fflush(stdout);
nag_gen_complex_mat_print_comp(order,
                               Nag_GeneralMatrix,
                               Nag_NonUnitDiag,
                               m,
                               n,
                               u,
                               pdu,
                               Nag_BracketForm,
                               "%7.4f",
                               "Example 1: matrix Q",
                               Nag_IntegerLabels, 0,
                               Nag_IntegerLabels, 0, 80, 0, 0,
                               &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complex_mat_print_comp (x04dbc)."
           "\n%s\n", fail.message);
    exit_status = 1;
goto ENDL;
}
else
{
    /* Compute the LQ factorization of A */
    /* nag_zgelqf (f08avc). */
    /* LQ factorization of complex general rectangular matrix */
nag_zgelqf(order, m, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zgelqf (f08avc).\n%s\n", fail.message);
    exit_status = 1;
goto ENDL;
}

    /* Copy A to PH */
for (i = 1; i <= m; ++i)
{
    for (j = 1; j <= n; ++j)
    {
        PH(i, j).re = A(i, j).re;
        PH(i, j).im = A(i, j).im;
    }
/* Form Q explicitly, storing the result in PH */
/* nag_zunglq (f08awc).
* Form all or part of unitary Q from LQ factorization
* determined by nag_zgelqf (f08avc)
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zunglq (f08awc).
%s
", fail.message);
    exit_status = 1;
    goto ENDL;
}
/* Copy L to U (used as workspace) */
for (i = 1; i <= m; ++i)
{
    for (j = 1; j <= i; ++j)
    {
        U(i, j).re = A(i, j).re;
        U(i, j).im = A(i, j).im;
    }
}
/* Set the strictly upper triangular part of L to zero */
for (i = 1; i <= m - 1; ++i)
{
    for (j = i + 1; j <= m; ++j)
    {
        U(i, j).re = 0.0;
        U(i, j).im = 0.0;
    }
}
/* Bidiagonalize L */
/* nag_zgebrd (f08ksc), see above. */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zgebrd (f08ksc).
%s
", fail.message);
    exit_status = 1;
    goto ENDL;
}
/* Update P**H, storing the result in PH */
/* nag_zunmbr (f08kuc), see above. */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zunmbr (f08kuc).
%s
", fail.message);
    exit_status = 1;
    goto ENDL;
}
/* Print bidiagonal form and matrix P**H */
printf("Example 2: bidiagonal matrix B

Diagonal
");
for (i = 1; i <= m; ++i)
    printf("%.4f%"d[i-1, i%8 == 0?"
": " ");
printf("Super-diagonal\n");
for (i = 1; i <= m - 1; ++i)
    printf("%.4f%"e[i-1, i%8 == 0?"
": " ");
/* nag_gen_complex_mat_print_comp (x04dbc), see above. */
cflush(stdout);
}
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n" "\n\n", fail.message);
    exit_status = 1;
    goto ENDL;
}

ENDL:
NAG_FREE(a);
NAG_FREE(ph);
NAG_FREE(tau);
NAG_FREE(taup);
NAG_FREE(tauq);
NAG_FREE(u);
NAG_FREE(d);
NAG_FREE(e);

NAG_FREE(a);
NAG_FREE(ph);
NAG_FREE(tau);
NAG_FREE(taup);
NAG_FREE(tauq);
NAG_FREE(u);
NAG_FREE(d);
NAG_FREE(e);
return exit_status;

10.2 Program Data

nag_zunmbr (f08kuc) Example Program Data

Example 1: bidiagonal matrix B
Diagonal
-3.0870 -2.0660 -1.8731 -2.0022
Super-diagonal
2.1126 -1.2628 1.6126

10.3 Program Results

nag_zunmbr (f08kuc) Example Program Results

Example 1: matrix Q
Example 1: matrix Q

1 (-0.3110, 0.2624) ( 0.6521, 0.5532) ( 0.0427, 0.0361) (-0.2634,-0.0741)
2 ( 0.3175,-0.6414) ( 0.3488, 0.0721) ( 0.2287, 0.0069) ( 0.1101,-0.0326)
3 (-0.2008, 0.1490) (-0.3103, 0.0230) ( 0.1855,-0.1817) (-0.2956, 0.5648)
4 ( 0.1199,-0.1231) (-0.0046,-0.0005) (-0.3305, 0.4821) (-0.0675, 0.3464)
5 (-0.2689,-0.1652) ( 0.1794,-0.0586) (-0.5235,-0.2580) ( 0.3927, 0.1450)
6 (-0.3499, 0.0907) ( 0.0829,-0.0506) ( 0.3202, 0.3038) ( 0.3174, 0.3241)
Example 2: bidiagonal matrix B
Diagonal
2.7615  1.6298  -1.3275
Super-diagonal
-0.9500  -1.0183

Example 2: matrix $P^*H$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-0.1258, 0.1618)</td>
<td>(-0.2247, 0.3864)</td>
<td>(0.3460, 0.2157)</td>
<td>(-0.7099, -0.2966)</td>
</tr>
<tr>
<td>2</td>
<td>(0.4148, 0.1795)</td>
<td>(0.1368, -0.3976)</td>
<td>(0.6885, 0.3386)</td>
<td>(0.1667, -0.0494)</td>
</tr>
<tr>
<td>3</td>
<td>(0.4575, -0.4807)</td>
<td>(-0.2733, 0.4981)</td>
<td>(-0.0230, 0.3861)</td>
<td>(0.1730, 0.2395)</td>
</tr>
</tbody>
</table>