NAG C Library Function Document

nag_zunmbr (f08kuc)

1 Purpose

nag_zunmbr (f08kuc) multiplies an arbitrary complex 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

```
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

$$QC,\ Q^HC,\ CQ,\ CQ^H,\ PC,\ P^HC,\ CP \textit{ or } CP^H,$$

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

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Parameters

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: **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: **vect** – Nag_VectType *Input*

On entry: indicates whether Q or Q^H or P or P^H is to be applied to C as follows:

if **vect** = **Nag_ApplyQ**, Q or Q^H is applied to C;

if **vect** = **Nag_ApplyP**, P or P^H is applied to C.

Constraint: **vect** = **Nag_ApplyQ** or **Nag_ApplyP**.

3: **side** – Nag_SideType *Input*

On entry: indicates how Q or Q^H or P or P^H is to be applied to C as follows:

if **side** = **Nag_LeftSide**, Q or Q^H or P or P^H is applied to C from the left;

if **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 *Input*

On entry: indicates whether Q or P or Q^H or P^H is to be applied to C as follows:

if **trans** = **Nag_NoTrans**, Q or P is applied to C;

if **trans** = **Nag_ConjTrans**, Q^H or P^H is applied to C.

Constraint: **trans** = **Nag_NoTrans** or **Nag_ConjTrans**.

5: **m** – Integer *Input*

On entry: m_C, the number of rows of the matrix C.

Constraint: **m** ≥ 0.

6: **n** – Integer *Input*

On entry: n_C, the number of columns of the matrix C.

Constraint: **n** ≥ 0.

7: **k** – Integer *Input*

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** ≥ 0.

8: **a**[dim] – Complex *Input/Output*

Note: the dimension, dim, of the array **a** must be at least
max(1, **pda** × max(1, min(r, **k**))) when **vect** = **Nag_ApplyQ** and **order** = **Nag_ColMajor**;
max(1, **pda** × r) when **vect** = **Nag_ApplyQ** and **order** = **Nag_RowMajor**;
max(1, **pda** × r) when **vect** = **Nag_ApplyP** and **order** = **Nag_ColMajor**;
max(1, **pda** × min(r, **k**))) 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).

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

9: **pda** – Integer *Input*

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

Constraints:

if **order** = **Nag_ColMajor**,
 if **vect** = **Nag_ApplyQ**, **pda** $\geq$ max(1, r);
 if **vect** = **Nag_ApplyP**, **pda** $\geq$ max(1, min(r, **k**));

if **order** = **Nag_RowMajor**,
 if **vect** = **Nag_ApplyQ**, **pda** $\geq$ max(1, min(r, **k**));
 if **vect** = **Nag_ApplyP**, **pda** $\geq$ 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 parameter **tauq** if **vect** = **Nag_ApplyQ**, or in its parameter **taup** if **vect** = **Nag_ApplyP**.

11: **c**[dim] – Complex *Input/Output*

Note: the dimension, dim, of the array **c** must be at least $\max(1, \mathbf{pdc} \times \mathbf{n})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdc} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix C is stored in $\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix C is stored in $\mathbf{c}[(i-1) \times \mathbf{pdc} + j - 1]$.

On entry: the matrix C.

On exit: C is overwritten by QC or $Q^H C$ or CQ or CQ^H or PC or $P^H C$ or CP or CP^H as specified by **vect**, **side** and **trans**.

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

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

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

On entry, **k** = $\langle value \rangle$.
Constraint: $\mathbf{k} \geq 0$.

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

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

NE_INT_2

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

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

NE_ENUM_INT_2

On entry, **vect** = $\langle value \rangle$, **k** = $\langle value \rangle$, **pda** = $\langle value \rangle$.
Constraint: if **vect** = **Nag_ApplyQ**, $\mathbf{pda} \geq \max(1, r)$;
if **vect** = **Nag_ApplyP**, $\mathbf{pda} \geq \max(1, \min(r, \mathbf{k}))$.

On entry, **vect** = $\langle value \rangle$, **k** = $\langle value \rangle$, **pda** = $\langle value \rangle$.
Constraint: if **vect** = **Nag_ApplyQ**, $\mathbf{pda} \geq \max(1, \min(r, \mathbf{k}))$;
if **vect** = **Nag_ApplyP**, $\mathbf{pda} \geq \max(1, r)$.

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

$$\begin{array}{ll} 8n_C k(2m_C - k), & \text{if } \mathbf{side} = \mathbf{Nag\_LeftSide} \text{ and } m_C \geq k; \\ 8m_C k(2n_C - k), & \text{if } \mathbf{side} = \mathbf{Nag\_RightSide} \text{ and } n_C \geq k; \\ 8m_C^2 n_C, & \text{if } \mathbf{side} = \mathbf{Nag\_LeftSide} \text{ and } m_C < k; \\ 8m_C n_C^2, & \text{if } \mathbf{side} = \mathbf{Nag\_RightSide} \text{ and } n_C < k; \end{array}$$

where k is the value of the parameter **k**.

The real analogue of this function is nag_dormbr (f08kgc).

9 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 $A = Q_a R$ and then reduces the factor R to bidiagonal form B: $R = Q_b B P^H$. Finally it forms Q_a and calls nag_zunmbr (f08kuc) to form $Q = Q_a Q_b$.

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 $A = LP_a^H$ and then reduces the factor L to bidiagonal form B: $L = QBP_b^H$. Finally it forms P_b^H and calls nag_zunmbr (f08kuc) to form $P^H = P_b^H P_a^H$.

9.1 Program Text

```
/* nag_zunmbr (f08kuc) Example Program.
 *
 * Copyright 2001 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;

#ifdef NAG_COLUMN_MAJOR
#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);
  Vprintf("f08kuc Example Program Results\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  for (ic = 1; ic <= 2; ++ic)
    {
      Vscanf("%ld%ld%*[^\n] ", &m, &n);

#ifdef 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)) ||
           !(taup = NAG_ALLOC(taup_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)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A from data file */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
        Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
    }
  Vscanf("%*[^\n] ");
  if (m >= n)
    {
      /* Compute the QR factorization of A */
      f08asc(order, m, n, a, pda, tau, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08asc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* 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 */
      f08atc(order, m, n, n, u, pdu, tau, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08atc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* 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 */
      f08ksc(order, n, n, ph, pdph, d, e, tauq, taup, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08ksc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* Update Q, storing the result in U */
      f08kuc(order, Nag_FormQ, Nag_RightSide, Nag_NoTrans,
             m, n, n, ph, pdph, tauq, u, pdu, &fail);
```

```
        if (fail.code != NE_NOERROR)
          {
            Vprintf("Error from f08kuc.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
          }
        /* Print bidiagonal form and matrix Q */
        Vprintf("\nExample 1: bidiagonal matrix B\nDiagonal\n");
        for (i = 1; i <= n; ++i)
          Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
        Vprintf("\nSuper-diagonal\n");
        for (i = 1; i <= n - 1; ++i)
          Vprintf("%8.4f%s", e[i-1], i%8 == 0 ?"\n":" ");
        Vprintf("\n\n");
        x04dbc(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)
          {
            Vprintf("Error from x04dbc.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
          }
      }
    else
      {
        /* Compute the LQ factorization of A */
        f08avc(order, m, n, a, pda, tau, &fail);
        if (fail.code != NE_NOERROR)
          {
            Vprintf("Error from f08avc.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
          }
        /* 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 */
        f08awc(order, m, n, m, ph, pdph, tau, &fail);
        if (fail.code != NE_NOERROR)
          {
            Vprintf("Error from f08awc.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
          }
        /* 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 */
```

```
          f08ksc(order, m, m, u, pdu, d, e, tauq, taup, &fail);
          if (fail.code != NE_NOERROR)
            {
              Vprintf("Error from f08ksc.\n%s\n", fail.message);
              exit_status = 1;
              goto END;
            }
          /* Update P**H, storing the result in PH */
          f08kuc(order, Nag_FormP, Nag_LeftSide, Nag_ConjTrans,
                 m, n, m, u, pdu, taup, ph, pdph, &fail);
          if (fail.code != NE_NOERROR)
            {
              Vprintf("Error from f08kuc.\n%s\n", fail.message);
              exit_status = 1;
              goto END;
            }

          /* Print bidiagonal form and matrix P**H */
          Vprintf("\nExample 2: bidiagonal matrix B\n%s\n",
                  "Diagonal");
          for (i = 1; i <= m; ++i)
            Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
          Vprintf("\nSuper-diagonal\n");
          for (i = 1; i <= m - 1; ++i)
            Vprintf("%8.4f%s", e[i-1], i%8==0 ?"\n":" ");
          Vprintf("\n\n");
          x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                 m, n, ph, pdph, Nag_BracketForm, "%7.4f",
                 "Example 2: matrix P**H", 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 (ph) NAG_FREE(ph);
      if (tau) NAG_FREE(tau);
      if (taup) NAG_FREE(taup);
      if (tauq) NAG_FREE(tauq);
      if (u) NAG_FREE(u);
      if (d) NAG_FREE(d);
      if (e) NAG_FREE(e);
    }
  return exit_status;
}
```

9.2 Program Data

```
f08kuc Example Program Data
  6  4                                        :Values of M and N, Example 1
 ( 0.96,-0.81) (-0.03, 0.96) (-0.91, 2.06) (-0.05, 0.41)
 (-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42) (-0.81, 0.56)
 ( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17) (-1.11, 0.60)
 (-0.37, 0.38) ( 0.19,-0.54) (-0.98,-0.36) ( 0.22,-0.20)
 ( 0.83, 0.51) ( 0.20, 0.01) (-0.17,-0.46) ( 1.47, 1.59)
 ( 1.08,-0.28) ( 0.20,-0.12) (-0.07, 1.23) ( 0.26, 0.26)   :End of matrix A
  3  4                                        :Values of M and N, Example 2
 ( 0.28,-0.36) ( 0.50,-0.86) (-0.77,-0.48) ( 1.58, 0.66)
 (-0.50,-1.10) (-1.21, 0.76) (-0.32,-0.24) (-0.27,-1.15)
 ( 0.36,-0.51) (-0.07, 1.33) (-0.75, 0.47) (-0.08, 1.01)   :End of matrix A
```

9.3 Program Results

```
f08kuc Example Program Results

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

 Example 1: matrix Q
                    1                   2                   3                   4
 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
                    1                   2                   3                   4
 1  (-0.1258, 0.1618)  (-0.2247, 0.3864)  ( 0.3460, 0.2157)  (-0.7099,-0.2966)
 2  ( 0.4148, 0.1795)  ( 0.1368,-0.3976)  ( 0.6885, 0.3386)  ( 0.1667,-0.0494)
 3  ( 0.4575,-0.4807)  (-0.2733, 0.4981)  (-0.0230, 0.3861)  ( 0.1730, 0.2395)
```