NAG C Library Function Document

nag_zungbr (f08ktc)

1 Purpose

nag_zungbr (f08ktc) generates one of the complex unitary matrices Q or P^H which were determined by nag_zgebrd (f08ksc) when reducing a complex matrix to bidiagonal form.

2 Specification

```
void nag_zungbr (Nag_OrderType order, Nag_VectType vect, Integer m, Integer n,
     Integer k, Complex a[], Integer pda, const Complex tau[], NagError *fail)
```

3 Description

nag_zungbr (f08ktc) 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 generate Q or P^H explicitly as square matrices, or in some cases just the leading columns of Q or the leading rows of P^H.

The various possibilities are specified by the parameters **vect**, **m**, **n** and **k**. The appropriate values to cover the most likely cases are as follows (assuming that A was an m by n matrix):

1. To form the full m by m matrix Q:

   ```
   nag_zungbr (order,Nag_FormQ,m,m,n,...)
   ```

   (note that the array **a** must have at least m columns).

2. If $m > n$, to form the n leading columns of Q:

   ```
   nag_zungbr (order,Nag_FormQ,m,n,n,...)
   ```

3. To form the full n by n matrix P^H:

   ```
   nag_zungbr (order,Nag_FormP,n,n,m,...)
   ```

   (note that the array **a** must have at least n rows).

4. If $m < n$, to form the m leading rows of P^H:

   ```
   nag_zungbr (order,Nag_FormP,m,n,m,...)
   ```

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: **vect** – Nag_VectType *Input*

On entry: indicates whether the unitary matrix Q or P^H is generated as follows:

if **vect** = **Nag_FormQ**, Q is generated;

if **vect** = **Nag_FormP**, P^H is generated.

Constraint: **vect** = **Nag_FormQ** or **Nag_FormP**.

3: **m** – Integer *Input*

On entry: the number of rows of the unitary matrix Q or P^H to be returned.

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

4: **n** – Integer *Input*

On entry: the number of columns of the unitary matrix Q or P^H to be returned.

Constraints:

$\mathbf{n} \geq 0$;
if **vect** = **Nag_FormQ** and $\mathbf{m} > \mathbf{k}$, $\mathbf{m} \geq \mathbf{n} \geq \mathbf{k}$;
if **vect** = **Nag_FormQ** and $\mathbf{m} \leq \mathbf{k}$, $\mathbf{m} = \mathbf{n}$;
if **vect** = **Nag_FormP** and $\mathbf{n} > \mathbf{k}$, $\mathbf{n} \geq \mathbf{m} \geq \mathbf{k}$;
if **vect** = **Nag_FormP** and $\mathbf{n} \leq \mathbf{k}$, $\mathbf{n} = \mathbf{m}$.

5: **k** – Integer *Input*

On entry: if **vect** = **Nag_FormQ**, the number of columns in the original matrix A; if **vect** = **Nag_FormP**, the number of rows in the original matrix A.

Constraint: $\mathbf{k} \geq 0$.

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

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

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

On entry: details of the vectors which define the elementary reflectors, as returned by nag_zgebrd (f08ksc).

On exit: the unitary matrix Q or P^H, or the leading rows or columns thereof, as specified by **vect**, **m** and **n**.

7: **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**.

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{m})$.

8: **tau**[dim] – const Complex *Input*

Note: the dimension, dim, of the array **tau** must be at least $(1, \min(\mathbf{m}, \mathbf{k}))$ when **vect** = **Nag_FormQ** and at least $(1, \min(\mathbf{n}, \mathbf{k}))$ when **vect** = **Nag_FormP**.

On entry: further details of the elementary reflectors, as returned by nag_zgebrd (f08ksc) in its parameter **tauq** if **vect** = **Nag_FormQ**, or in its parameter **taup** if **vect** = **Nag_FormP**.

9: **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, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 0.

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

NE_INT_2

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

NE_ENUM_INT_3

On entry, **vect** = ⟨*value*⟩, **m** = ⟨*value*⟩, **n** = ⟨*value*⟩, **k** = ⟨*value*⟩.
Constraint: **n** ≥ 0 and if **vect** = **Nag_FormQ** and **m** > **k**, **m** $\geq$ **n** $\geq$ **k**;
if **vect** = **Nag_FormQ** and **m** $\leq$ **k**, **m** = **n**;
if **vect** = **Nag_FormP** and **n** > **k**, **n** $\geq$ **m** $\geq$ **k**;
if **vect** = **Nag_FormP** and **n** $\leq$ **k**, **n** = **m**.

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 matrix Q differs from an exactly unitary matrix by a matrix E such that

$$\|E\|_2 = O(\epsilon),$$

where ϵ is the ***machine precision***. A similar statement holds for the computed matrix P^H.

8 Further Comments

The total number of real floating-point operations for the cases listed in Section 3 are approximately as follows:

1. To form the whole of Q:

   $\frac{16}{3}n(3m^2 - 3mn + n^2)$ if $m > n$,

   $\frac{16}{3}m^3$ if $m \leq n$;

2. To form the n leading columns of Q when $m > n$:

   $\frac{8}{3}n^2(3m - n)$;

3. To form the whole of P^H:

$\frac{16}{3}n^3$ if $m \geq n$,

$\frac{16}{3}m^3(3n^2 - 3mn + m^2)$ if $m < n$;

4. To form the m leading rows of P^H when $m < n$:

$\frac{8}{3}m^2(3n - m)$.

The real analogue of this function is nag_dorgbr (f08kfc).

9 Example

For this function two examples are presented, both of which involve computing the singular value decomposition of a matrix A, where

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

in the first example and

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

in the second. A must first be reduced to tridiagonal form by nag_zgebrd (f08ksc). The program then calls nag_zungbr (f08ktc) twice to form Q and P^H, and passes these matrices to nag_zbdsqr (f08msc), which computes the singular value decomposition of A.

9.1 Program Text

```
/* nag_zungbr (f08ktc) 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, ic, j, m, n, pda, pdc, pdu, pdvt, d_len;
  Integer  e_len, tauq_len, taup_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex *a=0, *c=0, *taup=0, *tauq=0, *u=0, *vt=0;
  double  *d=0, *e=0;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
#define VT(I,J) vt[(J-1)*pdvt + I - 1]
#define U(I,J) u[(J-1)*pdu + I - 1]
  order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define VT(I,J) vt[(I-1)*pdvt + J - 1]
```

```
#define U(I,J) u[(I-1)*pdu + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("f08ktc Example Program Results\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");

  for (ic = 1; ic <= 2; ++ic)
    {
      Vscanf("%ld%ld%*[^\n] ", &m, &n);
      d_len = n;
#ifdef NAG_COLUMN_MAJOR
      pda = m;
      pdc = n;
      pdu = m;
      pdvt = m;
      e_len = n-1;
      tauq_len = n;
      taup_len = n;
#else
      pda = n;
      pdc = n;
      pdu = n;
      pdvt = n;
      e_len = n-1;
      tauq_len = n;
      taup_len = n;
#endif
      /* Allocate memory */
      if ( !(a = NAG_ALLOC(m * n, Complex)) ||
           !(c = NAG_ALLOC(n * n, Complex)) ||
           !(taup = NAG_ALLOC(taup_len, Complex)) ||
           !(tauq = NAG_ALLOC(tauq_len, Complex)) ||
           !(u = NAG_ALLOC(m * n, Complex)) ||
           !(vt = NAG_ALLOC(m * n, 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] ");
      /* Reduce A to bidiagonal form */
      f08ksc(order, m, n, a, pda, d, e, tauq, taup, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08ksc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      if (m >= n)
        {
          /* Copy A to VT and U */
          for (i = 1; i <= n; ++i)
            {
              for (j = i; j <= n; ++j)
                {
                  VT(i,j).re = A(i,j).re;
                  VT(i,j).im = A(i,j).im;
                }
            }
```

```
      for (i = 1; i <= m; ++i)
        {
          for (j = 1; j <= MIN(i,n); ++j)
            {
              U(i,j).re = A(i,j).re;
              U(i,j).im = A(i,j).im;
            }
        }
      /* Form P**H explicitly, storing the result in VT */
      f08ktc(order, Nag_FormP, n, n, m, vt, pdvt, taup, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08ktc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Form Q explicitly, storing the result in U */
      f08ktc(order, Nag_FormQ, m, n, n, u, pdu, tauq, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08ktc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* Compute the SVD of A */
      f08msc(order, Nag_Upper, n, n, m, 0, d, e, vt, pdvt, u,
             pdu, c, pdc, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08msc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Print singular values, left & right singular vectors */
      Vprintf("\nExample 1: singular values\n");
      for (i = 1; i <= n; ++i)
        Vprintf("%8.4f%s", d[i-1], i%8==0?"\n":" ");
      Vprintf("\n\n");
      x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
             n, n, vt, pdvt, Nag_BracketForm, "%7.4f",
             "Example 1: right singular vectors, by row",
             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;
        }
      Vprintf("\n");
      x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
             m, n, u, pdu, Nag_BracketForm, "%7.4f",
             "Example 1: left singular vectors, by column",
             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
    {
      /* Copy A to VT and U */
      for (i = 1; i <= m; ++i)
        {
          for (j = i; j <= n; ++j)
            {
```

```
              VT(i,j).re = A(i,j).re;
              VT(i,j).im = A(i,j).im;
            }
        }
      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;
            }
        }
      /* Form P**H explicitly, storing the result in VT */
      f08ktc(order, Nag_FormP, m, n, m, vt, pdvt, taup, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08ktc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* Form Q explicitly, storing the result in U */
      f08ktc(order, Nag_FormQ, m, m, n, u, pdu, tauq, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08ktc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* Compute the SVD of A */
      f08msc(order, Nag_Lower, m, n, m, 0, d, e, vt, pdvt, u,
             pdu, c, pdc, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from f08msc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      /* Print singular values, left & right singular vectors */
      Vprintf("\nExample 2: singular values\n");
      for (i = 1; i <= m; ++i)
        Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
      Vprintf("\n\n");
      x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
             m, n, vt, pdvt, Nag_BracketForm, "%7.4f",
             "Example 2: right singular vectors, by row",
             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;
        }
      Vprintf("\n");
      x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
             m, m, u, pdu, Nag_BracketForm, "%7.4f",
             "Example 2: left singular vectors, by column",
             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 (c) NAG_FREE(c);
   if (taup) NAG_FREE(taup);
   if (tauq) NAG_FREE(tauq);
```

```
      if (u) NAG_FREE(u);
      if (vt) NAG_FREE(vt);
      if (d) NAG_FREE(d);
      if (e) NAG_FREE(e);
    }
  return exit_status;
}
```

9.2 Program Data

```
f08ktc 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

```
f08ktc Example Program Results

Example 1: singular values
  3.9994   3.0003   1.9944   0.9995

 Example 1: right singular vectors, by row
                   1                  2                  3                  4
 1  (-0.6971,-0.0000)  (-0.0867,-0.3548)  ( 0.0560,-0.5400)  (-0.1878,-0.2253)
 2  ( 0.2403, 0.0000)  ( 0.0725,-0.2336)  (-0.2477,-0.5291)  ( 0.7026, 0.2177)
 3  (-0.5123, 0.0000)  (-0.3030,-0.1735)  ( 0.0678, 0.5162)  ( 0.4418, 0.3864)
 4  (-0.4403, 0.0000)  ( 0.5294, 0.6361)  (-0.3027,-0.0346)  ( 0.1667, 0.0258)

 Example 1: left singular vectors, by column
                   1                  2                  3                  4
 1  (-0.5634, 0.0016)  (-0.2687,-0.2749)  ( 0.2451, 0.4657)  ( 0.3787, 0.2987)
 2  ( 0.1205,-0.6108)  (-0.2909, 0.1085)  ( 0.4329,-0.1758)  (-0.0182,-0.0437)
 3  (-0.0816, 0.1613)  (-0.1660, 0.3885)  (-0.4667, 0.3821)  (-0.0800,-0.2276)
 4  ( 0.1441,-0.1532)  ( 0.1984,-0.1737)  (-0.0034, 0.1555)  ( 0.2608,-0.5382)
 5  (-0.2487,-0.0926)  ( 0.6253, 0.3304)  ( 0.2643,-0.0194)  ( 0.1002, 0.0140)
 6  (-0.3758, 0.0793)  (-0.0307,-0.0816)  ( 0.1266, 0.1747)  (-0.4175,-0.4058)

Example 2: singular values
  3.0004   1.9967   0.9973

 Example 2: right singular vectors, by row
                   1                  2                  3                  4
 1  ( 0.2454,-0.0001)  ( 0.2942,-0.5843)  ( 0.0162,-0.0810)  ( 0.6794, 0.2083)
 2  (-0.1692, 0.5194)  ( 0.1915,-0.4374)  ( 0.5205,-0.0244)  (-0.3149,-0.3208)
 3  (-0.5553, 0.1403)  ( 0.1438,-0.1507)  (-0.5684,-0.5505)  (-0.0318,-0.0378)

 Example 2: left singular vectors, by column
                   1                  2                  3
 1  ( 0.6518, 0.0000)  (-0.4312, 0.0000)  ( 0.6239, 0.0000)
 2  (-0.4437,-0.5027)  (-0.3794, 0.1026)  ( 0.2014, 0.5961)
 3  (-0.2012, 0.2916)  (-0.8122, 0.0030)  (-0.3511,-0.3026)
```