NAG C Library Function Document

nag_zgbbrd (f08lsc)

1 Purpose

nag_zgbbrd (f08lsc) reduces a complex m by n band matrix to real upper bidiagonal form.

2 Specification

```
void nag_zgbbrd (Nag_OrderType order, Nag_VectType vect, Integer m, Integer n,
      Integer ncc, Integer kl, Integer ku, Complex ab[], Integer pdab, double d[],
      double e[], Complex q[], Integer pdq, Complex pt[], Integer pdpt, Complex c[],
      Integer pdc, NagError *fail)
```

3 Description

nag_zgbbrd (f08lsc) reduces a complex m by n band matrix to real upper bidiagonal form B by a unitary transformation: $A = QBP^H$. The unitary matrices Q and P^H, of order m and n respectively, are determined as a product of Givens rotation matrices, and may be formed explicitly by the function if required. A matrix C may also be updated to give $\tilde{C} = Q^H C$.

The function uses a vectorisable form of the reduction.

4 References

None.

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 matrices Q and/or P^H are generated:

if **vect** = **Nag_DoNotForm**, then neither Q nor P^H is generated;

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

if **vect** = **Nag_FormP**, then P^H is generated;

if **vect** = **Nag_FormBoth**, then both Q and P^H are generated.

Constraint: **vect** = **Nag_DoNotForm**, **Nag_FormQ**, **Nag_FormP** or **Nag_FormBoth**.

3: **m** – Integer *Input*

On entry: m, the number of rows of the matrix A.

Constraint: **m** ≥ 0.

4: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

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

5: **ncc** – Integer *Input*

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

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

6: **kl** – Integer *Input*

On entry: k_l, the number of sub-diagonals within the band of A.

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

7: **ku** – Integer *Input*

On entry: k_u, the number of super-diagonals within the band of A.

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

8: **ab**$[dim]$ – Complex *Input/Output*

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

On entry: the original m by n band matrix A. This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements a_{ij}, for $i = 1, \ldots, m$ and $j = \max(1, i - k_l), \ldots, \min(n, i + k_u)$, depends on the **order** parameter as follows:

if **order** $=$ **Nag_ColMajor**, a_{ij} is stored as $\mathbf{ab}[(j-1) \times \mathbf{pdab} + \mathbf{ku} + i - j]$;

if **order** $=$ **Nag_RowMajor**, a_{ij} is stored as $\mathbf{ab}[(i-1) \times \mathbf{pdab} + \mathbf{kl} + j - i]$.

On exit: A is overwritten by values generated during the reduction.

9: **pdab** – Integer *Input*

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

Constraint: $\mathbf{pdab} \geq \mathbf{kl} + \mathbf{ku} + 1$.

10: **d**$[dim]$ – double *Output*

Note: the dimension, dim, of the array **d** must be at least $\max(1, \min(\mathbf{m}, \mathbf{n}))$.

On exit: the diagonal elements of the bidiagonal matrix B.

11: **e**$[dim]$ – double *Output*

Note: the dimension, dim, of the array **e** must be at least $\max(1, \min(\mathbf{m}, \mathbf{n}) - 1)$.

On exit: the super-diagonal elements of the bidiagonal matrix B.

12: **q**$[dim]$ – Complex *Output*

Note: the dimension, dim, of the array **q** must be at least $\max(1, \mathbf{pdq} \times \mathbf{m})$ when **vect** $=$ **Nag_FormQ** or **Nag_FormBoth** and at least 1 otherwise.

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

On exit: the m by m unitary matrix Q, if **vect** $=$ **Nag_FormQ** or **Nag_FormBoth**.

q is not referenced if **vect** = **Nag_DoNotForm** or **Nag_FormP**.

13: **pdq** – Integer *Input*

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

Constraints:

if **vect** = **Nag_FormQ** or **Nag_FormBoth**, **pdq** $\geq \max(1, \mathbf{m})$;
otherwise **pdq** ≥ 1.

14: **pt**[dim] – Complex *Output*

Note: the dimension, dim, of the array **pt** must be at least $\max(1, \mathbf{pdpt} \times \mathbf{n})$ when **vect** = **Nag_FormP** or **Nag_FormBoth** and at least 1 otherwise.

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

On exit: the n by n unitary matrix P^H, if **vect** = **Nag_FormP** or **Nag_FormBoth**.

pt is not referenced if **vect** = **Nag_DoNotForm** or **Nag_FormQ**.

15: **pdpt** – Integer *Input*

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

Constraints:

if **vect** = **Nag_FormP** or **Nag_FormBoth**, **pdpt** $\geq \max(1, \mathbf{n})$;
otherwise **pdpt** ≥ 1.

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

Note: the dimension, dim, of the array **c** must be at least $\max(1, \mathbf{pdc} \times \mathbf{ncc})$ 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 **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 **c**$[(i-1) \times \mathbf{pdc} + j - 1]$.

On entry: an m by n_C matrix C.

On exit: C is overwritten by $Q^H C$.

c is not referenced if **ncc** = 0.

17: **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**,
  if **ncc** > 0, **pdc** $\geq \max(1, \mathbf{m})$;
  if **ncc** $= 0$, **pdc** ≥ 1;

if **order** = **Nag_RowMajor**, **pdc** $\geq \max(1, \mathbf{ncc})$.

18: **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, **ncc** = $\langle value\rangle$.
Constraint: **ncc** ≥ 0.

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

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

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

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

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

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

NE_INT_2

On entry, **pdq** $= \langle value\rangle$, **m** $= \langle value\rangle$.
Constraint: if **vect** = **Nag_FormQ** or **Nag_FormBoth**, **pdq** $\geq \max(1, \mathbf{m})$;
otherwise **pdq** ≥ 1.

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

NE_INT_3

On entry, **kl** $= \langle value\rangle$, **ku** $= \langle value\rangle$, **pdab** $= \langle value\rangle$.
Constraint: **pdab** $\geq$ **kl** + **ku** + 1.

On entry, **m** $= \langle value\rangle$, **ncc** $= \langle value\rangle$, **pdc** $= \langle value\rangle$.
Constraint: if **ncc** > 0, **pdc** $\geq \max(1, \mathbf{m})$;
if **ncc** $= 0$, **pdc** ≥ 1.

NE_ENUM_INT_2

On entry, **vect** $= \langle value\rangle$, **m** $= \langle value\rangle$, **pdq** $= \langle value\rangle$.
Constraint: if **vect** = **Nag_FormQ** or **Nag_FormBoth**, **pdq** $\geq \max(1, \mathbf{m})$;
otherwise **pdq** ≥ 1.

On entry, **vect** $= \langle value\rangle$, **n** $= \langle value\rangle$, **pdpt** $= \langle value\rangle$.
Constraint: if **vect** = **Nag_FormP** or **Nag_FormBoth**, **pdpt** $\geq \max(1, \mathbf{n})$;
otherwise **pdpt** ≥ 1.

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 bidiagonal form B satisfies $QBP^H = A + E$, where

$$\|E\|_2 \le c(n)\epsilon\|A\|_2,$$

$c(n)$ is a modestly increasing function of n, and ϵ is the ***machine precision***.

The elements of B themselves may be sensitive to small perturbations in A or to rounding errors in the computation, but this does not affect the stability of the singular values and vectors.

The computed matrix Q differs from an exactly unitary matrix by a matrix F such that

$$\|F\|_2 = O(\epsilon).$$

A similar statement holds for the computed matrix P^H.

8 Further Comments

The total number of real floating-point operations is approximately the sum of:

$20n^2k$, if **vect** = **Nag_DoNotForm** and **ncc** $= 0$, and

$10n^2n_C(k-1)/k$, if C is updated, and

$10n^3(k-1)/k$ if either Q or P^H is generated (double this if both),

where $k = k_l + k_u$, assuming $n \gg k$. For this section we assume that $m = n$.

The real analogue of this function is nag_dgbbrd (f08lec).

9 Example

To reduce the matrix A to upper bidiagonal form, where

$$A = \begin{pmatrix} 0.96-0.81i & -0.03+0.96i & 0.00+0.00i & 0.00+0.00i \\ -0.98+1.98i & -1.20+0.19i & -0.66+0.42i & 0.00+0.00i \\ 0.62-0.46i & 1.01+0.02i & 0.63-0.17i & -1.11+0.60i \\ 0.00+0.00i & 0.19-0.54i & -0.98-0.36i & 0.22-0.20i \\ 0.00+0.00i & 0.00+0.00i & -0.17-0.46i & 1.47+1.59i \\ 0.00+0.00i & 0.00+0.00i & 0.00+0.00i & 0.26+0.26i \end{pmatrix}.$$

9.1 Program Text

```
/* nag_zgbbrd (f08lsc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer  i, j, kl, ku, m, n, ncc, pdab, pdc, pdq, pdpt;
  Integer  d_len, e_len;
  Integer  exit_status=0;
```

```
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex *ab=0, *c=0, *pt=0, *q=0;
  double  *d=0, *e=0;

#ifdef NAG_COLUMN_MAJOR
#define AB(I,J) ab[(J-1)*pdab + ku + I - J]
  order = Nag_ColMajor;
#else
#define AB(I,J) ab[(I-1)*pdab + kl + J - I]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("f08lsc Example Program Results\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%ld%ld%ld%ld%*[^\n] ", &m, &n, &kl, &ku, &ncc);
#ifdef NAG_COLUMN_MAJOR
  pdab = kl + ku + 1;
  pdq = m;
  pdpt = n;
  pdc = m;
#else
  pdab = kl + ku + 1;
  pdq = m;
  pdpt = n;
  pdc = MAX(1,ncc);
#endif
  d_len = MIN(m,n);
  e_len = MIN(m,n)-1;

  /* Allocate memory */
  if ( !(ab = NAG_ALLOC((kl+ku+1) * m, Complex)) ||
       !(c = NAG_ALLOC(m * MAX(1,ncc), Complex)) ||
       !(d = NAG_ALLOC(d_len, double)) ||
       !(e = NAG_ALLOC(e_len, double)) ||
       !(pt = NAG_ALLOC(n * n, Complex)) ||
       !(q = NAG_ALLOC(m * m, Complex)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A from data file */
  for (i = 1; i <= m; ++i)
    {
      for (j = MAX(1,i-kl); j <= MIN(n,i+ku); ++j)
        Vscanf(" ( %lf , %lf )", &AB(i,j).re, &AB(i,j).im);
    }
  Vscanf("%*[^\n] ");
  /* Reduce A to bidiagonal form */
  f08lsc(order, Nag_DoNotForm, m, n, ncc, kl, ku, ab,
         pdab, d, e, q, pdq, pt, pdpt, c, pdc, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08lsc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print bidiagonal form */
  Vprintf("\nDiagonal\n");
  for (i = 1; i <= MIN(m,n); ++i)
    Vprintf("%9.4f%s", d[i-1], i%8==0 ?"\n":" ");
  if (m >= n)
    Vprintf("\nSuper-diagonal\n");
  else
```

```
    Vprintf("\nSub-diagonal\n");
  for (i = 1; i <= MIN(m,n) - 1; ++i)
    Vprintf("%9.4f%s", e[i-1], i%8==0 ?"\n":" ");
  Vprintf("\n");

 END:
  if (ab) NAG_FREE(ab);
  if (c) NAG_FREE(c);
  if (d) NAG_FREE(d);
  if (e) NAG_FREE(e);
  if (pt) NAG_FREE(pt);
  if (q) NAG_FREE(q);

  return exit_status;
}
```

9.2 Program Data

```
f08lsc Example Program Data
  6  4  2  1  0                                 :Values of M, N, KL, KU and NCC
 ( 0.96,-0.81) (-0.03, 0.96)
 (-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42)
 ( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17) (-1.11, 0.60)
               ( 0.19,-0.54) (-0.98,-0.36) ( 0.22,-0.20)
                             (-0.17,-0.46) ( 1.47, 1.59)
                                           ( 0.26, 0.26)   :End of matrix A
```

9.3 Program Results

```
f08lsc Example Program Results

Diagonal
   2.6560    1.7501    2.0607    0.8658
Super-diagonal
   1.7033    1.2800    0.1467
```