NAG Library Function Document

nag_zgebrd (f08ksc)

1 Purpose

nag_zgebrd (f08ksc) reduces a complex m by n matrix to bidiagonal form.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zgebrd (Nag_OrderType order, Integer m, Integer n, Complex a[],
     Integer pda, double d[], double e[], Complex tauq[], Complex taup[],
     NagError *fail)
```

3 Description

nag_zgebrd (f08ksc) reduces a complex m by n matrix A to real bidiagonal form B by a unitary transformation: $A = QBP^{\mathrm{H}}$, where Q and P^{H} are unitary matrices of order m and n respectively.

If $m \geq n$, the reduction is given by:

$$A = Q \begin{pmatrix} B_1 \\ 0 \end{pmatrix} P^{\mathrm{H}} = Q_1 B_1 P^{\mathrm{H}},$$

where B_1 is a real n by n upper bidiagonal matrix and Q_1 consists of the first n columns of Q.

If $m < n$, the reduction is given by

$$A = Q \begin{pmatrix} B_1 & 0 \end{pmatrix} P^{\mathrm{H}} = Q B_1 P_1^{\mathrm{H}},$$

where B_1 is a real m by m lower bidiagonal matrix and P_1^{H} consists of the first m rows of P^{H}.

The unitary matrices Q and P are not formed explicitly but are represented as products of elementary reflectors (see the f08 Chapter Introduction for details). Functions are provided to work with Q and P in this representation (see Section 9).

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 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: **m** – Integer *Input*

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

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

3: **n** – Integer *Input*

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

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

4: **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;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix A is stored in

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the m by n matrix A.

On exit: if $m \geq n$, the diagonal and first superdiagonal are overwritten by the upper bidiagonal matrix B, elements below the diagonal are overwritten by details of the unitary matrix Q and elements above the first superdiagonal are overwritten by details of the unitary matrix P.

If $m < n$, the diagonal and first subdiagonal are overwritten by the lower bidiagonal matrix B, elements below the first subdiagonal are overwritten by details of the unitary matrix Q and elements above the diagonal are overwritten by details of the unitary matrix P.

5: **pda** – Integer *Input*

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

Constraints:

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

6: **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.

7: **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 off-diagonal elements of the bidiagonal matrix B.

8: **tauq**[dim] – Complex *Output*

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

On exit: further details of the unitary matrix Q.

9: **taup**[dim] – Complex *Output*

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

On exit: further details of the unitary matrix P.

10: **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 $\langle value \rangle$ had an illegal value.

NE_INT

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

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

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

NE_INT_2

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

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

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 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 bidiagonal form B satisfies $QBP^{\mathrm{H}} = A + E$, where

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

$c(n)$ 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.

8 Parallelism and Performance

nag_zgebrd (f08ksc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgebrd (f08ksc) 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 $16n^2(3m - n)/3$ if $m \geq n$ or $16m^2(3n - m)/3$ if $m < n$.

If $m \gg n$, it can be more efficient to first call nag_zgeqrf (f08asc) to perform a QR factorization of A, and then to call nag_zgebrd (f08ksc) to reduce the factor R to bidiagonal form. This requires approximately $8n^2(m + n)$ floating-point operations.

If $m \ll n$, it can be more efficient to first call nag_zgelqf (f08avc) to perform an LQ factorization of A, and then to call nag_zgebrd (f08ksc) to reduce the factor L to bidiagonal form. This requires approximately $8m^2(m + n)$ operations.

To form the unitary matrices $P^{\rm H}$ and/or Q nag_zgebrd (f08ksc) may be followed by calls to nag_zungbr (f08ktc):

to form the m by m unitary matrix Q

```
nag_zungbr(order,Nag_FormQ,m,m,n,&a,pda,tauq,&fail)
```

but note that the second dimension of the array **a** must be at least **m**, which may be larger than was required by nag_zgebrd (f08ksc);

to form the n by n unitary matrix $P^{\rm H}$

```
nag_zungbr(order,Nag_FormP,n,n,m,&a,pda,taup,&fail)
```

but note that the first dimension of the array **a**, specified by the argument **pda**, must be at least **n**, which may be larger than was required by nag_zgebrd (f08ksc).

To apply Q or P to a complex rectangular matrix C, nag_zgebrd (f08ksc) may be followed by a call to nag_zunmbr (f08kuc).

The real analogue of this function is nag_dgebrd (f08kec).

10 Example

This example reduces the matrix A to bidiagonal form, where

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

10.1 Program Text

```
/* nag_zgebrd (f08ksc) Example Program.
 *
 * Copyright 2014 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, m, n, pda, d_len, e_len, tauq_len, taup_len;
  Integer       exit_status = 0;
  NagError      fail;
  Nag_OrderType order;
  /* Arrays */
  Complex       *a = 0, *taup = 0, *tauq = 0;
  double        *d = 0, *e = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J - 1) * pda + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I - 1) * pda + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zgebrd (f08ksc) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &m, &n);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &m, &n);
#endif
#ifdef NAG_COLUMN_MAJOR
  pda = m;
#else
  pda = n;
#endif
  d_len = MIN(m, n);
  e_len = MIN(m, n)-1;
  tauq_len = MIN(m, n);
  taup_len = MIN(m, n);

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m * n, Complex)) ||
      !(d = NAG_ALLOC(d_len, double)) ||
      !(e = NAG_ALLOC(e_len, double)) ||
      !(taup = NAG_ALLOC(taup_len, Complex)) ||
      !(tauq = NAG_ALLOC(tauq_len, Complex)))
    {
      printf("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)
#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 bidiagonal form */
  /* nag_zgebrd (f08ksc).
```

```
   * Unitary reduction of complex general rectangular matrix
   * to bidiagonal form
   */
  nag_zgebrd(order, m, n, a, pda, d, e, tauq, taup, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgebrd (f08ksc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

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

 END:
  NAG_FREE(a);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(taup);
  NAG_FREE(tauq);

  return exit_status;
}
```

10.2 Program Data

```
nag_zgebrd (f08ksc) Example Program Data
  6  4                                                    :Values of M and N
 ( 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
```

10.3 Program Results

```
nag_zgebrd (f08ksc) Example Program Results

Diagonal
  -3.0870    2.0660    1.8731    2.0022
Super-diagonal
   2.1126    1.2628   -1.6126
```