NAG Library Function Document

nag_dgbbrd (f08lec)

1 Purpose

nag_dgbbrd (f08lec) reduces a real m by n band matrix to upper bidiagonal form.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_dgbbrd (Nag_OrderType order, Nag_VectType vect, Integer m, Integer n,
     Integer ncc, Integer kl, Integer ku, double ab[], Integer pdab, double d[],
     double e[], double q[], Integer pdq, double pt[], Integer pdpt, double c[],
     Integer pdc, NagError *fail)
```

3 Description

nag_dgbbrd (f08lec) reduces a real m by n band matrix to upper bidiagonal form B by an orthogonal transformation: $A = QBP^{\mathrm{T}}$. The orthogonal matrices Q and P^{T}, 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 $\tilde{C} = Q^{\mathrm{T}}C$.

The function uses a vectorizable form of the reduction.

4 References

None.

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: **vect** – Nag_VectType *Input*

On entry: indicates whether the matrices Q and/or P^{T} are generated.

vect = Nag_DoNotForm
Neither Q nor P^{T} is generated.

vect = Nag_FormQ
Q is generated.

vect = Nag_FormP
P^{T} is generated.

vect = Nag_FormBoth
Both Q and P^{T} 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: $\mathbf{m} \geq 0$.

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: the number of subdiagonals, k_l, within the band of A.

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

7: **ku** – Integer *Input*

On entry: the number of superdiagonals, k_u, within the band of A.

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

8: **ab**[*dim*] – double *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;
$\max(1, \mathbf{m} \times \mathbf{pdab})$ 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 row $i = 1, \ldots, m$ and column $j = \max(1, i - k_l), \ldots, \min(n, i + k_u)$, depends on the **order** argument 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: **ab** 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**[$\min(\mathbf{m}, \mathbf{n})$] – double *Output*

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

11: **e**[$\min(\mathbf{m}, \mathbf{n}) - 1$] – double *Output*

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

12: **q**[*dim*] – double *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;
1 otherwise.

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

q$[(j-1) \times$ **pdq** $+ i - 1]$ when **order** = Nag_ColMajor;
q$[(i-1) \times$ **pdq** $+ j - 1]$ when **order** = Nag_RowMajor.

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

If **vect** = Nag_DoNotForm or Nag_FormP, **q** is not referenced.

13: **pdq** – Integer *Input*

On entry: the stride separating 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*] – double *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;
1 otherwise.

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

pt$[(j-1) \times$ **pdpt** $+ i - 1]$ when **order** = Nag_ColMajor;
pt$[(i-1) \times$ **pdpt** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the n by n orthogonal matrix P^{T}, if **vect** = Nag_FormP or Nag_FormBoth. If **vect** = Nag_DoNotForm or Nag_FormQ, **pt** is not referenced.

15: **pdpt** – Integer *Input*

On entry: the stride separating 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*] – double *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;
$\max(1, \mathbf{m} \times \mathbf{pdc})$ when **order** = Nag_RowMajor.

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

c$[(j-1) \times$ **pdc** $+ i - 1]$ when **order** = Nag_ColMajor;
c$[(i-1) \times$ **pdc** $+ j - 1]$ when **order** = Nag_RowMajor.

On entry: an m by n_C matrix C.

On exit: **c** is overwritten by $Q^{\mathrm{T}}C$. If **ncc** = 0, **c** is not referenced.

17: **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,
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 * *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.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_2

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

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

NE_INT

On entry, **kl** = ⟨*value*⟩.
Constraint: **kl** ≥ 0.

On entry, **ku** = ⟨*value*⟩.
Constraint: **ku** ≥ 0.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **ncc** = ⟨*value*⟩.
Constraint: **ncc** ≥ 0.

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

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

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

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

NE_INT_2

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

NE_INT_3

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

On entry, **pdab** = ⟨*value*⟩, **kl** = ⟨*value*⟩ and **ku** = ⟨*value*⟩.
Constraint: **pdab** $\geq$ **kl** + **ku** + 1.

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.

7 Accuracy

The computed bidiagonal form B satisfies $QBP^{\mathrm{T}} = A + E$, 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.

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

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

A similar statement holds for the computed matrix P^{T}.

8 Further Comments

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

$6n^2k$, if **vect** = Nag_DoNotForm and **ncc** $= 0$, and

$3n^2n_C(k-1)/k$, if C is updated, and

$3n^3(k-1)/k$, if either Q or P^{T} 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 complex analogue of this function is nag_zgbbrd (f08lsc).

9 Example

This example reduces the matrix A to upper bidiagonal form, where

$$A = \begin{pmatrix} -0.57 & -1.28 & 0.00 & 0.00 \\ -1.93 & 1.08 & -0.31 & 0.00 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ 0.00 & 0.64 & -0.66 & 0.08 \\ 0.00 & 0.00 & 0.15 & -2.13 \\ -0.00 & 0.00 & 0.00 & 0.50 \end{pmatrix}.$$

9.1 Program Text

```
/* nag_dgbbrd (f08lec) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
```

```
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double        alpha, beta, norm;
  Integer       i, j, kl, ku, m, n, ncc, pdab, pdaw, pdc, pdf, pdq;
  Integer       pdpt, d_len, e_len;
  Integer       exit_status = 0;
  NagError      fail;
  Nag_OrderType order;
  /* Arrays */
  double        *ab = 0, *aw = 0, *c = 0, *d = 0, *e = 0, *f = 0, *pt = 0, *q =
0;

#ifdef NAG_COLUMN_MAJOR
#define AB(I, J) ab[(J - 1) * pdab + ku + I - J]
#define AW(I, J) aw[(J - 1) * pdaw + I - 1]
#define F(I, J) f[(J - 1) * pdf + I - 1]
#define Q(I, J) q[(J - 1) * pdq + I - 1]
  order = Nag_ColMajor;
#else
#define AB(I, J) ab[(I - 1) * pdab + kl + J - I]
#define AW(I, J) aw[(I - 1) * pdaw + J - 1]
#define F(I, J) f[(I - 1) * pdf + J - 1]
#define Q(I, J) q[(I - 1) * pdq + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dgbbrd (f08lec) Example Program Results\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%ld%ld%*[^\n] ",
        &m, &n, &kl, &ku, &ncc);
#ifdef NAG_COLUMN_MAJOR
  pdab = kl + ku + 1;
  pdaw = m;
  pdf = m;
  pdq = m;
  pdpt = n;
  pdc = m;
#else
  pdab = kl + ku + 1;
  pdaw = n;
  pdf = n;
  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, double)) ||
      !(aw = NAG_ALLOC(m * n, double)) ||
      !(f = NAG_ALLOC(m * n, double)) ||
      !(c = NAG_ALLOC(m * MAX(1, ncc), double)) ||
      !(d = NAG_ALLOC(d_len, double)) ||
      !(e = NAG_ALLOC(e_len, double)) ||
      !(pt = NAG_ALLOC(n * n, double)) ||
      !(q = NAG_ALLOC(m * m, double)))
    {
```

```
      printf("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)
        scanf("%lf", &AB(i, j));
    }
  scanf("%*[^\n] ");

    /* Copy AB into AW */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
        {
          if(j >= MAX(1, i - kl) && j<= MIN(n, i + ku))
            AW(i, j) = AB(i, j);
          else
            AW(i, j) = 0;
        }
    }

  /* nag_gen_real_mat_print (x04cac): Print Matrix A. */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n,
                         aw, pdaw, "Matrix A", 0, &fail);
  printf("\n");
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  /* Reduce A to bidiagonal form */
  /* nag_dgbbrd (f08lec).
   * Reduction of real rectangular band matrix to upper
   * bidiagonal form
   */
  nag_dgbbrd(order, Nag_FormBoth, m, n, ncc, kl, ku, ab,
             pdab, d, e, q, pdq, pt, pdpt, c, pdc, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dgbbrd (f08lec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* F = Q*B */
  for(i = 1; i <= m; i++)
    {
      F(i, 1) = Q(i, 1) * d[0];
      for(j = 2; j <= n; j++)
        {
          F(i, j) = (Q(i, j) * d[j-1]) + (Q(i, j-1) * e[j-2]);
        }
    }

  /* nag_dgemm (f16yac): Compute A - Q*B*P^T from the factorization of A */
  /* and store in matrix AW */
  alpha = -1.0;
  beta = 1.0;
  nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, m, n, n, alpha, f, pdf,
            pt, pdpt, beta, aw, pdaw, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dgemm (f16yac).\n%s\n",
```

```
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_dge_norm (f16rac): Find norm of matrix AW and print warning if */
  /* it is too large*/
  nag_dge_norm(order, Nag_OneNorm, m, n, aw, pdaw, &norm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dge_norm (f16rac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  if (norm > pow(x02ajc(),0.8))
    {
      printf("%s\n%s\n","Norm of A-(Q*B*P^T) is much greater than 0.",
              "Schur factorization has failed.");
    }

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

  return exit_status;
}
```

9.2 Program Data

```
nag_dgbbrd (f08lec) Example Program Data
  6  4  2  1  0              :Values of M, N, KL, KU and NCC
 -0.57  -1.28
 -1.93   1.08  -0.31
  2.30   0.24   0.40  -0.35
         0.64  -0.66   0.08
                0.15  -2.13
                       0.50   :End of matrix A
```

9.3 Program Results

```
nag_dgbbrd (f08lec) Example Program Results
 Matrix A
             1          2          3          4
 1     -0.5700    -1.2800     0.0000     0.0000
 2     -1.9300     1.0800    -0.3100     0.0000
 3      2.3000     0.2400     0.4000    -0.3500
 4      0.0000     0.6400    -0.6600     0.0800
 5      0.0000     0.0000     0.1500    -2.1300
 6      0.0000     0.0000     0.0000     0.5000
```