NAG Library Function Document

nag_dormbr (f08kgc)

1 Purpose

nag_dormbr (f08kgc) multiplies an arbitrary real m by n matrix C by one of the real orthogonal matrices Q or P which were determined by nag_dgebrd (f08kec) when reducing a real matrix to bidiagonal form.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_dormbr (Nag_OrderType order, Nag_VectType vect, Nag_SideType side,
     Nag_TransType trans, Integer m, Integer n, Integer k, const double a[],
     Integer pda, const double tau[], double c[], Integer pdc,
     NagError *fail)
```

3 Description

nag_dormbr (f08kgc) is intended to be used after a call to nag_dgebrd (f08kec), which reduces a real rectangular matrix A to bidiagonal form B by an orthogonal transformation: $A = QBP^{\mathrm{T}}$. nag_dgebrd (f08kec) represents the matrices Q and P^{T} as products of elementary reflectors.

This function may be used to form one of the matrix products

$$QC, Q^{\mathrm{T}}C, CQ, CQ^{\mathrm{T}}, PC, P^{\mathrm{T}}C, CP \text{ or } CP^{\mathrm{T}},$$

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

4 References

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

5 Arguments

Note: in the descriptions below, r denotes the order of Q or P^{T}: if **side** = Nag_LeftSide, $r =$ **m** and if **side** = Nag_RightSide, $r =$ **n**.

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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 Q or Q^{T} or P or P^{T} is to be applied to C.

vect = Nag_ApplyQ
Q or Q^{T} is applied to C.

vect = Nag_ApplyP
P or P^{T} is applied to C.

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

3: **side** – Nag_SideType *Input*

On entry: indicates how Q or Q^{T} or P or P^{T} is to be applied to C.

side = Nag_LeftSide
Q or Q^{T} or P or P^{T} is applied to C from the left.

side = Nag_RightSide
Q or Q^{T} or P or P^{T} 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^{T} or P^{T} is to be applied to C.

trans = Nag_NoTrans
Q or P is applied to C.

trans = Nag_Trans
Q^{T} or P^{T} is applied to C.

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

5: **m** – Integer *Input*

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

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

6: **n** – Integer *Input*

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

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

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

8: **a**$[dim]$ – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least

$\max(1, \mathbf{pda} \times \min(r, \mathbf{k}))$ when **vect** = Nag_ApplyQ and **order** = Nag_ColMajor;
$\max(1, r \times \mathbf{pda})$ when **vect** = Nag_ApplyQ and **order** = Nag_RowMajor;
$\max(1, \mathbf{pda} \times r)$ when **vect** = Nag_ApplyP and **order** = Nag_ColMajor;
$\max(1, \min(r, \mathbf{k}) \times \mathbf{pda})$ when **vect** = Nag_ApplyP and **order** = Nag_RowMajor.

On entry: details of the vectors which define the elementary reflectors, as returned by nag_dgebrd (f08kec).

9: **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,

if **vect** = Nag_ApplyQ, **pda** $\geq \max(1, r)$;
if **vect** = Nag_ApplyP, **pda** $\geq \max(1, \min(r, \mathbf{k}))$.;
if **order** = Nag_RowMajor,

if **vect** = Nag_ApplyQ, **pda** $\geq \max(1, \min(r, \mathbf{k}))$;
if **vect** = Nag_ApplyP, **pda** $\geq \max(1, r)$..

10: **tau**[dim] – const double *Input*

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

On entry: further details of the elementary reflectors, as returned by nag_dgebrd (f08kec) in its argument **tauq** if **vect** = Nag_ApplyQ, or in its argument **taup** if **vect** = Nag_ApplyP.

11: **c**[dim] – double *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;
$\max(1, \mathbf{m} \times \mathbf{pdc})$ when **order** = Nag_RowMajor.

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

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

On entry: the matrix C.

On exit: **c** is overwritten by QC or $Q^{\mathrm{T}}C$ or CQ or $C^{\mathrm{T}}Q$ or PC or $P^{\mathrm{T}}C$ or CP or $C^{\mathrm{T}}P$ as specified by **vect**, **side** and **trans**.

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

13: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_ENUM_INT_2

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

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

NE_INT

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

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

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

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

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

NE_INT_2

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

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

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

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 Parallelism and Performance

nag_dormbr (f08kgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dormbr (f08kgc) 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 floating-point operations is approximately

if **side** = Nag_LeftSide and $m \geq k$, $2nk(2m-k)$;

if **side** = Nag_RightSide and $n \geq k$, $2mk(2n-k)$;

if **side** = Nag_LeftSide and $m < k$, $2m^2n$;

if **side** = Nag_RightSide and $n < k$, $2mn^2$,

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

The complex analogue of this function is nag_zunmbr (f08kuc).

10 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

$$A = \begin{pmatrix} -0.57 & -1.28 & -0.39 & 0.25 \\ -1.93 & 1.08 & -0.31 & -2.14 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ -1.93 & 0.64 & -0.66 & 0.08 \\ 0.15 & 0.30 & 0.15 & -2.13 \\ -0.02 & 1.03 & -1.43 & 0.50 \end{pmatrix}.$$

The function first performs a QR factorization of A as $A = Q_a R$ and then reduces the factor R to bidiagonal form B: $R = Q_b B P^{\mathrm{T}}$. Finally it forms Q_a and calls nag_dormbr (f08kgc) to form $Q = Q_a Q_b$.

In the second example, $m < n$, and

$$A = \begin{pmatrix} -5.42 & 3.28 & -3.68 & 0.27 & 2.06 & 0.46 \\ -1.65 & -3.40 & -3.20 & -1.03 & -4.06 & -0.01 \\ -0.37 & 2.35 & 1.90 & 4.31 & -1.76 & 1.13 \\ -3.15 & -0.11 & 1.99 & -2.70 & 0.26 & 4.50 \end{pmatrix}.$$

The function first performs an LQ factorization of A as $A = L P_a^{\mathrm{T}}$ and then reduces the factor L to bidiagonal form B: $L = Q B P_b^{\mathrm{T}}$. Finally it forms P_b^{T} and calls nag_dormbr (f08kgc) to form $P^{\mathrm{T}} = P_b^{\mathrm{T}} P_a^{\mathrm{T}}$.

10.1 Program Text

```
/* nag_dormbr (f08kgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

```
int main(void)
{
  /* Scalars */
  Integer i, ic, j, m, n, pda, pdpt, pdu;
  Integer d_len, e_len, tau_len, tauq_len, taup_len;
  Integer exit_status = 0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double *a = 0, *d = 0, *e = 0, *pt = 0, *tau = 0, *taup = 0, *tauq = 0;
  double *u = 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 PT(I, J) pt[(J - 1) * pdpt + 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 PT(I, J) pt[(I - 1) * pdpt + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dormbr (f08kgc) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  for (ic = 1; ic <= 2; ++ic) {
#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
    pdu = m;
    pdpt = n;
    taup_len = n;
    tauq_len = n;
    tau_len = n;
    d_len = n;
    e_len = n - 1;

    /* Allocate memory */
    if (!(a = NAG_ALLOC(m * n, double)) ||
        !(d = NAG_ALLOC(d_len, double)) ||
        !(e = NAG_ALLOC(e_len, double)) ||
        !(pt = NAG_ALLOC(n * n, double)) ||
        !(tau = NAG_ALLOC(tau_len, double)) ||
        !(taup = NAG_ALLOC(taup_len, double)) ||
        !(tauq = NAG_ALLOC(tauq_len, double)) ||
        !(u = 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 = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s("%lf", &A(i, j));
#else
        scanf("%lf", &A(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
    if (m >= n) {
      /* Example 1. */

      /* nag_dgeqrf (f08aec): Compute the QR factorization of A */
      nag_dgeqrf(order, m, n, a, pda, tau, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_dgeqrf (f08aec).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
      /* Copy A to U */
      for (i = 1; i <= m; ++i) {
        for (j = 1; j <= MIN(i, n); ++j)
          U(i, j) = A(i, j);
      }
      /* nag_dorgqr (f08afc):                                 */
      /*        Form Q explicitly, storing the result in U */
      nag_dorgqr(order, m, m, n, u, pdu, tau, &fail);
      if (fail.code != NE_NOERROR) {
        printf("order=%d\n", order);
        printf("Error from nag_dorgqr (f08afc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
      /* Copy R to PT (used as workspace) */
      nag_dtr_copy(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n, a,
                   pda, pt, pdpt, &fail);
      /* 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)
          PT(i, j) = 0.0;
      }
      /* nag_dgebrd (f08kec): Bidiagonalize R. */
      nag_dgebrd(order, n, n, pt, pdpt, d, e, tauq, taup, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_dgebrd (f08kec).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
      /* nag_dormbr (f08kgc): Update Q, storing the result in U. */
      nag_dormbr(order, Nag_ApplyQ, Nag_RightSide, Nag_NoTrans,
                 m, n, n, pt, pdpt, tauq, u, pdu, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_dormbr (f08kgc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
      /* Print bidiagonal form and matrix Q */
      printf("\nExample 1: bidiagonal matrix B\nDiagonal\n");
      for (i = 1; i <= n; ++i)
        printf("%8.4f%s", d[i - 1], i % 8 == 0 ? "\n" : " ");
      printf("\nSuperdiagonal\n");
      for (i = 1; i <= n - 1; ++i)
        printf("%8.4f%s", e[i - 1], i % 8 == 0 ? "\n" : " ");
      printf("\n\n");
      /* nag_gen_real_mat_print (x04cac): Print Q as stored in u. */
      fflush(stdout);
      nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                             m, n, u, pdu, "Example 1: matrix Q", 0, &fail);
```

```
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
}
else {
  /* Example 2. */

  /* nag_dgelqf (f08ahc): Compute the LQ factorization of A. */
  nag_dgelqf(order, m, n, a, pda, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgelqf (f08ahc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Copy A to PT */
  for (i = 1; i <= m; ++i) {
    for (j = i; j <= n; ++j)
      PT(i, j) = A(i, j);
  }
  /* nag_dorglq (f08ajc):                                  */
  /*       Form Q explicitly, storing the result in PT. */
  nag_dorglq(order, n, n, m, pt, pdpt, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dorglq (f08ajc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Copy L to U (used as workspace) */
  nag_dtr_copy(order, Nag_Lower, Nag_NoTrans, Nag_NonUnitDiag, m, a,
               pda, u, pdu, &fail);
  /* 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) = 0.0;
  }
  /* nag_dgebrd (f08kec): Bidiagonalize L. */
  nag_dgebrd(order, m, m, u, pdu, d, e, tauq, taup, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgebrd (f08kec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* nag_dormbr (f08kgc):Update P^T, storing the result in PT. */
  nag_dormbr(order, Nag_ApplyP, Nag_LeftSide, Nag_Trans, m, n, m, u,
             pdu, taup, pt, pdpt, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dormbr (f08kgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print bidiagonal form and matrix P^T */
  printf("\nExample 2: bidiagonal matrix B\n%s\n", "Diagonal\n");
  for (i = 1; i <= m; ++i)
    printf("%8.4f%s", d[i - 1], i % 8 == 0 ? "\n" : " ");
  printf("\nSuperdiagonal\n");
  for (i = 1; i <= m - 1; ++i)
    printf("%8.4f%s", e[i - 1], i % 8 == 0 ? "\n" : " ");
  printf("\n\n");

  /* nag_gen_real_mat_print (x04cac), Print pt. */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                         m, n, pt, pdpt, "Example 2: matrix P^T",
                         0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
           fail.message);
```

```
        exit_status = 1;
        goto END;
      }
    }
  END:
    NAG_FREE(a);
    NAG_FREE(d);
    NAG_FREE(e);
    NAG_FREE(pt);
    NAG_FREE(tau);
    NAG_FREE(taup);
    NAG_FREE(tauq);
    NAG_FREE(u);
  }
  return exit_status;
}

#undef A
#undef U
#undef PT
```

10.2 Program Data

```
nag_dormbr (f08kgc) Example Program Data
  6  4                                        :Values of M and N, Example 1
 -0.57  -1.28  -0.39   0.25
 -1.93   1.08  -0.31  -2.14
  2.30   0.24   0.40  -0.35
 -1.93   0.64  -0.66   0.08
  0.15   0.30   0.15  -2.13
 -0.02   1.03  -1.43   0.50                  :End of matrix A
  4  6                                        :Values of M and N, Example 2
 -5.42   3.28  -3.68   0.27   2.06   0.46
 -1.65  -3.40  -3.20  -1.03  -4.06  -0.01
 -0.37   2.35   1.90   4.31  -1.76   1.13
 -3.15  -0.11   1.99  -2.70   0.26   4.50   :End of matrix A
```

10.3 Program Results

```
nag_dormbr (f08kgc) Example Program Results

Example 1: bidiagonal matrix B
Diagonal
  3.6177  -2.4161   1.9213  -1.4265
Superdiagonal
  1.2587  -1.5262   1.1895

 Example 1: matrix Q
         1       2       3       4
 1  -0.1576 -0.2690  0.2612  0.8513
 2  -0.5335  0.5311 -0.2922  0.0184
 3   0.6358  0.3495 -0.0250 -0.0210
 4  -0.5335  0.0035  0.1537 -0.2592
 5   0.0415  0.5572 -0.2917  0.4523
 6  -0.0055  0.4614  0.8585 -0.0532

Example 2: bidiagonal matrix B
Diagonal

 -7.7724   6.1573  -6.0576   5.7933
Superdiagonal
  1.1926   0.5734  -1.9143

 Example 2: matrix P^T
```

```
         1       2       3       4       5       6
 1  -0.7104  0.4299 -0.4824  0.0354  0.2700  0.0603
 2   0.3583  0.1382 -0.4110  0.4044  0.0951 -0.7148
 3  -0.0507  0.4244  0.3795  0.7402 -0.2773  0.2203
 4   0.2442  0.4016  0.4158 -0.1354  0.7666 -0.0137
```