NAG Library Function Document

nag_dorgbr (f08kfc)

1 Purpose

nag_dorgbr (f08kfc) generates one of the real orthogonal matrices Q or P^{T} 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_dorgbr (Nag_OrderType order, Nag_VectType vect, Integer m,
     Integer n, Integer k, double a[], Integer pda, const double tau[],
     NagError *fail)
```

3 Description

nag_dorgbr (f08kfc) 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 generate Q or P^{T} explicitly as square matrices, or in some cases just the leading columns of Q or the leading rows of P^{T}.

The various possibilities are specified by the arguments **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_dorgbr(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_dorgbr(order,Nag_FormQ,m,n,n,...)
   ```

3. To form the full n by n matrix P^{T}:

   ```
   nag_dorgbr(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^{T}:

   ```
   nag_dorgbr(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 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 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 the orthogonal matrix Q or P^{T} is generated.

vect = Nag_FormQ
Q is generated.

vect = Nag_FormP
P^{T} is generated.

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

3: **m** – Integer *Input*

On entry: m, the number of rows of the orthogonal matrix Q or P^{T} to be returned.

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

4: **n** – Integer *Input*

On entry: n, the number of columns of the orthogonal matrix Q or P^{T} 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*] – double *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.

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

On exit: the orthogonal matrix Q or P^{T}, or the leading rows or columns thereof, as specified by **vect**, **m** and **n**.

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

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

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

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

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

$\max(1, \min(\mathbf{m}, \mathbf{k}))$ when **vect** = Nag_FormQ;
$\max(1, \min(\mathbf{n}, \mathbf{k}))$ when **vect** = Nag_FormP.

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

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

NE_ENUM_INT_3

On entry, **vect** = $\langle value \rangle$, **m** = $\langle value \rangle$, **n** = $\langle value \rangle$ and **k** = $\langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$ and
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 $\mathbf{n} \leq \mathbf{k}$, $\mathbf{n} = \mathbf{m}$.

NE_INT

On entry, **k** = $\langle value \rangle$.
Constraint: $\mathbf{k} \geq 0$.

On entry, **m** = $\langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

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 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 matrix Q differs from an exactly orthogonal 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^{T}.

8 Parallelism and Performance

nag_dorgbr (f08kfc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dorgbr (f08kfc) 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 for the cases listed in Section 3 are approximately as follows:

1. To form the whole of Q:

   $\frac{4}{3}n\left(3m^2 - 3mn + n^2\right)$ if $m > n$,

   $\frac{4}{3}m^3$ if $m \leq n$;

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

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

3. To form the whole of P^{T}:

   $\frac{4}{3}n^3$ if $m \geq n$,

   $\frac{4}{3}m\left(3n^2 - 3mn + m^2\right)$ if $m < n$;

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

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

The complex analogue of this function is nag_zungbr (f08ktc).

10 Example

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

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

in the first example and

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

in the second. A must first be reduced to tridiagonal form by nag_dgebrd (f08kec). The program then calls nag_dorgbr (f08kfc) twice to form Q and P^{T}, and passes these matrices to nag_dbdsqr (f08mec), which computes the singular value decomposition of A.

10.1 Program Text

```
/* nag_dorgbr (f08kfc) 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 <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 */
  double *a = 0, *c = 0, *d = 0, *e = 0, *taup = 0, *tauq = 0, *u = 0;
  double *vt = 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);

  printf("nag_dorgbr (f08kfc) Example Program Results\n\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;
    pdu = m;
    pdvt = m;
#else
    pda = n;
    pdu = n;
    pdvt = n;
#endif
    pdc = n;
    d_len = n;
    e_len = n - 1;
    tauq_len = n;
    taup_len = n;

    /* Allocate memory */
    if (!(a = NAG_ALLOC(m * n, double)) ||
        !(c = NAG_ALLOC(n * n, double)) ||
        !(d = NAG_ALLOC(d_len, double)) ||
        !(e = NAG_ALLOC(e_len, double)) ||
        !(taup = NAG_ALLOC(taup_len, double)) ||
        !(tauq = NAG_ALLOC(tauq_len, double)) ||
        !(u = NAG_ALLOC(m * n, double)) || !(vt = NAG_ALLOC(m * n, 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

    /* Reduce A to bidiagonal form using  nag_dgebrd (f08kec). */
    nag_dgebrd(order, m, n, a, pda, 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;
    }
    if (m >= n) {
      /* Example 1 */
      /* Copy A to VT and U */
      for (i = 1; i <= n; ++i) {
        for (j = i; j <= n; ++j)
          VT(i, j) = A(i, j);
      }
```

```
    for (i = 1; i <= m; ++i) {
      for (j = 1; j <= MIN(i, n); ++j)
        U(i, j) = A(i, j);
    }
    /* nag_dorgbr (f08kfc):                                  */
    /*       Form P^T explicitly, storing the result in VT */
    nag_dorgbr(order, Nag_FormP, n, n, m, vt, pdvt, taup, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dorgbr (f08kfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* nag_dorgbr (f08kfc):                                */
    /*       Form Q explicitly, storing the result in U */
    nag_dorgbr(order, Nag_FormQ, m, n, n, u, pdu, tauq, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dorgbr (f08kfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* nag_dbdsqr (f08mec): Compute the SVD of A.        */
    nag_dbdsqr(order, Nag_Upper, n, n, m, 0, d, e, vt, pdvt, u,
               pdu, c, pdc, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dbdsqr (f08mec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Print singular values, left & right singular vectors */
    printf("\n Example 1: singular values\n");
    for (i = 1; i <= n; ++i)
      printf("%8.4f%s", d[i - 1], i % 8 == 0 ? "\n" : " ");
    printf("\n\n");

    /* nag_gen_real_mat_print (x04cac): Print VT. */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           n, n, vt, pdvt,
                           "Example 1: right singular vectors, by row",
                           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;
    }
    printf("\n");

    /* nag_gen_real_mat_print (x04cac): Print U. */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           m, n, u, pdu,
                           "Example 1: left singular vectors, by column",
                           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 */
    /* Copy A to VT and U */
    for (i = 1; i <= m; ++i) {
      for (j = i; j <= n; ++j)
        VT(i, j) = A(i, j);
    }
```

```
    for (i = 1; i <= m; ++i) {
      for (j = 1; j <= i; ++j)
        U(i, j) = A(i, j);
    }
    /* nag_dorgbr (f08kfc):                                  */
    /*        Form P^T explicitly, storing the result in VT */
    nag_dorgbr(order, Nag_FormP, m, n, m, vt, pdvt, taup, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dorgbr (f08kfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
    /* nag_dorgbr (f08kfc):                                 */
    /*        Form Q explicitly, storing the result in U */
    nag_dorgbr(order, Nag_FormQ, m, m, n, u, pdu, tauq, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dorgbr (f08kfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
    /* nag_dbdsqr (f08mec): Compute the SVD of A */
    nag_dbdsqr(order, Nag_Lower, m, n, m, 0, d, e, vt, pdvt, u,
               pdu, c, pdc, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dbdsqr (f08mec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Print singular values, left & right singular vectors */
    printf("\n Example 2: singular values\n");
    for (i = 1; i <= m; ++i)
      printf("%8.4f%s", d[i - 1], i % 8 == 0 ? "\n" : " ");
    printf("\n\n");
    /* nag_gen_real_mat_print (x04cac): Print VT */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           m, n, vt, pdvt,
                           "Example 2: right singular vectors, by row",
                           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;
    }
    printf("\n");
    /* nag_gen_real_mat_print (x04cac): print U */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           m, m, u, pdu,
                           "Example 2: left singular vectors, by column",
                           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(c);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(taup);
  NAG_FREE(tauq);
  NAG_FREE(u);
  NAG_FREE(vt);
}
return exit_status;
```

```
}

#undef A
#undef U
#undef VT
```

10.2 Program Data

```
nag_dorgbr (f08kfc) 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_dorgbr (f08kfc) Example Program Results


 Example 1: singular values
  3.9987   3.0005   1.9967   0.9999

 Example 1: right singular vectors, by row
          1       2       3       4
 1   0.8251 -0.2794  0.2048  0.4463
 2  -0.4530 -0.2121 -0.2622  0.8252
 3  -0.2829 -0.7961  0.4952 -0.2026
 4   0.1841 -0.4931 -0.8026 -0.2807

 Example 1: left singular vectors, by column
          1       2       3       4
 1  -0.0203  0.2794  0.4690  0.7692
 2  -0.7284 -0.3464 -0.0169 -0.0383
 3   0.4393 -0.4955 -0.2868  0.0822
 4  -0.4678  0.3258 -0.1536 -0.1636
 5  -0.2200 -0.6428  0.1125  0.3572
 6  -0.0935  0.1927 -0.8132  0.4957

 Example 2: singular values
  7.9987   7.0059   5.9952   4.9989

 Example 2: right singular vectors, by row
          1       2       3       4       5       6
 1  -0.7933  0.3163 -0.3342 -0.1514  0.2142  0.3001
 2   0.1002  0.6442  0.4371  0.4890  0.3771  0.0501
 3   0.0111  0.1724 -0.6367  0.4354 -0.0430 -0.6111
 4   0.2361  0.0216 -0.1025 -0.5286  0.7460 -0.3120

 Example 2: left singular vectors, by column
          1       2       3       4
 1   0.8884  0.1275  0.4331  0.0838
 2   0.0733 -0.8264  0.1943 -0.5234
 3  -0.0361  0.5435  0.0756 -0.8352
 4   0.4518 -0.0733 -0.8769 -0.1466
```