NAG C 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
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: $A = QB^T$. 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 parameters 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 $Q$:
   
   
   
   
   
   
   
   
   
   
   
   

   (note that the array a must have at least $m$ columns).

2. If $m > n$, to form the $n$ leading columns of $Q$:
   
   
   
   
   
   

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

   (note that the array a must have at least $n$ rows).

4. If $m < n$, to form the $m$ leading rows of $P^T$:
   
   
   
   

4 References
Baltimore

5 Parameters
1: order – Nag_OrderType

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 orthogonal matrix \( Q \) or \( P^T \) is generated as follows:
- if vect = Nag_FormQ, \( Q \) is generated;
- if vect = Nag_FormP, \( P^T \) is generated.

*Constraint*: vect = Nag_FormQ or Nag_FormP.

3: m – Integer

*Input*

*On entry*: the number of rows of the orthogonal matrix \( Q \) or \( P^T \) to be returned.

*Constraint*: \( m \geq 0 \).

4: n – Integer

*Input*

*On entry*: the number of columns of the orthogonal matrix \( Q \) or \( P^T \) to be returned.

*Constraints*:
- \( n \geq 0 \);
- if vect = Nag_FormQ and \( m > k \), \( m \geq n \geq k \);
- if vect = Nag_FormQ and \( m \leq k \), \( m = n \);
- if vect = Nag_FormP and \( n > k \), \( n \geq m \geq k \);
- if vect = Nag_FormP and \( n \leq k \), \( n = m \).

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*: \( k \geq 0 \).

6: a[dim] – double

*Input/Output*

*Note*: the dimension, \( dim \), of the array \( a \) must be at least \( \max(1, pda \times n) \) when \( order = Nag_ColMajor \) and at least \( \max(1, pda \times m) \) when \( order = Nag_RowMajor \).

If \( order = Nag_ColMajor \), the \((i, j)\)th element of the matrix \( A \) is stored in \( a[(j - 1) \times pda + i - 1] \) and if \( order = Nag_RowMajor \), the \((i, j)\)th element of the matrix \( A \) is stored in \( a[(i - 1) \times pda + j - 1] \).

*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.

7: pda – Integer

*Input*

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

*Constraints*:
- if \( order = Nag_ColMajor \), \( pda \geq \max(1, m) \);
- if \( order = Nag_RowMajor \), \( pda \geq \max(1, n) \).

8: tau[dim] – const double

*Input*

*Note*: the dimension, \( dim \), of the array \( tau \) must be at least \( \max(1, \min(m, k)) \) when vect = Nag_FormQ and at least \( \max(1, \min(n, k)) \) when vect = Nag_FormP.

*On entry*: further details of the elementary reflectors, as returned by nag_dgebrd (f08kec) in its parameter \( tauq \) if vect = Nag_FormQ, or in its parameter \( taup \) if vect = Nag_FormP.
6 Error Indicators and Warnings

**NE_INT**

On entry, \(m = \langle \text{value} \rangle\).
Constraint: \(m \geq 0\).

On entry, \(k = \langle \text{value} \rangle\).
Constraint: \(k \geq 0\).

On entry, \(pda = \langle \text{value} \rangle\).
Constraint: \(pda > 0\).

**NE_INT_2**

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

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

**NE_ENUM_INT_3**

On entry, \(\text{vect} = \langle \text{value} \rangle, m = \langle \text{value} \rangle, n = \langle \text{value} \rangle, k = \langle \text{value} \rangle\).
Constraint: \(n \geq 0\) and if \(\text{vect} = \text{Nag\_FormQ}\) and \(m > k, m \geq n \geq k\); if \(\text{vect} = \text{Nag\_FormQ}\) and \(m \leq k, m = n\); if \(\text{vect} = \text{Nag\_FormP}\) and \(n > k, n \geq m \geq k\); if \(\text{vect} = \text{Nag\_FormP}\) and \(n \leq k, n = m\).

**NE_ALLOC_FAIL**

Memory allocation failed.

**NE_BAD_PARAM**

On entry, parameter \(\langle \text{value} \rangle\) 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 matrix \(Q\) differs from an exactly orthogonal matrix by a matrix \(E\) such that

\[
\|E\|_2 = O(\epsilon),
\]

where \(\epsilon\) is the *machine precision*. A similar statement holds for the computed matrix \(P^T\).

8 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 \(Q\):
   \[
   \frac{4}{3} n(3m^2 - 3mn + n^2) \text{ if } m > n, \quad \frac{4}{3} m^3 \text{ if } m \leq n;
   \]
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 \( m \geq n \),
   - \( \frac{4}{3}m(3n^2 - 3mn + m^2) \) 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 \textit{nag_zungbr} (f08ktc).

9 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 \textit{nag_dgebrd} (f08kec). The program then calls \textit{nag_dorgbr} (f08kfc) twice to form \( Q \) and \( P^T \), and passes these matrices to \textit{nag_dbdsqr} (f08mec), which computes the singular value decomposition of \( A \).

9.1 Program Text

/* nag_dorgbr (f08kfc) Example Program. */
* Copyright 2001 Numerical Algorithms Group.
* * Mark 7, 2001. */
#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, *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]
    #else
    #define A(I,J) a[(I-1)*pda + J - 1]
    #define VT(I,J) vt[(I-1)*pdvt + J - 1]
    #endif

#define U(I,J) u[(J-1)*pd_u + I - 1]
#define A(I,J) a[(I-1)*pd_a + J - 1]
#define VT(I,J) vt[(I-1)*pd_vt + J - 1]
#define U(I,J) u[(I-1)*pd_u + J - 1]

order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pd_a + J - 1]
#define VT(I,J) vt[(I-1)*pd_vt + J - 1]
#define U(I,J) u[(I-1)*pd_u + J - 1]

order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
Vprintf("f08kfc Example Program Results\n\n");

/* Skip heading in data file */
Vscanf("%*[\n ] ");

for (ic = 1; ic <= 2; ++ic)
{
  Vscanf("%ld%ld%*[\n ] ", &m, &n);
  d_len = n;
  #ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdc = n;
    pdu = m;
    pdvt = m;
    e_len = n-1;
    tauq_len = n;
    taup_len = n;
  #else
    pda = n;
    pdc = n;
    pdu = n;
    pdvt = n;
    e_len = n-1;
    tauq_len = n;
    taup_len = n;
  #endif
  /* 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)) )
  {
    Vprintf("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)
      Vscanf("%lf", &A(i,j));
  }
  Vscanf("%*[\n ] ");
  /* Reduce A to bidiagonal form */
  f08kec(order, m, n, a, pda, d, e, tauq, taup, &fail);
  if (fail.code != NE_NOERROR)
  {
    Vprintf("Error from f08kec.\n\n");
    exit_status = 1;
    goto END;
  }
  if (m >= n)
  {
    /* 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);
}
/* Form P**T explicitly, storing the result in VT */
f08kfc(order, Nag_FormP, n, n, m, vt, pdvt, taup, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08kfc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Form Q explicitly, storing the result in U */
f08kfc(order, Nag_FormQ, m, n, n, u, pdu, tauq, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08kfc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute the SVD of A */
f08mec(order, Nag_Upper, n, n, m, 0, d, e, vt, pdvt, u,
pdu, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08mec.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print singular values, left & right singular vectors */
Vprintf("\nExample 1: singular values\n");
for (i = 1; i <= n; ++i)
    Vprintf("%8.4f%s", d[i-1], i%8==0?"\n": "");
Vprintf("\n\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
n, n, vt, pdvt,
"Example 1: right singular vectors, by row", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
Vprintf("\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m, n, u, pdu,
"Example 1: left singular vectors, by column", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
else
{
    /* 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);
    }
}
else
{
    /* Form P**T explicitly, storing the result in VT */
f08kfc(order, Nag_FormP, n, n, m, vt, pdvt, taup, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08kfc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Form Q explicitly, storing the result in U */
f08kfc(order, Nag_FormQ, m, n, n, u, pdu, tauq, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08kfc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute the SVD of A */
f08mec(order, Nag_Upper, n, n, m, 0, d, e, vt, pdvt, u,
pdu, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08mec.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print singular values, left & right singular vectors */
Vprintf("\nExample 1: singular values\n");
for (i = 1; i <= n; ++i)
    Vprintf("%8.4f%s", d[i-1], i%8==0?"\n": "");
Vprintf("\n\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
n, n, vt, pdvt,
"Example 1: right singular vectors, by row", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
Vprintf("\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m, n, u, pdu,
"Example 1: left singular vectors, by column", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
}
U(i,j) = A(i,j);

} /* Form P**T explicitly, storing the result in VT */
f08kfc(order, Nag_FormP, m, n, m, vt, pdvt, taup, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f08kfc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

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

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

} /* Print singular values, left & right singular vectors */
Vprintf("\nExample 2: singular values\n");
for (i = 1; i <= m; ++i)
  Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
Vprintf("\n\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m, n, vt, pdvt,
"Example 2: right singular vectors, by row", 0, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from x04cac.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
Vprintf("\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m, m, u, pdu,
"Example 2: left singular vectors, by column", 0, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from x04cac.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

END:
if (a) NAG_FREE(a);
if (c) NAG_FREE(c);
if (d) NAG_FREE(d);
if (e) NAG_FREE(e);
if (taup) NAG_FREE(taup);
if (tauq) NAG_FREE(tauq);
if (u) NAG_FREE(u);
if (vt) NAG_FREE(vt);
return exit_status;
9.2 Program Data

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

9.3 Program Results

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