NAG Library Function Document

nag_dorghr (f08nfc)

1 Purpose

nag_dorghr (f08nfc) generates the real orthogonal matrix \( Q \) which was determined by nag_dgehrd (f08nec) when reducing a real general matrix \( A \) to Hessenberg form.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_dorghr (Nag_OrderType order, Integer n, Integer ilo, Integer ihi,
                double a[], Integer pda, const double tau[], NagError *fail)
```

3 Description

nag_dorghr (f08nfc) is intended to be used following a call to nag_dgehrd (f08nec), which reduces a real general matrix \( A \) to upper Hessenberg form \( H \) by an orthogonal similarity transformation: \( A = QHQ^T \).

nag_dgehrd (f08nec) represents the matrix \( Q \) as a product of \( i_{hi} / i_{lo} \) elementary reflectors. Here \( i_{lo} \) and \( i_{hi} \) are values determined by nag_dgebal (f08nhc) when balancing the matrix; if the matrix has not been balanced, \( i_{lo} = 1 \) and \( i_{hi} = n \).

This function may be used to generate \( Q \) explicitly as a square matrix. \( Q \) has the structure:

\[
Q = \begin{pmatrix}
I & 0 & 0 \\
0 & Q_{22} & 0 \\
0 & 0 & I
\end{pmatrix}
\]

where \( Q_{22} \) occupies rows and columns \( i_{lo} \) to \( i_{hi} \).

4 References


5 Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

On entry: the \textbf{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 \textbf{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: \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{n} – Integer \hspace{1cm} \textit{Input}

On entry: \( n \), the order of the matrix \( Q \).

Constraint: \( n \geq 0 \).
3:  ilo – Integer
4:  ihi – Integer

*Input*

On entry: these must be the same arguments ilo and ihi, respectively, as supplied to nag_dgehrd (f08nec).

*Constraints:*

- if $n > 0$, $1 \leq ilo \leq ihi \leq n$;
- if $n = 0$, $ilo = 1$ and $ihi = 0$.

5:  a[dim] – double

*Input/Output*

*Note: the dimension, dim, of the array a must be at least max(1, pda × n).*

On entry: details of the vectors which define the elementary reflectors, as returned by nag_dgehrd (f08nec).

On exit: the $n$ by $n$ orthogonal matrix $Q$.

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

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

6:  pda – Integer

*Input*

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

*Constraint: pda ≥ max(1, n).*

7:  tau[dim] – const double

*Input*

*Note: the dimension, dim, of the array tau must be at least max(1, n - 1).*

On entry: further details of the elementary reflectors, as returned by nag_dgehrd (f08nec).

8:  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, $n = \langle value \rangle$.

*Constraint: $n \geq 0$.*

On entry, $pda = \langle value \rangle$.

*Constraint: $pda > 0$.*

**NE_INT_2**

On entry, $pda = \langle value \rangle$ and $n = \langle value \rangle$.

*Constraint: $pda \geq \max(1, n)$.*
NE_INT_3

On entry, \( n = \langle \text{value} \rangle \), \( \text{ilo} = \langle \text{value} \rangle \) and \( \text{ihi} = \langle \text{value} \rangle \).
Constraint: if \( n > 0 \), \( 1 \leq \text{ilo} \leq \text{ihi} \leq n \);
if \( n = 0 \), \( \text{ilo} = 1 \) and \( \text{ihi} = 0 \).

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the
function 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 matrix \( Q \) differs from an exactly orthogonal matrix by a matrix \( E \) such that
\[
\| E \|_2 = O(\varepsilon),
\]
where \( \varepsilon \) is the machine precision.

8 Parallelism and Performance

\text{nag
\_dorgr (f08nfc)} is threaded by NAG for parallel execution in multithreaded implementations of the
NAG Library.

\text{nag
\_dorgr (f08nfc)} 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 \( \frac{4}{3}n^3 \), where \( q = \text{ihi} - \text{ilo} \).
The complex analogue of this function is \text{nag zesthr (f08ntc)}.

10 Example

This example computes the Schur factorization of the matrix \( A \), where
\[
A = \begin{pmatrix}
0.35 & 0.45 & -0.14 & -0.17 \\
0.09 & 0.07 & -0.54 & 0.35 \\
-0.44 & -0.33 & -0.03 & 0.17 \\
0.25 & -0.32 & -0.13 & 0.11
\end{pmatrix}
\]

Here \( A \) is general and must first be reduced to Hessenberg form by \text{nag
\_dgehrd (f08nec)}. The program
then calls \text{nag
\_dorgr (f08nfc)} to form \( Q \), and passes this matrix to \text{nag
\_dlseqr (f08pec)} which computes
the Schur factorization of \( A \).
10.1 Program Text

/* nag_dorghr (f08nfc) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 7b revised, 2004.
 */

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

int main(void)
{
    /* Scalars */
    double norm, alpha, beta;
    Integer i, j, n, pda, pdc, pdd, pdz, tau_len, wi_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *a = 0, *c = 0, *d = 0, *tau = 0, *wi = 0, *wr = 0, *z = 0;

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

    INIT_FAIL(fail);
    printf("nag_dorghr (f08nfc) Example Program Results\n\n");

    /* Skip heading in data file */
    ifndef _WIN32
    scanf_s("%*[\n]");
    ifndef _WIN32
    scanf("%*[\n]");
    ifndef _WIN32
    scanf_s("\%"NAG_IFMT"%*[\n]", &n);
    ifndef _WIN32
    scanf("\%"NAG_IFMT"%*[\n]", &n);
    endif
    endif
    endif
    pda = n;
    pdc = n;
    pdd = n;
    pdz = n;
    tau_len = n - 1;
    wi_len = n;

    /* Allocate memory */
    if (!((a = NAG_ALLOC(n * n, double)) ||
         (c = NAG_ALLOC(n * n, double)) ||
         (d = NAG_ALLOC(n * n, double)) ||
         (tau = NAG_ALLOC(tau_len, double)) ||
         (wi = NAG_ALLOC(wi_len, double))) ||
    }}
}}

f08nfc.4    Mark 25
!(wr = NAG_ALLOC(wi_len, double)) ||
!(z = NAG_ALLOC(n * n, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A from data file */
for (i = 1; i <= n; ++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

/* Copy A into D */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
        D(i, j) = A(i, j);
}

/* nag_gen_real_mat_print (x04cac): Print Matrix A. */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
a, pda, "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;
}

/* nag_dgehrd (f08nec): Reduce A to upper Hessenberg form H = (Q**T)*A*Q */
nag_dgehrd(order, n, 1, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgehrd (f08nec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Copy A into Z */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
        Z(i, j) = A(i, j);
}

/* nag_dorghr (f08nfc): Form Q explicitly, storing the result in Z */
nag_dorghr(order, n, 1, n, z, pdz, tau, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dorghr (f08nfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dhseqr (f08pec): Calculate the Schur factorization of H = Y*T*(Y**T) and form

Mark 25
f08nfc.5
Z = Q \cdot Y \text{ explicitly. Note that } A = Z \cdot T \cdot (Z^* \cdot T).

```c
nag_dhseqr(order, Nag_Schur, Nag_UpdateZ, n, 1, n, a, pda,
   wr, wi, z, pdz, &fail);
if (fail.code != NE_NOERROR)
   {
      printf("Error from nag_dhseqr (f08pec).\n\n", fail.message);
      exit_status = 1;
      goto END;
   }
/* nag_dgemm (f16yac): Compute } A - Z \cdot T \cdot Z^* \text{ from the factorization of } A \text{ and store in matrix } D */
alpha = 1.0;
beta = 0.0;
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, alpha, z, pdz,
   a, pda, beta, c, pdc, &fail);
if (fail.code != NE_NOERROR)
   {
      printf("Error from nag_dgemm (f16yac).\n\n", fail.message);
      exit_status = 1;
      goto END;
   }
alpha = -1.0;
beta = 1.0;
nag_dgemm(order, Nag_NoTrans, Nag_Trans, n, n, n, alpha, c, pdc, z,
   pdz, beta, d, pdd, &fail);
if (fail.code != NE_NOERROR)
   {
      printf("Error from nag_dgemm (f16yac).\n\n", fail.message);
      exit_status = 1;
      goto END;
   }
/* nag_dge_norm (f16rac): Find norm of matrix } D \text{ and print warning if } * \text{ it is too large } */
nag_dge_norm(order, Nag_OneNorm, n, n, d, pdd, &norm, &fail);
if (fail.code != NE_NOERROR)
   {
      printf("Error from nag_dge_norm (f16rac).\n\n", fail.message);
      exit_status = 1;
      goto END;
   }
if (norm>pow(x02ajc(),0.8))
   {
      printf("\n\n","Norm of } A - (Z \cdot T \cdot Z^*) \text{ is much greater than } 0.\text{,}"
         "Schur factorization has failed.");
   }
END:
NAG_FREE(a);
NAG_FREE(c);
NAG_FREE(d);
NAG_FREE(tau);
NAG_FREE(wi);
NAG_FREE(wr);
NAG_FREE(z);
return exit_status;
```

#undef A
#undef D
#undef Z
10.2 Program Data

nag_dorghr (f08nfc) Example Program Data

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.35</td>
<td>0.45</td>
<td>-0.14</td>
<td>-0.17</td>
</tr>
<tr>
<td>2</td>
<td>0.09</td>
<td>0.07</td>
<td>-0.54</td>
<td>0.35</td>
</tr>
<tr>
<td>3</td>
<td>-0.44</td>
<td>-0.33</td>
<td>-0.03</td>
<td>0.17</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>-0.32</td>
<td>-0.13</td>
<td>0.11</td>
</tr>
</tbody>
</table>

:End of matrix A

10.3 Program Results

nag_dorghr (f08nfc) Example Program Results

Matrix A

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3500</td>
<td>0.4500</td>
<td>-0.1400</td>
<td>-0.1700</td>
</tr>
<tr>
<td>2</td>
<td>0.0900</td>
<td>0.0700</td>
<td>-0.5400</td>
<td>0.3500</td>
</tr>
<tr>
<td>3</td>
<td>-0.4400</td>
<td>-0.3300</td>
<td>-0.0300</td>
<td>0.1700</td>
</tr>
<tr>
<td>4</td>
<td>0.2500</td>
<td>-0.3200</td>
<td>-0.1300</td>
<td>0.1100</td>
</tr>
</tbody>
</table>