

# 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

```
#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

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: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $Q$ .  
*Constraint:*  $n \geq 0$ .

- 3: **ilo** – Integer *Input*  
 4: **ihi** – Integer *Input*

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

*Constraints:*

if  $n > 0$ ,  $1 \leq \mathbf{ilo} \leq \mathbf{ihi} \leq n$ ;  
 if  $n = 0$ ,  $\mathbf{ilo} = 1$  and  $\mathbf{ihi} = 0$ .

- 5: **a**[*dim*] – double *Input/Output*

**Note:** the dimension, *dim*, of the array **a** must be at least  $\max(1, \mathbf{pda} \times n)$ .

*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  $\mathbf{a}[(j - 1) \times \mathbf{pda} + i - 1]$ .

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

- 6: **pda** – Integer *Input*

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

*Constraint:*  $\mathbf{pda} \geq \max(1, n)$ .

- 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 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 3.2.1.2 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\_INT

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

Constraint:  $n \geq 0$ .

On entry,  $\mathbf{pda} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} > 0$ .

**NE\_INT\_2**

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

**NE\_INT\_3**

On entry, **n** =  $\langle value \rangle$ , **ilo** =  $\langle value \rangle$  and **ihi** =  $\langle value \rangle$ .  
 Constraint: if **n** > 0,  $1 \leq \mathbf{ilo} \leq \mathbf{ihi} \leq \mathbf{n}$ ;  
 if **n** = 0, **ilo** = 1 and **ihi** = 0.

**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 3.6.6 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 3.6.5 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  $\epsilon$  is the *machine precision*.

**8 Parallelism and Performance**

nag\_dorghr (f08nfc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dorghr (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  $\frac{4}{3}q^3$ , where  $q = i_{hi} - i_{lo}$ .

The complex analogue of this function is nag\_zunghr (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 nag\_dgehrd (f08nec). The program then calls nag\_dorghr (f08nfc) to form  $Q$ , and passes this matrix to nag\_dhseqr (f08pec) which computes the Schur factorization of  $A$ .

## 10.1 Program Text

```

/* nag_dorghr (f08nfc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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 */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &n);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &n);
#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)) ||
        !(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
     * Z=Q*Y explicitly. Note that A = Z*T*(Z^T).
     */
    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%s\n", fail.message);
        exit_status = 1;
    }

```

```

    goto END;
}

/* nag_dgemm (f16yac): Compute A - Z*T*Z^T from the factorization of */
/* A 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%s\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%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dge_norm (f16rac): Find norm of matrix D and print warning if */
/* 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%s\n", fail.message);
    exit_status = 1;
    goto END;
}
if (norm > pow(x02ajc(), 0.8)) {
    printf("%s\n%s\n", "Norm of A-(Z*T*Z^T) is much greater than 0.",
          "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
4                               :Value of N
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 of matrix A

```

### **10.3 Program Results**

nag\_dorghr (f08nfc) Example Program Results

Matrix A

	1	2	3	4
1	0.3500	0.4500	-0.1400	-0.1700
2	0.0900	0.0700	-0.5400	0.3500
3	-0.4400	-0.3300	-0.0300	0.1700
4	0.2500	-0.3200	-0.1300	0.1100

---