NAG C Library Function Document

nag_dgeqrf (f08aec)

1 Purpose

nag_dgeqrf (f08aec) computes the QR factorization of a real m by n matrix.

2 Specification

```
void nag_dgeqrf (Nag_OrderType order, Integer m, Integer n, double a[],
     Integer pda, double tau[], NagError *fail)
```

3 Description

nag_dgeqrf (f08aec) forms the QR factorization of an arbitrary rectangular real m by n matrix. No pivoting is performed.

If $m \geq n$, the factorization is given by:

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix},$$

where R is an n by n upper triangular matrix and Q is an m by m orthogonal matrix. It is sometimes more convenient to write the factorization as

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}\begin{pmatrix} R \\ 0 \end{pmatrix},$$

which reduces to

$$A = Q_1 R,$$

where Q_1 consists of the first n columns of Q, and Q_2 the remaining $m - n$ columns.

If $m < n$, R is trapezoidal, and the factorization can be written

$$A = Q\begin{pmatrix} R_1 & R_2 \end{pmatrix},$$

where R_1 is upper triangular and R_2 is rectangular.

The matrix Q is not formed explicitly but is represented as a product of $\min(m, n)$ elementary reflectors (see the f08 Chapter Introduction for details). Functions are provided to work with Q in this representation (see Section 8).

Note also that for any $k < n$, the information returned in the first k columns of the array **a** represents a QR factorization of the first k columns of the original matrix A.

4 References

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

5 Parameters

1: **order** – Nag_OrderType *Input*

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: **m** – Integer *Input*

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

Constraint: **m** ≥ 0.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **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** and at least $\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.

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

On entry: the m by n matrix A.

On exit: if $m \geq n$, the elements below the diagonal are overwritten by details of the orthogonal matrix Q and the upper triangle is overwritten by the corresponding elements of the n by n upper triangular matrix R.

If $m < n$, the strictly lower triangular part is overwritten by details of the orthogonal matrix Q and the remaining elements are overwritten by the corresponding elements of the m by n upper trapezoidal matrix R.

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

6: **tau**$[dim]$ – double *Output*

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

On exit: further details of the orthogonal matrix Q.

7: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

NE_INT_2

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

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ 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 factorization is the exact factorization of a nearby matrix $A + E$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ is the ***machine precision***.

8 Further Comments

The total number of floating-point operations is approximately $\frac{2}{3}n^2(3m - n)$ if $m \geq n$ or $\frac{2}{3}m^2(3n - m)$ if $m < n$.

To form the orthogonal matrix Q this function may be followed by a call to nag_dorgqr (f08afc):

```
nag_dorgqr (order,m,m,MIN(m,n),&a,pda,tau,&fail)
```

but note that the second dimension of the array **a** must be at least **m**, which may be larger than was required by nag_dgeqrf (f08aec).

When $m \geq n$, it is often only the first n columns of Q that are required, and they may be formed by the call:

```
nag_dorgqr (order,m,n,n,&a,pda,tau,&fail)
```

To apply Q to an arbitrary real rectangular matrix C, this function may be followed by a call to nag_dormqr (f08agc). For example,

```
nag_dormqr (order,Nag_LeftSide,Nag_Trans,m,p,MIN(m,n),&a,pda,
tau,&c,pdc,&fail)
```

forms $C = Q^T C$, where C is m by p.

To compute a QR factorization with column pivoting, use nag_dgeqpf (f08bec).

The complex analogue of this function is nag_zgeqrf (f08asc).

9 Example

To solve the linear least-squares problem

$$\text{minimize } \|Ax_i - b_i\|_2, \quad i = 1, 2$$

where b_1 and b_2 are the columns of the matrix B,

$$A = \begin{pmatrix} -0.57 & -1.28 & -0.39 & 0.25 \\ -1.93 & 1.08 & -0.31 & -2.14 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ -1.93 & 0.64 & -0.66 & 0.08 \\ 0.15 & 0.30 & 0.15 & -2.13 \\ -0.02 & 1.03 & -1.43 & 0.50 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -3.15 & 2.19 \\ -0.11 & -3.64 \\ 1.99 & 0.57 \\ -2.70 & 8.23 \\ 0.26 & -6.35 \\ 4.50 & -1.48 \end{pmatrix}.$$

9.1 Program Text

```
/* nag_dgeqrf (f08aec) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer  i, j, m, n, nrhs, pda, pdb, tau_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  double *a=0, *b=0, *tau=0;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
#define B(I,J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("f08aec Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%ld%ld%*[^\n] ", &m, &n, &nrhs);
#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = m;
#else
  pda = n;
  pdb = nrhs;
#endif
  tau_len = MIN(m,n);

  /* Allocate memory */
  if ( !(a = NAG_ALLOC(m * n, double)) ||
       !(b = NAG_ALLOC(m * nrhs, double)) ||
       !(tau = NAG_ALLOC(tau_len, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Read A and B from data file */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
        Vscanf("%lf", &A(i,j));
    }
  Vscanf("%*[^\n] ");
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        Vscanf("%lf", &B(i,j));
```

```
    }
  Vscanf("%*[^\n] ");

  /* Compute the QR factorization of A */
  f08aec(order, m, n, a, pda, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08aec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute C = (Q**T)*B, storing the result in B */
  f08agc(order, Nag_LeftSide, Nag_Trans, m, nrhs, n, a, pda,
         tau, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08agc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute least-squares solution by backsubstitution in R*X = C */
  f07tec(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs,
          a, pda, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07tec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print least-squares solution(s) */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb,
         "Least-squares solution(s)", 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 (b) NAG_FREE(b);
  if (tau) NAG_FREE(tau);
  return exit_status;
}
```

9.2 Program Data

```
f08aec Example Program Data
  6  4  2                         :Values of M, N and NRHS
 -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
 -3.15   2.19
 -0.11  -3.64
  1.99   0.57
 -2.70   8.23
  0.26  -6.35
  4.50  -1.48                  :End of matrix B
```

9.3 Program Results

```
f08aec Example Program Results

 Least-squares solution(s)
             1           2
 1      1.5146     -1.5838
 2      1.8621      0.5536
 3     -1.4467      1.3491
 4      0.0396      2.9600
```