1 Purpose

nag_dgelqf (f08ahc) computes the $LQ$ factorization of a real $m$ by $n$ matrix.

2 Specification

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

3 Description

nag_dgelqf (f08ahc) forms the $LQ$ factorization of an arbitrary rectangular real $m$ by $n$ matrix. No pivoting is performed.

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

$$A = (L \ 0)Q$$

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

$$A = (L \ 0)\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}$$

which reduces to

$$A = LQ_1,$$

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

If $m > n$, $L$ is trapezoidal, and the factorization can be written

$$A = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix}Q$$

where $L_1$ is lower triangular and $L_2$ is rectangular.

The $LQ$ factorization of $A$ is essentially the same as the $QR$ factorization of $A^T$, since

$$A = (L \ 0)Q \iff A^T = Q^T \begin{pmatrix} L^T \\ 0 \end{pmatrix}.$$ 

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 $k < m$, the information returned in the first $k$ rows of the array $a$ represents an $LQ$ factorization of the first $k$ rows of the original matrix $A$.

4 References

None.

5 Parameters

1: \hspace{1em} order – Nag_OrderType \hspace{1em} \textit{Input} \\
   \hspace{1em} \textit{On entry:} the \textit{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
**f08ahc**

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  
   On entry: m, the number of rows of the matrix $A$.
   **Constraint:** $m \geq 0$.

3: **n** – Integer  
   On entry: n, the number of columns of the matrix $A$.
   **Constraint:** $n \geq 0$.

4: **a[**dim**]** – double  
   On entry: the $m$ by $n$ matrix $A$.
   On exit: if $m \leq n$, the elements above the diagonal are overwritten by details of the orthogonal matrix $Q$ and the lower triangle is overwritten by the corresponding elements of the $m$ by $m$ lower triangular matrix $L$.
   If $m > n$, the strictly upper 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 $n$ lower trapezoidal matrix $L$.

5: **pda** – Integer  
   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, $pda \geq \max(1,n)$.

6: **tau[**dim**]** – double  
   On entry: the stride separating matrix row or column elements (depending on the value of order) in the array $a$.
   On exit: further details of the orthogonal matrix $Q$.

7: **fail** – NagError *  
   The NAG error parameter (see the Essential Introduction).

### 6 Error Indicators and Warnings

**NE_INT**

On entry, $m = \langle value \rangle$.
   Constraint: $m \geq 0$.

On entry, $n = \langle value \rangle$.
   Constraint: $n \geq 0$.

On entry, $pda = \langle value \rangle$.
   Constraint: $pda > 0$.
NE_INT_2

On entry, \( pda = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \).
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_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 factorization is the exact factorization of a nearby matrix \( A + E \), where
\[
\|E\|_2 = O(\epsilon)\|A\|_2,
\]
and \( \epsilon \) is the \textit{machine precision}.

8 Further Comments

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

To form the orthogonal matrix \( Q \) this function may be followed by a call to \texttt{nag_dorglq} (f08ajc):
\[
\texttt{nag_dorglq}(\text{order}, n, n, \text{MIN}(m,n), &a, pda, tau, &fail)
\]
but note that the first dimension of the array \( a \), specified by the parameter \( pda \), must be at least \( n \), which may be larger than was required by \texttt{nag_dgelqf} (f08ahc).

When \( m \leq n \), it is often only the first \( m \) rows of \( Q \) that are required, and they may be formed by the call:
\[
\texttt{nag_dorglq}(\text{order}, m, n, m, &a, pda, tau, &fail)
\]
To apply \( Q \) to an arbitrary real rectangular matrix \( C \), this function may be followed by a call to \texttt{nag_dormlq} (f08akc). For example,
\[
\texttt{nag_dormlq}(\text{order}, \text{Nag_LeftSide}, \text{Nag_Trans}, m, p, \text{MIN}(m,n), &a, pda, tau, &c, pdc, &fail)
\]
forms the matrix product \( C = Q^T C \), where \( C \) is \( m \) by \( p \).

The complex analogue of this function is \texttt{nag_zgelqf} (f08avc).

9 Example

To find the minimum-norm solutions of the under-determined systems of linear equations \( Ax_1 = b_1 \) and \( Ax_2 = b_2 \)

where \( b_1 \) and \( b_2 \) are the columns of the matrix \( B \),

\[
A = \begin{pmatrix}
-5.42 & 3.28 & -3.68 & 0.27 & 2.06 & 0.46 \\
-1.65 & -3.40 & -3.20 & -1.03 & -4.06 & -0.01 \\
-0.37 & 2.35 & 1.90 & 4.31 & -1.76 & 1.13 \\
-3.15 & -0.11 & 1.99 & -2.70 & 0.26 & 4.50
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
-2.87 & -5.23 \\
1.63 & 0.29 \\
-3.52 & 4.76 \\
0.45 & -8.41
\end{pmatrix}
\]
9.1 Program Text

/*! nag_dgelqf (f08ahc) Example Program.
   * Copyright 2001 Numerical Algorithms Group.
   */

#include <stdio.h>
#include <nag.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("f08ahc 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 = n;
    #else
    pda = n;
    pdb = nrhs;
    #endif
    tau_len = MIN(m,n);
    /* Allocate memory */
    if ( !(a = NAG_ALLOC(m * n, double)) ||
        !(b = NAG_ALLOC(n * 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)
```c
    { for (j = 1; j <= nrhs; ++j)
      Vscanf("%lf", &B(i,j));
    }

    Vscanf("%*[\n] ");

    /* Compute the LQ factorization of A */
    f08ahc(order, m, n, a, pda, tau, &fail);
    if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08ahc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Solve L*Y = B, storing the result in B */
    f07tec(order, Nag_Lower, Nag_NoTrans, Nag_NoUnitDiag, m,
           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;
    }

    /* Set rows (M+1) to N of B to zero */
    if (m < n)
    {
      for (i = m + 1; i <= n; ++i)
      {
        for (j = 1; j <= nrhs; ++j)
          B(i,j) = 0.0;
      }
    }

    /* Compute minimum-norm solution X = (Q**T)*B in B */
    f08akc(order, Nag_LeftSide, Nag_Trans, n, nrhs, m, a, pda,
            tau, b, pdb, &fail);
    if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08akc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Print minimum-norm solution(s) */
    x04cac(order, Nag_GeneralMatrix, Nag_NoUnitDiag, n, nrhs, b, pdb,
            "Minimum-norm 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

**f08ahc Example Program Data**

```
4 6 2  :Values of M, N and NRHS
-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
-2.87  -5.23  1.63  0.29
-3.52  4.76
  0.45  -8.41  :End of matrix B
```

```
9.3 Program Results

f08ahc Example Program Results

Minimum-norm solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2371</td>
<td>0.7383</td>
</tr>
<tr>
<td>2</td>
<td>-0.4575</td>
<td>0.0158</td>
</tr>
<tr>
<td>3</td>
<td>-0.0085</td>
<td>-0.0161</td>
</tr>
<tr>
<td>4</td>
<td>-0.5192</td>
<td>1.0768</td>
</tr>
<tr>
<td>5</td>
<td>0.0239</td>
<td>-0.6436</td>
</tr>
<tr>
<td>6</td>
<td>-0.0543</td>
<td>-0.6613</td>
</tr>
</tbody>
</table>