NAG Library Function Document

nag_dgeqp3 (f08bfc)

1 Purpose

nag_dgeqp3 (f08bfc) computes the QR factorization, with column pivoting, of a real \( m \) by \( n \) matrix.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_dgeqp3 (Nag_OrderType order, Integer m, Integer n, double a[],
                   Integer pda, Integer jpvt[], double tau[], NagError *fail)
```

3 Description

nag_dgeqp3 (f08bfc) forms the QR factorization, with column pivoting, of an arbitrary rectangular real \( m \) by \( n \) matrix. If \( m \geq n \), the factorization is given by:

\[
AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix},
\]

where \( R \) is an \( n \) by \( n \) upper triangular matrix, \( Q \) is an \( m \) by \( m \) orthogonal matrix and \( P \) is an \( n \) by \( n \) permutation matrix. It is sometimes more convenient to write the factorization as

\[
AP = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix},
\]

which reduces to

\[
AP = Q_1 R,
\]

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

\[
AP = Q \begin{pmatrix} R_1 & R_2 \end{pmatrix},
\]

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 9).

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 permuted matrix \( AP \).

The function allows specified columns of \( A \) to be moved to the leading columns of \( AP \) at the start of the factorization and fixed there. The remaining columns are free to be interchanged so that at the \( i \)th stage the pivot column is chosen to be the column which maximizes the 2-norm of elements \( i \) to \( m \) over columns \( i \) to \( n \).
4 References

5 Arguments

1: order – Nag_OrderType
   
   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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.
   
   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: m – Integer
   
   On entry: m, the number of rows of the matrix A.
   
   Constraint: m ≥ 0.

3: n – Integer
   
   On entry: n, the number of columns of the matrix A.
   
   Constraint: n ≥ 0.

4: a[dim] – double
   
   Note: the dimension, dim, of the array a must be at least max(1, pda × n) when order = Nag_ColMajor;
   max(1, m × pda) when order = Nag_RowMajor.
   
   The (i, j)th element of the matrix A is stored in
   
   a[(j - 1) × pda + i - 1] when order = Nag_ColMajor;
   a[(i - 1) × pda + j - 1] when order = Nag_RowMajor.
   
   On entry: the m by n matrix A.
   
   On exit: if m ≥ 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
   
   On entry: the stride separating row or column elements (depending on the value of order) in the array a.

   Constraints:
   
   if order = Nag_ColMajor, pda ≥ max(1, m);
   if order = Nag_RowMajor, pda ≥ max(1, n).

6: jpvt[dim] – Integer
   
   Note: the dimension, dim, of the array jpvt must be at least max(1, n).
On entry: if \( \text{jptv}[j-1] \neq 0 \), then the \( j \) th column of \( A \) is moved to the beginning of \( AP \) before the decomposition is computed and is fixed in place during the computation. Otherwise, the \( j \) th column of \( A \) is a free column (i.e., one which may be interchanged during the computation with any other free column).

On exit: details of the permutation matrix \( P \). More precisely, if \( \text{jptv}[j-1] = k \), then the \( k \)th column of \( A \) is moved to become the \( j \)th column of \( AP \); in other words, the columns of \( AP \) are the columns of \( A \) in the order \( \text{jptv}[0], \text{jptv}[1], \ldots, \text{jptv}[n-1] \).

7: \( \text{tau}[\text{dim}] \) – double  
Output

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

On exit: the scalar factors of the elementary reflectors.

8: \( \text{fail} \) – 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 \( \langle \text{value} \rangle \) had an illegal value.

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

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

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

NE_INT_2
On entry, \( \text{pda} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: \( \text{pda} \geq \max(1, m) \).

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

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 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 factorization is the exact factorization of a nearby matrix \((A + E)\), where
\[
\|E\|_2 = O(\epsilon)\|A\|_2,
\]
and \(\epsilon\) is the machine precision.

8 Parallelism and Performance
nag_dgeqp3 (f08bfc) is threaded by NAG for parallel execution in multithreaded implementations of the
NAG Library.
nag_dgeqp3 (f08bfc) 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
\[
\frac{2}{3}n^2(3n - m) \quad \text{if} \quad m \geq n
\]
\[
\frac{2}{3}m^2(3n - m) \quad \text{if} \quad m < n.
\]
To form the orthogonal matrix \(Q\) nag_dgeqp3 (f08bfc) 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_dgeqp3 (f08bfc).
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 \(C\), nag_dgeqp3 (f08bfc) 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 \(C = Q^TC\), where \(C\) is \(m\) by \(p\).
To compute a \(QR\) factorization without column pivoting, use nag_dgeqrf (f08aec).
The complex analogue of this function is nag_zgeqp3 (f08btc).

10 Example
This example solves the linear least squares problems
\[
\min_{x} \|b_j - Ax_j\|_2, \quad j = 1, 2
\]
for the basic solutions \(x_1\) and \(x_2\), where
\[
A = \begin{pmatrix}
-0.09 & 0.14 & -0.46 & 0.68 & 1.29 \\
-1.56 & 0.20 & 0.29 & 1.09 & 0.51 \\
-1.48 & -0.43 & 0.89 & -0.71 & -0.96 \\
-1.09 & 0.84 & 0.77 & 2.11 & -1.27 \\
0.08 & 0.55 & -1.13 & 0.14 & 1.74 \\
-1.59 & -0.72 & 1.06 & 1.24 & 0.34
\end{pmatrix}
\text{and} \quad B = \begin{pmatrix}
7.4 & 2.7 \\
4.2 & -3.0 \\
-8.3 & -9.6 \\
1.8 & 1.1 \\
8.6 & 4.0 \\
2.1 & -5.7
\end{pmatrix}
\]
and \(b_j\) is the \(j\)th column of the matrix \(B\). The solution is obtained by first obtaining a \(QR\) factorization.
with column pivoting of the matrix $A$. A tolerance of 0.01 is used to estimate the rank of $A$ from the upper triangular factor, $R$.

10.1 Program Text

/* nag_dgeqp3 (f08bfc) Example Program.  *
* Copyright 2014 Numerical Algorithms Group.  *
* Mark 23, 2011.  */

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

int main(void)
{
    /* Scalars */
    double d, f, tol;
    Integer i, j, k, m, n, nrhs, pda, pdb;
    Integer exit_status = 0;
    /* Arrays */
    double *a = 0, *b = 0, *rnorm = 0, *tau = 0, *work = 0;
    Integer *jpvt = 0;
    /* Nag Types */
    Nag_OrderType order;
    NagError fail;

#ifndef NAG_COLUMN_MAJOR
#define A(I, J) a[(J - 1) * pda + I - 1]
#define B(I, J) b[(J - 1) * pdb + I - 1]
#else
#define A(I, J) a[(I - 1) * pda + J - 1]
#define B(I, J) b[(I - 1) * pdb + J - 1]
#endif

#include "nag.h"
#include "nagf08.h"

#include "nagf16.h"
#include "nagx04.h"

    if (!(a = NAG_ALLOC(m * n, double)) ||
            !(b = NAG_ALLOC(m * nrhs, double)) ||
    #endif
    #ifdef NAG_COLUMN_MAJOR
        pda = m;
    #else
        pda = n;
    #endif
    pdb = nrhs;

    INIT_FAIL(fail);
    printf("nag_dgeqp3 (f08bfc) Example Program Results\n\n");

#ifndef _WIN32
    scanf("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifndef _WIN32
    scanf("%"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT"%*[\n]", &m, &n, &nrhs);
#else
    scanf("%"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT"%*[\n]", &m, &n, &nrhs);
#endif
    order = Nag_ColMajor;
    
    #define B(I, J) b[(I - 1) * pdb + J - 1]
    order = Nag_RowMajor;
    
}
(rnrm = NAG_ALLOC(nrhs, double)) ||
(tau = NAG_ALLOC(n, double)) ||
(work = NAG_ALLOC(n, double)) ||
(jpvt = NAG_ALLOC(n, integer))
{
    printf("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)
#ifdef _WIN32
    scanf_s("%lf", &A(i, j));
#else
    scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[\n");
#else
    scanf("%*[\n");
#endif
for (i = 1; i <= m; ++i)
    for (j = 1; j <= nrhs; ++j)
#ifdef _WIN32
    scanf_s("%lf", &B(i, j));
#else
    scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[\n");
#else
    scanf("%*[\n");
#endif

/* nag_iload (f16dbc).  
 * Initialize jpvt to be zero so that all columns are free.  
 */
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_iload (f16dbc).\n\n", fail.message);
        exit_status = 1;
        goto END;
    }

/* nag_dgeqp3 (f08bfc).  
 * Compute the QR factorization of A.  
 */
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dgeqp3 (f08bfc).\n\n", fail.message);
        exit_status = 1;
        goto END;
    }

/* nag_dormqr (f08agc).  
 * Compute C = (C1) = (Q**T)*B, storing the result in B.  
 * (C2)  
 */
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dormqr (f08agc).\n\n", fail.message);
        exit_status = 1;
        goto END;
    }
/* Choose tol to reflect the relative accuracy of the input data */
tol = 0.01;

/* Determine and print the rank, k, of R relative to tol */
for (k = 1; k <= n; ++k)
  if ((d = A(k, k), fabs(d)) <= tol * (f = A(1, 1), fabs(f)))
    break;
--k;

printf("Tolerance used to estimate the rank of A\n");
printf("%11.2e\n", tol);

for (j = 1; j <= nrhs; ++j) {
  nag_dge_norm(order, Nag_FrobeniusNorm, m - k, 1, &B(k + 1, j), pdb,
               &rnorm[j - 1], &fail);
  if (fail.code != NE_NOERROR)
    {printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
     exit_status = 1;
     goto END;}
}

/* Permute the least-squares solutions stored in B to give X = P*Y */
for (j = 1; j <= nrhs; ++j) {
  for (i = 1; i <= n; ++i)
    work[jpvt[i - 1] - 1] = B(i, j);
  for (i = 1; i <= n; ++i)
    B(i, j) = work[i - 1];
}

/* Print least-squares solutions. */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b,
                       pdb, "Least-squares solution(s)", 0, &fail);
if (fail.code != NE_NOERROR)
  {printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
   exit_status = 1; goto END;}
{  
    printf("Error from nag_gen_real_mat_print (x04cac).\n\n", fail.message);  
    exit_status = 1;  
    goto END;  
}

/* Print the square roots of the residual sums of squares */
printf("\nSquare root(s) of the residual sum(s) of squares\n");
for (j = 0; j < nrhs; ++j)
    printf("%11.2e%s", rnorm[j], (j+1)%6 == 0?"\n":" ");

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(rnorm);
NAG_FREE(tau);
NAG_FREE(work);
NAG_FREE(jpvt);

return exit_status;
}

#undef A
#undef B

10.2 Program Data

nag_dgeqp3 (f08bfc) Example Program Data

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>nrhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

:Values of m, n and nrhs

-0.09 0.14 -0.46 0.68 1.29
-1.56 0.20 0.29 1.09 0.51
-1.48 -0.43 0.89 -0.71 -0.96
-1.09 0.84 0.77 2.11 -1.27
0.08 0.55 -1.13 0.14 1.74
-1.59 -0.72 1.06 1.24 0.34 :End of matrix A

7.4 2.7
4.2 -3.0
-8.3 -9.6
1.8 1.1
8.6 4.0
2.1 -5.7 :End of matrix B

10.3 Program Results

nag_dgeqp3 (f08bfc) Example Program Results

Tolerance used to estimate the rank of A
1.00e-02

Estimated rank of A
4

Least-squares solution(s)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9767</td>
</tr>
<tr>
<td>2</td>
<td>1.9861</td>
</tr>
<tr>
<td>3</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>2.9927</td>
</tr>
<tr>
<td>5</td>
<td>4.0272</td>
</tr>
</tbody>
</table>

Square root(s) of the residual sum(s) of squares

2.54e-02 3.65e-02