

NAG C Library Function Document

nag_dgeqpf (f08bec)

1 Purpose

nag_dgeqpf (f08bec) computes the QR factorization, with column pivoting, of a real m by n matrix.

2 Specification

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

3 Description

nag_dgeqpf (f08bec) 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

$$AP = (Q_1 \quad Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix}$$

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

$$AP = Q (R_1 \quad R_2),$$

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 \mathbf{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

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: $m \geq 0$.
- 3: **n** – Integer *Input*
On entry: n , the number of columns of the matrix A .
Constraint: $n \geq 0$.
- 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**, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **order = Nag_RowMajor**, $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 6: **jpvt**[*dim*] – Integer *Input/Output*
Note: the dimension, *dim*, of the array **jpvt** must be at least $\max(1, \mathbf{n})$.
On entry: if **jpvt**[i] $\neq 0$, then the i 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 i 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 **jpvt**[$i-1$] = k , then the k th column of A is moved to become the i th column of AP ; in other words, the columns of AP are the columns of A in the order **jpvt**[0], **jpvt**[1], ..., **jpvt**[$n-1$].
- 7: **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 .

8: **fail** – NagError *

Output

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **m** = $\langle value \rangle$.

Constraint: **m** ≥ 0 .

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

Constraint: **n** ≥ 0 .

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

Constraint: **pda** > 0 .

NE_INT_2

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

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

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

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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 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_dgeqpf (f08bec).

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` (f08age). 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 without column pivoting, use `nag_dgeqrf` (f08aec).

The complex analogue of this function is `nag_zgeqpf` (f08bsc).

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 ,

where

$$A = \begin{pmatrix} -0.09 & 0.14 & -0.46 & 0.68 & 1.29 \\ -1.56 & 0.20 & 0.29 & 1.09 & 0.51 \\ -1.48 & -0.43 & 0.89 & -0.71 & -0.96 \\ -1.09 & 0.84 & 0.77 & 2.11 & -1.27 \\ 0.08 & 0.55 & -1.13 & 0.14 & 1.74 \\ -1.59 & -0.72 & 1.06 & 1.24 & 0.34k \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -0.01 & -0.04 \\ 0.04 & -0.03 \\ 0.05 & 0.01 \\ -0.03 & -0.02 \\ 0.02 & 0.05 \\ -0.06 & 0.07k \end{pmatrix}.$$

Here A is approximately rank-deficient, and hence it is preferable to use `nag_dgeqpf` (f08bec) rather than `nag_dgeqrf` (f08aec).

9.1 Program Text

```
/* nag_dgeqpf (f08bec) 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 <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double tol;
    Integer i, j, jpvt_len, k, m, n, nrhs;
    Integer pda, pdb, pdx, tau_len;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *a=0, *b=0, *tau=0, *x=0;
    Integer *jpvt=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]
#define X(I,J) x[(J-1)*pdx + 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]
#define X(I,J) x[(I-1)*pdx + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);
    Vprintf("f08bec 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;
    pdx = m;
#else
    pda = n;
    pdb = nrhs;
    pdx = nrhs;
#endif
    tau_len = MIN(m,n);
    jpvt_len = n;

    /* Allocate memory */
    if ( !(a = NAG_ALLOC(m * n, double)) ||
        !(b = NAG_ALLOC(m * nrhs, double)) ||
        !(tau = NAG_ALLOC(tau_len, double)) ||
        !(x = NAG_ALLOC(m * nrhs, double)) ||
        !(jpvt = NAG_ALLOC(jpvt_len, Integer)) )
    {
        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] ");

    /* Initialize JPVT to be zero so that all columns are free */
    fl6dbc(n, 0, jpvt, 1, &fail);
    /* Compute the QR factorization of A */
    f08bec(order, m, n, a, pda, jpvt, tau, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08bec.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Choose TOL to reflect the relative accuracy of the input data */
    tol = 0.01;

    /* Determine which columns of R to use */
    for (k = 1; k <= n; ++k)
    {
        if (ABS(A(k, k)) <= tol * ABS(A(1, 1)) )
            break;
    }
    --k;

```

```

/* 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*B = C */
f07tec(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, k, 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;
  }
for (i = k + 1; i <= n; ++i)
  {
    for (j = 1; j <= nrhs; ++j)
      B(i,j) = 0.0;
  }

/* Unscramble the least-squares solution stored in B */
for (i = 1; i <= n; ++i)
  {
    for (j = 1; j <= nrhs; ++j)
      X(jpvt[i - 1], j) = B(i, j);
  }

/* Print least-squares solution */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
      "Least-squares solution", 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);
if (x) NAG_FREE(x);
if (jpvt) NAG_FREE(jpvt);
return exit_status;
}

```

9.2 Program Data

```

f08bec Example Program Data
  6 5 2                               :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
-0.01 -0.04
 0.04 -0.03
 0.05  0.01
-0.03 -0.02
 0.02  0.05
-0.06  0.07                               :End of matrix B

```

9.3 Program Results

f08bec Example Program Results

Least-squares solution

	1	2
1	-0.0370	-0.0044
2	0.0647	-0.0335
3	0.0000	0.0000
4	-0.0515	0.0018
5	0.0066	0.0102
