NAG Library Function Document

nag_dgeqrt (f08abc)

1 Purpose

nag_dgeqrt (f08abc) recursively computes, with explicit blocking, the QR factorization of a real m by n matrix.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
```

```
void nag_dgeqrt (Nag_OrderType order, Integer m, Integer n, Integer nb,
     double a[], Integer pda, double t[], Integer pdt, NagError *fail)
```

3 Description

nag_dgeqrt (f08abc) forms the QR factorization of an arbitrary rectangular real m by n matrix. No pivoting is performed.

It differs from nag_dgeqrf (f08aec) in that it: requires an explicit block size; stores reflector factors that are upper triangular matrices of the chosen block size (rather than scalars); and recursively computes the QR factorization based on the algorithm of Elmroth and Gustavson (2000).

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 upper 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 9).

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

4 References

Elmroth E and Gustavson F (2000) Applying Recursion to Serial and Parallel QR Factorization Leads to Better Performance *IBM Journal of Research and Development. (Volume 44)* **4** 605–624

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th 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: **m** – Integer *Input*

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

Constraint: $\mathbf{m} \geq 0$.

3: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

4: **nb** – Integer *Input*

On entry: the explicitly chosen block size to be used in computing the QR factorization. See Section 9 for details.

Constraints:

$\mathbf{nb} \geq 1$;
if $\min(\mathbf{m}, \mathbf{n}) > 0$, $\mathbf{nb} \leq \min(\mathbf{m}, \mathbf{n})$.

5: $\mathbf{a}[dim]$ – 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;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix A is stored in

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

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.

6: **pda** – Integer *Input*

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

7: **t**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **t** must be at least

$\max(1, \mathbf{pdt} \times \min(\mathbf{m}, \mathbf{n}))$ when **order** $=$ Nag_ColMajor;
$\max(1, \mathbf{nb} \times \mathbf{pdt})$ when **order** $=$ Nag_RowMajor.

The (i, j)th element of the matrix T is stored in

$\mathbf{t}[(j-1) \times \mathbf{pdt} + i - 1]$ when **order** $=$ Nag_ColMajor;
$\mathbf{t}[(i-1) \times \mathbf{pdt} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: further details of the orthogonal matrix Q. The number of blocks is $b = \left\lceil \frac{k}{\mathbf{nb}} \right\rceil$, where $k = \min(m, n)$ and each block is of order **nb** except for the last block, which is of order $k - (b-1) \times \mathbf{nb}$. For each of the blocks, an upper triangular block reflector factor is computed: $\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_b$. These are stored in the **nb** by n matrix T as $\boldsymbol{T} = [\boldsymbol{T}_1 | \boldsymbol{T}_2 | \ldots | \boldsymbol{T}_b]$.

8: **pdt** – Integer *Input*

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

Constraints:

if **order** $=$ Nag_ColMajor, **pdt** $\geq$ **nb**;
if **order** $=$ Nag_RowMajor, **pdt** $\geq \max(1, \min(\mathbf{m}, \mathbf{n}))$.

9: **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 ⟨*value*⟩ had an illegal value.

NE_INT

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

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

NE_INT_2

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

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

On entry, **pdt** $= \langle value \rangle$ and **nb** $= \langle value \rangle$.
Constraint: **pdt** $\geq$ **nb**.

NE_INT_3

On entry, **nb** $= \langle value \rangle$, **m** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **nb** ≥ 1 and
if $\min(\mathbf{m}, \mathbf{n}) > 0$, $\mathbf{nb} \leq \min(\mathbf{m}, \mathbf{n})$.

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

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 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 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 Parallelism and Performance

nag_dgeqrt (f08abc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

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

To apply Q to an arbitrary real rectangular matrix C, nag_dgeqrt (f08abc) may be followed by a call to nag_dgemqrt (f08acc). For example,

```
nag_dgemqrt(order,Nag_LeftSide,Nag_Trans,m,p,MIN(m,n),nb,a,pda,t,pdt,
 c,pdc,&fail)
```

forms $C = Q^{\mathrm{T}}C$, where C is m by p.

To form the orthogonal matrix Q explicitly, simply initialize the m by m matrix C to the identity matrix and form $C = QC$ using nag_dgemqrt (f08acc) as above.

The block size, **nb**, used by nag_dgeqrt (f08abc) is supplied explicitly through the interface. For moderate and large sizes of matrix, the block size can have a marked effect on the efficiency of the algorithm with the optimal value being dependent on problem size and platform. A value of $\mathbf{nb} = 64 \ll \min(m, n)$ is likely to achieve good efficiency and it is unlikely that an optimal value would exceed 340.

To compute a QR factorization with column pivoting, use nag_dtpqrt (f08bbc) or nag_dgeqpf (f08bec).

The complex analogue of this function is nag_zgeqrt (f08apc).

10 Example

This example solves the linear least squares problems

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

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

$$A = \begin{pmatrix} -0.57 & -1.28 & -0.39 & 0.25 \\ -1.93 & 1.08 & -0.31 & -2.14 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ -1.93 & 0.64 & -0.66 & 0.08 \\ 0.15 & 0.30 & 0.15 & -2.13 \\ -0.02 & 1.03 & -1.43 & 0.50 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -2.67 & 0.41 \\ -0.55 & -3.10 \\ 3.34 & -4.01 \\ -0.77 & 2.76 \\ 0.48 & -6.17 \\ 4.10 & 0.21 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dgeqrt (f08abc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double rnorm;
  Integer exit_status = 0;
  Integer pda, pdb, pdt;
  Integer i, j, m, n, nb, nrhs;
  /* Arrays */
  double *a = 0, *b = 0, *t = 0;
  /* Nag Types */
  Nag_OrderType order;
  NagError fail;

#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);

  printf("nag_dgeqrt (f08abc) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
```

```
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m, &n, &nrhs);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m, &n, &nrhs);
#endif
  nb = MIN(m, n);
  if (!(a = NAG_ALLOC(m * n, double)) ||
      !(b = NAG_ALLOC(m * nrhs, double)) ||
      !(t = NAG_ALLOC(nb * MIN(m, n), double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = m;
  pdt = nb;
#else
  pda = n;
  pdb = nrhs;
  pdt = MIN(m, n);
#endif

  /* 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_dgeqrt (f08abc).
   * Compute the QR factorization of A by recursive algorithm.
   */
  nag_dgeqrt(order, m, n, nb, a, pda, t, pdt, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgeqrt (f08abc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_dgemqrt (f08acc).
   * Compute C = (C1) = (Q^T)*B, storing the result in B
   *             (C2)
   * by applying Q^T from left.
   */
  nag_dgemqrt(order, Nag_LeftSide, Nag_Trans, m, nrhs, n, nb, a, pda, t, pdt,
              b, pdb, &fail);
```

```
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgemqrt (f08acc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }

  /* nag_dtrtrs (f07tec).
   * Compute least squares solutions by back-substitution in R*X = C1.
   */
  nag_dtrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs, a, pda,
             b, pdb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrtrs (f07tec).\n%s\n", fail.message);
    exit_status = 3;
    goto END;
  }

  /* nag_gen_real_mat_print (x04cac).
   * Print least squares solutions.
   */
  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 = 4;
    goto END;
  }

  printf("\n Square root(s) of the residual sum(s) of squares\n");
  for (j = 1; j <= nrhs; j++) {
    /* nag_dge_norm (f16rac).
     * Compute and print estimate of the square root of the residual
     * sum of squares.
     */
    nag_dge_norm(order, Nag_FrobeniusNorm, m - n, 1, &B(n + 1, j), pdb,
                 &rnorm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("\nError from nag_dge_norm (f16rac).\n%s\n", fail.message);
      exit_status = 5;
      goto END;
    }
    printf("  %11.2e ", rnorm);
  }
  printf("\n");

END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(t);

  return exit_status;
}
```

10.2 Program Data

```
nag_dgeqrt (f08abc) Example Program Data

  6      4      2           : 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 : matrix A

 -2.67   0.41
```

```
-0.55  -3.10
 3.34  -4.01
-0.77   2.76
 0.48  -6.17
 4.10   0.21               : matrix B
```

10.3 Program Results

```
nag_dgeqrt (f08abc) Example Program Results

 Least squares solution(s)
             1          2
 1      1.5339    -1.5753
 2      1.8707     0.5559
 3     -1.5241     1.3119
 4      0.0392     2.9585

 Square root(s) of the residual sum(s) of squares
     2.22e-02      1.38e-02
```
