NAG Library Function Document

nag_dggqrf (f08zec)

1 Purpose

nag_dggqrf (f08zec) computes a generalized QR factorization of a real matrix pair (A, B), where A is an n by m matrix and B is an n by p matrix.

2 Specification

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

```
void nag_dggqrf (Nag_OrderType order, Integer n, Integer m, Integer p,
     double a[], Integer pda, double taua[], double b[], Integer pdb,
     double taub[], NagError *fail)
```

3 Description

nag_dggqrf (f08zec) forms the generalized QR factorization of an n by m matrix A and an n by p matrix B

$$A = QR, \quad B = QTZ,$$

where Q is an n by n orthogonal matrix, Z is a p by p orthogonal matrix and R and T are of the form

$$R = \begin{cases} \begin{array}{r} \\ m \\ n-m \end{array} \begin{array}{c} m \\ \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{array}, & \text{if } n \geq m; \\ \begin{array}{r} \\ n \end{array} \begin{array}{c} \begin{array}{cc} n & m-n \end{array} \\ \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \end{array}, & \text{if } n < m, \end{cases}$$

with R_{11} upper triangular,

$$T = \begin{cases} \begin{array}{r} \\ n \end{array} \begin{array}{c} \begin{array}{cc} p-n & n \end{array} \\ \begin{pmatrix} 0 & T_{12} \end{pmatrix} \end{array}, & \text{if } n \leq p, \\ \begin{array}{r} \\ n-p \\ p \end{array} \begin{array}{c} p \\ \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \end{array}, & \text{if } n > p, \end{cases}$$

with T_{12} or T_{21} upper triangular.

In particular, if B is square and nonsingular, the generalized QR factorization of A and B implicitly gives the QR factorization of $B^{-1}A$ as

$$B^{-1}A = Z^{\mathrm{T}}\left(T^{-1}R\right).$$

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Anderson E, Bai Z and Dongarra J (1992) Generalized *QR* factorization and its applications *Linear Algebra Appl. (Volume 162–164)* 243–271

Hammarling S (1987) The numerical solution of the general Gauss-Markov linear model *Mathematics in Signal Processing* (eds T S Durrani, J B Abbiss, J E Hudson, R N Madan, J G McWhirter and T A Moore) 441–456 Oxford University Press

Paige C C (1990) Some aspects of generalized QR factorizations . *In Reliable Numerical Computation* (eds M G Cox and S Hammarling) 73–91 Oxford University Press

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 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: **n** – Integer *Input*

On entry: n, the number of rows of the matrices A and B.

Constraint: **n** ≥ 0.

3: **m** – Integer *Input*

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

Constraint: **m** ≥ 0.

4: **p** – Integer *Input*

On entry: p, the number of columns of the matrix B.

Constraint: **p** ≥ 0.

5: **a**[*dim*] – double *Input/Output*

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

$\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pda})$ 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 n by m matrix A.

On exit: the elements on and above the diagonal of the array contain the $\min(n, m)$ by m upper trapezoidal matrix R (R is upper triangular if $n \geq m$); the elements below the diagonal, with the array **taua**, represent the orthogonal matrix Q as a product of $\min(n, m)$ elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

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

7: **taua**[**min**(**n**, **m**)] – double *Output*

On exit: the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

8: **b**[dim] – double *Input/Output*

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

$\max(1, \mathbf{pdb} \times \mathbf{p})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

Where $\mathbf{B}(i,j)$ appears in this document, it refers to the array element

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

On entry: the n by p matrix B.

On exit: if $n \leq p$, the upper triangle of the subarray $\mathbf{B}(1:n, p-n+1:p)$ contains the n by n upper triangular matrix T_{12}.

If $n > p$, the elements on and above the $(n-p)$th subdiagonal contain the n by p upper trapezoidal matrix T; the remaining elements, with the array **taub**, represent the orthogonal matrix Z as a product of elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

9: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdb} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdb} \geq \max(1, \mathbf{p})$.

10: **taub**[**min**(**n**, **p**)] – double *Output*

On exit: the scalar factors of the elementary reflectors which represent the orthogonal matrix Z.

11: **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 ⟨*value*⟩ had an illegal value.

NE_INT

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

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

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

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

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

NE_INT_2

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

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

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

On entry, **pdb** $= \langle value \rangle$ and **p** $= \langle value \rangle$.
Constraint: **pdb** $\geq \max(1, \mathbf{p})$.

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 generalized QR factorization is the exact factorization for nearby matrices $(A + E)$ and $(B + F)$, where

$$\|E\|_2 = O\,\epsilon\|A\|_2 \quad \text{and} \quad \|F\|_2 = O\,\epsilon\|B\|_2,$$

and ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_dggqrf (f08zec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dggqrf (f08zec) 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 orthogonal matrices Q and Z may be formed explicitly by calls to nag_dorgqr (f08afc) and nag_dorgrq (f08cjc) respectively. nag_dormqr (f08agc) may be used to multiply Q by another matrix and nag_dormrq (f08ckc) may be used to multiply Z by another matrix.

The complex analogue of this function is nag_zggqrf (f08zsc).

10 Example

This example solves the general Gauss–Markov linear model problem

$$\min_x \|y\|_2 \quad \text{subject to} \quad d = Ax + By$$

where

$$A = \begin{pmatrix} -0.57 & -1.28 & -0.39 \\ -1.93 & 1.08 & -0.31 \\ 2.30 & 0.24 & -0.40 \\ -0.02 & 1.03 & -1.43 \end{pmatrix}, \quad B = \begin{pmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 2.0 & 0 \\ 0 & 0 & 0 & 5.0 \end{pmatrix} \quad \text{and} \quad d = \begin{pmatrix} 1.32 \\ -4.00 \\ 5.52 \\ 3.24 \end{pmatrix}.$$

The solution is obtained by first computing a generalized QR factorization of the matrix pair (A, B). The example illustrates the general solution process, although the above data corresponds to a simple weighted least squares problem.

10.1 Program Text

```
/* nag_dggqrf (f08zec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{
  /* Scalars */
  double        alpha, beta, rnorm;
  const double  zero = 0.0;
  Integer       i, j, m, n, nm, p, pda, pdb, pdd, pnm, zrow;
  Integer       exit_status = 0;

  /* Arrays */
  double        *a = 0, *b = 0, *d = 0, *taua = 0, *taub = 0, *y = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

#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_dggqrf (f08zec) Example Program Results\n\n");

  /* 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]", &n, &m, &p);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &m, &p);
#endif
  if (n < 0 || m < 0 || p < 0)
    {
      printf("Invalid n, m or p\n");
      exit_status = 1;
      goto END;
    }

#ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdb = n;
  pdd = n;
#else
  pda = m;
  pdb = p;
  pdd = 1;
#endif

  /* Allocate memory */
  if (!(a    = NAG_ALLOC(n*m, double)) ||
      !(b    = NAG_ALLOC(n*p, double)) ||
      !(d    = NAG_ALLOC(MAX(n, m), double)) ||
      !(taua = NAG_ALLOC(MIN(m, n), double)) ||
      !(taub = NAG_ALLOC(MIN(n, p), double)) ||
      !(y    = NAG_ALLOC(p, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A, B and d from data file */
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= m; ++j) scanf_s("%lf", &A(i, j));
#else
    for (j = 1; j <= m; ++j) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= p; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= p; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n; ++i) scanf_s("%lf", &d[i]);
#else
  for (i = 0; i < n; ++i) scanf("%lf", &d[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Compute the generalized QR factorization of (A,B) as
   * A = Q*(R),   B = Q*(T11 T12)*Z
   *       (0)          ( 0  T22)
```

```
 * using  nag_dggqrf (f08zec).
 */
nag_dggqrf(order, n, m, p, a, pda, taua, b, pdb, taub, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dggqrf (f08zec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Solve weighted least-squares problem for case n > m */
if (n <= m) goto END;

nm = n - m;
pnm = p - nm;
/* Multiply Q^T through d = Ax + By to get
 *     (c1) = Q^T * d = (R) * x + (T11 T12) * Z * (y1)
 *     (c2)             (0)       ( 0  T22)       (y2)
 * Compute C using nag_dormqr (f08agc).
 */
nag_dormqr(order, Nag_LeftSide, Nag_Trans, n, 1, m, a, pda, taua, d, pdd,
           &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dormqr (f08agc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Let Z*(y1) = (w1) and solving for w2 we have to solve the triangular sytem
 *       (y2) = (w2)
 *                     T22 * w2 = c2
 * This is done by putting c2 in y2 and backsolving to get w2 in y2.
 *
 * Copy c2 (at d[m]) into y2 using nag_dge_copy (f16qfc).
 */
nag_dge_copy(Nag_ColMajor, Nag_NoTrans, nm, 1, &d[m], n-m, &y[pnm], nm,
             &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dge_copy (f16qfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
/* Solve T22*w2 = c2 using nag_dtrtrs (f07tec).
 * T22 is stored in a submatrix of matrix B of dimension n-m by n-m
 * with first element at B(m+1,p-(n-m)+1). y2 is stored from y[p-(n-m)].
 */
nag_dtrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, nm, 1,
           &B(m + 1, pnm + 1), pdb, &y[pnm], nm, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dtrtrs (f07tec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* set w1 = 0 for minimum norm y. */
nag_dload(m + p - n, zero, y, 1, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dload.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Compute estimate of the square root of the residual sum of squares
 * norm(y) = norm(w2) with y1 = 0 using  nag_dge_norm (f16rac).
 */
nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, n-m, 1, &y[pnm],
             nm, &rnorm, &fail);
```

```
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* The top half of the system remains:
   *     (c1) = Q^T * d = (R) * x + (T11 T12) * ( 0)
   *                                            (w2)
   * =>      c1 = R * x + T12 * w2
   * =>   R * x = c1    - T12 * w2;
   *
   * first form d = c1 - T12*w2 where c1 is stored in d
   * using nag_dgemv (f16pac).
   */
  alpha = -1.0;
  beta = 1.0;
  nag_dgemv(order, Nag_NoTrans, m, nm, alpha, &B(1, pnm + 1), pdb, &y[pnm], 1,
            beta, d, 1, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dgemv (f16pac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Next, solve R * x = d for x (in d) where R is stored in leading submatrix
   * of A in a. This gives the least squares solution x in d.
   * Using nag_dtrtrs (f07tec).
   */
  nag_dtrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, m, 1, a, pda, d,
             pdd, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtrtrs (f07tec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute the minimum norm residual vector y = (Z**T)*w
   * using nag_dormrq (f08ckc).
   */
  zrow = MAX(1, n - p + 1);
  nag_dormrq(order, Nag_LeftSide, Nag_Trans, p, 1, MIN(n, p), &B(zrow, 1), pdb,
             taub, y, pdd, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dormrq (f08ckc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print least squares solution x */
  printf("Generalized least squares solution\n");
  for (i = 0; i < m; ++i) printf(" %11.4f%s", d[i], i%7 == 6?"\n":"");

  /* Print residual vector y */
  printf("\n");
  printf("\nResidual vector\n");
  for (i = 0; i < p; ++i) printf(" %10.2e%s", y[i], i%7 == 6?"\n":"");

  /* Print estimate of the square root of the residual sum of  squares. */
  printf("\n\nSquare root of the residual sum of squares\n");
  printf("%11.2e\n", rnorm);

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(d);
  NAG_FREE(taua);
```

```
  NAG_FREE(taub);
  NAG_FREE(y);

  return exit_status;
}
```

10.2 Program Data

```
nag_dggqrf (f08zec) Example Program Data

  4      3      4           : n, m and p

 -0.57  -1.28  -0.39
 -1.93   1.08  -0.31
  2.30   0.24  -0.40
 -0.02   1.03  -1.43        : matrix A

  0.50   0.00   0.00   0.00
  0.00   1.00   0.00   0.00
  0.00   0.00   2.00   0.00
  0.00   0.00   0.00   5.00 : matrix B

  1.32
 -4.00
  5.52
  3.24                      : vector d
```

10.3 Program Results

```
nag_dggqrf (f08zec) Example Program Results

Generalized least squares solution
      1.9889     -1.0058     -2.9911

Residual vector
  -6.37e-04  -2.45e-03  -4.72e-03   7.70e-03

Square root of the residual sum of squares
   9.38e-03
```