

# NAG Library Function Document

## nag\_dgelsy (f08bac)

### 1 Purpose

nag\_dgelsy (f08bac) computes the minimum norm solution to a real linear least squares problem

$$\min_x \|b - Ax\|_2$$

using a complete orthogonal factorization of  $A$ .  $A$  is an  $m$  by  $n$  matrix which may be rank-deficient. Several right-hand side vectors  $b$  and solution vectors  $x$  can be handled in a single call.

### 2 Specification

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

void nag_dgelsy (Nag_OrderType order, Integer m, Integer n, Integer nrhs,
                double a[], Integer pda, double b[], Integer pdb, Integer jpvt[],
                double rcond, Integer *rank, NagError *fail)
```

### 3 Description

The right-hand side vectors are stored as the columns of the  $m$  by  $r$  matrix  $B$  and the solution vectors in the  $n$  by  $r$  matrix  $X$ .

nag\_dgelsy (f08bac) first computes a  $QR$  factorization with column pivoting

$$AP = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$

with  $R_{11}$  defined as the largest leading sub-matrix whose estimated condition number is less than  $1/\mathbf{rcond}$ . The order of  $R_{11}$ , **rank**, is the effective rank of  $A$ .

Then,  $R_{22}$  is considered to be negligible, and  $R_{12}$  is annihilated by orthogonal transformations from the right, arriving at the complete orthogonal factorization

$$AP = Q \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z.$$

The minimum norm solution is then

$$X = PZ^T \begin{pmatrix} T_{11}^{-1} Q_1^T b \\ 0 \end{pmatrix}$$

where  $Q_1$  consists of the first **rank** columns of  $Q$ .

### 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>

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd 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:*  $m \geq 0$ .
- 3: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $A$ .  
*Constraint:*  $n \geq 0$ .
- 4: **nrhs** – Integer *Input*  
*On entry:*  $r$ , the number of right-hand sides, i.e., the number of columns of the matrices  $B$  and  $X$ .  
*Constraint:* **nrhs**  $\geq 0$ .
- 5: **a**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **a** must be at least  
 $\max(1, \mathbf{pda} \times \mathbf{n})$  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:* **a** has been overwritten by details of its complete orthogonal factorization.
- 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: **b**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **b** must be at least  
 $\max(1, \mathbf{pdb} \times \mathbf{nrhs})$  when **order** = Nag\_ColMajor;  
 $\max(1, \max(1, \mathbf{m}, \mathbf{n}) \times \mathbf{pdb})$  when **order** = Nag\_RowMajor.  
The ( $i, j$ )th element of the matrix  $B$  is stored in  
 $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:* the  $m$  by  $r$  right-hand side matrix  $B$ .  
*On exit:* the  $n$  by  $r$  solution matrix  $X$ .

- 8: **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, **pdb**  $\geq \max(1, \mathbf{m}, \mathbf{n})$ ;  
 if **order** = Nag\_RowMajor, **pdb**  $\geq \max(1, \mathbf{nrhs})$ .
- 9: **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* – 1]  $\neq 0$ , the *i*th column of *A* is permuted to the front of *AP*, otherwise column *i* is a free column.  
*On exit:* if **jpvt**[*i* – 1] = *k*, then the *i*th column of *AP* was the *k*th column of *A*.
- 10: **rcond** – double *Input*  
*On entry:* used to determine the effective rank of *A*, which is defined as the order of the largest leading triangular sub-matrix  $R_{11}$  in the *QR* factorization of *A*, whose estimated condition number is  $< 1/\mathbf{rcond}$ .  
*Suggested value:* if the condition number of **a** is not known then  $\mathbf{rcond} = \sqrt{(\epsilon)/2}$  (where  $\epsilon$  is *machine precision*, see nag\_machine\_precision (X02AJC)) is a good choice. Negative values or values less than *machine precision* should be avoided since this will cause **a** to have an effective rank =  $\min(\mathbf{m}, \mathbf{n})$  that could be larger than its actual rank, leading to meaningless results.
- 11: **rank** – Integer \* *Output*  
*On exit:* the effective rank of *A*, i.e., the order of the sub-matrix  $R_{11}$ . This is the same as the order of the sub-matrix  $T_{11}$  in the complete orthogonal factorization of *A*.
- 12: **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  $\langle value \rangle$  had an illegal value.

### NE\_INT

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

Constraint: **m**  $\geq 0$ .

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

Constraint: **n**  $\geq 0$ .

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

Constraint: **nrhs**  $\geq 0$ .

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

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, **m**).

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

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

### NE\_INT\_3

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

See Section 4.5 of Anderson *et al.* (1999) for details of error bounds.

## 8 Parallelism and Performance

nag\_dgelsy (f08bac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dgelsy (f08bac) 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 complex analogue of this function is nag\_zgelsy (f08bnc).

## 10 Example

This example solves the linear least squares problem

$$\min_x \|b - Ax\|_2$$

for the solution,  $x$ , of minimum norm, where

$$A = \begin{pmatrix} -0.09 & 0.14 & -0.46 & 0.68 & 1.29 \\ -1.56 & 0.20 & 0.29 & 1.09 & 0.51 \\ -1.48 & -0.43 & 0.89 & -0.71 & -0.96 \\ -1.09 & 0.84 & 0.77 & 2.11 & -1.27 \\ 0.08 & 0.55 & -1.13 & 0.14 & 1.74 \\ -1.59 & -0.72 & 1.06 & 1.24 & 0.34 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 7.4 \\ 4.2 \\ -8.3 \\ 1.8 \\ 8.6 \\ 2.1 \end{pmatrix}.$$

A tolerance of 0.01 is used to determine the effective rank of  $A$ .

## 10.1 Program Text

```

/* nag_dgelsy (f08bac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
    /* Scalars */
    double rcond;
    Integer exit_status = 0, i, j, m, n, nrhs, rank, pda, pdb;
    /* Arrays */
    double *a = 0, *b = 0;
    Integer *jpvt = 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_dgelsy (f08bac) 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]", &m, &n, &nrhs);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n]", &m, &n, &nrhs);
#endif

    /* Allocate memory */
    if (!(a = NAG_ALLOC(m * n, double)) ||
        !(b = NAG_ALLOC(m * nrhs, double)) || !(jpvt = NAG_ALLOC(n, Integer)))
    {

```

```

    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

#ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdb = m;
#else
    pda = n;
    pdb = nrhs;
#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_iloadd (f16dbc).
     * Initialize jpvvt to be zero so that all columns are free.
     */
    nag_iloadd(n, 0, jpvvt, 1, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_iloadd (f16dbc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

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

    /* nag_dgelsy (f08bac).
     * Solve the least squares problem  $\min(\text{norm2}(b - Ax))$  for the  $x$ 
     * of minimum norm.
     */
    nag_dgelsy(order, m, n, nrhs, a, pda, b, pdb, jpvvt, rcond, &rank, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dgelsy (f08bac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print solution */
    printf("Least squares solution\n");
    for (i = 1; i <= n; ++i) {
        for (j = 1; j <= nrhs; ++j)
            printf("%11.4f%s", B(i, j), j % 7 == 0 ? "\n" : " ");
        printf("\n");
    }
}

```

```

/* Print the effective rank of A */
printf("\nTolerance used to estimate the rank of A\n");
printf("%11.2e\n", rcond);
printf("Estimated rank of A\n%6" NAG_IFMT "\n", rank);

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

  return exit_status;
}

#undef A
#undef B

```

## 10.2 Program Data

nag\_dgelsy (f08bac) Example Program Data

```

  6      5      1                :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
  4.2
 -8.3
  1.8
  8.6
  2.1                :End of vector b

```

## 10.3 Program Results

nag\_dgelsy (f08bac) Example Program Results

Least squares solution

```

  0.6344
  0.9699
 -1.4402
  3.3678
  3.3992

```

Tolerance used to estimate the rank of A

```

  1.00e-02

```

Estimated rank of A

```

  4

```

---