# NAG Library Function Document

# nag_dgelsd (f08kcc)

## 1 Purpose

nag_dgelsd (f08kcc) computes the minimum norm solution to a real linear least squares problem

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

## 2 Specification

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

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

## 3 Description

nag_dgelsd (f08kcc) uses the singular value decomposition (SVD) of $A$, where $A$ is a real $m$ 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; they are stored as the columns of the $m$ by $r$ right-hand side matrix $B$ and the $n$ by $r$ solution matrix $X$.

The problem is solved in three steps:

1. reduce the coefficient matrix $A$ to bidiagonal form with Householder transformations, reducing the original problem into a 'bidiagonal least squares problem' (BLS);
2. solve the BLS using a divide-and-conquer approach;
3. apply back all the Householder transformations to solve the original least squares problem.

The effective rank of $A$ is determined by treating as zero those singular values which are less than **rcond** times the largest singular value.

## 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 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 *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: **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*: $\mathbf{nrhs} \geq 0$.

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 $n$ coefficient matrix $A$.

*On exit*: the contents of **a** are destroyed.

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

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*: **b** is overwritten by the $n$ by $r$ solution matrix $X$. If $m \geq n$ and $\mathbf{rank} = n$, the residual sum of squares for the solution in the $i$th column is given by the sum of squares of elements $n+1, \ldots, m$ in that column.

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: **s**[$dim$] – double *Output*

**Note**: the dimension, *dim*, of the array **s** must be at least $\max(1, \min(\mathbf{m}, \mathbf{n}))$.

*On exit*: the singular values of $A$ in decreasing order.

10: **rcond** – double *Input*

*On entry*: used to determine the effective rank of $A$. Singular values $\mathbf{s}[i-1] \leq \mathbf{rcond} \times \mathbf{s}[0]$ are treated as zero. If $\mathbf{rcond} < 0$, ***machine precision*** is used instead.

11: **rank** – Integer * *Output*

*On exit*: the effective rank of $A$, i.e., the number of singular values which are greater than $\mathbf{rcond} \times \mathbf{s}[0]$.

12: **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 $\langle value \rangle$ had an illegal value.

**NE_CONVERGENCE**

The algorithm for computing the SVD failed to converge; $\langle value \rangle$ off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

**NE_INT**

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{nrhs} = \langle value \rangle$.
Constraint: $\mathbf{nrhs} \geq 0$.

On entry, $\mathbf{pda} = \langle value \rangle$.
Constraint: $\mathbf{pda} > 0$.

On entry, $\mathbf{pdb} = \langle value \rangle$.
Constraint: $\mathbf{pdb} > 0$.

**NE_INT_2**

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

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

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

**NE_INT_3**

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

**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 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

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

## 8 Parallelism and Performance

nag_dgelsd (f08kcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dgelsd (f08kcc) 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_zgelsd (f08kqc).

## 10 Example

This example solves the linear least squares problem

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

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

$$A = \begin{pmatrix} -0.09 & -1.56 & -1.48 & -1.09 & 0.08 & -1.59 \\ 0.14 & 0.20 & -0.43 & 0.84 & 0.55 & -0.72 \\ -0.46 & 0.29 & 0.89 & 0.77 & -1.13 & 1.06 \\ 0.68 & 1.09 & -0.71 & 2.11 & 0.14 & 1.24 \\ 1.29 & 0.51 & -0.96 & -1.27 & 1.74 & 0.34 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 7.4 \\ 4.3 \\ -8.1 \\ 1.8 \\ 8.7 \end{pmatrix}.$$

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

### 10.1 Program Text

```
/* nag_dgelsd (f08kcc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagf08.h>
#include <nag_stdlib.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, *s = 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_dgelsd (f08kcc) 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
  if (m < 0 || n < 0 || nrhs < 0)
    {
      printf("Invalid m, n or nrhs\n");
      exit_status = 1;
      goto END;
    }

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

```
#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = MAX(m, n);
#else
  pda = n;
  pdb = nrhs;
#endif

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

  for (i = 1; i <= m; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  rcond = 0.01;
  nag_dgelsd(order, m, n, 1, a, pda, b, pdb, s, rcond, &rank, &fail);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dgelsd (f08kcc).\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(" %10.4f%s", B(i, j), j%7==0?"\n":"");
      printf("\n");
    }

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

  /* Print singular values of A */
  printf("\nSingular values of A\n");
  for (i = 0; i < m; ++i) printf(" %10.4f%s", s[i], i%7 == 6?"\n":"");
  printf("\n");

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

  return exit_status;
}
#undef A
#undef B
```

### 10.2 Program Data

```
nag_dgelsd (f08kcc) Example Program Data

  5      6      1                          : m, n, nrhs

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

  7.4
  4.3
 -8.1
  1.8
  8.7                                      : vector b
```

### 10.3 Program Results

```
nag_dgelsd (f08kcc) Example Program Results

Least squares solution
     1.5938
    -0.1180
    -3.1501
     0.1554
     2.5529
    -1.6730


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

Estimated rank of A
     4

Singular values of A
     3.9997     2.9962     2.0001     0.9988     0.0025
```