NAG Library Function Document

nag_zgelsy (f08bnc)

1 Purpose

nag_zgelsy (f08bnc) computes the minimum norm solution to a complex 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_zgelsy (Nag_OrderType order, Integer m, Integer n, Integer nrhs,
     Complex a[], Integer pda, Complex 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_zgelsy (f08bnc) 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 $1/$**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

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

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 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: **m** ≥ 0.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 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** ≥ 0.

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

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

max$(1,$ **pda** $\times$ **n**$)$ when **order** $=$ Nag_ColMajor;
max$(1,$ **m** $\times$ **pda**$)$ when **order** $=$ Nag_RowMajor.

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

a$[(j-1) \times$ **pda** $+ i - 1]$ when **order** $=$ Nag_ColMajor;
a$[(i-1) \times$ **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** $\geq \max(1,$ **m**$)$;
if **order** $=$ Nag_RowMajor, **pda** $\geq \max(1,$ **n**$)$.

7: **b**[*dim*] – Complex *Input/Output*

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

max$(1,$ **pdb** $\times$ **nrhs**$)$ when **order** $=$ Nag_ColMajor;
max$(1, \max(1,$ **m**, **n**$) \times$ **pdb**$)$ when **order** $=$ Nag_RowMajor.

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

b$[(j-1) \times$ **pdb** $+ i - 1]$ when **order** $=$ Nag_ColMajor;
b$[(i-1) \times$ **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 ith 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 ith column of AP was the kth 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 $< 1/$**rcond**.

Suggested value: if the condition number of **a** is not known then **rcond** $= \sqrt{(\epsilon)/2}$ (where ϵ 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 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_INT

On entry, **m** $= \langle value \rangle$.
Constraint: **m** ≥ 0.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

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

NE_INT_3

On entry, **pdb** = ⟨*value*⟩, **m** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
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 of error bounds.

8 Parallelism and Performance

nag_zgelsy (f08bnc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgelsy (f08bnc) 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 real analogue of this function is nag_dgelsy (f08bac).

10 Example

This example solves the linear least squares problem

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

for the solution, x, of minimum norm, where

$$A = \begin{pmatrix} 0.47 - 0.34i & -0.40 + 0.54i & 0.60 + 0.01i & 0.80 - 1.02i \\ -0.32 - 0.23i & -0.05 + 0.20i & -0.26 - 0.44i & -0.43 + 0.17i \\ 0.35 - 0.60i & -0.52 - 0.34i & 0.87 - 0.11i & -0.34 - 0.09i \\ 0.89 + 0.71i & -0.45 - 0.45i & -0.02 - 0.57i & 1.14 - 0.78i \\ -0.19 + 0.06i & 0.11 - 0.85i & 1.44 + 0.80i & 0.07 + 1.14i \end{pmatrix}$$

and

$$b = \begin{pmatrix} -1.08 - 2.59i \\ -2.61 - 1.49i \\ 3.13 - 3.61i \\ 7.33 - 8.01i \\ 9.12 + 7.63i \end{pmatrix}.$$

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

10.1 Program Text

```
/* nag_zgelsy (f08bnc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#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, pda, pdb, rank;
  /* Arrays */
  Complex       *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_zgelsy (f08bnc) 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, Complex)) ||
      !(b = NAG_ALLOC(m * nrhs, Complex)) ||
      !(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 , %lf )", &A(i, j).re, &A(i, j).im);
#else
      scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#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 , %lf )", &B(i, j).re, &B(i, j).im);
#else
      scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* nag_iload (f16dbc).
   * Initialize jpvt to be zero so that all columns are free.
   */
  nag_iload(n, 0, jpvt, 1, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_iload (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_zgelsy (f08bnc).
   * Solve the least squares problem min( norm2(b - Ax) ) for the x
   * of minimum norm.
   */
  nag_zgelsy(order, m, n, nrhs, a, pda, b, pdb, jpvt, rcond, &rank, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgelsy (f08bnc).\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("(%7.4f, %7.4f)%s", B(i, j).re, B(i, j).im, j%4 == 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");
  printf("%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_zgelsy (f08bnc) Example Program Data

   5            4            1                        :Values of m, n and nrhs

 ( 0.47,-0.34) (-0.40, 0.54) ( 0.60, 0.01) ( 0.80,-1.02)
 (-0.32,-0.23) (-0.05, 0.20) (-0.26,-0.44) (-0.43, 0.17)
 ( 0.35,-0.60) (-0.52,-0.34) ( 0.87,-0.11) (-0.34,-0.09)
 ( 0.89, 0.71) (-0.45,-0.45) (-0.02,-0.57) ( 1.14,-0.78)
 (-0.19, 0.06) ( 0.11,-0.85) ( 1.44, 0.80) ( 0.07, 1.14) :End of matrix A

 (-1.08,-2.59)
 (-2.61,-1.49)
 ( 3.13,-3.61)
 ( 7.33,-8.01)
 ( 9.12, 7.63)                                         :End of vector b
```

10.3 Program Results

```
nag_zgelsy (f08bnc) Example Program Results

Least squares solution
( 1.1669, -3.3224)
( 1.3486,  5.5027)
( 4.1764,  2.3435)
( 0.6467,  0.0107)

Tolerance used to estimate the rank of A
   1.00e-02
Estimated rank of A
     3
```