

NAG Library Function Document

nag_zgelss (f08knc)

1 Purpose

nag_zgelss (f08knc) computes the minimum norm solution to a complex linear least squares problem

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

2 Specification

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

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

3 Description

nag_zgelss (f08knc) uses the singular value decomposition (SVD) of A , where 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; 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 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: $m \geq 0$.

- 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** ≥ 0 .
- 5: **a**[*dim*] – Complex *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: the first $\min(m, n)$ rows of A are overwritten with its right singular vectors, stored row-wise.
- 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*] – Complex *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 **rank** = n , the residual sum of squares for the solution in the i th column is given by the sum of squares of the modulus of elements $n+1, \dots, m$ 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 $s[i - 1] \leq \mathbf{rcond} \times s[0]$ are treated as zero. If **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 $\mathbf{rcond} \times s[0]$.

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, **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 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_zgelss (f08knc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgelss (f08knc) 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_dgelss (f08kac).

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_zgelss (f08knc) 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, rnorm;
    Integer     exit_status = 0, i, j, m, n, nrhs, rank, pda, pdb;

    /* Arrays */
    Complex     *a = 0, *b = 0;
    double      *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_zgelss (f08knc) 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, Complex)) ||
        !(b = NAG_ALLOC(MAX(m, n) * nrhs, Complex)) ||
        !(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)
        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
#endif

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

    /* Solve the least squares problem min( norm2(b - Ax) ) for the x
     * of minimum norm.
     * nag_zgelss (f08knc).
     */
    nag_zgelss(order, m, n, nrhs, a, pda, b, pdb, s, rcond, &rank, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zgelss (f08knc).\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%11.2e\n", rcond);

    printf("Estimated rank of A\n%6"NAG_IFMT"\n", rank);

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

    /* Compute and print estimate of the square root of the residual sum of

```

```

    * squares using nag_zge_norm (f16uac) with Frobenius norm.*/
    if (rank == n)
    {
        nag_zge_norm(order, Nag_FrobeniusNorm, m - n, 1, &B(n + 1, 1), pdb,
                    &rnorm, &fail);
        printf("\nSquare root of the residual sum of squares\n%11.2e\n", rnorm);
    }

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

    return exit_status;
}
#undef A
#undef B

```

10.2 Program Data

nag_zgelss (f08knc) Example Program Data

```

    5           4           1           : m, n, 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) : matrix A

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

```

10.3 Program Results

nag_zgelss (f08knc) Example Program Results

Least squares solution

```

( 1.1673, -3.3222)
( 1.3480,  5.5028)
( 4.1762,  2.3434)
( 0.6465,  0.0105)

```

Tolerance used to estimate the rank of A

```
1.00e-02
```

Estimated rank of A

```
3
```

Singular values of A

```
2.9979    1.9983    1.0044    0.0064
```
