NAG Library Function Document

nag_zgglse (f08znc)

1 Purpose

nag_zgglse (f08znc) solves a complex linear equality-constrained least squares problem.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zgglse (Nag_OrderType order, Integer m, Integer n, Integer p,
     Complex a[], Integer pda, Complex b[], Integer pdb, Complex c[],
     Complex d[], Complex x[], NagError *fail)
```

3 Description

nag_zgglse (f08znc) solves the complex linear equality-constrained least squares (LSE) problem

$$\underset{x}{\text{minimize}} \|c - Ax\|_2 \quad \text{subject to} \quad Bx = d$$

where A is an m by n matrix, B is a p by n matrix, c is an m element vector and d is a p element vector. It is assumed that $p \leq n \leq m + p$, $\text{rank}(B) = p$ and $\text{rank}(\text{E}) = n$, where $E = \begin{pmatrix} A \\ B \end{pmatrix}$. These conditions ensure that the LSE problem has a unique solution, which is obtained using a generalized RQ factorization of the matrices B and A.

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

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

EldÉn L (1980) Perturbation theory for the least squares problem with linear equality constraints *SIAM J. Numer. Anal.* **17** 338–350

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** ≥ 0.

3: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

4: **p** – Integer *Input*

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

Constraint: $0 \leq \mathbf{p} \leq \mathbf{n} \leq \mathbf{m} + \mathbf{p}$.

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: **a** is overwritten.

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] – Complex *Input/Output*

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

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

On exit: **b** is overwritten.

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

9: **c**[**m**] – Complex *Input/Output*

On entry: the right-hand side vector c for the least squares part of the LSE problem.

On exit: the residual sum of squares for the solution vector x is given by the sum of squares of elements $\mathbf{c}[\mathbf{n} - \mathbf{p}], \mathbf{c}[\mathbf{n} - \mathbf{p} + 1], \ldots, \mathbf{c}[\mathbf{m} - 1]$; the remaining elements are overwritten.

10: **d**[**p**] – Complex *Input/Output*

On entry: the right-hand side vector d for the equality constraints.

On exit: **d** is overwritten.

11: **x**[**n**] – Complex *Output*

On exit: the solution vector x of the LSE problem.

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 2.3.1.2 in How to Use the NAG Library and its Documentation 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, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

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

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

NE_INT_3

On entry, **p** = ⟨*value*⟩, **m** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: $0 \leq \mathbf{p} \leq \mathbf{n} \leq \mathbf{m} + \mathbf{p}$.

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 2.7.6 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 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_SINGULAR

The $(N-P)$ by $(N-P)$ part of the upper trapezoidal factor T associated with A in the generalized RQ factorization of the pair (B,A) is singular, so that the rank of the matrix (E) comprising the rows of A and B is less than n; the least squares solutions could not be computed.

The upper triangular factor R associated with B in the generalized RQ factorization of the pair (B,A) is singular, so that rank$(B) < p$; the least squares solution could not be computed.

7 Accuracy

For an error analysis, see Anderson *et al.* (1992) and EldÉn (1980). See also Section 4.6 of Anderson *et al.* (1999).

8 Parallelism and Performance

nag_zgglse (f08znc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgglse (f08znc) 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

When $m \geq n = p$, the total number of real floating-point operations is approximately $\frac{8}{3}n^2(6m+n)$; if $p \ll n$, the number reduces to approximately $\frac{8}{3}n^2(3m-n)$.

10 Example

This example solves the least squares problem

$$\underset{x}{\text{minimize}} \|c - Ax\|_2 \quad \text{subject to} \quad Bx = d$$

where

$$c = \begin{pmatrix} -2.54 + 0.09i \\ 1.65 - 2.26i \\ -2.11 - 3.96i \\ 1.82 + 3.30i \\ -6.41 + 3.77i \\ 2.07 + 0.66i \end{pmatrix},$$

and

$$A = \begin{pmatrix} 0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\ -0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\ 0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\ 0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\ 0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\ 1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i \end{pmatrix},$$

$$B = \begin{pmatrix} 1.0 + 0.0i & 0 & -1.0 + 0.0i & 0 \\ 0 & 1.0 + 0.0i & 0 & -1.0 + 0.0i \end{pmatrix}$$

and

$$d = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

The constraints $Bx = d$ correspond to $x_1 = x_3$ and $x_2 = x_4$.

10.1 Program Text

```
/* nag_zgglse (f08znc) 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 rnorm;
  Integer i, j, m, n, p, pda, pdb;
  Integer exit_status = 0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex *a = 0, *b = 0, *c = 0, *d = 0, *x = 0;

#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_zgglse (f08znc) 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, &p);
```

```
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &m, &n, &p);
#endif

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

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m * n, Complex)) ||
      !(b = NAG_ALLOC(p * n, Complex)) ||
      !(c = NAG_ALLOC(m, Complex)) ||
      !(d = NAG_ALLOC(p, Complex)) || !(x = NAG_ALLOC(n, Complex)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read A, B, C and D 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 <= p; ++i) {
    for (j = 1; j <= n; ++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

  for (i = 1; i <= m; ++i)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &c[i - 1].re, &c[i - 1].im);
#else
    scanf(" ( %lf , %lf )", &c[i - 1].re, &c[i - 1].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 1; i <= p; ++i)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &d[i - 1].re, &d[i - 1].im);
#else
    scanf(" ( %lf , %lf )", &d[i - 1].re, &d[i - 1].im);
#endif
#ifdef _WIN32
```

```
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Solve the equality-constrained least squares problem    */
  /* minimize ||c - A*x|| (in the 2-norm) subject to B*x = D */
  nag_zgglse(order, m, n, p, a, pda, b, pdb, c, d, x, &fail);

  if (fail.code == NE_NOERROR) {
    /* Print least squares solution */
    printf("%s\n", "Constrained least squares solution");
    for (i = 1; i <= n; ++i)
      printf("(%7.4f, %7.4f)%s", x[i - 1].re, x[i - 1].im,
             i % 4 == 0 || i == n ? "\n" : " ");

    /* Compute the square root of the residual sum of squares */
    nag_zge_norm(Nag_ColMajor, Nag_FrobeniusNorm, 1, m - n + p, &c[n - p], 1,
                 &rnorm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    printf("\nSquare root of the residual sum of squares\n");
    printf("%11.2e\n", rnorm);
  }
  else {
    printf("Error from nag_zgglse (f08znc).\n%s\n", fail.message);
    exit_status = 1;
  }

END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(c);
  NAG_FREE(d);
  NAG_FREE(x);

  return exit_status;
}
```

10.2 Program Data

```
nag_zgglse (f08znc) Example Program Data

  6             4             2                          :Values of M, N and P

( 0.96,-0.81) (-0.03, 0.96) (-0.91, 2.06) (-0.05, 0.41)
(-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42) (-0.81, 0.56)
( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17) (-1.11, 0.60)
( 0.37, 0.38) ( 0.19,-0.54) (-0.98,-0.36) ( 0.22,-0.20)
( 0.83, 0.51) ( 0.20, 0.01) (-0.17,-0.46) ( 1.47, 1.59)
( 1.08,-0.28) ( 0.20,-0.12) (-0.07, 1.23) ( 0.26, 0.26) :End of matrix A

( 1.00, 0.00) ( 0.00, 0.00) (-1.00, 0.00) ( 0.00, 0.00)
( 0.00, 0.00) ( 1.00, 0.00) ( 0.00, 0.00) (-1.00, 0.00) :End of matrix B

(-2.54, 0.09)
( 1.65,-2.26)
(-2.11,-3.96)
( 1.82, 3.30)
(-6.41, 3.77)
( 2.07, 0.66)                                           :End of vector c

( 0.00, 0.00)
( 0.00, 0.00)                                           :End of vector d
```

10.3 Program Results

```
nag_zgglse (f08znc) Example Program Results

Constrained least squares solution
( 1.0874, -1.9621) (-0.7409,  3.7297) ( 1.0874, -1.9621) (-0.7409,  3.7297)

Square root of the residual sum of squares
   1.59e-01
```