NAG Library Function Document

nag_zggglm (f08zpc)

1 Purpose

nag_zggglm (f08zpc) solves a complex general Gauss–Markov linear (least squares) model problem.

2 Specification

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

void nag_zggglm (Nag_OrderType order, Integer m, Integer n, Integer p,
    Complex a[], Integer pda, Complex b[], Integer pdb, Complex d[],
    Complex x[], Complex y[], NagError *fail)
```

3 Description

nag_zggglm (f08zpc) solves the complex general Gauss–Markov linear model (GLM) problem

```
minimize \|y\|_2 \quad \text{subject to} \quad d = Ax + By
```

where $A$ is an $m$ by $n$ matrix, $B$ is an $m$ by $p$ matrix and $d$ is an $m$ element vector. It is assumed that

$n \leq m \leq n + p$, rank($A$) = $n$ and rank($E$) = $m$, where $E = (A \ B)$. Under these assumptions, the problem has a unique solution $x$ and a minimal 2-norm solution $y$, which is obtained using a generalized QR factorization of the matrices $A$ and $B$.

In particular, if the matrix $B$ is square and nonsingular, then the GLM problem is equivalent to the weighted linear least squares problem

```
\minimize_x \|B^{-1}(d - Ax)\|_2.
```

4 References


5 Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{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: \textbf{m} – Integer \hspace{1cm} \textit{Input}

On entry: $m$, the number of rows of the matrices $A$ and $B$.

Constraint: $m \geq 0$. 
3: \(\text{n} \) – Integer

*Input*

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

*Constraint*: \(0 \leq n \leq m\).

4: \(\text{p} \) – Integer

*Input*

On entry: \(p\), the number of columns of the matrix \(B\).

*Constraint*: \(p \geq m - n\).

5: \(\text{a}[\text{dim}] \) – Complex

*Input/Output*

*Note*: the dimension, \(\text{dim}\), of the array \(\text{a}\) must be at least

\[
\max(1, \text{pda} \times n) \text{ when } \text{order} = \text{Nag_ColMajor};
\max(1, m \times \text{pda}) \text{ when } \text{order} = \text{Nag_RowMajor}.
\]

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

\[
\begin{align*}
\text{a}[(j - 1) \times \text{pda} + i - 1] & \text{ when } \text{order} = \text{Nag_ColMajor}; \\
\text{a}[(i - 1) \times \text{pda} + j - 1] & \text{ when } \text{order} = \text{Nag_RowMajor}.
\end{align*}
\]

*On entry*: the \(m\) by \(n\) matrix \(A\).

*On exit*: \(\text{a}\) is overwritten.

6: \(\text{pda} \) – Integer

*Input*

On entry: the stride separating row or column elements (depending on the value of \text{order}) in the array \(\text{a}\).

*Constraints*:

\[
\begin{align*}
\text{if } \text{order} = \text{Nag_ColMajor}, & \text{ pda} \geq \max(1, m); \\
\text{if } \text{order} = \text{Nag_RowMajor}, & \text{ pda} \geq \max(1, n).
\end{align*}
\]

7: \(\text{b}[\text{dim}] \) – Complex

*Input/Output*

*Note*: the dimension, \(\text{dim}\), of the array \(\text{b}\) must be at least

\[
\max(1, \text{pdb} \times p) \text{ when } \text{order} = \text{Nag_ColMajor};
\max(1, m \times \text{pdb}) \text{ when } \text{order} = \text{Nag_RowMajor}.
\]

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

\[
\begin{align*}
\text{b}[(j - 1) \times \text{pdb} + i - 1] & \text{ when } \text{order} = \text{Nag_ColMajor}; \\
\text{b}[(i - 1) \times \text{pdb} + j - 1] & \text{ when } \text{order} = \text{Nag_RowMajor}.
\end{align*}
\]

*On entry*: the \(m\) by \(p\) matrix \(B\).

*On exit*: \(\text{b}\) is overwritten.

8: \(\text{pdb} \) – Integer

*Input*

On entry: the stride separating row or column elements (depending on the value of \text{order}) in the array \(\text{b}\).

*Constraints*:

\[
\begin{align*}
\text{if } \text{order} = \text{Nag_ColMajor}, & \text{ pdb} \geq \max(1, m); \\
\text{if } \text{order} = \text{Nag_RowMajor}, & \text{ pdb} \geq \max(1, p).
\end{align*}
\]

9: \(\text{d}[\text{m}] \) – Complex

*Input/Output*

On entry: the left-hand side vector \(d\) of the GLM equation.

On exit: \(\text{d}\) is overwritten.
10: \( \mathbf{x}[\mathbf{n}] \) – Complex

On exit: the solution vector \( \mathbf{x} \) of the GLM problem.

11: \( \mathbf{y}[\mathbf{p}] \) – Complex

On exit: the solution vector \( \mathbf{y} \) of the GLM problem.

12: \( \text{fail} \) – NagError *

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 \( \langle \text{value} \rangle \) had an illegal value.

NE_INT
On entry, \( \mathbf{m} = \langle \text{value} \rangle \).
Constraint: \( \mathbf{m} \geq 0 \).

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

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

NE_INT_2
On entry, \( \mathbf{m} = \langle \text{value} \rangle \) and \( \mathbf{n} = \langle \text{value} \rangle \).
Constraint: \( 0 \leq \mathbf{n} \leq \mathbf{m} \).

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

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

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

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

NE_INT_3
On entry, \( \mathbf{p} = \langle \text{value} \rangle \), \( \mathbf{m} = \langle \text{value} \rangle \) and \( \mathbf{n} = \langle \text{value} \rangle \).
Constraint: \( \mathbf{p} \geq \mathbf{m} - \mathbf{n} \).

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.
The bottom \((N - M)\) by \((N - M)\) part of the upper trapezoidal factor \(T\) associated with \(B\) in the generalized QR factorization of the pair \((A, B)\) is singular, so that \(\text{rank}(A \quad B) < n\); the least squares solutions could not be computed.

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 \(\text{rank}(B \quad A) < n\); the least squares solutions could not be computed.

7 Accuracy

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

8 Parallelism and Performance

\texttt{nag_zggglm (f08zpc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag_zggglm (f08zpc)} 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 \(p = m \geq n\), the total number of real floating-point operations is approximately \(\frac{8}{3}(2m^3 - n^3) + 16nm^2\); when \(p = m = n\), the total number of real floating-point operations is approximately \(56m^3\).

10 Example

This example solves the weighted least squares problem

\[
\text{minimize}_{x} \|B^{-1}(d - Ax)\|_2,
\]

where

\[
B = \begin{pmatrix}
0.5 - 1.0i & 2.0 - 3.0i \\
1.0 - 2.0i & 5.0 - 4.0i
\end{pmatrix},
\]

\[
d = \begin{pmatrix}
6.00 - 0.40i \\
-5.27 + 0.90i \\
2.72 - 2.13i \\
-1.30 - 2.80i
\end{pmatrix},
\]

and
\[
A = \\
\begin{pmatrix}
0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i \\
-0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i \\
0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i \\
1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i \\
\end{pmatrix}.
\]

### 10.1 Program Text

```c
/* nag_zggglm (f08zpc) Example Program. *
* Copyright 2014 Numerical Algorithms Group.
* Mark 9, 2009. */

#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, *d = 0, *x = 0, *y = 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 ]
    #else
    #define A(I, J) a[(I-1)*pda +J-1 ]
    #define B(I, J) b[(I-1)*pdb +J-1 ]
    #endif

    INIT_FAIL(fail);
    printf("nag_zggglm (f08zpc) 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] ", &n, &m, &p);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &n, &m, &p);
    #endif

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

    /* Allocate memory */
    if (!(a = NAG_ALLOC(n*m, Complex)) ||
        !(b = NAG_ALLOC(n*p, Complex)) ||
        !(d = NAG_ALLOC(n, Complex)) ||
        !(x = NAG_ALLOC(p, Complex)) ||
        !(y = NAG_ALLOC(p, Complex)))
        fail = Nag_Fail;
    #endif
```

Mark 25
!(d = NAG_ALLOC(n, Complex)) ||!
!(x = NAG_ALLOC(m, Complex)) ||!
!(y = NAG_ALLOC(p, Complex))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Read A, B and D from data file */
for (i = 1; i <= n; ++i)
{
  for (j = 1; j <= m; ++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 <= n; ++i)
  #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 )", &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 weighted least-squares problem */
    /* minimize ||inv(B)*(d - A*x)|| (in the 2-norm) */
    nag_zggglm(order, n, m, p, a, pda, b, pdb, d, x, y, &fail);
    if (fail.code == NE_NOERROR)
    {
      /* Print least-squares solution */
      printf("Weighted least-squares solution\n");
      for (i = 1; i <= m; ++i)
        printf("(%9.4f, %9.4f)%s", x[i - 1].re, x[i - 1].im,
                   i%3 == 0 || i == m?"\n":" ");
      /* Print residual vector y = inv(B)*(d - A*x) */
      printf("\nResidual vector\n");
      for (i = 1; i <= p; ++i)
        printf("(%11.2e, %11.2e)%s", y[i - 1].re, y[i - 1].im,
                   i%3 == 0 || i == p?"\n":" ");
      /* Compute and print the square root of the residual sum of */
/* squares */

nag_zge_norm(Nag_ColMajor, Nag_FrobeniusNorm, 1, p, y, 1, &rnorm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zge_norm (f16uac).\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("Square root of the residual sum of squares\n");
printf("%11.2e\n", rnorm);
}
else
{
    printf("Error from nag_zggglm (f08zpc).\n", fail.message);
    exit_status = 1;
}

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

return exit_status;

10.2 Program Data

nag_zggglm (f08zpc) Example Program Data

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0.96, -0.81)</td>
<td>(-0.03, 0.96)</td>
<td>(-0.91, 2.06)</td>
</tr>
<tr>
<td>(-0.98, 1.98)</td>
<td>(-1.20, 0.19)</td>
<td>(-0.66, 0.42)</td>
</tr>
<tr>
<td>( 0.62, -0.46)</td>
<td>( 1.01, 0.02)</td>
<td>( 0.63, -0.17)</td>
</tr>
<tr>
<td>( 1.08, -0.28)</td>
<td>( 0.20, -0.12)</td>
<td>(-0.07, 1.23)</td>
</tr>
</tbody>
</table>

:End of matrix A

| ( 0.50, -1.00) | ( 0.00, 0.00) | ( 0.00, 0.00) | ( 0.00, 0.00) |
| ( 0.00, 0.00) | ( 1.00, -2.00) | ( 0.00, 0.00) | ( 0.00, 0.00) |
| ( 0.00, 0.00) | ( 0.00, 0.00) | ( 2.00, -3.00) | ( 0.00, 0.00) |
| ( 0.00, 0.00) | ( 0.00, 0.00) | ( 0.00, 0.00) | ( 5.00, -4.00) |

:End of matrix B

| ( 6.00, -0.40) |
| (-5.27, 0.90) |
| ( 2.72, -2.13) |
| (-1.30, -2.80) |

:End of vector d

10.3 Program Results

nag_zggglm (f08zpc) Example Program Results

Weighted least-squares solution

| ( -0.9846, 1.9950) | ( 3.9929, -4.9748) | ( -3.0026, 0.9994) |

Residual vector

| ( 1.26e-04, -4.66e-04) | ( 1.11e-03, -8.61e-04) | ( 3.84e-03, -1.82e-03) |
| ( 2.03e-03, 3.02e-03) |

Square root of the residual sum of squares

5.79e-03