NAG Library Function Document

nag_dggglm (f08zbc)

1 Purpose
nag_dggglm (f08zbc) solves a real general Gauss–Markov linear (least squares) model problem.

2 Specification
#include <nag.h>
#include <nagf08.h>

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

3 Description
nag_dggglm (f08zbc) solves the real general Gauss–Markov linear model (GLM) problem

\[
\minimize_x \|y\|_2 \quad \text{subject to} \quad d = Ax + By
\]

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 \(n \leq m \leq n + p\), \(\text{rank}(A) = n\) and \(\text{rank}(E) = m\), 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
\textit{Input}

\textit{On entry:} the \textbf{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 \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{m} – Integer
\textit{Input}

\textit{On entry:} \(m\), the number of rows of the matrices \(A\) and \(B\).

\textit{Constraint:} \(m \geq 0\).
3:  \textbf{n} – Integer  
\textit{Input}

\textit{On entry:} \( n \), the number of columns of the matrix \( A \).
\textit{Constraint:} \( 0 \leq n \leq m \).

4:  \textbf{p} – Integer  
\textit{Input}

\textit{On entry:} \( p \), the number of columns of the matrix \( B \).
\textit{Constraint:} \( p \geq m - n \).

5:  \textbf{a}[\textit{dim}] – double  
\textit{Input/Output}

\textit{Note:} the dimension, \( \textit{dim} \), of the array \( a \) must be at least
\( \max(1, pda \times n) \) when \( \text{order} = \text{Nag\_ColMajor} \);
\( \max(1, m \times pda) \) when \( \text{order} = \text{Nag\_RowMajor} \).

The \((i, j)\)th element of the matrix \( A \) is stored in
\( a[(j - 1) \times pda + i - 1] \) when \( \text{order} = \text{Nag\_ColMajor} \);
\( a[(i - 1) \times pda + j - 1] \) when \( \text{order} = \text{Nag\_RowMajor} \).

\textit{On entry:} the \( m \) by \( n \) matrix \( A \).
\textit{On exit:} \( a \) is overwritten.

6:  \textbf{pda} – Integer  
\textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( a \).
\textit{Constraints:}
\if \text{order} = \text{Nag\_ColMajor}, \ \text{pda} \geq \max(1, m); \n\else \ \text{pda} \geq \max(1, n). \fi

7:  \textbf{b}[\textit{dim}] – double  
\textit{Input/Output}

\textit{Note:} the dimension, \( \textit{dim} \), of the array \( b \) must be at least
\( \max(1, pdb \times p) \) when \( \text{order} = \text{Nag\_ColMajor} \);
\( \max(1, m \times pdb) \) when \( \text{order} = \text{Nag\_RowMajor} \).

The \((i, j)\)th element of the matrix \( B \) is stored in
\( b[(j - 1) \times pdb + i - 1] \) when \( \text{order} = \text{Nag\_ColMajor} \);
\( b[(i - 1) \times pdb + j - 1] \) when \( \text{order} = \text{Nag\_RowMajor} \).

\textit{On entry:} the \( m \) by \( p \) matrix \( B \).
\textit{On exit:} \( b \) is overwritten.

8:  \textbf{pdb} – Integer  
\textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( b \).
\textit{Constraints:}
\if \text{order} = \text{Nag\_ColMajor}, \ \text{pdb} \geq \max(1, m); \n\else \ \text{pdb} \geq \max(1, p). \fi

9:  \textbf{d}[m] – double  
\textit{Input/Output}

\textit{On entry:} the left-hand side vector \( d \) of the GLM equation.
\textit{On exit:} \( d \) is overwritten.
10: \( \mathbf{x}[\mathbf{n}] \) – double

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

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

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

12: \( \text{fail} \) – NagError *

*Input/Output*
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

**NE_ALLOCFAIL**
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, \( \mathbf{pdb} = \langle \text{value} \rangle \) 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.
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.

NE_SINGULAR
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 $\text{rank}(A \ B) < n$; 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 $\text{rank}(B \ A) < n$; 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
nag_dggglm (f08zbc) is threaded by NAG for parallel execution in multithreaded implementations of the
NAG Library.

nag_dggglm (f08zbc) 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 floating-point operations is approximately $\frac{2}{3}(2m^3 - n^3) + 4nm^2$;
when $p = m = n$, the total number of floating-point operations is approximately $\frac{14}{3}m^3$.

10 Example
This example solves the weighted least squares problem

$$\min_{x} \|B^{-1}(d - Ax)\|_2,$$

where

$$B = \begin{pmatrix}
0.5 & 0.0 & 0.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 2.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 5.0
\end{pmatrix}, \quad
d = \begin{pmatrix}
1.32 \\
-4.00 \\
5.52 \\
3.24
\end{pmatrix} \quad \text{and} \quad
A = \begin{pmatrix}
-0.57 & -1.28 & -0.39 \\
-1.93 & 1.08 & -0.31 \\
2.30 & 0.24 & -0.40 \\
-0.02 & 1.03 & -1.43
\end{pmatrix}.$$
#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 */
    double  *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 ]
    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_dggglm (f08zbc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf("%*[\n] ");
    #else
    scanf("%*[\n] ");
    #endif

    #ifdef _WIN32
    scanf("%"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 = m;
    #else
    pda = n;
    pdb = p;
    #endif

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

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

END:
exit_status = 0;
return exit_status;
}
for (i = 1; i <= m; ++i)
    for (j = 1; j <= p; ++j)
#ifdef _WIN32
    scanf_s("%lf", &B(i, j));
#else
    scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[΄\n "");
#else
    scanf("%*[΄\n "");
#endif
for (i = 0; i < m; ++i)
#ifdef _WIN32
    scanf_s("%lf", &d[i]);
#else
    scanf("%lf", &d[i]);
#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) using nag_dggglm (f08zbc).
*/
    nag_dggglm(order, m, n, p, a, pda, b, pdb, d, x, y, &fail);
if (fail.code == NE_NOERROR)
{
    /* Print least squares solution, x. */
    printf("Weighted least-squares solution\n");
    for (i = 0; i < n; ++i)
        printf(" %11.4f%s", x[i], i%7 == 6 ?"\n ":"" );
/* Print residual vector y = inv(B)*(d - A*x). */
    printf("\n");
    printf("%s", "Residual vector");
    for (i = 0; i < p; ++i)
        printf(" %11.2e%s", y[i], i%7 == 6 ?"\n ":"" );
/* Compute and print the square root of the residual sum of squares using
 * nag_dge_norm (f16rac).
*/
    nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, 1, p, y, 1, &rnorm, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    printf("\nSquare root of the residual sum of squares\n %11.2e\n", rnorm);
}
else
{
    printf("Error from nag_dggglm (f08zbc).\n%s\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_dggglm (f08zbc) Example Program Data

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>-0.57</td>
<td>-1.28</td>
<td>-0.39</td>
</tr>
<tr>
<td>-1.93</td>
<td>1.08</td>
<td>-0.31</td>
</tr>
<tr>
<td>2.30</td>
<td>0.24</td>
<td>-0.40</td>
</tr>
<tr>
<td>-0.02</td>
<td>1.03</td>
<td>-1.43</td>
</tr>
</tbody>
</table>

: (m by n) matrix A

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>2.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

: (m by p) matrix B

<table>
<thead>
<tr>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.32</td>
</tr>
<tr>
<td>-4.00</td>
</tr>
<tr>
<td>5.52</td>
</tr>
<tr>
<td>3.24</td>
</tr>
</tbody>
</table>

: m-vector d

## 10.3 Program Results

nag_dggglm (f08zbc) Example Program Results

Weighted least-squares solution

1.9889   -1.0058   -2.9911

Residual vector

-6.37e-04  -2.45e-03  -4.72e-03  7.70e-03

Square root of the residual sum of squares

9.38e-03