NAG Library Function Document

nag_dgglse (f08zac)

1 Purpose
nag_dgglse (f08zac) solves a real linear equality-constrained least squares problem.

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

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

3 Description
nag_dgglse (f08zac) solves the real linear equality-constrained least squares (LSE) problem

\[
\text{minimize } \|c - Ax\|_2 \quad \text{subject to } \quad Bx = d
\]

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 \(\text{rank}(E) = n\), 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


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 matrix \(A\).

\textit{Constraint:} \(m \geq 0\).
3: \( \text{n} \) — Integer  
*On entry:* \( n \), the number of columns of the matrices \( A \) and \( B \).  
*Constraint:* \( n \geq 0 \).

4: \( \text{p} \) — Integer  
*On entry:* \( p \), the number of rows of the matrix \( B \).  
*Constraint:* \( 0 \leq p \leq n \leq m + p \).

5: \( \text{a}[\text{dim}] \) — double  
*Input/Output*  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{a} \) must be at least  
\[
\max(1, \text{pda} \times n) \quad \text{when } \text{order} = \text{Nag\_ColMajor};  
\max(1, m \times \text{pda}) \quad \text{when } \text{order} = \text{Nag\_RowMajor}. 
\]
The \((i,j)\)th element of the matrix \( A \) is stored in  
\[
\text{a}[(j-1) \times \text{pda} + i - 1] \quad \text{when } \text{order} = \text{Nag\_ColMajor};  
\text{a}[(i-1) \times \text{pda} + j - 1] \quad \text{when } \text{order} = \text{Nag\_RowMajor}. 
\]
*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:*  
\[
\begin{align*}
\text{if } \text{order} = \text{Nag\_ColMajor}, & \quad \text{pda} \geq \max(1, m);  
\text{if } \text{order} = \text{Nag\_RowMajor}, & \quad \text{pda} \geq \max(1, n). 
\end{align*}
\]

7: \( \text{b}[\text{dim}] \) — double  
*Input/Output*  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{b} \) must be at least  
\[
\max(1, \text{pdb} \times n) \quad \text{when } \text{order} = \text{Nag\_ColMajor};  
\max(1, p \times \text{pdb}) \quad \text{when } \text{order} = \text{Nag\_RowMajor}. 
\]
The \((i,j)\)th element of the matrix \( B \) is stored in  
\[
\text{b}[(j-1) \times \text{pdb} + i - 1] \quad \text{when } \text{order} = \text{Nag\_ColMajor};  
\text{b}[(i-1) \times \text{pdb} + j - 1] \quad \text{when } \text{order} = \text{Nag\_RowMajor}. 
\]
*On entry:* the \( p \) by \( n \) 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:*  
\[
\begin{align*}
\text{if } \text{order} = \text{Nag\_ColMajor}, & \quad \text{pdb} \geq \max(1, p);  
\text{if } \text{order} = \text{Nag\_RowMajor}, & \quad \text{pdb} \geq \max(1, n). 
\end{align*}
\]

9: \( \text{c}[\text{m}] \) — double  
*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 \( c[n-p], c[n-p+1], \ldots, c[m-1] \); the remaining elements are overwritten.
10: \( d[p] \) – double

Input/Output

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

On exit: \( d \) is overwritten.

11: \( x[n] \) – double

Output

On exit: the solution vector \( x \) of the LSE 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_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 \(<\text{value}>\) had an illegal value.

NE_INT

On entry, \( m = <\text{value}> \).

Constraint: \( m \geq 0 \).

On entry, \( n = <\text{value}> \).

Constraint: \( n \geq 0 \).

On entry, \( pda = <\text{value}> \).

Constraint: \( pda > 0 \).

On entry, \( pdb = <\text{value}> \).

Constraint: \( pdb > 0 \).

NE_INT_2

On entry, \( pda = <\text{value}> \) and \( m = <\text{value}> \).

Constraint: \( pda \geq \max(1, m) \).

On entry, \( pda = <\text{value}> \) and \( n = <\text{value}> \).

Constraint: \( pda \geq \max(1, n) \).

On entry, \( pdb = <\text{value}> \) and \( n = <\text{value}> \).

Constraint: \( pdb \geq \max(1, n) \).

On entry, \( pdb = <\text{value}> \) and \( p = <\text{value}> \).

Constraint: \( pdb \geq \max(1, p) \).

NE_INT_3

On entry, \( p = <\text{value}> \), \( m = <\text{value}> \) and \( n = <\text{value}> \).

Constraint: \( 0 \leq p \leq n \leq m + p \).

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 3.6.6 in the Essential Introduction for further information.
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.

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 \(\text{rank}(B) < p\); 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**

\(\text{nag\_dgglse (f08zac)}\) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\(\text{nag\_dgglse (f08zac)}\) 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 floating-point operations is approximately \(\frac{3}{2}n^2(6m + n)\); if \(p \ll n\), the number reduces to approximately \(\frac{3}{2}n^2(3m - n)\).

\(\text{nag\_opt\_lin\_lsq (e04ncc)}\) may also be used to solve LSE problems. It differs from \(\text{nag\_dgglse (f08zac)}\) in that it uses an iterative (rather than direct) method, and that it allows general upper and lower bounds to be specified for the variables \(x\) and the linear constraints \(Bx\).

**10 Example**

This example solves the least squares problem

\[
\min_x ||c - Ax||_2 \quad \text{subject to} \quad Bx = d
\]

where

\[
c = \begin{pmatrix} -1.50 \\ -2.14 \\ 1.23 \\ -0.54 \\ -1.68 \\ 0.82 \end{pmatrix}
\]
\[ A = \begin{pmatrix} -0.57 & -1.28 & -0.39 & 0.25 \\ -1.93 & 1.08 & -0.31 & -2.14 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ -1.93 & 0.64 & -0.66 & 0.08 \\ 0.15 & 0.30 & 0.15 & -2.13 \\ -0.02 & 1.03 & -1.43 & 0.50 \end{pmatrix}, \]

\[ B = \begin{pmatrix} 1.0 & 0 & -1.0 & 0 \\ 0 & 1.0 & 0 & -1.0 \end{pmatrix} \]

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

```c
#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 */
    double *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_dgglse (f08zac) Example Program Results\n\n");
    /* Skip heading in data file */
    #ifdef __WAVE32
    scanf_s("%*[\n"]");
    #else
    scanf("%*[\n"]");
    #endif
    #ifdef __WAVE32
    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, double)) ||
    !(b = NAG_ALLOC(p*n, double)) ||
    !(c = NAG_ALLOC(m, double)) ||
    !(d = NAG_ALLOC(p, double)) ||
    !(x = NAG_ALLOC(n, double)))
{
    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", &A(i, j));
        #else
            scanf("%lf", &A(i, j));
        #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", &B(i, j));
            #else
                scanf("%lf", &B(i, j));
            #endif
        }
    }

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

    #ifdef _WIN32
        scanf_s("%*[\n] ");
    #else
        scanf("%*[\n] ");
    #endif
    for (i = 1; i <= p; ++i)
    {
        scanf("%lf", &d[i - 1]);
    }

f08zac.6  Mark 25
else
    scanf("%lf", &d[i - 1]);
#endif
#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_dgglse(order, m, n, p, a, pda, b, pdb, c, d, x, &fail);
if (fail.code == NE_NOERROR)
{
    /* Print least-squares solution */
    printf("Constrained least-squares solution\n", &fail.message);
    for (i = 1; i <= n; ++i)
        printf("%11.4f\n", x[i - 1], i%7 == 0 || i == n?"\n":" ");
    /* Compute the square root of the residual sum of squares */
    nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, 1, m - n + p, &c[n - p], 1,
                 &rnorm, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dge_norm (f16rac).\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_dgglse (f08zac).\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_dgglse (f08zac) Example Program Data

<table>
<thead>
<tr>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<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>-1.93</td>
<td>0.64</td>
<td>-0.66</td>
</tr>
<tr>
<td>0.15</td>
<td>0.30</td>
<td>0.15</td>
</tr>
<tr>
<td>-0.02</td>
<td>1.03</td>
<td>-1.43</td>
</tr>
<tr>
<td>1.00</td>
<td>0.00</td>
<td>-1.00</td>
</tr>
<tr>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

-1.50
-2.14
1.23
-0.54
10.3 Program Results

nag_dgglse (f08zac) Example Program Results

Constrained least-squares solution

\[ 0.4890 \quad 0.9975 \quad 0.4890 \quad 0.9975 \]

Square root of the residual sum of squares

\[ 2.51e-02 \]