NAG Library Function Document

nag_dgelsd (f08kcc)

1 Purpose
nag_dgelsd (f08kcc) computes the minimum norm solution to a real linear least squares problem
\[
\min_x \| b - Ax \|_2.
\]

2 Specification
```
#include <nag.h>
#include <nagf08.h>
void nag_dgelsd (Nag_OrderType order, Integer m, Integer n, Integer nrhs,
                 double a[], Integer pda, double b[], Integer pdb, double s[],
                 double rcond, Integer *rank, NagError *fail)
```

3 Description
nag_dgelsd (f08kcc) uses the singular value decomposition (SVD) of \( A \), where \( A \) is a real \( m \) 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 problem is solved in three steps:
1. reduce the coefficient matrix \( A \) to bidiagonal form with Householder transformations, reducing the original problem into a ‘bidiagonal least squares problem’ (BLS);
2. solve the BLS using a divide-and-conquer approach;
3. apply back all the Householder transformations to solve the original least squares problem.

The effective rank of \( A \) is determined by treating as zero those singular values which are less than \( \text{rcond} \) times the largest singular value.

4 References


5 Arguments
1: \( \text{order} \) – Nag_OrderType
   
   On entry: the \( \text{order} \) 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
   \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed
   explanation of the use of this argument.

   Constraint: \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).
2: \textbf{m} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \textbf{m}, the number of rows of the matrix \textbf{A}.
\textit{Constraint:} \textbf{m} \geq 0.

3: \textbf{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \textbf{n}, the number of columns of the matrix \textbf{A}.
\textit{Constraint:} \textbf{n} \geq 0.

4: \textbf{nrhs} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \textbf{r}, the number of right-hand sides, i.e., the number of columns of the matrices \textbf{B} and \textbf{X}.
\textit{Constraint:} \textbf{nrhs} \geq 0.

5: \textbf{a}[^{\text{dim}}] – double \hspace{1cm} \textit{Input/Output}

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

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

\textit{On entry:} the \textbf{m} by \textbf{n} coefficient matrix \textbf{A}.
\textit{On exit:} the contents of \textbf{a} are destroyed.

6: \textbf{pda} – Integer \hspace{1cm} \textit{Input}

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

\textit{Constraints:}
\begin{align*}
\text{if} \ \text{order} = \text{Nag\_ColMajor}, & \text{ pda} \geq \max(1, \textbf{m}); \\
\text{if} \ \text{order} = \text{Nag\_RowMajor}, & \text{ pda} \geq \max(1, \textbf{n}).
\end{align*}

7: \textbf{b}[^{\text{dim}}] – double \hspace{1cm} \textit{Input/Output}

\textit{Note:} the dimension, \textit{dim}, of the array \textbf{b} must be at least
\begin{align*}
\max(1, \text{pdb} \times \textbf{nrhs}) & \quad \text{when} \ \text{order} = \text{Nag\_ColMajor}; \\
\max(1, \max(1, \textbf{m}, \textbf{n}) \times \text{pdb}) & \quad \text{when} \ \text{order} = \text{Nag\_RowMajor}.
\end{align*}

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

\textit{On entry:} the \textbf{m} by \textbf{r} right-hand side matrix \textbf{B}.
\textit{On exit:} \textbf{b} is overwritten by the \textbf{n} by \textbf{r} solution matrix \textbf{X}. If \textbf{m} \geq \textbf{n} and \textbf{rank} = \textbf{n}, the residual sum of squares for the solution in the \textit{i}th column is given by the sum of squares of elements \textbf{n} + 1, \ldots, \textbf{m} in that column.

8: \textbf{pdb} – Integer \hspace{1cm} \textit{Input}

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

if order = Nag.ColMajor,  

\[ \text{pdb} \geq \max(1, \text{m}, \text{n}) \];

if order = Nag.RowMajor,  

\[ \text{pdb} \geq \max(1, \text{nrhs}) \].

9:  \( s[\text{dim}] \) – double  \hspace{1em} \text{Output}

\text{Note:} \text{the dimension,} \text{dim, of the array} \ s \text{ must be at least} \max(1, \min(\text{m}, \text{n})). \text{On exit: } \text{the singular values of} \ A \text{ in decreasing order.}

10:  \( \text{rcond} \) – double  \hspace{1em} \text{Input}

\text{On entry:} \text{used to determine the effective rank of} \ A. \text{Singular values} \ s[i - 1] \leq \text{rcond} \times s[0] \text{ are treated as zero. If} \ \text{rcond} < 0, \text{machine precision is used instead.}

11:  \( \text{rank} \) – Integer *  \hspace{1em} \text{Output}

\text{On exit:} \text{the effective rank of} \ A, \text{i.e., the number of singular values which are greater than} \ \text{rcond} \times s[0].

12:  \( \text{fail} \) – NagError *  \hspace{1em} \text{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, \( \text{m} = \langle value\rangle \).

Constraint: \( \text{m} \geq 0 \).

On entry, \( \text{n} = \langle value\rangle \).

Constraint: \( \text{n} \geq 0 \).

On entry, \( \text{nrhs} = \langle value\rangle \).

Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pda} = \langle value\rangle \).

Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdb} = \langle value\rangle \).

Constraint: \( \text{pdb} > 0 \).

NE_INT_2

On entry, \( \text{pda} = \langle value\rangle \) and \( \text{m} = \langle value\rangle \).

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

On entry, \( \text{pda} = \langle value\rangle \) and \( \text{n} = \langle value\rangle \).

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

**NE_INT_3**
On entry, \( \text{pdb} = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} \geq \max(1, m, n) \).

**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
\( \text{nag_dgelsd (f08kcc)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
\( \text{nag_dgelsd (f08kcc)} \) 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 complex analogue of this function is \( \text{nag_zgelsd (f08kqc)} \).

10 Example
This example solves the linear least squares problem
\[
\min_x \| b - Ax \|_2
\]
for the solution, \( x \), of minimum norm, where
\[
A = \begin{pmatrix}
-0.09 & -1.56 & -1.48 & -1.09 & 0.08 & -1.59 \\
0.14 & 0.20 & -0.43 & 0.84 & 0.55 & -0.72 \\
-0.46 & 0.29 & 0.89 & 0.77 & -1.13 & 1.06 \\
0.68 & 1.09 & -0.71 & 2.11 & 0.14 & 1.24 \\
1.29 & 0.51 & -0.96 & -1.27 & 1.74 & 0.34
\end{pmatrix}
\text{and } b = \begin{pmatrix}
7.4 \\
4.3 \\
-8.1 \\
1.8 \\
8.7
\end{pmatrix}.
\]
A tolerance of 0.01 is used to determine the effective rank of \( A \).
10.1 Program Text

/* nag_dgelsd (f08kcc) Example Program. 
 * Copyright 2014 Numerical Algorithms Group. 
 * Mark 23, 2011. 
*/
#include <stdio.h>
#include <nag.h>
#include <nagf08.h>
#include <nag_stdblib.h>

int main(void)
{
    /* Scalars */
    double rcond;
    Integer exit_status = 0, i, j, m, n, nrhs, rank, pda, pdb;
    /* Arrays */
    double *a = 0, *b = 0, *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_dgelsd (f08kcc) 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, double)) ||
        !(b = NAG_ALLOC(MAX(m, n) * nrhs, double)) ||
        !(s = NAG_ALLOC(MIN(m, n), double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}

f08 – Least-squares and Eigenvalue Problems (LAPACK)  f08kcc
/* Read A and B from data file */
for (i = 1; i <= m; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &A(i, j));
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
for (i = 1; i <= m; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
rcond = 0.01;
nag_dgelsd(order, m, n, 1, a, pda, b, pdb, s, rcond, &rank, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgelsd (f08kcc).\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("%10.4f%s", B(i, j), j%7==0?"\n":"");
    printf("\n");
}

/* Print the effective rank of A */
printf("\n\nTolerance used to estimate the rank of A\n11.2e\n", rcond);
printf("\nEstimated rank of A\n%6"NAG_IFMT"\n", rank);

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

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(s);
return exit_status;
}
#endif
### 10.2 Program Data

nag_dgelsd (f08kcc) Example Program Data

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>-0.09</td>
<td>-1.56</td>
<td>-1.48</td>
</tr>
<tr>
<td>0.14</td>
<td>0.20</td>
<td>-0.43</td>
</tr>
<tr>
<td>-0.46</td>
<td>0.29</td>
<td>0.89</td>
</tr>
<tr>
<td>0.68</td>
<td>1.09</td>
<td>-0.71</td>
</tr>
<tr>
<td>1.29</td>
<td>0.51</td>
<td>-0.96</td>
</tr>
</tbody>
</table>

: m, n, nrhs

-0.09  -1.56  -1.48  -1.09  0.08  -1.59
0.14   0.20   0.84   0.55  -0.72
-0.46  0.29   0.89   0.77  -1.13  1.06
0.68   1.09  -0.71   2.11  0.14  1.24
1.29   0.51  -0.96  -1.27  1.74  0.34 : matrix A

7.4
4.3
-8.1
1.8
8.7

: vector b

### 10.3 Program Results

nag_dgelsd (f08kcc) Example Program Results

Least squares solution

1.5938
-0.1180
-3.1501
0.1554
2.5529
-1.6730

Tolerance used to estimate the rank of A

1.00e-02

Estimated rank of A

4

Singular values of A

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9997</td>
<td>2.9962</td>
<td>2.0001</td>
<td>0.9988</td>
<td>0.0025</td>
</tr>
</tbody>
</table>