NAG Library Function Document

nag_real_gen_lin_solve (f04bac)

1 Purpose

nag_real_gen_lin_solve (f04bac) computes the solution to a real system of linear equations \( AX = B \), where \( A \) is an \( n \) by \( n \) matrix and \( X \) and \( B \) are \( n \) by \( r \) matrices. An estimate of the condition number of \( A \) and an error bound for the computed solution are also returned.

2 Specification

```c
#include <nag.h>
#include <nagf04.h>

void nag_real_gen_lin_solve (Nag_OrderType order, Integer n, Integer nrhs,
                          double a[], Integer pda, Integer ipiv[], double b[], Integer pdb,
                          double *rcond, double *errbnd, NagError *fail)
```

3 Description

The \( LU \) decomposition with partial pivoting and row interchanges is used to factor \( A \) as \( A = PLU \), where \( P \) is a permutation matrix, \( L \) is unit lower triangular, and \( U \) is upper triangular. The factored form of \( A \) is then used to solve the system of equations \( AX = B \).

4 References


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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: `order = Nag_ROW_MAJOR` or `Nag_COL_MAJOR`.

2: `n` – Integer

   `Input`

   On entry: the number of linear equations \( n \), i.e., the order of the matrix \( A \).

   Constraint: \( n \geq 0 \).

3: `nrhs` – Integer

   `Input`

   On entry: the number of right-hand sides \( r \), i.e., the number of columns of the matrix \( B \).

   Constraint: \( nrhs \geq 0 \).

4: `a[dim]` – double

   `Input/Output`

   Note: the dimension, \( dim \), of the array \( a \) must be at least \( \max(1, pda \times n) \).
The $(i, j)\text{th}$ element of the matrix $A$ is stored in

\[ a[(j - 1) \times \text{pda} + i - 1] \text{ when order = Nag\_ColMajor; } \]
\[ a[(i - 1) \times \text{pda} + j - 1] \text{ when order = Nag\_RowMajor. } \]

\textbf{On entry:} the $n$ by $n$ coefficient matrix $A$.

\textbf{On exit:} if \texttt{fail\_code == NE\_NOERROR}, the factors $L$ and $U$ from the factorization $A = PLU$. The unit diagonal elements of $L$ are not stored.

5: $\text{pda} \rightarrow \text{Integer}$ \hspace{1cm} \textit{Input}

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

\textit{Constraint: $\text{pda} \geq \max(1, n)$.}

6: $\text{ipiv}[n] \rightarrow \text{Integer}$ \hspace{1cm} \textit{Output}

\textit{On exit:} if \texttt{fail\_code == NE\_NOERROR}, the pivot indices that define the permutation matrix $P$; at the $r$th step row $i$ of the matrix was interchanged with row $\text{ipiv}[i - 1]$. $\text{ipiv}[i - 1] = i$ indicates a row interchange was not required.

7: $\text{b}[\text{dim}] \rightarrow \text{double}$ \hspace{1cm} \textit{Input/Output}

\textit{Note:} the dimension, \texttt{dim}, of the array $b$ must be at least

\[
\max(1, \text{pdb} \times \text{nrhs}) \text{ when order = Nag\_ColMajor; } \\
\max(1, n \times \text{pdb}) \text{ when order = Nag\_RowMajor. }
\]

\textit{On entry:} the $n$ by $r$ matrix of right-hand sides $B$.

\textit{On exit:} if \texttt{fail\_code == NE\_NOERROR} or \texttt{NE\_RCOND}, the $n$ by $r$ solution matrix $X$.

8: $\text{pdb} \rightarrow \text{Integer}$ \hspace{1cm} \textit{Input}

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

\textit{Constraints:}

\[ \text{if order = Nag\_ColMajor, pdb} \geq \max(1, n); \]
\[ \text{if order = Nag\_RowMajor, pdb} \geq \max(1, \text{nrhs}). \]

9: $\text{rcond} \rightarrow \text{double} *$ \hspace{1cm} \textit{Output}

\textit{On exit:} if no constraints are violated, an estimate of the reciprocal of the condition number of the matrix $A$, computed as $\text{rcond} = 1/\left(\|A\|_1 \|A^{-1}\|_1\right)$.

10: $\text{errbnd} \rightarrow \text{double} *$ \hspace{1cm} \textit{Output}

\textit{On exit:} if \texttt{fail\_code == NE\_NOERROR} or \texttt{NE\_RCOND}, an estimate of the forward error bound for a computed solution $\hat{x}$, such that $\|\hat{x} - x\|_1/\|x\|_1 \leq \text{errbnd}$, where $\hat{x}$ is a column of the computed solution returned in the array $b$ and $x$ is the corresponding column of the exact solution $X$. If $\text{rcond}$ is less than \textit{machine precision}, then $\text{errbnd}$ is returned as unity.

11: $\text{fail} \rightarrow \text{NagError} *$ \hspace{1cm} \textit{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_INT
On entry, \( n = \langle value \rangle \).
Constraint: \( n \geq 0 \).

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

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

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

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_RCOND
A solution has been computed, but \texttt{rcond} is less than \texttt{machine precision} so that the matrix \( A \) is numerically singular.

NE_SINGULAR
Diagonal element \( \langle value \rangle \) of the upper triangular factor is zero. The factorization has been completed, but the solution could not be computed.

7 Accuracy
The computed solution for a single right-hand side, \( \hat{x} \), satisfies an equation of the form

\[(A + E)\hat{x} = b,\]

where
\[ \|E\|_1 = O(\varepsilon)\|A\|_1 \]

and \( \varepsilon \) is the \textbf{machine precision}. An approximate error bound for the computed solution is given by

\[ \frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A)\|E\|_1 / \|A\|_1, \]

where \( \kappa(A) = \|A^{-1}\|_1 \|A\|_1 \), the condition number of \( A \) with respect to the solution of the linear equations. \text{nag_real_gen_lin_solve} (f04bac) uses the approximation \( \|E\|_1 = \varepsilon \|A\|_1 \) to estimate \text{errbnd}.

See Section 4.4 of Anderson et al. (1999) for further details.

### 8 Parallelism and Performance

\text{nag_real_gen_lin_solve} (f04bac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\text{nag_real_gen_lin_solve} (f04bac) 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 Integer allocatable memory required is \( n \), and the double allocatable memory required is \( 4 \times n \). In this case the factorization and the solution \( X \) have been computed, but \text{rcond} and \text{errbnd} have not been computed.

The total number of floating-point operations required to solve the equations \( AX = B \) is proportional to \( (\frac{2}{3}n^3 + n^2r) \). The condition number estimation typically requires between four and five solves and never more than eleven solves, following the factorization.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

The complex analogue of \text{nag_real_gen_lin_solve} (f04bac) is \text{nag_complex_gen_lin_solve} (f04cac).

### 10 Example

This example solves the equations

\[ AX = B, \]

where

\[
A = \begin{pmatrix}
1.80 & 2.88 & 2.05 & -0.89 \\
5.25 & -2.95 & -0.95 & -3.80 \\
1.58 & -2.69 & -2.90 & -1.04 \\
-1.11 & -0.66 & -0.59 & 0.80
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
9.52 & 18.47 \\
24.35 & 2.25 \\
0.77 & -13.28 \\
-6.22 & -6.21
\end{pmatrix}.
\]

An estimate of the condition number of \( A \) and an approximate error bound for the computed solutions are also printed.
10.1 Program Text

/* nag_real_gen_lin_solve (f04bac) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 8, 2004. */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf04.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double errbnd, rcond;
    Integer exit_status, i, j, n, nrhs, pda, pdb;
    /* Arrays */
    double *a = 0, *b = 0;
    Integer *ipiv = 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 ]
#else
#define A(I, J) a[(I-1)*pda+J-1 ]
#define B(I, J) b[(I-1)*pdb +J-1 ]
#endif

exit_status = 0;
INIT_FAIL(fail);
printf("nag_real_gen_lin_solve (f04bac) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &n, &nrhs);
#else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &n, &nrhs);
#endif
if (n >= 0 && nrhs >= 0)
{
    /* Allocate memory */
    if (!a || !(b = NAG_ALLOC(n*nrhs, double)) || !(ipiv = NAG_ALLOC(n, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#endif NAG_COLUMN_MAJOR
pda = n;
pdb = n;
#else
pda = n;
pdb = nrhs;
#endif
}
else
{
    printf("%s\n", "n and/or nrhs too small");
    exit_status = 1;
    return exit_status;
}

/* Read A and B from data file */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
    {
#ifdef _WIN32
    scanf_s("%lf", &A(i, j));
#else
    scanf("%lf", &A(i, j));
#endif
    }
#endif
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
        {
#ifdef _WIN32
        scanf_s("%lf", &B(i, j));
#else
        scanf("%lf", &B(i, j));
#endif
        }
#endif
        scanf("%*[\n ] ");
    }
    /* Solve the equations AX = B for X */
    /* nag_real_gen_lin_solve (f04bac).
    * Computes the solution and error-bound to a real system of
    * linear equations
    */
    nag_real_gen_lin_solve(order, n, nrhs, a, pda, ipiv, b, pdb, &rcond, &errbnd, &fail);
    if (fail.code == NE_NOERROR)
    {
        /* Print solution, estimate of condition number and approximate */
    /* error bound */
    /* nag_gen_real_mat_print (x04cac).
    * Print real general matrix (easy-to-use)
    */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb, "Solution", 0, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
        exit_status = 1;
goto END;
    }
}
printf("\n");
printf("%s
 %10.1e
\n\n\n", "Estimate of condition number", 1.0/rcond);
printf("%s
 %10.1e
\n", "Estimate of error bound for computed solutions", errbnd);

} else if (fail.code == NE_RCOND)
{
    /* Matrix A is numerically singular. Print estimate of */
    /* reciprocal of condition number and solution */
    printf("\n%s
%10.1e
\n\n\n", "Estimate of reciprocal of condition number ", rcond);
}

else if (fail.code == NE_SINGULAR)
{
    /* The upper triangular matrix U is exactly singular. Print */
    /* details of factorization */
    printf("\n");
    printf("%s
", "Details of factorization");
    printf("\n");
    printf("%s
", "Pivot indices");
    for (i = 1; i <= n; ++i)
    {
        printf("%11NAG_IFMT "%s", ipiv[i - 1], i%7 == 0 || i == n?"\n": "");
    }
    printf("\n");
}
else
{
    printf("Error from nag_real_gen_lin_solve (f04bac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(ipiv);
return exit_status;

Mark 25
10.2 Program Data

nag_real_gen_lin_solve (f04bac) Example Program Data

4 2 :Values of N and NRHS

1.80  2.88  2.05  -0.89
5.25 -2.95 -0.95  -3.80
1.58 -2.69 -2.90  -1.04
-1.11 -0.66 -0.59  0.80 :End of matrix A

9.52  18.47
24.35  2.25
 0.77 -13.28
-6.22 -6.21 :End of matrix B

10.3 Program Results

nag_real_gen_lin_solve (f04bac) Example Program Results

Solution
1 2
1  1.0000  3.0000
2 -1.0000  2.0000
3  3.0000  4.0000
4 -5.0000  1.0000

Estimate of condition number
1.5e+02

Estimate of error bound for computed solutions
1.7e-14