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

```
#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

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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_RowMajor or Nag_ColMajor.

2: **n** – Integer *Input*

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

Constraint: **n** ≥ 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** ≥ 0.

4: **a**$[dim]$ – double *Input/Output*

Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \textbf{pda} \times \textbf{n})$.

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

a$[(j-1)\times$ **pda** $+i-1]$ when **order** = Nag_ColMajor;
a$[(i-1)\times$ **pda** $+j-1]$ when **order** = Nag_RowMajor.

On entry: the n by n coefficient matrix A.

On exit: if **fail.code** = NE_NOERROR, the factors L and U from the factorization $A=PLU$. The unit diagonal elements of L are not stored.

5: **pda** – Integer *Input*

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

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

6: **ipiv**[**n**] – Integer *Output*

On exit: if **fail.code** = NE_NOERROR, the pivot indices that define the permutation matrix P; at the ith step row i of the matrix was interchanged with row **ipiv**$[i-1]$. **ipiv**$[i-1]=i$ indicates a row interchange was not required.

7: **b**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least

$\max(1,$ **pdb** $\times$ **nrhs**$)$ when **order** = Nag_ColMajor;
$\max(1,$ **n** $\times$ **pdb**$)$ when **order** = Nag_RowMajor.

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

b$[(j-1)\times$ **pdb** $+i-1]$ when **order** = Nag_ColMajor;
b$[(i-1)\times$ **pdb** $+j-1]$ when **order** = Nag_RowMajor.

On entry: the n by r matrix of right-hand sides B.

On exit: if **fail.code** = NE_NOERROR or NE_RCOND, the n by r solution matrix X.

8: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdb** $\geq \max(1,$ **n**$)$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1,$ **nrhs**$)$.

9: **rcond** – double * *Output*

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

10: **errbnd** – double * *Output*

On exit: if **fail.code** = NE_NOERROR or NE_RCOND, an estimate of the forward error bound for a computed solution $\hat{x}$, such that $\|\hat{x}-x\|_1/\|x\|_1 \leq$ **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 **rcond** is less than ***machine precision***, then **errbnd** is returned as unity.

11: **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.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{nrhs} = \langle value \rangle$.
Constraint: $\mathbf{nrhs} \geq 0$.

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

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

NE_INT_2

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

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

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.

NE_RCOND

A solution has been computed, but **rcond** is less than ***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

$$\|E\|_1 = O(\epsilon)\|A\|_1$$

and ϵ is the ***machine precision***. An approximate error bound for the computed solution is given by

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

where $\kappa(A) = \left\|A^{-1}\right\|_1 \|A\|_1$, the condition number of A with respect to the solution of the linear equations. nag_real_gen_lin_solve (f04bac) uses the approximation $\|E\|_1 = \epsilon\|A\|_1$ to estimate **errbnd**. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

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

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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of floating-point operations required to solve the equations $AX = B$ is proportional to $\left(\frac{2}{3}n^3 + n^2r\right)$. 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 nag_real_gen_lin_solve (f04bac) is 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 2004 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]
  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

  exit_status = 0;
  INIT_FAIL(fail);

  printf("nag_real_gen_lin_solve (f04bac) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%*[^\n] ", &n, &nrhs);
  if (n >= 0 && nrhs >= 0)
    {
      /* Allocate memory */
      if (!(a = NAG_ALLOC(n*n, double)) ||
          !(b = NAG_ALLOC(n*nrhs, double)) ||
          !(ipiv = NAG_ALLOC(n, Integer)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
#ifdef 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)
        {
          scanf("%lf", &A(i, j));
        }
    }
  scanf("%*[^\n] ");

  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        {
          scanf("%lf", &B(i, j));
        }
    }
  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\n      %10.1e\n\n\n", "Estimate of condition number",
            1.0/rcond);
    printf("%s\n      %10.1e\n\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\n%10.1e\n\n\n",
            "Estimate of reciprocal of condition number      ", rcond);

    /* nag_gen_real_mat_print (x04cac), see above. */
    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;
      }
  }
else if (fail.code == NE_SINGULAR)
  {
    /* The upper triangular matrix U is exactly singular.  Print */
    /* details of factorization */
    printf("\n");
    /* nag_gen_real_mat_print (x04cac), see above. */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                           a, pda, "Details of factorization", 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;
      }

    /* Print pivot indices */
    printf("\n");
    printf("%s\n", "Pivot indices");
    for (i = 1; i <= n; ++i)
      {
        printf("%11ld%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;
}
```

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
```