NAG Library Function Document

nag_real_sym_lin_solve (f04bhc)

1 Purpose

nag_real_sym_lin_solve (f04bhc) computes the solution to a real system of linear equations $AX = B$, where A is an n by n symmetric 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_sym_lin_solve (Nag_OrderType order, Nag_UploType uplo,
     Integer n, Integer nrhs, double a[], Integer pda, Integer ipiv[],
     double b[], Integer pdb, double *rcond, double *errbnd, NagError *fail)
```

3 Description

The diagonal pivoting method is used to factor A as $A = UDU^{\mathrm{T}}$, if **uplo** $=$ Nag_Upper, or $A = LDL^{\mathrm{T}}$, if **uplo** $=$ Nag_Lower, where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. 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: **uplo** – Nag_UploType *Input*

On entry: if **uplo** $=$ Nag_Upper, the upper triangle of the matrix A is stored.

If **uplo** $=$ Nag_Lower, the lower triangle of the matrix A is stored.

Constraint: **uplo** $=$ Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **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.

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

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

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

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

On entry: the n by n symmetric matrix A.

If **uplo** = Nag_Upper, the leading **n** by **n** upper triangular part of the array **a** contains the upper triangular part of the matrix A, and the strictly lower triangular part of **a** is not referenced.

If **uplo** = Nag_Lower, the leading **n** by **n** lower triangular part of the array **a** contains the lower triangular part of the matrix A, and the strictly upper triangular part of **a** is not referenced.

On exit: if **fail.code** = NE_NOERROR, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^{\mathrm{T}}$ or $A = LDL^{\mathrm{T}}$ as computed by nag_dsytrf (f07mdc).

6: **pda** – Integer *Input*

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

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

7: **ipiv**$[\mathbf{n}]$ – Integer *Output*

On exit: if **fail.code** = NE_NOERROR, details of the interchanges and the block structure of D, as determined by nag_dsytrf (f07mdc).

$\mathbf{ipiv}[k-1] > 0$
Rows and columns k and $\mathbf{ipiv}[k-1]$ were interchanged, and d_{kk} is a 1 by 1 diagonal block.

uplo = Nag_Upper and $\mathbf{ipiv}[k-1] = \mathbf{ipiv}[k-2] < 0$
Rows and columns $k-1$ and $-\mathbf{ipiv}[k-1]$ were interchanged and $d_{k-1:k,k-1:k}$ is a 2 by 2 diagonal block.

uplo = Nag_Lower and $\mathbf{ipiv}[k-1] = \mathbf{ipiv}[k] < 0$
Rows and columns $k+1$ and $-\mathbf{ipiv}[k-1]$ were interchanged and $d_{k:k+1,k:k+1}$ is a 2 by 2 diagonal block.

8: **b**$[dim]$ – double *Input/Output*

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

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

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

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

9: **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** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

10: **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** $= 1/\left(\|A\|_1 \|A^{-1}\|_1\right)$.

11: **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.

12: **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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **nrhs** = ⟨*value*⟩.
Constraint: **nrhs** ≥ 0.

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

NE_INT_2

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

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

On entry, **pdb** = ⟨*value*⟩ and **nrhs** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{nrhs})$.

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 **rcond** is less than ***machine precision*** so that the matrix A is numerically singular.

NE_SINGULAR

Diagonal block $\langle$*value*$\rangle$ of the block diagonal matrix 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_sym_lin_solve (f04bhc) 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_sym_lin_solve (f04bhc) is not threaded by NAG in any implementation.

nag_real_sym_lin_solve (f04bhc) 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 $\max(2\times\mathbf{n},\mathbf{lwork})$, where **lwork** is the optimum workspace required by nag_dsysv (f07mac). If this failure occurs it may be possible to solve the equations by calling the packed storage version of nag_real_sym_lin_solve (f04bhc), nag_real_sym_packed_lin_solve (f04bjc), or by calling nag_dsysv (f07mac) directly with less than the optimum workspace (see Chapter f07).

The total number of floating-point operations required to solve the equations $AX = B$ is proportional to $\left(\frac{1}{3}n^3 + 2n^2r\right)$. 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 analogues of nag_real_sym_lin_solve (f04bhc) are nag_herm_lin_solve (f04chc) for complex Hermitian matrices, and nag_complex_sym_lin_solve (f04dhc) for complex symmetric matrices.

10 Example

This example solves the equations

$$AX = B,$$

where A is the symmetric indefinite matrix

$$A = \begin{pmatrix} -1.81 & 2.06 & 0.63 & -1.15 \\ 2.06 & 1.15 & 1.87 & 4.20 \\ 0.63 & 1.87 & -0.21 & 3.87 \\ -1.15 & 4.20 & 3.87 & 2.07 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0.96 & 3.93 \\ 6.07 & 19.25 \\ 8.38 & 9.90 \\ 9.50 & 27.85 \end{pmatrix}.$$

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_sym_lin_solve (f04bhc) 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 */
  char          nag_enum_arg[40];
  double        *a = 0, *b = 0;
  Integer       *ipiv = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;
  Nag_UploType  uplo;

#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_sym_lin_solve (f04bhc) 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"%*[^\n] ", &n, &nrhs);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &nrhs);
#endif
  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;
    }

#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  if (uplo == Nag_Upper)
    {
      /* Read the upper triangular part of A from data file */
      for (i = 1; i <= n; ++i)
        {
          for (j = i; 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
```

```
    }
  else
    {
      /* Read the lower triangular part of A from data file */
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++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
    }

  /* Read B from data file */
  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
        }
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Solve the equations AX = B for X */
  /* nag_real_sym_lin_solve (f04bhc).
   * Computes the solution and error-bound to a real symmetric
   * system of linear equations
   */
  nag_real_sym_lin_solve(order, uplo, 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%s\n%6s%10.1e\n", "Estimate of condition number", "",
             1./rcond);
      printf("\n\n");
      printf("%s\n%6s%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");
      printf("%s\n%6s%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_UpperMatrix, 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%s\n", "Pivot indices");
      for (i = 1; i <= n; ++i)
        {
          printf("%11"NAG_IFMT"%s", ipiv[i-1], i%7 == 0 || i == n?"\n":" ");
        }
      printf("\n");
    }
  else
    {
      printf("Error from nag_real_sym_lin_solve (f04bhc).\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_sym_lin_solve (f04bhc) Example Program Data

  4      2                   :Values of n and nrhs
  Nag_Upper                  :Value of uplo
 -1.81   2.06   0.63  -1.15
         1.15   1.87   4.20
```

```
               -0.21   3.87
                       2.07 :End of matrix A

  0.96   3.93
  6.07  19.25
  8.38   9.90
  9.50  27.85               :End of matrix B
```

10.3 Program Results

```
nag_real_sym_lin_solve (f04bhc) Example Program Results

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

Estimate of condition number
         7.6e+01


Estimate of error bound for computed solutions
         8.4e-15
```