NAG C Library Function Document

nag_real_sym_packed_lin_solve (f04bjc)

1 Purpose

nag_real_sym_packed_lin_solve (f04bjc) computes the solution to a real system of linear equations $AX = B$, where A is an n by n symmetric matrix, stored in packed format 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_packed_lin_solve (Nag_OrderType order, Nag_UploType uplo,
     Integer n, Integer nrhs, double ap[], 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 URL: 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 2.2.1.4 of 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: **ap**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n}+1)/2)$.

On entry: the n by n symmetric matrix A, packed column-wise in a linear array. The jth column of the matrix A is stored in the array **ap** as follows:

if **uplo** = **Nag_Upper**, $\mathbf{ap}[i + (j-1)j/2] = a_{ij}$ for $1 \leq i \leq j$
if **uplo** = **Nag_Lower**, $\mathbf{ap}[i + (j-1)(2n-j)/2] = a_{ij}$ for $j \leq i \leq n$

See Section 8 below for further details.

On exit: if **fail.code** = **NE_NOERROR**, **NE_SINGULAR** or **NE_RCOND**, 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_dsptrf (f07pdc), stored as a packed triangular matrix in the same storage format as A.

6: **ipiv**[dim] – Integer *Output*

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

On exit: if **fail.code** = **NE_NOERROR**, **NE_SINGULAR** or **NE_RCOND**, details of the interchanges and the block structure of D, as determined by nag_dsptrf (f07pdc).

If $\mathbf{ipiv}[k-1] > 0$, then rows and columns k and $\mathbf{ipiv}[k-1]$ were interchanged, and d_{kk} is a 1 by 1 diagonal block;
if **uplo** = **Nag_Upper** and $\mathbf{ipiv}[k-1] = \mathbf{ipiv}[k-2] < 0$, then 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;
if **uplo** = **Nag_Lower** and $\mathbf{ipiv}[k-1] = \mathbf{ipiv}[k] < 0$, then 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.

7: **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**;
$\max(1, \mathbf{pdb} \times \mathbf{n})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i,j)th element of the matrix B is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = **Nag_RowMajor**, the (i,j)th element of the matrix B is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.

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 matrix row or column elements (depending on the value of **order**) in the array **b**.

Constraints:

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

9: **rcond** – double * *Output*

On exit: if **fail.code** = **NE_NOERROR**, **NE_SINGULAR** or **NE_RCOND**, an estimate of the reciprocal of the condition number of the matrix A, computed as $\mathbf{rcond} = 1/\left(\|A\|_1 \|A^{-1}\|_1\right)$.

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 2.6 of 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, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **nrhs** $= \langle value \rangle$.
Constraint: **nrhs** ≥ 0.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

NE_INT_2

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

On entry, **pdb** $= \langle value \rangle$, **nrhs** $= \langle value \rangle$.
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 consult 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 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

$$\|E\|_1 = \mathrm{O}(\epsilon)\|A\|_1$$

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 $\kappa(A) = \|A^{-1}\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. nag_real_sym_packed_lin_solve (f04bjc) uses the approximation $\|E\|_1 = \epsilon\|A\|_1$ to estimate **errbnd**. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Further Comments

The packed storage scheme is illustrated by the following example when $n = 4$ and **uplo** = **Nag_Upper**. Two-dimensional storage of the symmetric matrix A:

$$\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{array} \quad (a_{ij} = a_{ji})$$

Packed storage of the upper triangle of A:

$$\mathbf{ap} = [a_{11},\ a_{12},\ a_{22},\ a_{13},\ a_{23},\ a_{33},\ a_{14},\ a_{24},\ a_{34},\ a_{44}]$$

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_packed_lin_solve (f04bjc) are nag_herm_packed_lin_solve (f04cjc) for complex Hermitian matrices, and nag_complex_sym_packed_lin_solve (f04djc) for complex symmetric matrices.

9 Example

To solve 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.

9.1 Program Text

```
/* nag_real_sym_packed_lin_solve (f04bjc) 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, pdb;

  /* Arrays */
  char   uplo[2];
  double *ap=0, *b=0;
  Integer *ipiv=0;

  /* Nag types */
  NagError fail;
  Nag_OrderType order;
  Nag_UploType uplo_enum;


#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I,J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I,J) ap[(2*n-J)*(J-1)/2 + I - 1]
#define B(I,J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A_LOWER(I,J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I,J) ap[(2*n-I)*(I-1)/2 + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  exit_status = 0;
  INIT_FAIL(fail);

  Vprintf("nag_real_sym_packed_lin_solve (f04bjc) Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%ld%*[^\n] ",   &n,  &nrhs);
  if (n>0 && nrhs>0)
    {
      /* Allocate memory */
      if ( !(ap = NAG_ALLOC(n*(n+1)/2, double)) ||
           !(b = NAG_ALLOC(n*nrhs, double)) ||
           !(ipiv = NAG_ALLOC(n, Integer)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
#ifdef NAG_COLUMN_MAJOR
      pdb = n;
#else
      pdb = nrhs;
#endif
    }
  else
    {
      Vprintf("%s\n",   "n and/or nrhs too small");
      exit_status = 1;
      return exit_status;

    }

  Vscanf(" ' %1s '%*[^\n] ", uplo);
  if (*(unsigned char *)uplo == 'L')
    uplo_enum = Nag_Lower;
  else if (*(unsigned char *)uplo == 'U')
    uplo_enum = Nag_Upper;
  else
    {
      Vprintf("Unrecognised character for Nag_UploType type\n");
      exit_status = -1;
      goto END;
    }

  /* Read the upper or lower triangular part of the matrix A from */
  /* data file */

  if (uplo_enum == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            {
              Vscanf("%lf", &A_UPPER(i,j));
            }
        }
      Vscanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            {
              Vscanf("%lf", &A_LOWER(i,j));
            }
        }
      Vscanf("%*[^\n] ");
    }

  /* Read B from data file */

  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        {
          Vscanf("%lf", &B(i,j));
        }
    }
  Vscanf("%*[^\n] ");


  /* Solve the equations AX = B for X */
  /* nag_real_sym_packed_lin_solve (f04bjc).
   * Computes the solution and error-bound to a real symmetric
   * system of linear equations, packed storage
   */
  nag_real_sym_packed_lin_solve(order, uplo_enum, n, nrhs, ap, 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)
       */
      nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,  n,
                             nrhs, b, pdb, "Solution", 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      Vprintf("\n");
      Vprintf("%s\n%6s%9.1e\n", "Estimate of condition number", "", 1.0/rcond);
      Vprintf("\n\n");
      Vprintf("%s\n%6s%9.1e\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 */

      Vprintf("\n");
      Vprintf("%s\n%6s%9.1e\n\n", "Estimate of reciprocal of condition number",
              "",  rcond);
      Vprintf("\n");
      /* nag_gen_real_mat_print (x04cac), see above. */
      nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs,
                             b, pdb, "Solution", 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("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 */

      Vprintf("\n");
      /* nag_pack_real_mat_print (x04ccc).
       * Print real packed triangular matrix (easy-to-use)
       */
      nag_pack_real_mat_print(order, Nag_Upper, Nag_NonUnitDiag, n, ap,
                              "Details of factorization", 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_pack_real_mat_print (x04ccc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      /* Print pivot indices */
      Vprintf("\n");
      Vprintf("%s\n%3s",   "Pivot indices",  "");
      for (i = 1; i <= n; ++i)
        {
          Vprintf("%11ld%s", ipiv[i - 1], i%7 == 0 || i == n ?"\n":" ");
        }
      Vprintf("\n");
    }
 END:
  if (ap) NAG_FREE(ap);
  if (b) NAG_FREE(b);
  if (ipiv) NAG_FREE(ipiv);

  return exit_status;
}

#undef B
```

9.2 Program Data

```
nag_real_sym_packed_lin_solve (f04bjc) Example Program Data

  4      2                 :Values of N and NRHS
  'U'                      :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
```

9.3 Program Results

```
nag_real_sym_packed_lin_solve (f04bjc) 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
```