NAG Library Function Document

nag_real_sym_posdef_packed_lin_solve (f04bec)

1 Purpose

nag_real_sym_posdef_packed_lin_solve (f04bec) computes the solution to a real system of linear equations $AX = B$, where A is an n by n symmetric positive definite 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_posdef_packed_lin_solve (Nag_OrderType order,
     Nag_UploType uplo, Integer n, Integer nrhs, double ap[], double b[],
     Integer pdb, double *rcond, double *errbnd, NagError *fail)
```

3 Description

The Cholesky factorization is used to factor A as $A = U^{\mathrm{T}}U$, if **uplo** $=$ Nag_Upper, or $A = LL^{\mathrm{T}}$, if **uplo** $=$ Nag_Lower, where U is an upper triangular matrix and L is a lower triangular matrix. 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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: $\mathbf{n} \geq 0$.

4: **nrhs** – Integer *Input*

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

Constraint: $\mathbf{nrhs} \geq 0$.

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. The upper or lower triangular part of the symmetric matrix is packed column-wise in a linear array. The jth column of A is stored in the array **ap** as follows:

The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:

if **order** = Nag_ColMajor and **uplo** = Nag_Upper,
  A_{ij} is stored in $\mathbf{ap}[(j-1) \times j/2 + i - 1]$, for $i \leq j$;
if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
  A_{ij} is stored in $\mathbf{ap}[(2n-j) \times (j-1)/2 + i - 1]$, for $i \geq j$;
if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
  A_{ij} is stored in $\mathbf{ap}[(2n-i) \times (i-1)/2 + j - 1]$, for $i \leq j$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
  A_{ij} is stored in $\mathbf{ap}[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

On exit: if **fail.code** = NE_NOERROR or NE_RCOND, the factor U or L from the Cholesky factorization $A = U^{\mathrm{T}}U$ or $A = LL^{\mathrm{T}}$, in the same storage format as A.

6: **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.

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

8: **rcond** – double * *Output*

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

9: **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 $\|\hat{x} - x\|_1 / \|x\|_1 \leq \mathbf{errbnd}$, 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.

10: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
The Integer allocatable memory required is **n**, and the double allocatable memory required is $3 \times \mathbf{n}$. Allocation failed before the solution could be computed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

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

NE_INT_2

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

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

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_POS_DEF

The principal minor of order $\langle value \rangle$ of the matrix A is not positive definite. The factorization has not been completed and the solution could not be computed.

NE_RCOND

A solution has been computed, but **rcond** is less than ***machine precision*** so that the matrix A is numerically singular.

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 $\kappa(A) = \|A^{-1}\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. nag_real_sym_posdef_packed_lin_solve (f04bec) 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_posdef_packed_lin_solve (f04bec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_real_sym_posdef_packed_lin_solve (f04bec) 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 packed storage scheme is illustrated by the following example when $n = 4$ and **uplo** $=$ Nag_Upper. Two-dimensional storage of the symmetric matrix A:

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

Packed storage of the upper triangle of A:

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

The total number of floating-point operations required to solve the equations $AX = B$ is proportional to $\left(\frac{1}{3}n^3 + n^2r\right)$. 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_sym_posdef_packed_lin_solve (f04bec) is nag_herm_posdef_pack ed_lin_solve (f04cec).

10 Example

This example solves the equations

$$AX = B,$$

where A is the symmetric positive definite matrix

$$A = \begin{pmatrix} 4.16 & -3.12 & 0.56 & -0.10 \\ -3.12 & 5.03 & -0.83 & 1.18 \\ 0.56 & -0.83 & 0.76 & 0.34 \\ -0.10 & 1.18 & 0.34 & 1.18 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 8.70 & 8.30 \\ -13.35 & 2.13 \\ 1.89 & 1.61 \\ -4.14 & 5.00 \end{pmatrix}.$$

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_posdef_packed_lin_solve (f04bec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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 nag_enum_arg[40];
  double *ap = 0, *b = 0;

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

#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);

  printf("nag_real_sym_posdef_packed_lin_solve (f04bec) 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 (!(ap = NAG_ALLOC(n * (n + 1) / 2, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
#ifdef NAG_COLUMN_MAJOR
    pdb = n;
#else
    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, (unsigned)_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);

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

  if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i) {
      for (j = i; j <= n; ++j) {
#ifdef _WIN32
        scanf_s("%lf", &A_UPPER(i, j));
#else
        scanf("%lf", &A_UPPER(i, j));
#endif
      }
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }
  else {
    for (i = 1; i <= n; ++i) {
      for (j = 1; j <= i; ++j) {
#ifdef _WIN32
        scanf_s("%lf", &A_LOWER(i, j));
#else
        scanf("%lf", &A_LOWER(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_posdef_packed_lin_solve (f04bec).
   * Computes the solution and error-bound to a real symmetric
   * positive-definite system of linear equations, packed
   * storage
   */
  nag_real_sym_posdef_packed_lin_solve(order, uplo, n, nrhs, ap, 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%6s%10.1e\n\n\n", "Estimate of condition number", "",
           1.0 / rcond);
    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%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_POS_DEF) {
    /* The matrix A is not positive definite to working precision */
    printf("%s%3" NAG_IFMT "%s\n\n", "The leading minor of order ",
           fail.errnum, " is not positive definite");
  }
  else {
    printf("Error from "
           "nag_real_sym_posdef_packed_lin_solve (f04bec).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
END:
  NAG_FREE(ap);
  NAG_FREE(b);

  return exit_status;
}
```

10.2 Program Data

```
nag_real_sym_posdef_packed_lin_solve (f04bec) Example Program Data

   4      2                  :Values of n and nrhs
   Nag_Upper                 :Value of uplo
   4.16  -3.12   0.56  -0.10
          5.03  -0.83   1.18
                 0.76   0.34
                        1.18 :End of matrix A

   8.70   8.30
 -13.35   2.13
   1.89   1.61
  -4.14   5.00               :End of matrix B
```

10.3 Program Results

```
nag_real_sym_posdef_packed_lin_solve (f04bec) Example Program Results

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

Estimate of condition number
         9.7e+01


Estimate of error bound for computed solutions
         1.1e-14
```