NAG Library Function Document

nag_real_sym_posdef_band_lin_solve (f04bfc)

1 Purpose

nag_real_sym_posdef_band_lin_solve (f04bfc) computes the solution to a real system of linear equations $AX = B$, where A is an n by n symmetric positive definite band matrix of band width $2k + 1$, 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_band_lin_solve (Nag_OrderType order,
     Nag_UploType uplo, Integer n, Integer kd, Integer nrhs, double ab[],
     Integer pdab, 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 band matrix with k superdiagonals, and L is a lower triangular band matrix with k subdiagonals. 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: **kd** – Integer *Input*

On entry: the number of superdiagonals k (and the number of subdiagonals) of the band matrix A.

Constraint: **kd** ≥ 0.

5: **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.

6: **ab**[*dim*] – double *Input/Output*

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

On entry:

if **uplo** = Nag_Upper then

if **order** = Nag_ColMajor, a_{ij} is stored in $\mathbf{ab}[(j-1) \times \mathbf{pdab} + \mathbf{kd} + i - j]$;

if **order** = Nag_RowMajor, a_{ij} is stored in $\mathbf{ab}[(i-1) \times \mathbf{pdab} + j - i]$,

for $\max(1, j - \mathbf{kd}) \leq i \leq j$;

if **uplo** = Nag_Lower then

if **order** = Nag_ColMajor, a_{ij} is stored in $\mathbf{ab}[(j-1) \times \mathbf{pdab} + i - j]$;

if **order** = Nag_RowMajor, a_{ij} is stored in $\mathbf{ab}[(i-1) \times \mathbf{pdab} + \mathbf{kd} + j - i]$,

for $j \leq i \leq \min(n, j + \mathbf{kd})$,

where **pdab** $\geq$ **kd** $+ 1$ is the stride separating diagonal matrix elements in the array **ab**.

See Section 9 below for further details.

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.

7: **pdab** – Integer *Input*

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

Constraint: **pdab** $\geq$ **kd** $+ 1$.

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

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

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.

NE_BAD_PARAM

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

NE_INT

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

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

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

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

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

NE_INT_2

On entry, **pdab** $= \langle$*value*$\rangle$ and **kd** $= \langle$*value*$\rangle$.
Constraint: **pdab** $\geq$ **kd** $+ 1$.

On entry, **pdb** $= \langle$*value*$\rangle$ and **n** $= \langle$*value*$\rangle$.
Constraint: **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_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 $\kappa(A) = \left\|A^{-1}\right\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. nag_real_sym_posdef_band_lin_solve (f04bfc) 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_band_lin_solve (f04bfc) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library.

nag_real_sym_posdef_band_lin_solve (f04bfc) 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 band storage schemes for the array **ab** are identical to the storage schemesfor symmetric and Hermitian band matrices in Chapter f07. See Section 3.3.4 in the f07 Chapter Introduction for details of the storage schemes and an illustrated example.

If **uplo** $=$ Nag_Upper then the elements of the stored upper triangular part of A are overwritten by the corresponding elements of the upper triangular matrix U. Similarly, if **uplo** $=$ Nag_Lower then the elements of the stored lower triangular part of A are overwritten by the corresponding elements of the lower triangular matrix L.

Assuming that $n \gg k$, the total number of floating-point operations required to solve the equations $AX = B$ is approximately $n(k+1)^2$ for the factorization and $4nkr$ for the solution following the factorization. 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_band_lin_solve (f04bfc) is nag_herm_posdef_band_lin_solve (f04cfc).

10 Example

This example solves the equations

$$AX = B,$$

where A is the symmetric positive definite band matrix

$$A = \begin{pmatrix} 5.49 & 2.68 & 0 & 0 \\ 2.68 & 5.63 & -2.39 & 0 \\ 0 & -2.39 & 2.60 & -2.22 \\ 0 & 0 & -2.22 & 5.17 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 22.09 & 5.10 \\ 9.31 & 30.81 \\ -5.24 & -25.82 \\ 11.83 & 22.90 \end{pmatrix}.$$

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_band_lin_solve (f04bfc) 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, kd, n, nrhs, pdab, pdb;

  /* Arrays */
  char          nag_enum_arg[20];
  double        *ab = 0, *b = 0;

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

#ifdef NAG_COLUMN_MAJOR
#define AB_U(I, J) ab[(J-1)*pdab + kd + I - J]
#define AB_L(I, J) ab[(J-1)*pdab + I - J]
#define B(I, J)    b[(J-1)*pdb +  I - 1]
  order = Nag_ColMajor;
#else
#define AB_U(I, J) ab[(I-1)*pdab + J - I]
#define AB_L(I, J) ab[(I-1)*pdab + kd + J - I]
#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_band_lin_solve (f04bfc)"
         " Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%*[^\n] ", &n, &kd, &nrhs);
  if (n > 0 && kd > 0 && nrhs > 0)
    {
      /* Allocate memory */
      if (!(ab = NAG_ALLOC((kd+1)*n, double)) ||
          !(b = NAG_ALLOC(n*nrhs, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      pdab = kd+1;
#ifdef NAG_COLUMN_MAJOR
      pdb = n;
#else
      pdb = nrhs;
#endif
    }
  else
    {
      printf("%s\n", "One or more of n, kd and nrhs is too small");
      exit_status = 1;
      return exit_status;
    }

  /* Read uplo storage name for the matrix A and convert to value. */
  scanf("%19s%*[^\n] ", nag_enum_arg);

  /* 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)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= MIN(n, i + kd); ++j)
            {
              scanf("%lf", &AB_U(i, j));
            }
          scanf("%*[^\n] ");
        }
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = MAX(1, i - kd); j <= i; ++j)
            {
              scanf("%lf", &AB_L(i, j));
            }
          scanf("%*[^\n] ");
        }
    }

  /* Read B from data file */
  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_sym_posdef_band_lin_solve (f04bfc).
   * Computes the solution and error-bound to a real symmetric
   * positive-definite banded system of linear equations
   */
  nag_real_sym_posdef_band_lin_solve(order, uplo, n, kd, nrhs, ab, pdab,
                                     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\n", "Estimate of condition number", "",
              1.0/rcond);

      printf("\n%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%3ld%s\n\n", "The leading minor of order ",
              fail.errnum, " is not positive definite");
    }
  else
    {
      printf(
              "Error from nag_real_sym_posdef_band_lin_solve (f04bfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(ab);
  NAG_FREE(b);

  return exit_status;
}
```

10.2 Program Data

```
nag_real_sym_posdef_band_lin_solve (f04bfc) Example Program Data

  4      1      2           :Values of n, kd and nrhs
  Nag_Upper                 :Name of uplo
  5.49   2.68
         5.63  -2.39
                2.60  -2.22
                       5.17 :End of matrix A

 22.09   5.10
  9.31  30.81
 -5.24 -25.82
 11.83  22.90               :End of matrix B
```

10.3 Program Results

```
nag_real_sym_posdef_band_lin_solve (f04bfc) Example Program Results

 Solution
             1          2
 1      5.0000    -2.0000
 2     -2.0000     6.0000
 3     -3.0000    -1.0000
 4      1.0000     4.0000

Estimate of condition number
         7.4e+01


Estimate of error bound for computed solutions
         8.2e-15
```