NAG Library Function Document

nag_herm_posdef_packed_lin_solve (f04cec)

1 Purpose

nag_herm_posdef_packed_lin_solve (f04cec) computes the solution to a complex system of linear equations $AX = B$, where A is an n by n Hermitian 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_herm_posdef_packed_lin_solve (Nag_OrderType order,
     Nag_UploType uplo, Integer n, Integer nrhs, Complex ap[], Complex b[],
     Integer pdb, double *rcond, double *errbnd, NagError *fail)
```

3 Description

The Cholesky factorization is used to factor A as $A = U^{H}U$, if **uplo** $=$ Nag_Upper, or $A = LL^{H}$, 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 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: **ap**$[dim]$ – Complex *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 Hermitian matrix A. The upper or lower triangular part of the Hermitian matrix is packed column-wise in a linear array. The jth column of A is stored in the array **ap** as follows:

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

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

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 $A = U^{\mathrm{H}}U$ or $A = LL^{\mathrm{H}}$, in the same storage format as A.

6: **b**$[dim]$ – Complex *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

b$[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
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, **pdb** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **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 **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 $\|\hat{x} - x\|_1 / \|x\|_1 \leq$ **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 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

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

NE_INT_2

On entry, **pdb** $=$ ⟨*value*⟩ and **n** $=$ ⟨*value*⟩.
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 ⟨*value*⟩ 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_herm_posdef_packed_lin_solve (f04cec) 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_herm_posdef_packed_lin_solve (f04cec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_herm_posdef_packed_lin_solve (f04cec) 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 packed storage scheme is illustrated by the following example when $n = 4$ and **uplo** = Nag_Upper. Two-dimensional storage of the Hermitian matrix A:

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

Packed storage of the upper triangle of A:

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

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 real analogue of nag_herm_posdef_packed_lin_solve (f04cec) is nag_real_sym_posdef_packed_lin_solve (f04bec).

10 Example

This example solves the equations

$$AX = B,$$

where A is the Hermitian positive definite matrix

$$A = \begin{pmatrix} 3.23 & 1.51 - 1.92i & 1.90 + 0.84i & 0.42 + 2.50i \\ 1.51 + 1.92i & 3.58 & -0.23 + 1.11i & -1.18 + 1.37i \\ 1.90 - 0.84i & -0.23 - 1.11i & 4.09 & 2.33 - 0.14i \\ 0.42 - 2.50i & -1.18 - 1.37i & 2.33 + 0.14i & 4.29 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 3.93 - 6.14i & 1.48 + 6.58i \\ 6.17 + 9.42i & 4.65 - 4.75i \\ -7.17 - 21.83i & -4.91 + 2.29i \\ 1.99 - 14.38i & 7.64 - 10.79i \end{pmatrix}.$$

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_herm_posdef_packed_lin_solve (f04cec) 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           nag_enum_arg[40];
  char           *clabs = 0, *rlabs = 0;
  Complex        *ap = 0, *b = 0;

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

#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_herm_posdef_packed_lin_solve (f04cec) 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 (!(clabs = NAG_ALLOC(2, char)) ||
          !(rlabs = NAG_ALLOC(2, char)) ||
          !(ap = NAG_ALLOC(n*(n+1)/2, Complex)) ||
          !(b = NAG_ALLOC(n*nrhs, Complex)))
        {
          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;
    }
  scanf("%39s%*[^\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);
```

```
/* 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)
          {
            scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
                   &A_UPPER(i, j).im);
          }
      }
    scanf("%*[^\n] ");
  }
else
  {
    for (i = 1; i <= n; ++i)
      {
        for (j = 1; j <= i; ++j)
          {
            scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
                   &A_LOWER(i, j).im);
          }
      }
    scanf("%*[^\n] ");
  }

/* Read B from data file */
for (i = 1; i <= n; ++i)
  {
    for (j = 1; j <= nrhs; ++j)
      {
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
      }
  }
scanf("%*[^\n] ");

/* Solve the equations AX = B for X */
/* nag_herm_posdef_packed_lin_solve (f04cec).
 * Computes the solution and error-bound to a complex
 * Hermitian positive-definite system of linear equations,
 * packed storage
 */
nag_herm_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_complx_mat_print_comp (x04dbc).
     * Print complex general matrix (comprehensive)
     */
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                                  n, nrhs, b, pdb, Nag_BracketForm, "%7.4f",
                                  "Solution", Nag_IntegerLabels, 0,
                                  Nag_IntegerLabels, 0, 80, 0, 0,
                                  &fail);
    if (fail.code != NE_NOERROR)
      {
        printf(
                "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
      }
    printf("\n");
    printf("%s\n%4s%10.1e\n", "Estimate of condition number", "",
            1.0/rcond);
```

```
      printf("\n\n");
      printf("%s\n%4s%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%4s%10.1e\n\n\n",
             "Estimate of reciprocal of condition number", "", rcond);
      /* nag_gen_complx_mat_print_comp (x04dbc), see above. */
      fflush(stdout);
      nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                                    n, nrhs, b, pdb, Nag_BracketForm, "%7.4f",
                                    "Solution", Nag_IntegerLabels, 0,
                                    Nag_IntegerLabels, 0, 80, 0, 0,
                                    &fail);
      if (fail.code != NE_NOERROR)
        {
          printf(
                 "Error from nag_gen_complx_mat_print_comp (x04dbc).\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_herm_posdef_packed_lin_solve (f04cec).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(clabs);
  NAG_FREE(rlabs);
  NAG_FREE(ap);
  NAG_FREE(b);

  return exit_status;
}
```

10.2 Program Data

```
nag_herm_posdef_packed_lin_solve (f04cec) Example Program Data

   4             2                                     :Values of n and nrhs
   Nag_Upper                                           :Value of uplo
 ( 3.23,  0.00) ( 1.51, -1.92) ( 1.90,  0.84) ( 0.42,  2.50)
                ( 3.58,  0.00) (-0.23,  1.11) (-1.18,  1.37)
                               ( 4.09,  0.00) ( 2.33, -0.14)
                                              ( 4.29,  0.00) :End of matrix A

 ( 3.93, -6.14) ( 1.48,  6.58)
 ( 6.17,  9.42) ( 4.65, -4.75)
 (-7.17,-21.83) (-4.91,  2.29)
 ( 1.99,-14.38) ( 7.64,-10.79)                                :End of matrix B
```

10.3 Program Results

```
nag_herm_posdef_packed_lin_solve (f04cec) Example Program Results

 Solution
                    1                   2
 1  ( 1.0000,-1.0000)  (-1.0000, 2.0000)
 2  (-0.0000, 3.0000)  ( 3.0000,-4.0000)
 3  (-4.0000,-5.0000)  (-2.0000, 3.0000)
 4  ( 2.0000, 1.0000)  ( 4.0000,-5.0000)

Estimate of condition number
       1.5e+02


Estimate of error bound for computed solutions
       1.7e-14
```