NAG C Library Function Document

nag_herm_lin_solve (f04chc)

1 Purpose

nag_herm_lin_solve (f04chc) computes the solution to a complex system of linear equations $AX = B$, where A is an n by n Hermitian 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_herm_lin_solve (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer nrhs, Complex a[], Integer pda, Integer ipiv[], Complex b[],
     Integer pdb, double *rcond, double *errbnd, NagError *fail)
```

3 Description

The diagonal pivoting method is used to factor A as $A = UDU^{\mathrm{H}}$, if **uplo** = **Nag_Upper**, or $A = LDL^{\mathrm{H}}$, if **uplo** = **Nag_Lower**, where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian 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: **a**[dim] – Complex *Input/Output*

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

If **order** = **Nag_ColMajor**, the (i,j)th element of the matrix A is stored in **a**$[(j-1) \times \textbf{pda} + i - 1]$.

If **order** = **Nag_RowMajor**, the (i,j)th element of the matrix A is stored in **a**$[(i-1) \times \textbf{pda} + j - 1]$.

On entry: the n by n Hermitian 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**, **NE_SINGULAR** or **NE_RCOND**, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^{\mathrm{H}}$ or $A = LDL^{\mathrm{H}}$ as computed by nag_zhetrf (f07mrc).

6: **pda** – Integer *Input*

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

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

7: **ipiv**[dim] – Integer *Output*

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

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_zhetrf (f07mrc).

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

8: **b**[dim] – Complex *Input/Output*

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

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

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

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

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.

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

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 2.6 of 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, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

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

NE_INT_2

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

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

On entry, **pdb** = ⟨*value*⟩, **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 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 ⟨*value*⟩ 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 $\kappa(A) = \left\|A^{-1}\right\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. nag_herm_lin_solve (f04chc) 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 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.

Function nag_complex_sym_lin_solve (f04dhc) is for complex symmetric matrices, and the real analogue of nag_herm_lin_solve (f04chc) is nag_real_sym_lin_solve (f04bhc).

9 Example

To solve the equations

$$AX = B,$$

where A is the Hermitian indefinite matrix

$$A = \begin{pmatrix} -1.84 & 0.11 - 0.11i & -1.78 - 1.18i & 3.91 - 1.50i \\ 0.11 + 0.11i & -4.63 & -1.84 + 0.03i & 2.21 + 0.21i \\ -1.78 + 1.18i & -1.84 - 0.03i & -8.87 & 1.58 - 0.90i \\ 3.91 + 1.50i & 2.21 - 0.21i & 1.58 + 0.90i & -1.36 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 2.98 - 10.18i & 28.68 - 39.89i \\ -9.58 + 3.88i & -24.79 - 8.40i \\ -0.77 - 16.05i & 4.23 - 70.02i \\ 7.79 + 5.48i & -35.39 + 18.01i \end{pmatrix}.$$

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_herm_lin_solve (f04chc) 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, pda, pdb;

  /* Arrays */
  char   uplo[2];
  char *clabs=0, *rlabs=0;
  Complex *a=0, *b=0;
  Integer *ipiv=0;

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

#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);

  Vprintf("nag_herm_lin_solve (f04chc) 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 ( !(clabs = NAG_ALLOC(2, char)) ||
           !(rlabs = NAG_ALLOC(2, char)) ||
           !(a = NAG_ALLOC(n*n, Complex)) ||
           !(b = NAG_ALLOC(n*nrhs, Complex)) ||
           !(ipiv = NAG_ALLOC(n, Integer)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
#ifdef NAG_COLUMN_MAJOR
      pda = n;
      pdb = n;
#else
      pda = n;
```

```
      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;
    }

  if (uplo_enum == Nag_Upper)
    {
      /* Read the upper triangular part of A from data file */
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            {
              Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
            }
        }
      Vscanf("%*[^\n] ");
    }
  else
    {
      /* Read the lower triangular part of A from data file */
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            {
              Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
            }
        }
      Vscanf("%*[^\n] ");
    }

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

  /* Solve the equations AX = B for X */
  /* nag_herm_lin_solve (f04chc).
   * Computes the solution and error-bound to a complex
   * Hermitian system of linear equations
   */
  nag_herm_lin_solve(order, uplo_enum, 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_complx_mat_print_comp (x04dbc).
       * Print complex general matrix (comprehensive)
       */
```

```
      nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                                    n, nrhs, b, pdb, Nag_BracketForm, 0,
                                    "Solution", Nag_IntegerLabels, 0,
                                    Nag_IntegerLabels, 0,  80, 0, 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      Vprintf("\n");
      Vprintf("%s\n%8s%9.1e\n", "Estimate of condition number", "", 1.0/rcond);
      Vprintf("\n\n");
      Vprintf("%s\n%8s%9.1e\n\n",
              "Estimate of error bound for computed solutions", "",  errbnd);
    }
  if (fail.code == NE_RCOND)
    {

      /* Matrix A is numerically singular.  Print estimate of */
      /* reciprocal of condition number and solution */

      Vprintf("\n");
      Vprintf("%s\n%8s%9.1e\n\n\n",
              "Estimate of reciprocal of condition number", "", rcond);
      /* nag_gen_complx_mat_print_comp (x04dbc), see above. */
      nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                                    n, nrhs, b, pdb, Nag_BracketForm, 0,
                                    "Solution", Nag_IntegerLabels, 0,
                                    Nag_IntegerLabels, 0, 80, 0, 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
    }
  if (fail.code == NE_SINGULAR)
    {

      /* The upper triangular matrix U is exactly singular.  Print */
      /* details of factorization */

      Vprintf("\n");
      /* nag_gen_complx_mat_print_comp (x04dbc), see above. */
      nag_gen_complx_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, n,
                                    n, a, pda, Nag_BracketForm, 0,
                                    "Details of factorization",
                                    Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                                    80, 0, 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }


      /* Print pivot indices */


      Vprintf("\n");
      Vprintf("%s\n",   "Pivot indices");
      for (i = 1; i <= n; ++i)
        {
          Vprintf("%11ld%s", ipiv[i - 1], i%7 == 0 || i == n ?"\n":" ");
```

```
        }
      Vprintf("\n");
    }
 END:
  if (clabs) NAG_FREE(clabs);
  if (rlabs) NAG_FREE(rlabs);
  if (a) NAG_FREE(a);
  if (b) NAG_FREE(b);
  if (ipiv) NAG_FREE(ipiv);

  return exit_status;
}

#undef B
#undef A
```

9.2 Program Data

```
nag_herm_lin_solve (f04chc) Example Program Data

    4              2                                                :N and NRHS
    'U'                                                             :UPLO
 ( -1.84,  0.00) (  0.11, -0.11) ( -1.78, -1.18) (  3.91, -1.50)
                 ( -4.63 , 0.00) ( -1.84,  0.03) (  2.21,  0.21)
                                 ( -8.87,  0.00) (  1.58, -0.90)
                                                 ( -1.36 , 0.00) :End matrix A

 (  2.98,-10.18) ( 28.68,-39.89)
 ( -9.58,  3.88) (-24.79, -8.40)
 ( -0.77,-16.05) (  4.23,-70.02)
 (  7.79,  5.48) (-35.39, 18.01)                                 :End matrix B
```

9.3 Program Results

```
nag_herm_lin_solve (f04chc) Example Program Results

 Solution

                             1                         2
 1  (     2.0000,     1.0000)  (    -8.0000,     6.0000)
 2  (     3.0000,    -2.0000)  (     7.0000,    -2.0000)
 3  (    -1.0000,     2.0000)  (    -1.0000,     5.0000)
 4  (     1.0000,    -1.0000)  (     3.0000,    -4.0000)

Estimate of condition number
          6.7e+00


Estimate of error bound for computed solutions
          7.4e-16
```