# NAG Library Function Document

# nag_herm_packed_lin_solve (f04cjc)

## 1    Purpose

nag_herm_packed_lin_solve (f04cjc) computes the solution to a complex system of linear equations $AX = B$, where $A$ is an $n$ by $n$ complex Hermitian 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_packed_lin_solve (Nag_OrderType order, Nag_UploType uplo,
      Integer n, Integer nrhs, Complex ap[], 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 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*: $\mathbf{n} \geq 0$.

4:    **nrhs** – Integer                                                                                                  *Input*

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

Constraint: **nrhs** $\geq 0$.

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 $A$, packed column-wise in a linear array. The $j$th column of the matrix $A$ 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, 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_zhptrf (f07prc), stored as a packed triangular matrix in the same storage format as $A$.

6:    **ipiv**[**n**] – Integer                                                                                          *Output*

On exit: if **fail.code** = NE_NOERROR, details of the interchanges and the block structure of $D$, as determined by nag_zhptrf (f07prc).

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.

7:    **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 $X$.

8:    **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$(1, \mathbf{nrhs})$.

9:    **rcond** – double *                                                                                             *Output*

On exit: if no constraints are violated, an estimate of the reciprocal of the condition number of the matrix $A$, computed as **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 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** ≥ max(1, **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_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

$$\|E\|_1 = O(\epsilon)\|A\|_1$$

and $\epsilon$ 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_herm_packed_lin_solve (f04cjc) 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_packed_lin_solve (f04cjc) is not threaded by NAG in any implementation.

nag_herm_packed_lin_solve (f04cjc) 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 $A$:

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

Packed storage of the upper triangle of $A$:

$$\mathbf{ap} = \begin{bmatrix} a_{11}, & a_{12}, & a_{22}, & a_{13}, & a_{23}, & a_{33}, & a_{14}, & a_{24}, & a_{34}, & a_{44} \end{bmatrix}.$$

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_packed_lin_solve (f04djc) is for complex symmetric matrices, and the real analogue of nag_herm_packed_lin_solve (f04cjc) is nag_real_sym_packed_lin_solve (f04bjc).

## 10 Example

This example solves 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.

## 10.1 Program Text

```c
/* nag_herm_packed_lin_solve (f04cjc) 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;
  Integer       *ipiv = 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_herm_packed_lin_solve (f04cjc) 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)) ||
          !(ipiv = NAG_ALLOC(n, Integer)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
```

```
#ifdef NAG_COLUMN_MAJOR
      pdb = n;
#else
      pdb = nrhs;
#endif
      /* Read A and B from data file */
    }
  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_packed_lin_solve (f04cjc).
   * Computes the solution and error-bound to a complex
   * Hermitian system of linear equations, packed storage
   */
  nag_herm_packed_lin_solve(order, uplo, 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_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, 0,
                                    "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%8s%10.1e\n", "Estimate of condition number", "",
             1.0/rcond);
      printf("\n\n");
      printf("%s\n%8s%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%8s%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, 0,
                                    "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_SINGULAR)
    {
      /* The upper triangular matrix U is exactly singular.  Print */
      /* details of factorization */
      printf("\n");
      /* nag_pack_complx_mat_print_comp (x04ddc).
       * Print complex packed triangular matrix (comprehensive)
       */
      fflush(stdout);
      nag_pack_complx_mat_print_comp(order, Nag_Upper, Nag_NonUnitDiag, n, ap,
                                     Nag_BracketForm, 0,
                                     "Details of factorization",
                                     Nag_IntegerLabels, 0, Nag_IntegerLabels,
                                     0, 80, 0, 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf(
                  "Error from nag_pack_complx_mat_print_comp (x04ddc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      /* Print pivot indices */
      printf("\n");
      printf("%s\n", "Pivot indices");
      for (i = 1; i <= n; ++i)
```

```
      {
        printf("%11ld%s", ipiv[i - 1], i%7 == 0 || i == n?"\n":" ");
      }
    printf("\n");
  }
else
  {
    printf("Error from nag_herm_packed_lin_solve (f04cjc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

END:
 NAG_FREE(clabs);
 NAG_FREE(rlabs);
 NAG_FREE(ap);
 NAG_FREE(b);
 NAG_FREE(ipiv);

 return exit_status;
}

#undef B
```

## 10.2  Program Data

```
nag_herm_packed_lin_solve (f04cjc) Example Program Data

    4                 2                                      :n and nrhs
   Nag_Upper                                                 :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
```

## 10.3  Program Results

```
nag_herm_packed_lin_solve (f04cjc) 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
```