

## NAG Library Function Document

### nag\_complex\_sym\_lin\_solve (f04dhc)

#### 1 Purpose

nag\_complex\_sym\_lin\_solve (f04dhc) computes the solution to a complex system of linear equations  $AX = B$ , where  $A$  is an  $n$  by  $n$  complex symmetric 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_complex_sym_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  $A = UDU^T$ , if **uplo** = Nag\_Upper, or  $A = LDL^T$ , if **uplo** = Nag\_Lower, where  $U$  (or  $L$ ) is a product of permutation and unit upper (lower) triangular matrices, and  $D$  is symmetric 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:* **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: **a**[*dim*] – Complex *Input/Output*  
**Note:** the dimension, *dim*, of the array **a** must be at least  $\max(1, \mathbf{pda} \times \mathbf{n})$ .  
The ( $i, j$ )th element of the matrix  $A$  is stored in  

$$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor};$$

$$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*On entry:* the  $n$  by  $n$  complex symmetric 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, the block diagonal matrix  $D$  and the multipliers used to obtain the factor  $U$  or  $L$  from the factorization  $A = UDU^T$  or  $A = LDL^T$  as computed by nag\_zsytrf (f07nrc).
- 6: **pda** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **a**.  
*Constraint:* **pda**  $\geq \max(1, \mathbf{n})$ .
- 7: **ipiv**[ $\mathbf{n}$ ] – Integer *Output*  
*On exit:* if **fail.code** = NE\_NOERROR, details of the interchanges and the block structure of  $D$ , as determined by nag\_zsytrf (f07nrc).  
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  $-\mathbf{ipiv}[k-1]$  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  $-\mathbf{ipiv}[k-1]$  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  
 $\max(1, \mathbf{pdb} \times \mathbf{nrhs})$  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  

$$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor};$$

$$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.$$
*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 no constraints are violated, an estimate of the reciprocal of the condition number of the matrix  $A$ , computed as  $\mathbf{rcond} = 1 / (\|A\|_1 \|A^{-1}\|_1)$ .

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.

See Section 3.2.1.2 in the Essential Introduction for further information.

### NE\_BAD\_PARAM

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

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

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

On entry, **nrhs** =  $\langle value \rangle$ .

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

On entry, **pda** =  $\langle value \rangle$ .

Constraint: **pda**  $> 0$ .

On entry, **pdb** =  $\langle value \rangle$ .

Constraint: **pdb**  $> 0$ .

### NE\_INT\_2

On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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

On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **pdb**  $\geq \max(1, \mathbf{n})$ .

On entry, **pdb** =  $\langle value \rangle$  and **nrhs** =  $\langle value \rangle$ .

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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 3.6.6 in the Essential Introduction for further information.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
See Section 3.6.5 in the Essential Introduction for further information.

**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_complex_sym_lin_solve` (f04dhc) 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_complex_sym_lin_solve` (f04dhc) is not threaded by NAG in any implementation.

`nag_complex_sym_lin_solve` (f04dhc) 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 double allocatable memory required is **n**, and the Complex allocatable memory required is  $\max(2 \times \mathbf{n}, \mathbf{lwork})$ , where **lwork** is the optimum workspace required by `nag_zsysv` (f07nnc). If this failure occurs it may be possible to solve the equations by calling the packed storage version of `nag_complex_sym_lin_solve` (f04dhc), `nag_complex_sym_packed_lin_solve` (f04djc), or by calling `nag_zsysv` (f07nnc) directly with less than the optimum workspace (see Chapter f07).

The total number of floating-point operations required to solve the equations  $AX = B$  is proportional to  $(\frac{1}{3}n^3 + 2n^2r)$ . 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_herm_lin_solve` (f04chc) is for complex Hermitian matrices, and the real analogue of `nag_complex_sym_lin_solve` (f04dhc) is `nag_real_sym_lin_solve` (f04bhc).

## 10 Example

This example solves the equations

$$AX = B,$$

where  $A$  is the symmetric indefinite matrix

$$A = \begin{pmatrix} -0.56 + 0.12i & -1.54 - 2.86i & 5.32 - 1.59i & 3.80 + 0.92i \\ -1.54 - 2.86i & -2.83 - 0.03i & -3.52 + 0.58i & -7.86 - 2.96i \\ 5.32 - 1.59i & -3.52 + 0.58i & 8.86 + 1.81i & 5.14 - 0.64i \\ 3.80 + 0.92i & -7.86 - 2.96i & 5.14 - 0.64i & -0.39 - 0.71i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -6.43 + 19.24i & -4.59 - 35.53i \\ -0.49 - 1.47i & 6.95 + 20.49i \\ -48.18 + 66.00i & -12.08 - 27.02i \\ -55.64 + 41.22i & -19.09 - 35.97i \end{pmatrix}.$$

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_complex_sym_lin_solve (f04dhc) Example Program.
 *
 * Copyright 2014 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        nag_enum_arg[40];
    char        *clabs = 0, *rlabs = 0;
    Complex     *a = 0, *b = 0;
    Integer     *ipiv = 0;

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

#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);

    printf(

```

```

        "nag_complex_sym_lin_solve (f04dhc) 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 (!(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)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
#ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
#else
        pda = n;
        pdb = nrhs;
#endif
    }
    else
    {
        printf("%s\n", "n and/or nrhs too small");
        exit_status = 1;
        return exit_status;
    }

    /* Read A and B from data file */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, _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 triangular part of A from data file */
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
        {
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#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 , %lf )", &B(i, j).re, &B(i, j).im);
#else
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
        }
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Solve the equations AX = B for X */
/* nag_complex_sym_lin_solve (f04dhc).
 * Computes the solution and error-bound to a complex
 * symmetric system of linear equations
 */
nag_complex_sym_lin_solve(order, uplo, 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)
     */
    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_complex_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_gen_complex_mat_print_comp (x04dbc), see above. */
    fflush(stdout);
    nag_gen_complex_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)
    {
        printf(
            "Error from nag_gen_complex_mat_print_comp (x04dbc).\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("%11"NAG_IFMT"%s", ipiv[i - 1],
              i%7 == 0 || i == n?"\n":" ");
    }
    printf("\n");
}
else
{
    printf("Error from nag_complex_sym_lin_solve (f04dhc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(clabs);
NAG_FREE(rlabs);
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(ipiv);

return exit_status;
}

#undef B
#undef A

```



**10.2 Program Data**

nag\_complex\_sym\_lin\_solve (f04dhc) Example Program Data

```

      4          2          :n and nrhs
      Nag_Upper          :uplo
( -0.56,  0.12) ( -1.54, -2.86) (  5.32, -1.59) (  3.80,  0.92)
          ( -2.83 , -0.03) ( -3.52,  0.58) ( -7.86, -2.96)
          (  8.86,  1.81) (  5.14, -0.64)
          ( -0.39 , -0.71) :End matrix A

( -6.43, 19.24) ( -4.59, -35.53)
( -0.49, -1.47) (  6.95, 20.49)
(-48.18, 66.00) (-12.08, -27.02)
(-55.64, 41.22) (-19.09, -35.97)          :End matrix B

```

**10.3 Program Results**

nag\_complex\_sym\_lin\_solve (f04dhc) Example Program Results

```

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

Estimate of condition number
      2.1e+01

Estimate of error bound for computed solutions
      2.3e-15

```

---