

## NAG Library Function Document

### nag\_real\_sym\_posdef\_lin\_solve (f04bdc)

#### 1 Purpose

nag\_real\_sym\_posdef\_lin\_solve (f04bdc) computes the solution to a real system of linear equations  $AX = B$ , where  $A$  is an  $n$  by  $n$  symmetric positive definite 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_real_sym_posdef_lin_solve (Nag_OrderType order, Nag_UploType uplo,
    Integer n, Integer nrhs, double a[], Integer pda, double b[],
    Integer pdb, double *rcond, double *errbnd, NagError *fail)
```

#### 3 Description

The Cholesky factorization is used to factor  $A$  as  $A = U^T U$ , if **uplo** = Nag\_Upper, or  $A = LL^T$ , 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**  $\geq$  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*] – double *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  
 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:* the  $n$  by  $n$  symmetric matrix  $A$ .  
If **uplo** = Nag\_Upper, the leading  $\mathbf{n}$  by  $\mathbf{n}$  upper triangular part of **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  $\mathbf{n}$  by  $\mathbf{n}$  lower triangular part of **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 or NE\_RCOND, the factor  $U$  or  $L$  from the Cholesky factorization  $A = U^T U$  or  $A = LL^T$ .
- 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: **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  $X$ .
- 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 **fail.code** = NE\_NOERROR or NE\_RCOND, an estimate of the reciprocal of the condition number of the matrix  $A$ , computed as  $\mathbf{rcond} = 1 / (\|A\|_1 \|A^{-1}\|_1)$ .
- 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  $\|\hat{x} - x\|_1 / \|x\|_1 \leq \mathbf{errbnd}$ , 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.

- 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.  
 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\_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  $\epsilon$  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_real_sym_posdef_lin_solve` (f04bdc) 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_lin_solve` (f04bdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_real_sym_posdef_lin_solve` (f04bdc) 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 Integer allocatable memory required is  $\mathbf{n}$ , and the double allocatable memory required is  $3 \times \mathbf{n}$ . Allocation failed before the solution could be computed.

The total number of floating-point operations required to solve the equations  $AX = B$  is proportional to  $(\frac{1}{3}n^3 + n^2r)$ . 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.

The complex analogue of `nag_real_sym_posdef_lin_solve` (f04bdc) is `nag_herm_posdef_lin_solve` (f04cdc).

## 10 Example

This example solves the equations

$$AX = B,$$

where  $A$  is the symmetric positive definite matrix

$$A = \begin{pmatrix} 4.16 & -3.12 & 0.56 & -0.10 \\ -3.12 & 5.03 & -0.83 & 1.18 \\ 0.56 & -0.83 & 0.76 & 0.34 \\ -0.10 & 1.18 & 0.34 & 1.18 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 8.70 & 8.30 \\ -13.35 & 2.13 \\ 1.89 & 1.61 \\ -4.14 & 5.00 \end{pmatrix}.$$

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_lin_solve (f04bdc) 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];
    double       *a = 0, *b = 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_real_sym_posdef_lin_solve (f04bdc) 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 (!(a = NAG_ALLOC(n*n, double)) ||
            !(b = NAG_ALLOC(n*nrhs, double)))
        {
            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;
}

#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);

if (uplo == Nag_Upper)
{
    /* 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", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#endif
        }
    }
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
}
else
{
    /* Read the lower triangular part of A from data file */
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
#ifdef _WIN32
            scanf_s("%lf", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#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", &B(i, j));
#else

```

```

        scanf("%lf", &B(i, j));
#endif
    }
}
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Solve the equations AX = B for X */

/* nag_real_sym_posdef_lin_solve (f04bdc).
 * Computes the solution and error-bound to a real symmetric
 * positive-definite system of linear equations
 */
nag_real_sym_posdef_lin_solve(order, uplo, n, nrhs, a, pda, 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");
    printf("%s\n%6s%10.1e\n", "Estimate of condition number", "",
           1.0/rcond);
    printf("\n\n");

    printf("%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%3"NAG_IFMT"%s\n\n", "The leading minor of order ",
           fail.errnum, " is not positive definite");
}
else
{

```

```

    printf(
        "Error from nag_real_sym_posdef_lin_solve (f04bdc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(a);
NAG_FREE(b);

return exit_status;
}

```

## 10.2 Program Data

nag\_real\_sym\_posdef\_lin\_solve (f04bdc) Example Program Data

```

4      2      :Values of n and nrhs
Nag_Upper :Value of uplo
4.16  -3.12   0.56  -0.10
      5.03  -0.83   1.18
           0.76   0.34
           1.18 :End of matrix A

8.70   8.30
-13.35  2.13
 1.89   1.61
-4.14   5.00      :End of matrix B

```

## 10.3 Program Results

nag\_real\_sym\_posdef\_lin\_solve (f04bdc) Example Program Results

```

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

Estimate of condition number
      9.7e+01

Estimate of error bound for computed solutions
      1.1e-14

```

---