

# NAG Library Function Document

## nag\_dpotsr (f07fec)

### 1 Purpose

nag\_dpotsr (f07fec) solves a real symmetric positive definite system of linear equations with multiple right-hand sides,

$$AX = B,$$

where  $A$  has been factorized by nag\_dpotsr (f07fec).

### 2 Specification

```
#include <nag.h>
#include <nagf07.h>

void nag_dpotsr (Nag_OrderType order, Nag_UploType uplo, Integer n,
                Integer nrhs, const double a[], Integer pda, double b[], Integer pdb,
                NagError *fail)
```

### 3 Description

nag\_dpotsr (f07fec) is used to solve a real symmetric positive definite system of linear equations  $AX = B$ , this function must be preceded by a call to nag\_dpotsr (f07fec) which computes the Cholesky factorization of  $A$ . The solution  $X$  is computed by forward and backward substitution.

If **uplo** = Nag\_Upper,  $A = U^T U$ , where  $U$  is upper triangular; the solution  $X$  is computed by solving  $U^T Y = B$  and then  $UX = Y$ .

If **uplo** = Nag\_Lower,  $A = LL^T$ , where  $L$  is lower triangular; the solution  $X$  is computed by solving  $LY = B$  and then  $L^T X = Y$ .

### 4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

### 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:* specifies how  $A$  has been factorized.

**uplo** = Nag\_Upper

$A = U^T U$ , where  $U$  is upper triangular.

- uplo** = Nag\_Lower  
 $A = LL^T$ , where  $L$  is lower triangular.  
*Constraint:* **uplo** = Nag\_Upper or Nag\_Lower.
- 3: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $A$ .  
*Constraint:*  $n \geq 0$ .
- 4: **nrhs** – Integer *Input*  
*On entry:*  $r$ , the number of right-hand sides.  
*Constraint:* **nrhs**  $\geq 0$ .
- 5: **a**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least  $\max(1, \mathbf{pda} \times \mathbf{n})$ .  
*On entry:* the Cholesky factor of  $A$ , as returned by nag\_dpotr (f07fdc).
- 6: **pda** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) of the matrix 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  $r$  right-hand side matrix  $B$ .  
*On exit:* 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: **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,  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{n} \geq 0$ .

On entry,  $\mathbf{nrhs} = \langle value \rangle$ .

Constraint:  $\mathbf{nrhs} \geq 0$ .

On entry,  $\mathbf{pda} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} > 0$ .

On entry,  $\mathbf{pdb} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} > 0$ .

**NE\_INT\_2**

On entry,  $\mathbf{pda} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{nrhs} = \langle value \rangle$ .

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

**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.

**7 Accuracy**

For each right-hand side vector  $b$ , the computed solution  $x$  is the exact solution of a perturbed system of equations  $(A + E)x = b$ , where

if **uplo** = Nag\_Upper,  $|E| \leq c(n)\epsilon|U^T|U$ ;

if **uplo** = Nag\_Lower,  $|E| \leq c(n)\epsilon|L|L^T$ ,

$c(n)$  is a modest linear function of  $n$ , and  $\epsilon$  is the *machine precision*.

If  $\hat{x}$  is the true solution, then the computed solution  $x$  satisfies a forward error bound of the form

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x)\epsilon$$

where  $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \text{cond}(A) = \| |A^{-1}| |A| \|_\infty \leq \kappa_\infty(A)$ .

Note that  $\text{cond}(A, x)$  can be much smaller than  $\text{cond}(A)$ .

Forward and backward error bounds can be computed by calling `nag_dporfs` (f07fhc), and an estimate for  $\kappa_\infty(A)$  ( $= \kappa_1(A)$ ) can be obtained by calling `nag_dpocon` (f07fgc).

## 8 Parallelism and Performance

nag\_dpotsr (f07fec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dpotsr (f07fec) 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 total number of floating-point operations is approximately  $2n^2r$ .

This function may be followed by a call to nag\_dporfs (f07fhc) to refine the solution and return an error estimate.

The complex analogue of this function is nag\_zpotrs (f07fsc).

## 10 Example

This example solves the system of equations  $AX = B$ , where

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

Here  $A$  is symmetric positive definite and must first be factorized by nag\_dpotrf (f07fdc).

### 10.1 Program Text

```

/* nag_dpotsr (f07fec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer      i, j, n, nrhs, pda, pdb;
    Integer      exit_status = 0;
    NagError     fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    char         nag_enum_arg[40];
    double       *a = 0, *b = 0;

#ifdef NAG_LOAD_FP
    /* The following line is needed to force the Microsoft linker
       to load floating point support */
    float        force_loading_of_ms_float_support = 0;
#endif /* NAG_LOAD_FP */

#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

    INIT_FAIL(fail);

    printf("nag_dpotrs (f07fec) 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
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
#else
    pda = n;
    pdb = nrhs;
#endif
    /* Allocate memory */
    if (!(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* 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);

    if (uplo == Nag_Upper)
    {
        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
    {
        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
    }

    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
#endif

/* Factorize A */
/* nag_dpotrf (f07fdc).
 * Cholesky factorization of real symmetric
 * positive-definite matrix
 */
nag_dpotrf(order, uplo, n, a, pda, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dpotrf (f07fdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution */
/* nag_dpotrs (f07fec).
 * Solution of real symmetric positive-definite system of
 * linear equations, multiple right-hand sides, matrix
 * already factorized by nag_dpotrf (f07fdc)
 */
nag_dpotrs(order, uplo, n, nrhs, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dpotrs (f07fec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print solution */
/* 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(s)", 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;
}
}

```

```

END:
  NAG_FREE(a);
  NAG_FREE(b);
  return exit_status;
}

```

## 10.2 Program Data

```

nag_dpotrs (f07fec) Example Program Data
  4  2                               :Values of n and nrhs
  Nag_Lower                           :Value of uplo
  4.16
 -3.12  5.03
  0.56 -0.83  0.76
 -0.10  1.18  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_dpotrs (f07fec) Example Program Results

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

```

---