NAG C Library Function Document

nag_dsytrs (f07mec)

1 Purpose

nag_dsytrs (f07mec) solves a real symmetric indefinite system of linear equations with multiple right-hand sides, \( AX = B \), where \( A \) has been factorized by nag_dsytrf (f07mdc).

2 Specification

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

3 Description

To solve a real symmetric indefinite system of linear equations \( AX = B \), this function must be preceded by a call to nag_dsytrf (f07mdc) which computes the Bunch–Kaufman factorization of \( A \).

If \( \text{uplo} = \text{Nag Upper} \), \( A = PUDU^T P^T \), where \( P \) is a permutation matrix, \( U \) is an upper triangular matrix and \( D \) is a symmetric block diagonal matrix with 1 by 1 and 2 by 2 blocks; the solution \( X \) is computed by solving \( PUDY = B \) and then \( U^T P^T X = Y \).

If \( \text{uplo} = \text{Nag Lower} \), \( A = PLDL^T P^T \), where \( L \) is a lower triangular matrix; the solution \( X \) is computed by solving \( PLDY = B \) and then \( L^T P^T X = Y \).

4 References


5 Parameters

1: \( \text{order} \) – Nag_OrderType

**Input**

On entry: the \text{order} parameter specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by \text{order} = Nag_RowMajor. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: \text{order} = Nag_RowMajor or Nag_ColMajor.

2: \( \text{uplo} \) – Nag_UploType

**Input**

On entry: indicates how \( A \) has been factorized as follows:

- if \( \text{uplo} = \text{Nag Upper} \), \( A = PUDU^T P^T \), where \( U \) is upper triangular;
- if \( \text{uplo} = \text{Nag Lower} \), \( A = PLDL^T P^T \), where \( L \) is lower triangular.

Constraint: \text{uplo} = Nag_Upper or Nag_Lower.

3: \( n \) – Integer

**Input**

On entry: \( n \), the order of the matrix \( A \).

Constraint: \( n \geq 0 \).
**f07mec**

4: nrhs – Integer  
*Input*  
*On entry:* \( r \), the number of right-hand sides.  
*Constraint:* \( \text{nrhs} \geq 0 \).

5: a[\text{dim}] – const double  
*Input*  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{a} \) must be at least \( \max(1, \text{pda} \times \text{n}) \).  
*On entry:* details of the factorization of \( A \), as returned by nag_dsytrf (f07mdc).

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 \( \text{a} \).  
*Constraint:* \( \text{pda} \geq \max(1, \text{n}) \).

7: ipiv[\text{dim}] – const Integer  
*Input*  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{ipiv} \) must be at least \( \max(1, \text{n}) \).  
*On entry:* details of the interchanges and the block structure of \( D \), as returned by nag_dsytrf (f07mdc).

8: b[\text{dim}] – double  
*Input/Output*  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{b} \) must be at least \( \max(1, \text{pdb} \times \text{nrhs}) \) when order = Nag_ColMajor and at least \( \max(1, \text{pdb} \times \text{n}) \) when order = Nag_RowMajor.  
*On entry:* the \( n \) by \( r \) right-hand side matrix \( B \).  
*On exit:* 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 \( \text{b} \).  
*Constraints:*  
\[
\begin{align*}
\text{if order} & = \text{Nag_ColMajor}, \quad \text{pdb} \geq \max(1, \text{n}); \\
\text{if order} & = \text{Nag_RowMajor}, \quad \text{pdb} \geq \max(1, \text{nrhs}).
\end{align*}
\]

10: fail – NagError *  
*Output*  
The NAG error parameter (see the Essential Introduction).

**6 Error Indicators and Warnings**

**NE_INT**

On entry, \( \text{n} = \langle \text{value} \rangle \).  
Constraint: \( \text{n} \geq 0 \).  
On entry, \( \text{nrhs} = \langle \text{value} \rangle \).  
Constraint: \( \text{nrhs} \geq 0 \).  
On entry, \( \text{pda} = \langle \text{value} \rangle \).  
Constraint: \( \text{pda} > 0 \).  
On entry, \( \text{pdb} = \langle \text{value} \rangle \).  
Constraint: \( \text{pdb} > 0 \).
On entry, \( pda = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, n) \).

On entry, \( pdb = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \).
Constraint: \( pdb \geq \max(1, n) \).

On entry, \( pdb = \langle \text{value} \rangle \), \( nrhs = \langle \text{value} \rangle \).
Constraint: \( pdb \geq \max(1, nrhs) \).

Memory allocation failed.

On entry, parameter \( \langle \text{value} \rangle \) had an illegal value.

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.

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

\[
\text{if} \quad uplo = \text{Nag\_Upper}, \quad |E| \leq c(n)\epsilon P|U||D||U^T|P^T;
\]
\[
\text{if} \quad uplo = \text{Nag\_Lower}, \quad |E| \leq c(n)\epsilon P|L||D||L^T|P^T;
\]

\( c(n) \) 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 \text{nag\_dsyrfs (f07mhc)}, and an estimate for \( \kappa_\infty(A) \) \((= \kappa_1(A))\) can be obtained by calling \text{nag\_dsycon (f07mgc)}.

The total number of floating-point operations is approximately \( 2n^2r \).

This function may be followed by a call to \text{nag\_dsyrfs (f07mhc)} to refine the solution and return an error estimate.

The complex analogues of this function are \text{nag\_zhets (f07mcc)} for Hermitian matrices and \text{nag\_zsytrs (f07mcc)} for symmetric matrices.
9 Example

To solve the system of equations $AX = B$, where

\[
A = \begin{pmatrix}
2.07 & 3.87 & 4.20 & -1.15 \\
3.87 & -0.21 & 1.87 & 0.63 \\
4.20 & 1.87 & 1.15 & 2.06 \\
-1.15 & 0.63 & 2.06 & -1.81
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
-9.50 & 27.85 \\
-8.38 & 9.90 \\
-6.07 & 19.25 \\
-0.96 & 3.93
\end{pmatrix}
\]

Here $A$ is symmetric indefinite and must first be factorized by nag_dsytrf (f07mdc).

9.1 Program Text

/* nag_dsytrs (f07mec) Example Program. *
* Copyright 2001 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_OrderType order;
    /* Arrays */
    char uplo[2];
    Integer *ipiv=0;
    double *a=0, *b=0;
    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
    INIT_FAIL(fail);
    Vprintf("f07mec Example Program Results\n\n");
    /* Skip heading in data file */
    Vscanf("%*[\n]");
    Vscanf("%ld%ld%*[\n] ", &n, &nrhs);
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        pda = n;
        pdb = n;
    #else
        pda = n;
        pdb = nrhs;
    #endif
    /* Allocate memory */
    if ( ![NP3645/7]
/* Read A and B from data file */
Vscanf(" %ls %*[\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)
  {  
    for (i = 1; i <= n; ++i)
      {  
        for (j = i; j <= n; ++j)
          Vscanf("%lf", &A(i,j));
        Vscanf("%*[\n] ");
      }
  }
else
  {  
    for (i = 1; i <= n; ++i)
      {  
        for (j = 1; j <= i; ++j)
          Vscanf("%lf", &A(i,j));
        Vscanf("%*[\n] ");
      }
  }
for (i = 1; i <= n; ++i)
  {  
    for (j = 1; j <= nrhs; ++j)
      Vscanf("%lf", &B(i,j));
  }
Vscanf("%*[\n] ");

/* Factorize A */
f07mdc(order, uplo_enum, n, a, pda, ipiv, &fail);
if (fail.code != NE_NOERROR)
  {  
    Vprintf("Error from f07mdc.\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Compute solution */
f07mec(order, uplo_enum, n, nrhs, a, pda, ipiv, b, pdb, &fail);
if (fail.code != NE_NOERROR)
  {  
    Vprintf("Error from f07mec.\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Print solution */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb, "Solution(s)", 0, &fail);
if (fail.code != NE_NOERROR)
  {  
    Vprintf("Error from x04cac.\n", fail.message);
    exit_status = 1;
    goto END;
  }
END:
if (ipiv) NAG_FREE(ipiv);
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);
return exit_status;
}

9.2 Program Data

f07mec Example Program Data

\[\begin{array}{cccc}
5 & 2 & : Values of N and NRHS \\
\text{`L'} & : Value of UPLO \\
3.87 & -0.21 & 2.07 \\
4.20 & 1.87 & 1.20 \\
-1.15 & 0.63 & 2.06 & -1.81 \\
-9.50 & 27.85 \\
-8.38 & 9.90 \\
-6.07 & 19.25 \\
-0.96 & 3.93 \\
\end{array}\] 

: End of matrix A

\[\begin{array}{cccc}
2.07 & 3.87 & -0.21 & 4.20 \\
2.07 & 3.87 & -0.21 & 4.20 \\
2.07 & 3.87 & -0.21 & 4.20 \\
\end{array}\] 

: End of matrix B

9.3 Program Results

f07mec Example Program Results

Solution(s)

\[\begin{array}{cccc}
\text{Solution(s)} & 1 & 2 \\
1 & -4.0000 & 1.0000 \\
2 & -1.0000 & 4.0000 \\
3 & 2.0000 & 3.0000 \\
4 & 5.0000 & 2.0000 \\
\end{array}\]