NAG Library Function Document

nag_dspsv (f07pac)

1 Purpose

nag_dspsv (f07pac) computes the solution to a real system of linear equations

$$AX = B,$$

where $A$ is an $n$ by $n$ symmetric matrix stored in packed format and $X$ and $B$ are $n$ by $r$ matrices.

2 Specification

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

void nag_dspsv (Nag_OrderType order, Nag_UploType uplo, Integer n, 
    Integer nrhs, double ap[], Integer ipiv[], double b[], Integer pdb, 
    NagError *fail)
```

3 Description

nag_dspsv (f07pac) uses the diagonal pivoting method to factor $A$ as $A = UDU^T$ if $\text{uplo} = \text{Nag_Upper}$ or $A = LDL^T$ if $\text{uplo} = \text{Nag_Lower}$, where $U$ (or $L$) is a product of permutation and unit upper (lower) triangular matrices, $D$ 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


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 $A$ is stored.

   If uplo = Nag_Lower, the lower triangle of $A$ is stored.

   **Constraint:** uplo = Nag_Upper or Nag_Lower.
3: \( n \) – Integer

Input

On entry: \( n \), the number of linear equations, i.e., the order of the matrix \( A \).
Constraint: \( n \geq 0 \).

4: \( \text{nrhs} \) – Integer

Input

On entry: \( r \), the number of right-hand sides, i.e., the number of columns of the matrix \( B \).
Constraint: \( \text{nrhs} \geq 0 \).

5: \( \text{ap}[\text{dim}] \) – double

Input/Output

Note: the dimension, \( \text{dim} \), of the array \( \text{ap} \) must be at least \( \max(1, n \times (n+1)/2) \).

On entry: the \( n \) by \( n \) symmetric matrix \( A \), packed by rows or columns.

The storage of elements \( A_{ij} \) depends on the \( \text{order} \) and \( \text{uplo} \) arguments as follows:

if \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
\( A_{ij} \) is stored in \( \text{ap}[(j-1) \times j/2 + i - 1], \) for \( i \leq j \);
if \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
\( A_{ij} \) is stored in \( \text{ap}[(2n - j) \times (j-1)/2 + i - 1], \) for \( i \geq j \);
if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
\( A_{ij} \) is stored in \( \text{ap}[(2n - i) \times (i-1)/2 + j - 1], \) for \( i \leq j \);
if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
\( A_{ij} \) is stored in \( \text{ap}[(i-1) \times i/2 + j - 1], \) for \( i \geq j \).

On exit: the block diagonal matrix \( D \) and the multipliers used to obtain the factor \( U \) or \( L \) from the factorization \( A = U D U^T \) or \( A = L D L^T \) as computed by \text{nag}_\text{dsytrf} (f07pdc), stored as a packed triangular matrix in the same storage format as \( A \).

6: \( \text{ipiv}[\text{n}] \) – Integer

Output

On exit: details of the interchanges and the block structure of \( D \). More precisely,

if \( \text{ipiv}[i-1] = k > 0 \), \( d_{ii} \) is a 1 by 1 pivot block and the \( i \)th row and column of \( A \) were interchanged with the \( k \)th row and column;

if \( \text{uplo} = \text{Nag}_\text{Upper} \) and \( \text{ipiv}[i-2] = \text{ipiv}[i-1] = -l < 0 \), \( \begin{pmatrix} d_{i-1,i-1} & d_{i,i-1} \\ d_{i,i-1} & d_{ii} \end{pmatrix} \) is a 2 by 2 pivot block and the \((i-1)\)th row and column of \( A \) were interchanged with the \( l \)th row and column;

if \( \text{uplo} = \text{Nag}_\text{Lower} \) and \( \text{ipiv}[i-1] = \text{ipiv}[i] = -m < 0 \), \( \begin{pmatrix} d_{ii} & d_{i+1,i} \\ d_{i+1,i} & d_{i+1,i+1} \end{pmatrix} \) is a 2 by 2 pivot block and the \((i+1)\)th row and column of \( A \) were interchanged with the \( m \)th row and column.

7: \( \text{b}[\text{dim}] \) – 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 \( \text{order} = \text{Nag}_\text{ColMajor} \);
\( \max(1, n \times \text{pdb}) \) when \( \text{order} = \text{Nag}_\text{RowMajor} \).

The \((i,j)\)th element of the matrix \( B \) is stored in

\( \text{b}[(j-1) \times \text{pdb} + i - 1] \) when \( \text{order} = \text{Nag}_\text{ColMajor} \);
\( \text{b}[(i-1) \times \text{pdb} + j - 1] \) when \( \text{order} = \text{Nag}_\text{RowMajor} \).

On entry: the \( n \) by \( r \) right-hand side matrix \( B \).

On exit: if \( \text{fail.code} = \text{NE}_\text{NOERROR} \), 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* ≥ max(1, *n*);
     if *order* = Nag_RowMajor, *pdb* ≥ max(1, *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 ⟨*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*).

On entry, *pdb* = ⟨*value*⟩ and *nrhs* = ⟨*value*⟩.

Constraint: *pdb* ≥ max(1, *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_SINGULAR**

Element ⟨*value*⟩ of the diagonal is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so 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

\[
\frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A)\|E\|_1, \]

where \( \kappa(A) = \|A^{-1}\|_1\|A\|_1 \), the condition number of \( A \) with respect to the solution of the linear equations. See Section 4.4 of Anderson et al. (1999) and Chapter 11 of Higham (2002) for further details.

nag_dspsvx (f07pbc) is a comprehensive LAPACK driver that returns forward and backward error bounds and an estimate of the condition number. Alternatively, nag_real_sym_packed_lin_solve (f04bjc) solves \( AX = B \) and returns a forward error bound and condition estimate. nag_real_sym_packed_lin_solve (f04bjc) calls nag_dspsi (f07pac) to solve the equations.

8 Parallelism and Performance

nag_dspsi (f07pac) is not threaded by NAG in any implementation.

nag_dspsi (f07pac) 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 \( \frac{1}{3}n^3 + 2n^2r \), where \( r \) is the number of right-hand sides.

The complex analogues of nag_dspsi (f07pac) are nag_zhpsi (f07pnc) for Hermitian matrices, and nag_zpsi (f07qnc) for symmetric matrices.

10 Example

This example solves the equations

\[
Ax = b,
\]

where \( A \) is the symmetric matrix

\[
A = \begin{pmatrix}
-1.81 & 2.06 & 0.63 & -1.15 \\
2.06 & 1.15 & 1.87 & 4.20 \\
0.63 & 1.87 & -0.21 & 3.87 \\
-1.15 & 4.20 & 3.87 & 2.07
\end{pmatrix}

\text{and } b = \begin{pmatrix} 0.96 \\ 6.07 \\ 8.38 \\ 9.50 \end{pmatrix}.
\]

Details of the factorization of \( A \) are also output.
# 10.1 Program Text

/* nag_dspsv (f07pac) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 23, 2011.
 */

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

int main(void)
{

    /* Scalars */
    Integer exit_status = 0, i, j, n, nrhs, pdb;

    /* Arrays */
    double *ap = 0, *b = 0;
    Integer *ipiv = 0;
    char nag_enum_arg[40];

    /* 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

    INIT_FAIL(fail);

    printf("nag_dspsv (f07pac) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif

    #ifdef _WIN32
    scanf_s("%39s%*[\n]", &nag_enum_arg, _countof(nag_enum_arg));
    #else
    scanf("%39s%*[\n]", nag_enum_arg);
    #endif

    INIT_FAIL(fail);

    if (n < 0 || nrhs < 0)
    {
        printf("Invalid n or nrhs\n");
        exit_status = 1;
        goto END;
    }

    /* Convert NAG enum name to value */
    * Convert NAG enum member name to value
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

allocate memory

if (!(ap = NAG_ALLOC(n*(n+1)/2, double)) ||
    !(b = NAG_ALLOC(n*nrhs, double)) ||
    !(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 the 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_s("%lf", &A_UPPER(i, j));
else
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= i; ++j) scanf_s("%lf", &A_LOWER(i, j));
#endif

/* Read b from data file */
for (i = 1; i <= n; ++i)
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#endif

/* Solve the equations Ax = b for x using nag_dspsv (f07pac). */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dspsv (f07pac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
printf(" Solution\n");
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        printf("%10.4f", B(i, j));
    printf("\n");
}
END:
   NAG_FREE(ap);
   NAG_FREE(b);
   NAG_FREE(ipiv);

   return exit_status;
}
#undef A_UPPER
#undef A_LOWER
#undef B

10.2 Program Data
nag_dspsv (f07pac) Example Program Data
   4    : n, nrhs
   Nag_Lower : uplo
   -1.81
   2.06  1.15
   -1.15 4.20 3.87 2.07 : matrix A
   0.96 6.07 8.38 9.50 : vector b

10.3 Program Results
nag_dspsv (f07pac) Example Program Results
   Solution
   -5.0000
   -2.0000
   1.0000
   4.0000