NAG Library Function Document

nag_zspsv (f07qnc)

1 Purpose

nag_zspsv (f07qnc) computes the solution to a complex 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_zspsv (Nag_OrderType order, Nag_UploType uplo, Integer n,
                Integer nrhs, Complex ap[], Integer ipiv[], Complex b[], Integer pdb,
                NagError *fail)
```

3 Description

nag_zspsv (f07qnc) uses the diagonal pivoting method 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, \( D \) 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:  \texttt{order} – Nag_OrderType  

\textit{Input}

\textit{On entry:} the \texttt{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 \texttt{order = Nag_RowMajor}. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \texttt{order} = Nag_RowMajor or Nag_ColMajor.

2:  \texttt{uplo} – Nag_UploType  

\textit{Input}

\textit{On entry:} if \texttt{uplo} = Nag_Upper, the upper triangle of \( A \) is stored. If \texttt{uplo} = Nag_Lower, the lower triangle of \( A \) is stored.

\textit{Constraint:} \texttt{uplo} = Nag_Upper or Nag_Lower.
3: \ n \ – \ Integer \hspace{1cm} \textbf{Input}

\textit{On entry:} \( n \), the number of linear equations, i.e., the order of the matrix \( A \).

\textit{Constraint:} \( n \geq 0 \).

4: \ \textbf{nrhs} – Integer \hspace{1cm} \textbf{Input}

\textit{On entry:} \( r \), the number of right-hand sides, i.e., the number of columns of the matrix \( B \).

\textit{Constraint:} \( \textbf{nrhs} \geq 0 \).

5: \ \textbf{ap}[\text{dim}] – Complex \hspace{1cm} \textbf{Input/Output}

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

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

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

\begin{itemize}
  \item if \textbf{order} = Nag_ColMajor and \textbf{uplo} = Nag_Upper, \\
  \hspace{0.5cm} \( A_{ij} \) is stored in \textbf{ap}\((j - 1) \times j/2 + i - 1\), for \( i \leq j \);
  \item if \textbf{order} = Nag_ColMajor and \textbf{uplo} = Nag_Lower, \\
  \hspace{0.5cm} \( A_{ij} \) is stored in \textbf{ap}\((2n - j) \times (j - 1)/2 + i - 1\), for \( i \geq j \);
  \item if \textbf{order} = Nag_RowMajor and \textbf{uplo} = Nag_Upper, \\
  \hspace{0.5cm} \( A_{ij} \) is stored in \textbf{ap}\((2n - i) \times (i - 1)/2 + j - 1\), for \( i \leq j \);
  \item if \textbf{order} = Nag_RowMajor and \textbf{uplo} = Nag_Lower, \\
  \hspace{0.5cm} \( A_{ij} \) is stored in \textbf{ap}\((i - 1) \times i/2 + j - 1\), for \( i \geq j \).
\end{itemize}

\textit{On exit:} 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_zsptrf (f07qrc), stored as a packed triangular matrix in the same storage format as \( A \).

6: \ \textbf{ipiv}[\text{n}] – Integer \hspace{1cm} \textbf{Output}

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

\begin{itemize}
  \item if \textbf{ipiv}[\text{i} - 1] = \text{k} \geq 0, \( d_{ii} \) is a 1 by 1 pivot block and the \text{i}th row and column of \( A \) were interchanged with the \text{k}th row and column;
  \item if \textbf{uplo} = Nag_Upper and \textbf{ipiv}[\text{i} - 2] = \text{ipiv}[\text{i} - 1] = -\text{l} \leq 0, \begin{pmatrix} \frac{d_{i-1,i-1}}{d_{ii}} & \frac{d_{i,i-1}}{d_{ii}} \\ \frac{d_{i,i-1}}{d_{ii}} & \frac{d_{ii}}{d_{ii}} \end{pmatrix} \) is a 2 by 2 pivot block and the \( (\text{i} - 2)\)th row and column of \( A \) were interchanged with the \text{l}th row and column;
  \item if \textbf{uplo} = Nag_Lower and \textbf{ipiv}[\text{i} - 1] = \text{ipiv}[\text{i}] = -\text{m} \leq 0, \begin{pmatrix} \frac{d_{ii}}{d_{i+1,i}} & \frac{d_{i+1,i}}{d_{i+1,i}} \\ \frac{d_{i+1,i}}{d_{i+1,i}} & \frac{d_{i+1,i}}{d_{i+1,i}} \end{pmatrix} \) is a 2 by 2 pivot block and the \( (\text{i} + 1)\)th row and column of \( A \) were interchanged with the \text{m}th row and column.
\end{itemize}

7: \ \textbf{b}[\text{dim}] – Complex \hspace{1cm} \textbf{Input/Output}

\textit{Note:} the dimension, \textit{dim}, of the array \textbf{b} must be at least \( \max(1, \text{pdb} \times \text{nrhs}) \) when \textbf{order} = Nag_ColMajor; \( \max(1, n \times \text{pdb}) \) when \textbf{order} = Nag_RowMajor.

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

\begin{itemize}
  \item \textbf{b}\((j - 1) \times \text{pdb} + i - 1\) when \textbf{order} = Nag_ColMajor;
  \item \textbf{b}\((i - 1) \times \text{pdb} + j - 1\) when \textbf{order} = Nag_RowMajor.
\end{itemize}

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

\textit{On exit:} if \textbf{fail.code} = NE_NOERROR, the \( n \) by \( r \) solution matrix \( X \).
8: \( \text{pdb} \) – Integer

*Input*

*On entry:* the stride separating row or column elements (depending on the value of \text{order}) in the array \( b \).

*Constraints:*

if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdb} \geq \max(1, n) \);

if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdb} \geq \max(1, \text{nrhs}) \).

9: \( \text{fail} \) – 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 \( \langle \text{value} \rangle \) had an illegal value.

**NE\_INT**

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( \text{nrhs} = \langle \text{value} \rangle \).

Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).

Constraint: \( \text{pdb} > 0 \).

**NE\_INT\_2**

On entry, \( \text{pdb} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

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

On entry, \( \text{pdb} = \langle \text{value} \rangle \) and \( \text{nrhs} = \langle \text{value} \rangle \).

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

**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 \( \langle \text{value} \rangle \) 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

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

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_zspsvx (f07qpc) is a comprehensive LAPACK driver that returns forward and backward error bounds and an estimate of the condition number. Alternatively, nag_complex_sym_packed_lin_solve (f04djc) solves \( AX = B \) and returns a forward error bound and condition estimate. nag_complex_sym_packed_lin_solve (f04djc) calls nag_zspsv (f07qnc) to solve the equations.

8 Parallelism and Performance

nag_zspsv (f07qnc) is not threaded by NAG in any implementation.

nag_zspsv (f07qnc) 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 \( \frac{4}{3}n^3 + 8n^2r \), where \( r \) is the number of right-hand sides.

The real analogue of this function is nag_dspsv (f07pac). The complex Hermitian analogue of this function is nag_zhpsv (f07pnc).

10 Example

This example solves the equations

\[
Ax = b,
\]

where \( A \) is the complex symmetric 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

\[
b = \begin{pmatrix}
-6.43 + 19.24i \\
-0.49 - 1.47i \\
-48.18 + 66.00i \\
-55.64 + 41.22i
\end{pmatrix}.
\]

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

/* nag_zspsv (f07qnc) 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 */
    Complex *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_zspsv (f07qnc) Example Program Results\n\n");

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

    /* nag_enum_name_to_value (x04nac).
    * Converts NAG enum member name to value
    * Mark 25 f07qnc.5
    */
    #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)
    {
        printf("Invalid n or nrhs\n");
        exit_status = 1;
        goto END;
    }

    /* Convert NAG enum member name to value */
    *
    * f07qnc.5
    */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

/* Allocate memory */
if (! (ap = NAG_ALLOC(n*(n+1)/2, Complex)) ||
    !(b = NAG_ALLOC(n*nrhs, Complex)) ||
    !(ipiv = NAG_ALLOC(n, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
#endif 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)
            #ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
            #else
                scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
            #endif
else if (uplo == Nag_Lower)
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= i; ++j)
            #ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
            #else
                scanf(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
            #endif
#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
#endif

/* Solve the equations Ax = b for x using * nag_zspsv (f07qnc). */
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zspsv (f07qnc).\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(" (%7.4f, %7.4f)%s", B(i, j).re, B(i, j).im, j%4 == 0?"\n":"" );
print("\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_zspsv (f07qnc) Example Program Data

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_Lower</td>
<td>uplo</td>
</tr>
<tr>
<td>( -0.56, 0.12)</td>
<td></td>
</tr>
<tr>
<td>( -1.54, -2.86)</td>
<td>( -2.83, -0.03)</td>
</tr>
<tr>
<td>( 5.32, -1.59)</td>
<td>( -3.52, 0.58)</td>
</tr>
<tr>
<td>( 3.80, 0.92)</td>
<td>( -7.86, -2.96)</td>
</tr>
<tr>
<td>( -6.43, 19.24)</td>
<td>( -0.49, -1.47)</td>
</tr>
</tbody>
</table>

: n, nrhs

( -0.56, 0.12)
( -1.54, -2.86) ( -2.83, -0.03)
( 5.32, -1.59) ( -3.52, 0.58) ( 8.86, 1.81)
( 3.80, 0.92) ( -7.86, -2.96) ( 5.14, -0.03) ( -0.39, -0.71)
( -6.43, 19.24) ( -0.49, -1.47) ( -48.18, 66.00) ( -55.64, 41.22) : matrix A

10.3 Program Results

nag_zspsv (f07qnc) Example Program Results

Solution

( -4.0000, 3.0000)
( 3.0000, -2.0000)
( -2.0000, 5.0000)
( 1.0000, -1.0000)