NAG Library Function Document

nag_zspsvx (f07qpc)

1 Purpose
nag_zspsvx uses the diagonal pivoting factorization

\[ A = UDU^T \quad \text{or} \quad A = LDL^T \]

to compute 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. Error bounds on the solution and a condition estimate are also provided.

2 Specification

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

void nag_zspsvx (Nag_OrderType order, Nag_FactoredFormType fact, 
                 Nag_UploType uplo, Integer n, Integer nrhs, const Complex ap[], 
                 Complex afp[], Integer ipiv[], const Complex b[], Integer pdb, 
                 Complex x[], Integer pdx, double *rcond, double ferr[], double berr[], 
                 NagError *fail)
```

3 Description

nag_zspsvx performs the following steps:

1. If `fact = Nag_NotFactored`, the diagonal pivoting method is used 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 and \( D \) is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks.

2. If some \( d_{ii} = 0 \), so that \( D \) is exactly singular, then the function returns with `fail.errnum = i` and `fail.code = NE_SINGULAR`. Otherwise, the factored form of \( A \) is used to estimate the condition number of the matrix \( A \). If the reciprocal of the condition number is less than `machine precision`, `fail.code = NE_SINGULAR_WP` is returned as a warning, but the function still goes on to solve for \( X \) and compute error bounds as described below.

3. The system of equations is solved for \( X \) using the factored form of \( A \).

4. Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for it.

4 References


5 Arguments

1: order – Nag_OrderType

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: fact – Nag_FactoredFormType

On entry: specifies whether or not the factorized form of the matrix A has been supplied.

fact = Nag_Factored
afp and ipiv contain the factorized form of the matrix A. afp and ipiv will not be modified.

fact = Nag_NotFactored
The matrix A will be copied to afp and factorized.

Constraint: fact = Nag_Factored or Nag_NotFactored.

3: uplo – Nag_UploType

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.

4: n – Integer

On entry: n, the number of linear equations, i.e., the order of the matrix A.

Constraint: n ≥ 0.

5: nrhs – Integer

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

Constraint: nrhs ≥ 0.

6: ap[dim] – const Complex

Note: the dimension, dim, of the array ap must be at least max(1,n × (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 order and uplo arguments as follows:

if order = Nag_ColMajor and uplo = Nag_Upper,
A_{ij} is stored in ap[(j - 1) × j/2 + i - 1], for i ≤ j;
if order = Nag_ColMajor and uplo = Nag_Lower,
A_{ij} is stored in ap[(2n - j) × (j - 1)/2 + i - 1], for i ≥ j;
if order = Nag_RowMajor and uplo = Nag_Upper,
A_{ij} is stored in ap[(2n - i) × (i - 1)/2 + j - 1], for i ≤ j;
if order = Nag_RowMajor and uplo = Nag_Lower,
A_{ij} is stored in ap[(i - 1) × i/2 + j - 1], for i ≥ j.

7: afp[dim] – Complex

Note: the dimension, dim, of the array afp must be at least max(1,n × (n + 1)/2).

On entry: if fact = Nag_Factored, afp contains the block diagonal matrix D and the multipliers
used to obtain the factor U or L from the factorization A = U D^T or A = L D L^T as computed by
nag_zsptrf (f07qrc), stored as a packed triangular matrix in the same storage format as A.
On exit: if fact = Nag_NotFactored, afp contains 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 \( A \).

8: \( \text{ipiv}[n] \) – Integer

\textit{Input/Output}

\textit{On entry:} if fact = Nag_Factored, ipiv contains details of the interchanges and the block structure of \( D \), as determined by nag_zsptrf (f07qrc).

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 uplo = Nag_Upper and \( \text{ipiv}[i - 2] = \text{ipiv}[i - 1] = -l < 0 \), \( \begin{pmatrix} d_{i-l+1,i-l} & d_{i-l,i-l} \\ d_{i-l,i-l} & d_{ii} \end{pmatrix} \) 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 uplo = Nag_Lower and \( \text{ipiv}[i - 1] = \text{ipiv}[i] = -m < 0 \), \( \begin{pmatrix} d_{ii} & d_{i+1,i} \\ d_{i+1,i} & d_{ii+1} \end{pmatrix} \) 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.

\textit{On exit:} if fact = Nag_NotFactored, ipiv contains details of the interchanges and the block structure of \( D \), as determined by nag_zsptrf (f07qrc), as described above.

9: \( \text{b}[\text{dim}] \) – const Complex

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{b} \) must be at least

\[
\max(1, \text{pdb} \times \text{nrhs}) \text{ when order = Nag_ColMajor}; \\
\max(1, n \times \text{pdb}) \text{ when order = Nag_RowMajor}.
\]

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

\[
\text{b}[(j - 1) \times \text{pdb} + i - 1] \text{ when order = Nag_ColMajor}; \\
\text{b}[(i - 1) \times \text{pdb} + j - 1] \text{ when order = Nag_RowMajor}.
\]

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

10: \( \text{pdb} \) – Integer

\textit{Input}

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

\textit{Constraints:}

\[
\text{if order = Nag_ColMajor, pdb} \geq \max(1, n); \\
\text{if order = Nag_RowMajor, pdb} \geq \max(1, \text{nrhs}).
\]

11: \( \text{x}[\text{dim}] \) – Complex

\textit{Output}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{x} \) must be at least

\[
\max(1, \text{pdx} \times \text{nrhs}) \text{ when order = Nag_ColMajor}; \\
\max(1, n \times \text{pdx}) \text{ when order = Nag_RowMajor}.
\]

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

\[
\text{x}[(j - 1) \times \text{pdx} + i - 1] \text{ when order = Nag_ColMajor}; \\
\text{x}[(i - 1) \times \text{pdx} + j - 1] \text{ when order = Nag_RowMajor}.
\]

\textit{On exit:} if fail.code = NE_NOERROR or NE_SINGULAR_WP, the \( n \) by \( r \) solution matrix \( X \).

12: \( \text{pdx} \) – Integer

\textit{Input}

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

if order = Nag_ColMajor, \( \text{pdx} \geq \max(1, n) \);
if order = Nag_RowMajor, \( \text{pdx} \geq \max(1, \text{nrhs}) \).

13: \textbf{rcond} – double *

*Output*

On exit: the estimate of the reciprocal condition number of the matrix \( A \). If \text{rcond} = 0.0, the matrix may be exactly singular. This condition is indicated by \text{fail.code} = \text{NE_SINGULAR}. Otherwise, if \text{rcond} is less than the \text{machine precision}, the matrix is singular to working precision. This condition is indicated by \text{fail.code} = \text{NE_SINGULAR_WP}.

14: \textbf{ferr[nrhs]} – double

*Output*

On exit: if \text{fail.code} = \text{NE_NOERROR} or \text{NE_SINGULAR_WP}, an estimate of the forward error bound for each computed solution vector, such that \( \| \hat{x}_j - x_j \|_\infty / \| x_j \|_\infty \leq \text{ferr}[j - 1] \) where \( \hat{x}_j \) is the \( j \)th column of the computed solution returned in the array \( x \) and \( x_j \) is the corresponding column of the exact solution \( X \). The estimate is as reliable as the estimate for \text{rcond}, and is almost always a slight overestimate of the true error.

15: \textbf{berr[nrhs]} – double

*Output*

On exit: if \text{fail.code} = \text{NE_NOERROR} or \text{NE_SINGULAR_WP}, an estimate of the component-wise relative backward error of each computed solution vector \( \hat{x}_j \) (i.e., the smallest relative change in any element of \( A \) or \( B \) that makes \( \hat{x}_j \) an exact solution).

16: \textbf{fail} – NagError *

*Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

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

\textbf{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, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).
On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

\textbf{NE_INT_2}

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

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

**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 factor $D$ is exactly singular, so the solution and error bounds could not be computed. $\text{rcond} = 0.0$ is returned.

**NE_SINGULAR_WP**
$D$ is nonsingular, but $\text{rcond}$ is less than *machine precision*, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of $\text{rcond}$ would suggest.

7 **Accuracy**
For each right-hand side vector $b$, the computed solution $\hat{x}$ is the exact solution of a perturbed system of equations $(A + E)\hat{x} = b$, where $\|E\|_1 = O(\epsilon)\|A\|_1$, where $\epsilon$ is the *machine precision*. See Chapter 11 of Higham (2002) for further details.

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

$$\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq w_c \\text{cond}(A, \hat{x}, b)$$

where $\text{cond}(A, \hat{x}, b) = \frac{\|A^{-1}\|_1(\|A\|_1 + \|b\|_1)}{\|\hat{x}\|_\infty}$ and $\|\hat{x}\|_\infty \leq \text{cond}(A) = \frac{\|A^{-1}\|\|A\|_\infty}{\|\hat{x}\|_\infty} \leq \kappa_\infty(A)$. If $\hat{x}$ is the $j$th column of $X$, then $w_c$ is returned in $\text{berr}[j - 1]$ and a bound on $\|x - \hat{x}\|_\infty / \|\hat{x}\|_\infty$ is returned in $\text{ferr}[j - 1]$. See Section 4.4 of Anderson et al. (1999) for further details.

8 **Parallelism and Performance**
\text{nag_zspsvx} (f07qpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\text{nag_zspsvx} (f07qpc) 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 factorization of $A$ requires approximately $\frac{4}{3}n^3$ floating-point operations.

For each right-hand side, computation of the backward error involves a minimum of $16n^2$ floating-point operations. Each step of iterative refinement involves an additional $24n^2$ operations. At most five steps of iterative refinement are performed, but usually only one or two steps are required. Estimating the forward error involves solving a number of systems of equations of the form $Ax = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $8n^2$ operations.

The real analogue of this function is `nag_dspsvx (f07pbc)`. The complex Hermitian analogue of this function is `nag_zhpsvx (f07ppc)`.

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

$$B = \begin{pmatrix}
-6.43 + 19.24i & -4.59 - 35.53i \\
-0.49 - 1.47i & 6.95 + 20.49i \\
-48.18 + 66.00i & -12.08 - 27.02i \\
-55.64 + 41.22i & -19.09 - 35.97i
\end{pmatrix}.$$

Error estimates for the solutions, and an estimate of the reciprocal of the condition number of the matrix $A$ are also output.

10.1 Program Text

/* nag_zspsvx (f07qpc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group. *
 * Mark 23, 2011. */
#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf07.h>

int main(void)
{
    /* Scalars */
    double rcond;
    Integer exit_status = 0, i, j, n, nrhs, pdb, pdx;

    /* Arrays */
    Complex *afp = 0, *ap = 0, *b = 0, *x = 0;
    double *berr = 0, *ferr = 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_zspsvx (f07qpc) Example Program Results\n\n");

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

#ifndef _WIN32
scanf("%"NAG_FMT "%"NAG_FMT "%*[\n]", &n, &nrhs);
#else
scanf("%"NAG_FMT "%"NAG_FMT "%*[\n]", &n, &nrhs);
#endif
if (n < 0 || nrhs < 0)
{
    printf("Invalid n or nrhs\n");
    exit_status = 1;
    goto END;
}
#endif _WIN32

#ifndef _WIN32
scanf(" %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);

/* Allocate memory */
if (!afp = NAG_ALLOC(n*(n+1)/2, Complex)) ||
!ap = NAG_ALLOC(n*(n+1)/2, Complex)) ||
!b = NAG_ALLOC(n * nrhs, Complex)) ||
!x = NAG_ALLOC(n * nrhs, Complex)) ||
!berr = NAG_ALLOC(nrhs, double)) ||
!ferr = NAG_ALLOC(nrhs, double)) ||
!ipiv = NAG_ALLOC(n, Integer))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
#endif NAG_COLUMN_MAJOR

pdb = n;
pdx = n;
#else
pdb = nrhs;
pdx = 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
                #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 , %lf )", &B(i, j).re, &B(i, j).im);
                #else
                    scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
                #endif
                #ifdef _WIN32
                    scanf_s("%*[\n]" );
                #else
                    scanf("%*[\n]" );
                #endif
        /* Solve the equations AX = B for X using nag_zspsvx (f07qpc). */
        nag_zspsvx(order, Nag_NotFactored, uplo, n, nrhs, ap, afp, ipiv, b,
                    pdb, x, pdx, &rcond, ferr, berr, &fail);
        if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
            { printf("Error from nag_zspsvx (f07qpc).
            %s
            ", fail.message);
                exit_status = 1;
                goto END;
            }
        /* Print solution using nag_gen_complx_mat_print_comp (x04dbc). */
        printf("\n%11.1e%s", berr[j], j%7 == 6?"\n":" ");
        printf("%11.1e%s", ferr[j], j%7 == 6?"\n":" ");
        printf("%11.1e", rcond);
        if (fail.code == NE_SINGULAR)
            { printf("Error from nag_zspsvx (f07qpc).
            %s
            ", fail.message);
                exit_status = 1;
                goto END;
            }
NAG_FREE(afp);
NAG_FREE(ap);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(ipiv);
return exit_status;
}
#undef A_UPPER
#undef A_LOWER
#undef B

10.2 Program Data

nag_zspsvx (f07qpc) Example Program Data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nag_Upper</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( -0.56, 0.12)</td>
<td>( -1.54, -2.86)</td>
<td>( 5.32, -1.59)</td>
<td>( 3.80, 0.92)</td>
</tr>
<tr>
<td>( -2.83, -0.03)</td>
<td>( -1.52, 0.58)</td>
<td>( -7.86, -2.96)</td>
<td></td>
</tr>
<tr>
<td>( 8.86, 1.81)</td>
<td>( 5.14, -0.64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( -0.39, -0.71)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

: matrix A

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3 Program Results</td>
<td></td>
</tr>
</tbody>
</table>

nag_zspsvx (f07qpc) Example Program Results

Solution(s)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(-4.0000, 3.0000)</td>
<td>(-1.0000, 1.0000)</td>
</tr>
<tr>
<td>(3.0000, -2.0000)</td>
<td>(3.0000, 2.0000)</td>
</tr>
<tr>
<td>(-2.0000, 5.0000)</td>
<td>(1.0000, -3.0000)</td>
</tr>
<tr>
<td>(1.0000, -1.0000)</td>
<td>(-2.0000, -1.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

8.1e-17 3.0e-17

Estimated forward error bounds (machine-dependent)

1.2e-14 1.2e-14

Estimate of reciprocal condition number

4.9e-02