1 Purpose

nag_zptsvx (f07jpc) uses the factorization

\[ A = LDL^H \]

to compute the solution to a complex system of linear equations

\[ AX = B, \]

where \( A \) is an \( n \) by \( n \) Hermitian positive definite tridiagonal matrix 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_zptsvx (Nag_OrderType order, Nag_FactoredFormType fact, Integer n,
                Integer nrhs, const double d[], const Complex e[], double df[],
                Complex ef[], const Complex b[], Integer pdb, Complex x[], Integer pdx,
                double *rcond, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zptsvx (f07jpc) performs the following steps:

1. If \( \text{fact} = \text{Nag_NotFactored} \), the matrix \( A \) is factorized as \( A = LDL^H \), where \( L \) is a unit lower bidiagonal matrix and \( D \) is diagonal. The factorization can also be regarded as having the form \( A = U^HDU \).
2. If the leading \( i \) by \( i \) principal minor is not positive definite, then the function returns with \( \text{fail.errnum} = i \) and \( \text{fail.code} = \text{NE_MAT_NOT_POS_DEF} \). 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 \( \text{machine precision} \), \( \text{fail.code} = \text{NE_SINGULAR_WP} \) 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: \textbf{order} – Nag_OrderType \hspace{2cm} \textit{Input}

\textit{On entry:} the \textbf{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 \textbf{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:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{fact} – Nag_FactoredFormType \hspace{2cm} \textit{Input}

\textit{On entry:} specifies whether or not the factorized form of the matrix \( A \) has been supplied.

\textbf{fact} = Nag_Factored
\hspace{1cm} \textbf{df} and \textbf{ef} contain the factorized form of the matrix \( A \). \textbf{df} and \textbf{ef} will not be modified.

\textbf{fact} = Nag_NotFactored
\hspace{1cm} The matrix \( A \) will be copied to \textbf{df} and \textbf{ef} and factorized.

\textit{Constraint:} \textbf{fact} = Nag_Factored or Nag_NotFactored.

3: \textbf{n} – Integer \hspace{2cm} \textit{Input}

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

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

4: \textbf{nrhs} – Integer \hspace{2cm} \textit{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{d}[\textit{dim}] – const double \hspace{2cm} \textit{Input}

\textit{Note:} the dimension, \( \textit{dim} \), of the array \textbf{d} must be at least \( \max(1, n) \).

\textit{On entry:} the \( n \) diagonal elements of the tridiagonal matrix \( A \).

6: \textbf{e}[\textit{dim}] – const Complex \hspace{2cm} \textit{Input}

\textit{Note:} the dimension, \( \textit{dim} \), of the array \textbf{e} must be at least \( \max(1, n - 1) \).

\textit{On entry:} the \( (n - 1) \) subdiagonal elements of the tridiagonal matrix \( A \).

7: \textbf{df}[\textit{dim}] – double \hspace{2cm} \textit{Input/Output}

\textit{Note:} the dimension, \( \textit{dim} \), of the array \textbf{df} must be at least \( \max(1, n) \).

\textit{On entry:} if \textbf{fact} = Nag_Factored, \textbf{df} must contain the \( n \) diagonal elements of the diagonal matrix \( D \) from the \( LDL^H \) factorization of \( A \).

\textit{On exit:} if \textbf{fact} = Nag_NotFactored, \textbf{df} contains the \( n \) diagonal elements of the diagonal matrix \( D \) from the \( LDL^H \) factorization of \( A \).

8: \textbf{ef}[\textit{dim}] – Complex \hspace{2cm} \textit{Input/Output}

\textit{Note:} the dimension, \( \textit{dim} \), of the array \textbf{ef} must be at least \( \max(1, n - 1) \).

\textit{On entry:} if \textbf{fact} = Nag_Factored, \textbf{ef} must contain the \( (n - 1) \) subdiagonal elements of the unit bidiagonal factor \( L \) from the \( LDL^H \) factorization of \( A \).

\textit{On exit:} if \textbf{fact} = Nag_NotFactored, \textbf{ef} contains the \( (n - 1) \) subdiagonal elements of the unit bidiagonal factor \( L \) from the \( LDL^H \) factorization of \( A \).
9: \( \mathbf{b}[\text{dim}] \) – const Complex

**Note:** the dimension, \( \text{dim} \), of the array \( \mathbf{b} \) must be at least
\[
\max(1, \text{pdb} \times \text{nrhs}) \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor};
\]
\[
\max(1, \text{n} \times \text{pdb}) \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

The \((i,j)\)th element of the matrix \( \mathbf{B} \) is stored in
\[
\mathbf{b}[(j - 1) \times \text{pdb} + i - 1] \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor};
\]
\[
\mathbf{b}[(i - 1) \times \text{pdb} + j - 1] \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

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

10: \( \text{pdb} \) – Integer

**Input**

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

**Constraints:**
\[
\begin{align*}
\text{if} \quad \text{order} = \text{Nag\_ColMajor}, & \quad \text{pdb} \geq \max(1, \text{n}); \\
\text{if} \quad \text{order} = \text{Nag\_RowMajor}, & \quad \text{pdb} \geq \max(1, \text{nrhs}).
\end{align*}
\]

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

**Output**

**Note:** the dimension, \( \text{dim} \), of the array \( \mathbf{x} \) must be at least
\[
\max(1, \text{pdx} \times \text{nrhs}) \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor};
\]
\[
\max(1, \text{n} \times \text{pdx}) \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

The \((i,j)\)th element of the matrix \( \mathbf{X} \) is stored in
\[
\mathbf{x}[(j - 1) \times \text{pdx} + i - 1] \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor};
\]
\[
\mathbf{x}[(i - 1) \times \text{pdx} + j - 1] \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

**On exit:** if \( \text{fail\_code} = \text{NE\_NOERROR} \) or \( \text{NE\_SINGULAR\_WP} \), the \( n \) by \( r \) solution matrix \( \mathbf{X} \).

12: \( \text{pdx} \) – Integer

**Input**

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

**Constraints:**
\[
\begin{align*}
\text{if} \quad \text{order} = \text{Nag\_ColMajor}, & \quad \text{pdx} \geq \max(1, \text{n}); \\
\text{if} \quad \text{order} = \text{Nag\_RowMajor}, & \quad \text{pdx} \geq \max(1, \text{nrhs}).
\end{align*}
\]

13: \( \text{rcond} \) – double *

**Output**

**On exit:** the reciprocal condition number of the matrix \( \mathbf{A} \). If \( \text{rcond} \) is less than the machine precision (in particular, if \( \text{rcond} = 0.0 \)), the matrix is singular to working precision. This condition is indicated by a return code of \( \text{fail\_code} = \text{NE\_SINGULAR\_WP} \).

14: \( \text{ferr}[\text{nrhs}] \) – double

**Output**

**On exit:** the forward error bound for each solution vector \( \hat{x}_j \) (the \( j \)th column of the solution matrix \( \mathbf{X} \)). If \( x_j \) is the true solution corresponding to \( \hat{x}_j \), \( \text{ferr}[j - 1] \) is an estimated upper bound for the magnitude of the largest element in \((\hat{x}_j - x_j)\) divided by the magnitude of the largest element in \( \hat{x}_j \).

15: \( \text{berr}[\text{nrhs}] \) – double

**Output**

**On exit:** the component-wise relative backward error of each solution vector \( \hat{x}_j \) (i.e., the smallest relative change in any element of \( \mathbf{A} \) or \( \mathbf{B} \) that makes \( \hat{x}_j \) an exact solution).
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, \( n = \langle \text{value} \rangle \).
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: \( \text{pdb} > 0 \).
On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

**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: \( \text{pdb} \geq \max(1, \text{nrhs}) \).
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_MAT_NOT_POS_DEF**
The leading minor of order \( \langle \text{value} \rangle \) of \( A \) is not positive definite, so the factorization could not be completed, and the solution has not been computed. \( \text{rcond} = 0.0 \) is returned.

**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_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

$$|E| \leq c(n)\epsilon |R^T||R|,$$

where $R = D^{1/2}U$.

c(n) is a modest linear function of $n$, and $\epsilon$ is the machine precision. See Section 10.1 of Higham (2002) for further details.

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

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

where $\text{cond}(A, \hat{x}, b) = \frac{||A^{-1}||}{||A||}$ and $\text{cond}(A) = \frac{||A^{-1}||}{||A||}$, $\kappa_\infty(A)$. 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

nag_zptsvx (f07jpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zptsvx (f07jpc) 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 number of floating-point operations required for the factorization, and for the estimation of the condition number of $A$ is proportional to $n$. The number of floating-point operations required for the solution of the equations, and for the estimation of the forward and backward error is proportional to $nr$, where $r$ is the number of right-hand sides.

The condition estimation is based upon Equation (15.11) of Higham (2002). For further details of the error estimation, see Section 4.4 of Anderson et al. (1999).

The real analogue of this function is nag_dptsvx (f07jbc).

10 Example

This example solves the equations

$$AX = B,$$

where $A$ is the Hermitian positive definite tridiagonal matrix

$$A = \begin{pmatrix}
16.0 & 16.0 - 16.0i & 0 & 0 \\
16.0 + 16.0i & 41.0 & 18.0 + 9.0i & 0 \\
0 & 18.0 - 9.0i & 46.0 & 1.0 + 4.0i \\
0 & 0 & 1.0 - 4.0i & 21.0
\end{pmatrix}$$
and

\[ B = \begin{pmatrix} 64.0 + 16.0i & -16.0 - 32.0i \\ 93.0 + 62.0i & 61.0 - 66.0i \\ 78.0 - 80.0i & 71.0 - 74.0i \\ 14.0 - 27.0i & 35.0 + 15.0i \end{pmatrix}. \]

Error estimates for the solutions and an estimate of the reciprocal of the condition number of \( A \) are also output.

### 10.1 Program Text

```c
/* nag_zptsvx (f07jpc) 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>

int main(void)
{
    /* Scalars */
    double rcond;
    Integer exit_status = 0, i, j, n, nrhs, pdb, pdx;
    /* Arrays */
    Complex *b = 0, *e = 0, *ef = 0, *x = 0;
    double *berr = 0, *d = 0, *df = 0, *ferr = 0;
    /* Nag Types */
    NagError fail;
    Nag_OrderType order;

    INIT_FAIL(fail);

    printf("nag_zptsvx (f07jpc) Example Program Results\n\n");
    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*\n");
    #else
        scanf("%*\n");
    #endif
    #ifdef _WIN32
        scanf("%"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;
    }
```

---

**f07jpc.6**  
*Mark 25*
/* Allocate memory */
if (!(b = NAG_ALLOC(n * nrhs, Complex)) ||
    !(x = NAG_ALLOC(n * nrhs, Complex)) ||
    !(e = NAG_ALLOC(n-1, Complex)) ||
    !(ef = NAG_ALLOC(n-1, Complex)) ||
    !(d = NAG_ALLOC(n, double)) ||
    !(df = NAG_ALLOC(n, double)) ||
    !(berr = NAG_ALLOC(nrhs, double)) ||
    !(ferr = NAG_ALLOC(nrhs, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
#endif

nag_zptsvx(order, Nag_NotFactored, n, nrhs, d, e, df, ef, b, pdb, x, pdx,
         &rcond, ferr, berr, &fail);
if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
{
    printf("Error from nag_zptsvx (f07jpc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
#endif

/* Print solution, error bounds and condition number using
 * nag_gen_complx_mat_print_comp (x04dbc).
 */

Mark 25

f07jpc.7
fflush(stdout);

nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx, Nag_BracketForm, "%7.4f",
"Solution(s)", Nag_IntegerLabels, 0,
Nag_IntegerLabels, 0, 80, 0, 0, &fail);

if (fail.code != NE_NOERROR)
{
  printf("Error from nag_gen_complex_mat_print_comp (x04dbc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

printf("\nBackward errors (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", berr[j], j%7 == 6 ? "\n" : "");

printf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", ferr[j], j%7 == 6 ? "\n" : "");

if (fail.code == NE_SINGULAR)
{
  printf("Error from nag_zptsvx (f07jpc).\n%s\n", fail.message);
  exit_status = 1;
}

END:
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(e);
NAG_FREE(ef);
NAG_FREE(d);
NAG_FREE(df);
NAG_FREE(berr);
NAG_FREE(ferr);

return exit_status;
}

#define B

10.2 Program Data

nag_zptsvx (f07jpc) Example Program Data

4 2 : n, nrhs
16.0 41.0 46.0 21.0 : diagonal d
( 16.0, 16.0) ( 18.0, -9.0) ( 1.0, -4.0) : sub-diagonal e
( 64.0, 16.0) (-16.0,-32.0)
( 93.0, 62.0) ( 61.0,-66.0)
( 78.0,-80.0) ( 71.0,-74.0)
( 14.0,-27.0) ( 35.0, 15.0) : matrix b

10.3 Program Results

nag_zptsvx (f07jpc) Example Program Results

Solution(s)

1 2
1 ( 2.0000, 1.0000) (-3.0000,-2.0000)
2 ( 1.0000, 1.0000) ( 1.0000, 1.0000)
3 ( 1.0000,-2.0000) ( 1.0000,-2.0000)
4 ( 1.0000,-1.0000) ( 2.0000, 1.0000)

Backward errors (machine-dependent)
0.0e+00 0.0e+00

Estimated forward error bounds (machine-dependent)
9.0e-12 6.1e-12

Estimate of reciprocal condition number
1.1e-04