

# NAG Library Function Document

## nag\_zppsvx (f07gpc)

### 1 Purpose

nag\_zppsvx (f07gpc) uses the Cholesky factorization

$$A = U^H U \quad \text{or} \quad A = LL^H$$

to compute the solution to a complex system of linear equations

$$AX = B,$$

where  $A$  is an  $n$  by  $n$  Hermitian positive definite 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

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

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

### 3 Description

nag\_zppsvx (f07gpc) performs the following steps:

1. If **fact** = Nag\_EquilibrateAndFactor, real diagonal scaling factors,  $D_S$ , are computed to equilibrate the system:

$$(D_S A D_S)(D_S^{-1} X) = D_S B.$$

Whether or not the system will be equilibrated depends on the scaling of the matrix  $A$ , but if equilibration is used,  $A$  is overwritten by  $D_S A D_S$  and  $B$  by  $D_S B$ .

2. If **fact** = Nag\_NotFactored or Nag\_EquilibrateAndFactor, the Cholesky decomposition is used to factor the matrix  $A$  (after equilibration if **fact** = Nag\_EquilibrateAndFactor) as  $A = U^H U$  if **uplo** = Nag\_Upper or  $A = LL^H$  if **uplo** = Nag\_Lower, where  $U$  is an upper triangular matrix and  $L$  is a lower triangular matrix.
3. If the leading  $i$  by  $i$  principal minor of  $A$  is not positive definite, then the function returns with **fail.errnum** =  $i$  and **fail.code** = 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 *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.
4. The system of equations is solved for  $X$  using the factored form of  $A$ .
5. Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for it.
6. If equilibration was used, the matrix  $X$  is premultiplied by  $D_S$  so that it solves the original system before equilibration.

## 4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia <http://www.netlib.org/lapack/lug>

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

## 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: **fact** – Nag\_FactoredFormType *Input*

*On entry:* specifies whether or not the factorized form of the matrix  $A$  is supplied on entry, and if not, whether the matrix  $A$  should be equilibrated before it is factorized.

**fact** = Nag\_Factored

**afp** contains the factorized form of  $A$ . If **equed** = Nag\_Equilibrated, the matrix  $A$  has been equilibrated with scaling factors given by **s**. **ap** and **afp** will not be modified.

**fact** = Nag\_NotFactored

The matrix  $A$  will be copied to **afp** and factorized.

**fact** = Nag\_EquilibrateAndFactor

The matrix  $A$  will be equilibrated if necessary, then copied to **afp** and factorized.

*Constraint:* **fact** = Nag\_Factored, Nag\_NotFactored or Nag\_EquilibrateAndFactor.

3: **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.

4: **n** – Integer *Input*

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

*Constraint:*  $n \geq 0$ .

5: **nrhs** – Integer *Input*

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

*Constraint:* **nrhs**  $\geq 0$ .

6: **ap**[*dim*] – Complex *Input/Output*

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

*On entry:* if **fact** = Nag\_Factored and **equed** = Nag\_Equilibrated, **ap** must contain the equilibrated matrix  $D_S A D_S$ ; otherwise, **ap** must contain the  $n$  by  $n$  Hermitian 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** $[(j-1) \times j/2 + i - 1]$ , for  $i \leq j$ ;  
 if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap** $[(2n-j) \times (j-1)/2 + i - 1]$ , for  $i \geq j$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $A_{ij}$  is stored in **ap** $[(2n-i) \times (i-1)/2 + j - 1]$ , for  $i \leq j$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap** $[(i-1) \times i/2 + j - 1]$ , for  $i \geq j$ .

*On exit:* if **fact** = Nag\_Factored or Nag\_NotFactored, or if **fact** = Nag\_EquilibrateAndFactor and **equed** = Nag\_NoEquilibration, **ap** is not modified.

If **fact** = Nag\_EquilibrateAndFactor and **equed** = Nag\_Equilibrated, **ap** is overwritten by  $D_S A D_S$ .

7: **afp** $[dim]$  – Complex *Input/Output*

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

*On entry:* if **fact** = Nag\_Factored, **afp** contains the triangular factor  $U$  or  $L$  from the Cholesky factorization  $A = U^H U$  or  $A = LL^H$ , in the same storage format as **ap**. If **equed** = Nag\_Equilibrated, **afp** is the factorized form of the equilibrated matrix  $D_S A D_S$ .

*On exit:* if **fact** = Nag\_NotFactored or if **fact** = Nag\_EquilibrateAndFactor and **equed** = Nag\_NoEquilibration, **afp** returns the triangular factor  $U$  or  $L$  from the Cholesky factorization  $A = U^H U$  or  $A = LL^H$  of the original matrix  $A$ .

If **fact** = Nag\_EquilibrateAndFactor and **equed** = Nag\_Equilibrated, **afp** returns the triangular factor  $U$  or  $L$  from the Cholesky factorization  $A = U^H U$  or  $A = LL^H$  of the equilibrated matrix  $A$  (see the description of **ap** for the form of the equilibrated matrix).

8: **equed** – Nag\_EquilibrationType \* *Input/Output*

*On entry:* if **fact** = Nag\_NotFactored or Nag\_EquilibrateAndFactor, **equed** need not be set.

If **fact** = Nag\_Factored, **equed** must specify the form of the equilibration that was performed as follows:

if **equed** = Nag\_NoEquilibration, no equilibration;

if **equed** = Nag\_Equilibrated, equilibration was performed, i.e.,  $A$  has been replaced by  $D_S A D_S$ .

*On exit:* if **fact** = Nag\_Factored, **equed** is unchanged from entry.

Otherwise, if no constraints are violated, **equed** specifies the form of the equilibration that was performed as specified above.

*Constraint:* if **fact** = Nag\_Factored, **equed** = Nag\_NoEquilibration or Nag\_Equilibrated.

9: **s** $[dim]$  – double *Input/Output*

**Note:** the dimension,  $dim$ , of the array **s** must be at least  $\max(1, \mathbf{n})$ .

*On entry:* if **fact** = Nag\_NotFactored or Nag\_EquilibrateAndFactor, **s** need not be set.

If **fact** = Nag\_Factored and **equed** = Nag\_Equilibrated, **s** must contain the scale factors,  $D_S$ , for  $A$ ; each element of **s** must be positive.

*On exit:* if **fact** = Nag\_Factored, **s** is unchanged from entry.

Otherwise, if no constraints are violated and **equed** = Nag\_Equilibrated, **s** contains the scale factors,  $D_S$ , for  $A$ ; each element of **s** is positive.

- 10: **b**[*dim*] – Complex Input/Output
- Note:** the dimension, *dim*, of the array **b** must be at least
- $$\begin{aligned} & \max(1, \mathbf{pdb} \times \mathbf{nrhs}) \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \max(1, \mathbf{n} \times \mathbf{pdb}) \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$
- The (*i*, *j*)th element of the matrix *B* is stored in
- $$\begin{aligned} & \mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$
- On entry:* the *n* by *r* right-hand side matrix *B*.
- On exit:* if **equed** = Nag\_NoEquilibration, **b** is not modified.
- If **equed** = Nag\_Equilibrated, **b** is overwritten by  $D_S B$ .
- 11: **pdb** – Integer Input
- On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **b**.
- Constraints:*
- $$\begin{aligned} & \text{if } \mathbf{order} = \text{Nag\_ColMajor}, \mathbf{pdb} \geq \max(1, \mathbf{n}); \\ & \text{if } \mathbf{order} = \text{Nag\_RowMajor}, \mathbf{pdb} \geq \max(1, \mathbf{nrhs}). \end{aligned}$$
- 12: **x**[*dim*] – Complex Output
- Note:** the dimension, *dim*, of the array **x** must be at least
- $$\begin{aligned} & \max(1, \mathbf{pdx} \times \mathbf{nrhs}) \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \max(1, \mathbf{n} \times \mathbf{pdx}) \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$
- The (*i*, *j*)th element of the matrix *X* is stored in
- $$\begin{aligned} & \mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$
- On exit:* if **fail.code** = NE\_NOERROR or NE\_SINGULAR\_WP, the *n* by *r* solution matrix *X* to the original system of equations. Note that the arrays *A* and *B* are modified on exit if **equed** = Nag\_Equilibrated, and the solution to the equilibrated system is  $D_S^{-1} X$ .
- 13: **pdx** – Integer Input
- On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **x**.
- Constraints:*
- $$\begin{aligned} & \text{if } \mathbf{order} = \text{Nag\_ColMajor}, \mathbf{pdx} \geq \max(1, \mathbf{n}); \\ & \text{if } \mathbf{order} = \text{Nag\_RowMajor}, \mathbf{pdx} \geq \max(1, \mathbf{nrhs}). \end{aligned}$$
- 14: **rcond** – double \* Output
- On exit:* if no constraints are violated, an estimate of the reciprocal condition number of the matrix *A* (after equilibration if that is performed), computed as  $\mathbf{rcond} = 1.0 / (\|A\|_1 \|A^{-1}\|_1)$ .
- 15: **ferr**[**nrhs**] – double Output
- On exit:* if **fail.code** = NE\_NOERROR or NE\_SINGULAR\_WP, an estimate of the forward error bound for each computed solution vector, such that  $\|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq \mathbf{ferr}[j-1]$  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 **rcond**, and is almost always a slight overestimate of the true error.

- 16: **berr**[nrhs] – double *Output*  
*On exit:* if **fail.code** = NE\_NOERROR or 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).
- 17: **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  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .  
 Constraint: **n**  $\geq$  0.

On entry, **nrhs** =  $\langle value \rangle$ .  
 Constraint: **nrhs**  $\geq$  0.

On entry, **pdb** =  $\langle value \rangle$ .  
 Constraint: **pdb**  $>$  0.

On entry, **pdx** =  $\langle value \rangle$ .  
 Constraint: **pdx**  $>$  0.

### NE\_INT\_2

On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 Constraint: **pdb**  $\geq$  max(1, **n**).

On entry, **pdb** =  $\langle value \rangle$  and **nrhs** =  $\langle value \rangle$ .  
 Constraint: **pdb**  $\geq$  max(1, **nrhs**).

On entry, **pdx** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 Constraint: **pdx**  $\geq$  max(1, **n**).

On entry, **pdx** =  $\langle value \rangle$  and **nrhs** =  $\langle value \rangle$ .  
 Constraint: **pdx**  $\geq$  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\_MAT\_NOT\_POS\_DEF

The leading minor of order  $\langle value \rangle$  of  $A$  is not positive definite, so the factorization could not be completed, and the solution has not been computed. **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

$U$  (or  $L$ ) is nonsingular, but **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 **rcond** would suggest.

## 7 Accuracy

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

if **uplo** = Nag\_Upper,  $|E| \leq c(n)\epsilon|U^H||U|$ ;

if **uplo** = Nag\_Lower,  $|E| \leq c(n)\epsilon|L||L^H|$ ,

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

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

## 8 Parallelism and Performance

nag\_zppsvx (f07gpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_zppsvx (f07gpc) 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\_dpssvx (f07gbc).

## 10 Example

This example solves the equations

$$AX = B,$$

where  $A$  is the Hermitian positive definite matrix

$$A = \begin{pmatrix} 3.23 & 1.51 - 1.92i & 1.90 + 0.84i & 0.42 + 2.50i \\ 1.51 + 1.92i & 3.58 & -0.23 + 1.11i & -1.18 + 1.37i \\ 1.90 - 0.84i & -0.23 - 1.11i & 4.09 & 2.33 - 0.14i \\ 0.42 - 2.50i & -1.18 - 1.37i & 2.33 + 0.14i & 4.29 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 3.93 - 6.14i & 1.48 + 6.58i \\ 6.17 + 9.42i & 4.65 - 4.75i \\ -7.17 - 21.83i & -4.91 + 2.29i \\ 1.99 - 14.38i & 7.64 - 10.79i \end{pmatrix}.$$

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

## 10.1 Program Text

```

/* nag_zppsvx (f07gpc) 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         *afp = 0, *ap = 0, *b = 0, *x = 0;
    double          *berr = 0, *ferr = 0, *s = 0;
    char            nag_enum_arg[40];

    /* Nag Types */
    NagError        fail;
    Nag_OrderType   order;
    Nag_EquilibrationType equed;
    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_zppsvx (f07gpc) Example Program Results\n\n");
    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");

```

```

#endif

#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;
    }
#ifdef _WIN32
    scanf_s(" %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)) ||
    !(s = NAG_ALLOC(n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

#ifdef NAG_COLUMN_MAJOR
    pdb = n;
    pdx = n;
#else
    pdb = nrhs;
    pdx = nrhs;
#endif

/* Read the upper or lower 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_zppsvx (f07gpc). */
nag_zppsvx(order, Nag_EquilibrateAndFactor, uplo, n, nrhs, ap, afp,
           &equed, s, b, pdb, x, pdx, &rcond, ferr, berr, &fail);
if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
{
    printf("Error from nag_zppsvx (f07gpc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution using nag_gen_complx_mat_print_comp (x04dbc). */
fflush(stdout);
nag_gen_complx_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_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Print error bounds, condition number and the form of equilibration */
printf("\nBackward errors (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", berr[j], j%7 == 6?"\n":" ");

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

printf("\n\nEstimate of reciprocal condition number\n%11.1e\n\n", rcond);
if (equed == Nag_NoEquilibration)
    printf("A has not been equilibrated\n");
else if (equed == Nag_RowAndColumnEquilibration)
    printf("A has been row and column scaled as diag(S)*A*diag(S)\n");
if (fail.code == NE_SINGULAR)
{
    printf("Error from nag_zppsvx (f07gpc).\n%s\n", fail.message);
    exit_status = 1;
}
}
END:
NAG_FREE(afp);
NAG_FREE(ap);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(s);

return exit_status;
}

#undef A_UPPER
#undef A_LOWER
#undef B

```

## 10.2 Program Data

```
nag_zppsvx (f07gpc) Example Program Data
  4      2                                     : n, nrhs
  Nag_Upper                                     : uplo
  ( 3.23,  0.00) ( 1.51, -1.92) ( 1.90,  0.84) ( 0.42,  2.50)
                ( 3.58,  0.00) (-0.23,  1.11) (-1.18,  1.37)
                ( 4.09,  0.00) ( 2.33, -0.14)
                ( 4.29,  0.00) : matrix A

  ( 3.93, -6.14) ( 1.48,  6.58)
  ( 6.17,  9.42) ( 4.65, -4.75)
  (-7.17,-21.83) (-4.91,  2.29)
  ( 1.99,-14.38) ( 7.64,-10.79)           : matrix B
```

## 10.3 Program Results

nag\_zppsvx (f07gpc) Example Program Results

```
Solution(s)
          1          2
1 ( 1.0000,-1.0000) (-1.0000, 2.0000)
2 (-0.0000, 3.0000) ( 3.0000,-4.0000)
3 (-4.0000,-5.0000) (-2.0000, 3.0000)
4 ( 2.0000, 1.0000) ( 4.0000,-5.0000)

Backward errors (machine-dependent)
 1.1e-16      7.9e-17

Estimated forward error bounds (machine-dependent)
 6.1e-14      7.4e-14

Estimate of reciprocal condition number
 6.6e-03

A has not been equilibrated
```

---