NAG Library Function Document

nag_zpbsvx (f07hpc)

1 Purpose

nag_zpbsvx (f07hpc) uses the Cholesky factorization

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

to compute the solution to a complex system of linear equations

$$AX = B,$$

where A is an n by n Hermitian positive definite band matrix of bandwidth $(2k_d + 1)$ 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_zpbsvx (Nag_OrderType order, Nag_FactoredFormType fact,
     Nag_UploType uplo, Integer n, Integer kd, Integer nrhs, Complex ab[],
     Integer pdab, Complex afb[], Integer pdafb,
     Nag_EquilibrationType *equed, double s[], Complex b[], Integer pdb,
     Complex x[], Integer pdx, double *rcond, double ferr[], double berr[],
     NagError *fail)
```

3 Description

nag_zpbsvx (f07hpc) performs the following steps:

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

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

   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 $A = U^{\mathrm{H}}U$ if **uplo** $=$ Nag_Upper or $A = LL^{\mathrm{H}}$ 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
: **afb** contains the factorized form of A. If **equed** = Nag_Equilibrated, the matrix A has been equilibrated with scaling factors given by **s**. **ab** and **afb** will not be modified.

fact = Nag_NotFactored
: The matrix A will be copied to **afb** and factorized.

fact = Nag_EquilibrateAndFactor
: The matrix A will be equilibrated if necessary, then copied to **afb** 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** ≥ 0.

5: **kd** – Integer *Input*

On entry: k_d, the number of superdiagonals of the matrix A if **uplo** = Nag_Upper, or the number of subdiagonals if **uplo** = Nag_Lower.

Constraint: **kd** ≥ 0.

6: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0.

7: **ab**[*dim*] – Complex *Input/Output*

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

On entry: the upper or lower triangle of the Hermitian band matrix A, except if **fact** = Nag_Factored and **equed** = Nag_Equilibrated, in which case **ab** must contain the equilibrated matrix $D_S A D_S$.

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of A_{ij}, depends on the **order** and **uplo** arguments as follows:

if **order** = Nag_ColMajor and **uplo** = Nag_Upper,
A_{ij} is stored in $\mathbf{ab}[k_d + i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = \max(1, j - k_d), \ldots, j$;
if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ab}[i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, j + k_d)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
A_{ij} is stored in $\mathbf{ab}[j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_d)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ab}[k_d + j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_d), \ldots, i$.

On exit: if **fact** = Nag_EquilibrateAndFactor and **equed** = Nag_Equilibrated, **ab** is overwritten by $D_S A D_S$.

8: **pdab** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix A in the array **ab**.

Constraint: $\mathbf{pdab} \geq \mathbf{kd} + 1$.

9: **afb**[*dim*] – Complex *Input/Output*

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

On entry: if **fact** = Nag_Factored, **afb** contains the triangular factor U or L from the Cholesky factorization $A = U^{\mathrm{H}} U$ or $A = L L^{\mathrm{H}}$ of the band matrix A, in the same storage format as A. If **equed** = Nag_Equilibrated, **afb** is the factorized form of the equilibrated matrix A.

On exit: if **fact** = Nag_NotFactored, **afb** returns the triangular factor U or L from the Cholesky factorization $A = U^{\mathrm{H}} U$ or $A = L L^{\mathrm{H}}$.

If **fact** = Nag_EquilibrateAndFactor, **afb** returns the triangular factor U or L from the Cholesky factorization $A = U^{\mathrm{H}} U$ or $A = L L^{\mathrm{H}}$ of the equilibrated matrix A (see the description of **ab** for the form of the equilibrated matrix).

10: **pdafb** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix A in the array **afb**.

Constraint: $\mathbf{pdafb} \geq \mathbf{kd} + 1$.

11: **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.

12: **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.

13: **b**[*dim*] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

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

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

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 $D_S B$.

14: **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, $\mathbf{pdb} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.

15: **x**[*dim*] – Complex *Output*

Note: the dimension, *dim*, of the array **x** must be at least

$\max(1, \mathbf{pdx} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

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

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

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 $D_S^{-1} X$.

16: **pdx** – Integer *Input*

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

Constraints:

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

17: **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 **rcond** $= 1.0/\left(\|A\|_1\|A^{-1}\|_1\right)$.

18: **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 $\|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq$ **ferr**$[j-1]$ where $\hat{x}_j$ is the jth 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.

19: **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).

20: **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, **kd** = ⟨*value*⟩.
Constraint: **kd** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **nrhs** = ⟨*value*⟩.
Constraint: **nrhs** ≥ 0.

On entry, **pdab** = ⟨*value*⟩.
Constraint: **pdab** > 0.

On entry, **pdafb** = ⟨*value*⟩.
Constraint: **pdafb** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

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

NE_INT_2

On entry, **pdab** $= \langle value \rangle$ and **kd** $= \langle value \rangle$.
Constraint: **pdab** $\geq$ **kd** $+ 1$.

On entry, **pdafb** $= \langle value \rangle$ and **kd** $= \langle value \rangle$.
Constraint: **pdafb** $\geq$ **kd** $+ 1$.

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

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

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

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

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, $|E| \leq c(n)\epsilon|U^{\mathrm{H}}||U|$;

if **uplo** = Nag_Lower, $|E| \leq c(n)\epsilon|L||L^{\mathrm{H}}|$,

$c(n)$ is a modest linear function of n, and ϵ 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

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

where $\operatorname{cond}(A, \hat{x}, b) = \left\| |A^{-1}|(|A||\hat{x}| + |b|) \right\|_\infty / \|\hat{x}\|_\infty \leq \operatorname{cond}(A) = \left\| |A^{-1}||A| \right\|_\infty \leq \kappa_\infty(A)$. If $\hat{x}$ is the

jth 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_zpbsvx (f07hpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zpbsvx (f07hpc) 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

When $n \gg k$, the factorization of A requires approximately $4n(k+1)^2$ floating-point operations, where k is the number of superdiagonals.

For each right-hand side, computation of the backward error involves a minimum of $32nk$ floating-point operations. Each step of iterative refinement involves an additional $48nk$ 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 $16nk$ operations.

The real analogue of this function is nag_dpbsvx (f07hbc).

10 Example

This example solves the equations

$$AX = B,$$

where A is the Hermitian positive definite band matrix

$$A = \begin{pmatrix} 9.39 & 1.08-1.73i & 0 & 0 \\ 1.08+1.73i & 1.69 & -0.04+0.29i & 0 \\ 0 & -0.04-0.29i & 2.65 & -0.33+2.24i \\ 0 & 0 & -0.33-2.24i & 2.17 \end{pmatrix}$$

and

$$B = \begin{pmatrix} -12.42+68.42i & 54.30-56.56i \\ -9.93+\ 0.88i & 18.32+\ 4.76i \\ -27.30-\ 0.01i & -4.40+\ 9.97i \\ 5.31+23.63i & 9.43+\ 1.41i \end{pmatrix}.$$

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_zpbsvx (f07hpc) 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, kd, n, nrhs, pdab, pdafb, pdb,
                        pdx;

  /* Arrays */
  Complex               *ab = 0, *afb = 0, *b = 0, *x = 0;
  double                *berr = 0, *ferr = 0, *s = 0;
  char                   nag_enum_arg[40];

  /* Nag Types */
  NagError              fail;
  Nag_UploType          uplo;
  Nag_OrderType         order;
  Nag_EquilibrationType equed;

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I, J) ab[(J-1)*pdab + kd + I - J]
#define AB_LOWER(I, J) ab[(J-1)*pdab + I - J]
#define B(I, J)        b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define AB_UPPER(I, J) ab[(I-1)*pdab + J - I]
#define AB_LOWER(I, J) ab[(I-1)*pdab + kd + J - I]
#define B(I, J)        b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif
  INIT_FAIL(fail);

  printf("nag_zpbsvx (f07hpc) 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"%"NAG_IFMT"%*[^\n]", &n, &kd, &nrhs);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &kd, &nrhs);
#endif
  if (n < 0 || kd < 0 || nrhs < 0)
    {
      printf("%s\n", "Invalid n or kd or nrhs");
      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 (!(ab   = NAG_ALLOC((kd+1) * n, Complex)) ||
      !(afb  = NAG_ALLOC((kd+1) * n, 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;
    }
  pdab = kd+1;
  pdafb = kd+1;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
  pdx = n;
#else
  pdb = nrhs;
  pdx = nrhs;
#endif

  /* Read the upper or lower triangular part of the band matrix A */
  /* from data file */
  if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
      for (j = i; j <= MIN(n, i + kd); ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &AB_UPPER(i, j).re, &AB_UPPER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re, &AB_UPPER(i, j).im);
#endif
  else
    for (i = 1; i <= n; ++i)
      for (j = MAX(1, i - kd); j <= i; ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &AB_LOWER(i, j).re, &AB_LOWER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re, &AB_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_zpbsvx (f07hpc). */
  nag_zpbsvx(order, Nag_EquilibrateAndFactor, uplo, n, kd, nrhs, ab, pdab,
             afb, pdafb, &equed, s, b, pdb, x, pdx, &rcond, ferr, berr, &fail);
  if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
    {
      printf("Error from nag_zpbsvx (f07hpc).\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_zpbsvx (f07hpc).\n%s\n", fail.message);
      exit_status = 1;
    }
 END:
  NAG_FREE(ab);
  NAG_FREE(afb);
  NAG_FREE(b);
  NAG_FREE(x);
  NAG_FREE(berr);
  NAG_FREE(ferr);
  NAG_FREE(s);

  return exit_status;
}

#undef AB_UPPER
#undef AB_LOWER
#undef B
```

10.2 Program Data

```
nag_zpbsvx (f07hpc) Example Program Data
  4                1               2                          : n kd nrhs
  Nag_Upper                                                   : uplo
 (  9.39, 0.00) (  1.08, -1.73)
                (  1.69,  0.00) ( -0.04, 0.29)
                                (  2.65, 0.00) ( -0.33, 2.24)
                                               (  2.17, 0.00) : matrix A
 (-12.42,68.42) ( 54.30,-56.56)
 ( -9.93, 0.88) ( 18.32,  4.76)
 (-27.30,-0.01) ( -4.40,  9.97)
 (  5.31,23.63) (  9.43,  1.41)                               : matrix B
```

10.3 Program Results

```
nag_zpbsvx (f07hpc) Example Program Results

 Solution(s)
                    1                   2
 1  (-1.0000, 8.0000)  ( 5.0000,-6.0000)
 2  ( 2.0000,-3.0000)  ( 2.0000, 3.0000)
 3  (-4.0000,-5.0000)  (-8.0000, 4.0000)
 4  ( 7.0000, 6.0000)  (-1.0000,-7.0000)

Backward errors (machine-dependent)
    8.2e-17     5.4e-17

Estimated forward error bounds (machine-dependent)
    3.6e-14     3.0e-14
```

```
Estimate of reciprocal condition number
    7.6e-03

A has not been equilibrated
```