# NAG Library Function Document

# nag_zpbsv (f07hnc)

## 1 Purpose

nag_zpbsv (f07hnc) computes 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.

## 2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_zpbsv (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer kd, Integer nrhs, Complex ab[], Integer pdab, Complex b[],
     Integer pdb, NagError *fail)
```

## 3 Description

nag_zpbsv (f07hnc) uses the Cholesky decomposition to factor $A$ 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 band matrix, and $L$ is a lower triangular band matrix, with the same number of superdiagonals or subdiagonals as $A$. The factored form of $A$ is then used to solve the system of equations $AX = B$.

## 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

## 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: **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.

3: **n** – Integer *Input*

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

*Constraint*: $\mathbf{n} \geq 0$.

4: **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*: $\mathbf{kd} \geq 0$.

5: **nrhs** – Integer *Input*

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

*Constraint*: $\mathbf{nrhs} \geq 0$.

6: **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 $A$.

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 **fail.code** $=$ NE_NOERROR, the triangular factor $U$ or $L$ from the Cholesky factorization $A = U^{\mathrm{H}}U$ or $A = LL^{\mathrm{H}}$ of the band matrix $A$, in the same storage format as $A$.

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

8: **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 **fail.code** $=$ NE_NOERROR, the $n$ by $r$ solution matrix $X$.

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

10: **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, **kd** $= \langle value \rangle$.
Constraint: **kd** $\geq 0$.

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

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

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

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

**NE_INT_2**

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

**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.

**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.

# 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

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

where $\kappa(A)=\left\|A^{-1}\right\|_1\|A\|_1$, the condition number of $A$ with respect to the solution of the linear equations. See Section 4.4 of Anderson *et al.* (1999) for further details.

nag_zpbsvx (f07hpc) is a comprehensive LAPACK driver that returns forward and backward error bounds and an estimate of the condition number. Alternatively, nag_herm_posdef_band_lin_solve (f04cfc) solves $Ax=b$ and returns a forward error bound and condition estimate. nag_herm_posdef_band_lin_solve (f04cfc) calls nag_zpbsv (f07hnc) to solve the equations.

# 8 Parallelism and Performance

nag_zpbsv (f07hnc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zpbsv (f07hnc) 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 total number of floating-point operations is approximately $4n(k+1)^2+16nkr$, where $k$ is the number of superdiagonals and $r$ is the number of right-hand sides.

The real analogue of this function is nag_dpbsv (f07hac).

# 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

$$b = \begin{pmatrix} -12.42 + 68.42i \\ -9.93 + \ \ 0.88i \\ -27.30 - \ \ 0.01i \\ 5.31 + 23.63i \end{pmatrix}.$$

Details of the Cholesky factorization of $A$ are also output.

## 10.1 Program Text

```
/* nag_zpbsv (f07hnc) 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 */
  Integer       exit_status = 0, i, j, kd, n, nrhs, pdab, pdb;

  /* Arrays */
  Complex       *ab = 0, *b = 0;
  char          nag_enum_arg[40];

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

#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_zpbsv (f07hnc) 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("Invalid n, kd 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 (!(ab = NAG_ALLOC((kd+1) * n, Complex)) ||
      !(b =  NAG_ALLOC(n*nrhs, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  pdab = kd+1;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = 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_zpbsv (f07hnc). */
  nag_zpbsv(order, uplo, n, kd, nrhs, ab, pdab, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
```

```
      printf("Error from nag_zpbsv (f07hnc).\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":" ");
      printf("\n");
    }
  printf("\n");

  /* Print details of factorization using
   * nag_band_complx_mat_print_comp (x04dfc).
   */
  fflush(stdout);
  if (uplo == Nag_Upper)
      nag_band_complx_mat_print_comp(order, n, n, 0, kd, ab, pdab,
                                     Nag_BracketForm, "%7.4f",
                                     "Cholesky factor U", Nag_IntegerLabels, 0,
                                     Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  else
      nag_band_complx_mat_print_comp(order, n, n, kd, 0, ab, pdab,
                                     Nag_BracketForm, "%7.4f",
                                     "Cholesky factor L", Nag_IntegerLabels, 0,
                                     Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_band_complx_mat_print_comp (x04dfc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(ab);
  NAG_FREE(b);

  return exit_status;
}

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

## 10.2 Program Data

```
nag_zpbsv (f07hnc) Example Program Data
    4             1             1                         : 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) (-9.93, 0.88) (-27.30,-0.01) ( 5.31,23.63) : vector b
```

## 10.3 Program Results

```
nag_zpbsv (f07hnc) Example Program Results

Solution
(-1.0000,  8.0000)
( 2.0000, -3.0000)
(-4.0000, -5.0000)
( 7.0000,  6.0000)
```

```
 Cholesky factor U
                    1                   2                   3                   4
 1  ( 3.0643, 0.0000)  ( 0.3524,-0.5646)
 2                     ( 1.1167, 0.0000)  (-0.0358, 0.2597)
 3                                        ( 1.6066, 0.0000)  (-0.2054, 1.3942)
 4                                                           ( 0.4289, 0.0000)
```