NAG Library Function Document

nag_zpbsvx (f07hpc)

1 Purpose
nag_zpbsvx (f07hpc) 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 band matrix of bandwidth \((2kd + 1)\) 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_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 \( \text{fact} = \text{Nag\_EquilibrateAndFactor} \), real diagonal scaling factors, \( D_S \), are computed to equilibrate the system:

\[
(D_S AD_S)(D_S^3 X) = D_S B.
\]

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 \( D_S AD_S \) and \( B \) by \( D_S B \).

2. If \( \text{fact} = \text{Nag\_NotFactored} \) or \( \text{Nag\_EquilibrateAndFactor} \), the Cholesky decomposition is used to factor the matrix \( A \) (after equilibration if \( \text{fact} = \text{Nag\_EquilibrateAndFactor} \)) as \( A = U^H U \) if \( \text{uplo} = \text{Nag\_Upper} \) or \( A = LL^H \) if \( \text{uplo} = \text{Nag\_Lower} \), 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 \( \text{fail\_ernum} = i \) 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 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.

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


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 $A$ is supplied on entry, and if not, whether the matrix $A$ should be equilibrated before it is factorized.

fact = Nag_Factored

$A_f$ contains the factorized form of $A$. If equed = Nag_Equilibrated, the matrix $A$ has been equilibrated with scaling factors given by $s$. $A_{fb}$ and $A_f$ will not be modified.

fact = Nag_NotFactored

The matrix $A$ will be copied to $A_{fb}$ and factorized.

fact = Nag_EquilibrateAndFactor

The matrix $A$ will be equilibrated if necessary, then copied to $A_{fb}$ and factorized.

Constraint: fact = Nag_Factored, Nag_NotFactored or Nag_EquilibrateAndFactor.

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: $n \geq 0$.

5: kd – Integer

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: $k_d \geq 0$.

6: nrhs – Integer

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

Constraint: $nrhs \geq 0$. 

f07hpc.2 Mark 25
Factored and ColMajor and Equilibrated, Factored, EquilibrateAndFactor, RowMajor and NotFactored, RowMajor and Lower, Equilibrated, in which case EquilibrateAndFactor, Upper, EquilibrateAndFactor and Mark 25 f07hpc.3

f07 – Linear Equations (LAPACK)
f07hpc

11: boxed environment

Mark 25
if equed = Nag_NoEquilibration, no equilibration;

if equed = Nag_Equilibrated, equilibration was performed, i.e., $A$ has been replaced by $D_SAD_S$.

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 $\max(1, n)$.

On entry: if fact = Nag_NotFactored or Nag_EquilibrationAndFactor, $s$ 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 $\max(1, \text{pdb} \times \text{nrhs})$ when order = Nag_ColMajor; $\max(1, n \times \text{pdb})$ when order = Nag_RowMajor.

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

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

$b[(i - 1) \times \text{pdb} + j - 1]$ 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 $D_SB$.

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, pdb $\geq \max(1, n)$;

if order = Nag_RowMajor, pdb $\geq \max(1, \text{nrhs})$.

15: $x[dim]$ – Complex  
Output
Note: the dimension, $dim$, of the array $x$ must be at least $\max(1, \text{pdx} \times \text{nrhs})$ when order = Nag_ColMajor; $\max(1, n \times \text{pdx})$ when order = Nag_RowMajor.

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

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

$x[(i - 1) \times \text{pdx} + j - 1]$ 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  
   
   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 ≥ max(1, n);
   if order = Nag_RowMajor, pdx ≥ max(1, nrhs).

17: rcond — double *  
   
   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/(||A||_1 ||A^{-1}||_1).

18: ferr[nrhs] — double  
   
   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).

19: berr[nrhs] — double  
   
   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 *  
   
   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 \( \langle \text{value} \rangle \) had an illegal value.

NE_INT  
On entry, \( kd = \langle \text{value} \rangle \).  
Constraint: \( kd ≥ 0 \).

On entry, \( n = \langle \text{value} \rangle \).  
Constraint: \( n ≥ 0 \).

On entry, \( nrhs = \langle \text{value} \rangle \).  
Constraint: \( nrhs ≥ 0 \).

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

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

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

**NE_INT_2**

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

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

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

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**

$U$ (or $L$) 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 $x$ is the exact solution of a perturbed system of equations $(A + E)x = b$, where

if $\text{uplo} = \text{Nag}\_\text{Upper}$, $|E| \leq c(n)\epsilon|U^H||U|$;

if $\text{uplo} = \text{Nag}\_\text{Lower}$, $|E| \leq c(n)\epsilon|L||L^H|$,

$c(n)$ 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

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

where $\text{cond}(A, \hat{x}, b) = \frac{||A^{-1}(|A||\hat{x}| + |b|)||_\infty}{||\hat{x}||_\infty} \leq \text{cond}(A) = \frac{||A^{-1}||_1||A||_1}{\kappa_\infty(A)}$. If $\hat{x}$ is the
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>
```c
#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("%*[^
]");
    #else
    scanf("%*[^
]");
    #endif

    #ifdef _WIN32
    scanf_s("%39s%*[^
]", &nag_enum_arg, _countof(nag_enum_arg));
    #else
    scanf("%39s%*[^
]");
    #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 or kd or nrhs\n");
        exit_status = 1;
        goto END;
    }

    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))

    END:
    exit_status = 1;
    return exit_status;
}
```
/* 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("%*[^
"]);
#else
  scanf("%*[^
"]);
#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("%*[^
"]);
#else
  scanf("%*[^
"]);
#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_complex_mat_print_comp (x04dbc). */
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)
/* Print error bounds, condition number and the form of equilibration */
printf("Error from nag_gen_complex_mat_print_comp (x04dbc).\n\s\n", fail.message);
exit_status = 1;
goto END;
}

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;
}
#endif AB_UPPER
#endif AB_LOWER
#endif 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)
( 5.31, 23.63) ( 9.43, 1.41) : 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