NAG Library Function Document

nag_zpbtrs (f07hsc)

1 Purpose

nag_zpbtrs (f07hsc) solves a complex Hermitian positive definite band system of linear equations with multiple right-hand sides,

$$AX = B,$$

where A has been factorized by nag_zpbtrf (f07hrc).

2 Specification

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

3 Description

nag_zpbtrs (f07hsc) is used to solve a complex Hermitian positive definite band system of linear equations $AX = B$, the function must be preceded by a call to nag_zpbtrf (f07hrc) which computes the Cholesky factorization of A. The solution X is computed by forward and backward substitution.

If **uplo** $=$ Nag_Upper, $A = U^{H}U$, where U is upper triangular; the solution X is computed by solving $U^{H}Y = B$ and then $UX = Y$.

If **uplo** $=$ Nag_Lower, $A = LL^{H}$, where L is lower triangular; the solution X is computed by solving $LY = B$ and then $L^{H}X = Y$.

4 References

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: specifies how A has been factorized.

uplo $=$ Nag_Upper
: $A = U^{H}U$, where U is upper triangular.

uplo $=$ Nag_Lower
: $A = LL^{H}$, where L is lower triangular.

Constraint: **uplo** $=$ Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

4: **kd** – Integer *Input*

On entry: k_d, the number of superdiagonals or subdiagonals of the matrix A.

Constraint: $\mathbf{kd} \geq 0$.

5: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides.

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

6: **ab**[dim] – const Complex *Input*

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

On entry: the Cholesky factor of A, as returned by nag_zpbtrf (f07hrc).

7: **pdab** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix 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: 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, $\mathbf{pdb} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.

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.

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

NE_INT_2

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

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

On entry, **pdb** $=$ ⟨*value*⟩ and **nrhs** $=$ ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1,$ **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.

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, $|E| \leq c(k+1)\epsilon|U^{\mathrm{H}}||U|$;

if **uplo** $=$ Nag_Lower, $|E| \leq c(k+1)\epsilon|L||L^{\mathrm{H}}|$,

$c(k+1)$ is a modest linear function of $k+1$, and ϵ is the ***machine precision***.

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

$$\frac{\|x-\hat{x}\|_\infty}{\|x\|_\infty} \leq c(k+1)\,\mathrm{cond}(A,x)\epsilon$$

where $\mathrm{cond}(A,x) = \left\||A^{-1}||A||x|\right\|_\infty / \|x\|_\infty \leq \mathrm{cond}(A) = \left\||A^{-1}||A|\right\|_\infty \leq \kappa_\infty(A)$. Note that $\mathrm{cond}(A,x)$ can be much smaller than $\mathrm{cond}(A)$.

Forward and backward error bounds can be computed by calling nag_zpbrfs (f07hvc), and an estimate for $\kappa_\infty(A)$ $(=\kappa_1(A))$ can be obtained by calling nag_zpbcon (f07huc).

8 Parallelism and Performance

nag_zpbtrs (f07hsc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zpbtrs (f07hsc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of real floating-point operations is approximately $16nkr$, assuming $n \gg k$.

This function may be followed by a call to nag_zpbrfs (f07hvc) to refine the solution and return an error estimate.

The real analogue of this function is nag_dpbtrs (f07hec).

10 Example

This example solves the system of equations $AX = B$, where

$$A = \begin{pmatrix} 9.39 + 0.00i & 1.08 - 1.73i & 0.00 + 0.00i & 0.00 + 0.00i \\ 1.08 + 1.73i & 1.69 + 0.00i & -0.04 + 0.29i & 0.00 + 0.00i \\ 0.00 + 0.00i & -0.04 - 0.29i & 2.65 + 0.00i & -0.33 + 2.24i \\ 0.00 + 0.00i & 0.00 + 0.00i & -0.33 - 2.24i & 2.17 + 0.00i \end{pmatrix}$$

and

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

Here A is Hermitian positive definite, and is treated as a band matrix, which must first be factorized by nag_zpbtrf (f07hrc).

10.1 Program Text

```
/* nag_zpbtrs (f07hsc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer       i, j, k, kd, n, nrhs, pdab, pdb;
  Integer       exit_status = 0;
  Nag_UploType  uplo;
  NagError      fail;
  Nag_OrderType order;
  /* Arrays */
  char          nag_enum_arg[40];
  Complex       *ab = 0, *b = 0;

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I, J) ab[(J-1)*pdab + k + I - J - 1]
#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 + k + J - I - 1]
#define B(I, J)        b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zpbtrs (f07hsc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%*[^\n] ", &n, &kd, &nrhs);
  pdab = kd + 1;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif

  /* 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;
    }

  /* Read A from data file */
  scanf(" %39s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  k = kd + 1;
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= MIN(i+kd, n); ++j)
            scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re,
                  &AB_UPPER(i, j).im);
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = MAX(1, i-kd); j <= i; ++j)
            scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re,
                  &AB_LOWER(i, j).im);
        }
      scanf("%*[^\n] ");
    }
  /* Read B from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }
  scanf("%*[^\n] ");

  /* Factorize A */
  /* nag_zpbtrf (f07hrc).
   * Cholesky factorization of complex Hermitian
   * positive-definite band matrix
```

```
   */
  nag_zpbtrf(order, uplo, n, kd, ab, pdab, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpbtrf (f07hrc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute solution */
  /* nag_zpbtrs (f07hsc).
   * Solution of complex Hermitian positive-definite band
   * system of linear equations, multiple right-hand sides,
   * matrix already factorized by nag_zpbtrf (f07hrc)
   */
  nag_zpbtrs(order, uplo, n, kd, nrhs, ab, pdab, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpbtrs (f07hsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print solution */
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                nrhs, b, pdb, 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;
    }
 END:
  NAG_FREE(ab);
  NAG_FREE(b);
  return exit_status;
}
```

10.2 Program Data

```
nag_zpbtrs (f07hsc) Example Program Data
  4  1  2                                            :Values of n, kd and nrhs
  Nag_Lower                                          :Value of 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)  :End of 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)                            :End of matrix B
```

10.3 Program Results

```
nag_zpbtrs (f07hsc) 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)
```