NAG Library Function Document

nag_zpttrs (f07jsc)

1 Purpose

nag_zpttrs (f07jsc) computes the solution to a complex system of linear equations $AX = B$, where A is an n by n Hermitian positive definite tridiagonal matrix and X and B are n by r matrices, using the LDL^{H} factorization returned by nag_zpttrf (f07jrc).

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_zpttrs (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer nrhs, const double d[], const Complex e[], Complex b[],
     Integer pdb, NagError *fail)
```

3 Description

nag_zpttrs (f07jsc) should be preceded by a call to nag_zpttrf (f07jrc), which computes a modified Cholesky factorization of the matrix A as

$$A = LDL^{\mathrm{H}},$$

where L is a unit lower bidiagonal matrix and D is a diagonal matrix, with positive diagonal elements. nag_zpttrs (f07jsc) then utilizes the factorization to solve the required equations. Note that the factorization may also be regarded as having the form $U^{\mathrm{H}}DU$, where U is a unit upper bidiagonal matrix.

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

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 the form of the factorization as follows:

uplo = Nag_Upper
$A = U^{\mathrm{H}}DU$.

uplo = Nag_Lower
$A = LDL^{\mathrm{H}}$.

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: **nrhs** – Integer *Input*

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

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

5: **d**[dim] – const double *Input*

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

On entry: must contain the n diagonal elements of the diagonal matrix D from the LDL^{H} or $U^{\mathrm{H}}DU$ factorization of A.

6: **e**[dim] – const Complex *Input*

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

On entry: if **uplo** = Nag_Upper, **e** must contain the $(n-1)$ superdiagonal elements of the unit upper bidiagonal matrix U from the $U^{\mathrm{H}}DU$ factorization of A.

If **uplo** = Nag_Lower, **e** must contain the $(n-1)$ subdiagonal elements of the unit lower bidiagonal matrix L from the LDL^{H} factorization of A.

7: **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

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

On entry: the n by r matrix of right-hand sides B.

On exit: the n by r solution matrix X.

8: **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, $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.

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

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

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

NE_INT_2

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

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

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 ϵ 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.

Following the use of this function nag_zptcon (f07juc) can be used to estimate the condition number of A and nag_zptrfs (f07jvc) can be used to obtain approximate error bounds.

8 Parallelism and Performance

nag_zpttrs (f07jsc) is not threaded by NAG in any implementation.

nag_zpttrs (f07jsc) 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 floating-point operations required to solve the equations $AX = B$ is proportional to nr.

The real analogue of this function is nag_dpttrs (f07jec).

10 Example

This example solves the equations

$$AX = B,$$

where A is the Hermitian positive definite tridiagonal matrix

$$A = \begin{pmatrix} 16.0 & 16.0 - 16.0i & 0 & 0 \\ 16.0 + 16.0i & 41.0 & 18.0 + 9.0i & 0.0 \\ 0 & 18.0 - 9.0i & 46.0 & 1.0 + 4.0i \\ 0 & 0 & 1.0 - 4.0i & 21.0 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 64.0 + 16.0i & -16.0 - 32.0i \\ 93.0 + 62.0i & 61.0 - 66.0i \\ 78.0 - 80.0i & 71.0 - 74.0i \\ 14.0 - 27.0i & 35.0 + 15.0i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zpttrs (f07jsc) Example Program.
 *
 * Copyright 2004 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, n, nrhs, pdb;
  Nag_OrderType order;

  /* Arrays */
  Complex       *b = 0, *e = 0;
  double        *d = 0;

  /* Nag Types */
  NagError      fail;

#ifdef NAG_COLUMN_MAJOR
#define B(I, J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define B(I, J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zpttrs (f07jsc) Example Program Results\n\n");
  /* Skip heading in data file */
  scanf("%*[^\n]");
  scanf("%ld%ld%*[^\n]", &n, &nrhs);
  if (n < 0 || nrhs < 0)
    {
      printf("Invalid n or nrhs\n");
      exit_status = 1;
      goto END;
    }
```

```
  /* Allocate memory */
  if (!(b = NAG_ALLOC(n * nrhs, Complex)) ||
      !(e = NAG_ALLOC(n-1, Complex)) ||
      !(d = NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif

  /* Read the upper bidiagonal part of the tridiagonal matrix A from */
  /* data file */
  for (i = 0; i < n - 1; ++i) scanf(" ( %lf , %lf )", &e[i].re, &e[i].im);
  scanf("%*[^\n]");
  for (i = 0; i < n; ++i) scanf("%lf", &d[i]);
  scanf("%*[^\n]");

  /* Read the right hand matrix B */
  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 the tridiagonal matrix A using nag_zpttrf (f07jrc). */
  nag_zpttrf(n, d, e, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpttrf (f07jrc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Solve the equations AX = B using nag_zpttrs (f07jsc). */
  nag_zpttrs(order, Nag_Upper, n, nrhs, d, e, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpttrs (f07jsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print the solution using nag_gen_complx_mat_print_comp (x04dbc). */
  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(b);
  NAG_FREE(e);
  NAG_FREE(d);

  return exit_status;
}
#undef B
```

10.2 Program Data

```
nag_zpttrs (f07jsc) Example Program Data
    4             2                              : n,  nrhs
 ( 16.0,-16.0) ( 18.0,  9.0) (  1.0,  4.0)       : sub-diagonal e
   16.0          41.0          46.0          21.0  : diagonal D
 ( 64.0, 16.0) (-16.0,-32.0)
 ( 93.0, 62.0) ( 61.0,-66.0)
 ( 78.0,-80.0) ( 71.0,-74.0)
 ( 14.0,-27.0) ( 35.0, 15.0)                     : matrix B
```

10.3 Program Results

```
nag_zpttrs (f07jsc) Example Program Results

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