NAG Library Function Document

nag_zptrfs (f07jvc)

1 Purpose

nag_zptrfs (f07jvc) computes error bounds and refines 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 modified Cholesky factorization returned by nag_zpttrf (f07jrc) and an initial solution returned by nag_zpttrs (f07jsc). Iterative refinement is used to reduce the backward error as much as possible.

2 Specification

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

3 Description

nag_zptrfs (f07jvc) should normally be preceded by calls to nag_zpttrf (f07jrc) and nag_zpttrs (f07jsc). nag_zpttrf (f07jrc) 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 compute a solution, $\hat{X}$, to the required equations. Letting $\hat{x}$ denote a column of $\hat{X}$, nag_zptrfs (f07jvc) computes a *component-wise backward error*, β, the smallest relative perturbation in each element of A and b such that $\hat{x}$ is the exact solution of a perturbed system

$$(A + E)\hat{x} = b + f, \quad \text{with} \quad \left|e_{ij}\right| \leq \beta\left|a_{ij}\right|, \quad \text{and} \quad \left|f_j\right| \leq \beta\left|b_j\right|.$$

The function also estimates a bound for the *component-wise forward error* in the computed solution defined by $\max|x_i - \hat{x}_i|/\max|\hat{x}_i|$, where x is the corresponding column of the exact solution, X.

Note that the modified Cholesky factorization of A can also be expressed as

$$A = U^{\mathrm{H}}DU,$$

where U is unit upper bidiagonal.

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: **n** ≥ 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: **nrhs** ≥ 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 matrix 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 matrix A.

If **uplo** = Nag_Lower, **e** must contain the $(n - 1)$ subdiagonal elements of the matrix A.

7: **df**[dim] – const double *Input*

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

On entry: must contain the n diagonal elements of the diagonal matrix D from the LDL^{T} factorization of A.

8: **ef**[dim] – const Complex *Input*

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

On entry: if **uplo** = Nag_Upper, **ef** 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, **ef** must contain the $(n - 1)$ subdiagonal elements of the unit lower bidiagonal matrix L from the LDL^{H} factorization of A.

9: **b**[dim] – const Complex *Input*

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

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

10: **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** $\geq \max(1,$ **nrhs**$)$.

11: **x**$[dim]$ – Complex *Input/Output*

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

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

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

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

On entry: the n by r initial solution matrix X.

On exit: the n by r refined solution matrix X.

12: **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** $\geq \max(1,$ **n**$)$;
if **order** = Nag_RowMajor, **pdx** $\geq \max(1,$ **nrhs**$)$.

13: **ferr**[**nrhs**] – double *Output*

On exit: estimate of the forward error bound for each computed solution vector, such that $\left\|\hat{x}_j - x_j\right\|_\infty / \left\|\hat{x}_j\right\|_\infty \leq$ **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 almost always a slight overestimate of the true error.

14: **berr**[**nrhs**] – double *Output*

On exit: 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).

15: **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.

On entry, **pdx** = ⟨*value*⟩.
Constraint: **pdx** > 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**).

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

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

$$\|E\|_\infty = O(\epsilon)\|A\|_\infty$$

and ϵ is the ***machine precision***. An approximate error bound for the computed solution is given by

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

where $\kappa(A) = \left\|A^{-1}\right\|_\infty \|A\|_\infty$, 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.

Function nag_zptcon (f07juc) can be used to compute the condition number of A.

8 Parallelism and Performance

nag_zptrfs (f07jvc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zptrfs (f07jvc) 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. At most five steps of iterative refinement are performed, but usually only one or two steps are required.

The real analogue of this function is nag_dptrfs (f07jhc).

10 Example

This example solves the equations

$$AX = B,$$

where A is the Hermitian positive definite tridiagonal matrix

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

and

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

Estimates for the backward errors and forward errors are also output.

10.1 Program Text

```
/* nag_zptrfs (f07jvc) 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>
#include <nagf16.h>

int main(void)
{

  /* Scalars */
  Integer       exit_status = 0, i, j, n, nrhs, pdb, pdx;
  Nag_OrderType order;

  /* Arrays */
  Complex       *b = 0, *e = 0, *ef = 0, *x = 0;
  double        *berr = 0, *d = 0, *df = 0, *ferr = 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_zptrfs (f07jvc) 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;
    }
  if (!(b    = NAG_ALLOC(n * nrhs, Complex)) ||
      !(e    = NAG_ALLOC(n - 1, Complex)) ||
      !(ef   = NAG_ALLOC(n - 1, Complex)) ||
      !(x    = NAG_ALLOC(n * nrhs, Complex)) ||
      !(berr = NAG_ALLOC(nrhs, double)) ||
      !(d    = NAG_ALLOC(n, double)) ||
      !(df   = NAG_ALLOC(n, double)) ||
      !(ferr = NAG_ALLOC(nrhs, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
#ifdef NAG_COLUMN_MAJOR
      pdb = n;
      pdx = n;
#else
      pdb = nrhs;
      pdx = nrhs;
#endif

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

  /* Copy A into DF and EF, and copy B into X */
  for (i = 0; i < n; ++i) df[i] = d[i];
  for (i = 0; i < n - 1; ++i) {
    ef[i].re = e[i].re;
    ef[i].im = e[i].im;
  }

  /* Copy B into X using nag_zge_copy (f16tfc). */
  nag_zge_copy(order, Nag_NoTrans, n, nrhs, b, pdb, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Factorize the copy of the tridiagonal matrix A
   * using nag_zpttrf (f07jrc).
   */
  nag_zpttrf(n, df, ef, &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_Lower, n, nrhs, df, ef, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpttrs (f07jsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Improve the solution and compute error estimates
   * using nag_zptrfs (f07jvc).
   */
  nag_zptrfs(order, Nag_Lower, n, nrhs, d, e, df, ef, b, pdb, x, pdx,
             ferr, berr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zptrfs (f07jvc).\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, 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 the the forward and backward error estimates */
  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":" ");

 END:
  NAG_FREE(b);
  NAG_FREE(e);
  NAG_FREE(ef);
  NAG_FREE(x);
  NAG_FREE(berr);
  NAG_FREE(d);
  NAG_FREE(df);
  NAG_FREE(ferr);

  return exit_status;
}
#undef B
```

10.2 Program Data

```
nag_zptrfs (f07jvc) Example Program Data
    4            2                                  : n,  nrhs
   16.0          41.0          46.0          21.0 : diagonal d
 ( 16.0, 16.0) ( 18.0, -9.0) (  1.0, -4.0)        : sub-diagonal e
 ( 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_zptrfs (f07jvc) 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)

Backward errors (machine-dependent)
    0.0e+00     0.0e+00

Estimated forward error bounds (machine-dependent)
    9.0e-12     6.1e-12
```