NAG Library Function Document

nag_zptsvx (f07jpc)

1 Purpose

nag_zptsvx (f07jpc) uses the factorization

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

to compute 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. Error bounds on the solution and a condition estimate are also provided.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
```

```
void nag_zptsvx (Nag_OrderType order, Nag_FactoredFormType fact, Integer n,
     Integer nrhs, const double d[], const Complex e[], double df[],
     Complex ef[], const Complex b[], Integer pdb, Complex x[], Integer pdx,
     double *rcond, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zptsvx (f07jpc) performs the following steps:

1. If **fact** $=$ Nag_NotFactored, the matrix A is factorized as $A = LDL^{\mathrm{H}}$, where L is a unit lower bidiagonal matrix and D is diagonal. The factorization can also be regarded as having the form $A = U^{\mathrm{H}}DU$.
2. If the leading i by i principal minor is not positive definite, then the function returns with **fail**.**errnum** $= i$ and **fail**.**code** $=$ 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***, **fail**.**code** $=$ 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.
3. The system of equations is solved for X using the factored form of A.
4. Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for it.

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

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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: **fact** – Nag_FactoredFormType *Input*

On entry: specifies whether or not the factorized form of the matrix A has been supplied.

fact = Nag_Factored
: **df** and **ef** contain the factorized form of the matrix A. **df** and **ef** will not be modified.

fact = Nag_NotFactored
: The matrix A will be copied to **df** and **ef** and factorized.

Constraint: **fact** = Nag_Factored or Nag_NotFactored.

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: the n diagonal elements of the tridiagonal matrix 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: the $(n - 1)$ subdiagonal elements of the tridiagonal matrix A.

7: **df**[*dim*] – double *Input/Output*

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

On entry: if **fact** = Nag_Factored, **df** must contain the n diagonal elements of the diagonal matrix D from the LDL^{H} factorization of A.

On exit: if **fact** = Nag_NotFactored, **df** contains the n diagonal elements of the diagonal matrix D from the LDL^{H} factorization of A.

8: **ef**[*dim*] – Complex *Input/Output*

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

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

On exit: if **fact** = Nag_NotFactored, **ef** contains the $(n - 1)$ subdiagonal elements of the unit bidiagonal factor 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, **pdb** × **nrhs**) when **order** = Nag_ColMajor;
max(1, **n** × **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 right-hand side matrix 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** ≥ max(1, **n**);
if **order** = Nag_RowMajor, **pdb** ≥ max(1, **nrhs**).

11: **x**[*dim*] – Complex *Output*

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

max(1, **pdx** × **nrhs**) when **order** = Nag_ColMajor;
max(1, **n** × **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 exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, the n by r 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** ≥ max(1, **n**);
if **order** = Nag_RowMajor, **pdx** ≥ max(1, **nrhs**).

13: **rcond** – double * *Output*

On exit: the reciprocal condition number of the matrix A. If **rcond** is less than the ***machine precision*** (in particular, if **rcond** = 0.0), the matrix is singular to working precision. This condition is indicated by a return code of **fail.code** = NE_SINGULAR_WP.

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

On exit: the forward error bound for each solution vector $\hat{x}_j$ (the jth column of the solution matrix X). If x_j is the true solution corresponding to $\hat{x}_j$, **ferr**$[j-1]$ is an estimated upper bound for the magnitude of the largest element in $(\hat{x}_j - x_j)$ divided by the magnitude of the largest element in $\hat{x}_j$.

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

On exit: the component-wise relative backward error of each 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).

16: **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 $\langle value \rangle$ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{nrhs} = \langle value \rangle$.
Constraint: $\mathbf{nrhs} \geq 0$.

On entry, $\mathbf{pdb} = \langle value \rangle$.
Constraint: $\mathbf{pdb} > 0$.

On entry, $\mathbf{pdx} = \langle value \rangle$.
Constraint: $\mathbf{pdx} > 0$.

NE_INT_2

On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

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

On entry, $\mathbf{pdx} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdx} \geq \max(1, \mathbf{n})$.

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

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.

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. $\mathbf{rcond} = 0.0$ is returned.

NE_SINGULAR_WP

D is nonsingular, but **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 **rcond** would suggest.

7 Accuracy

For each right-hand side vector b, the computed solution $\hat{x}$ is the exact solution of a perturbed system of equations $(A + E)\hat{x} = b$, where

$$|E| \leq c(n)\epsilon |R^{\mathrm{T}}||R|, \text{ where } R = D^{\frac{1}{2}}U,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***. See Section 10.1 of Higham (2002) for further details.

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

$$\frac{\|x-\hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq w_c \, \mathrm{cond}(A,\hat{x},b)$$

where $\mathrm{cond}(A,\hat{x},b) = \left\| |A^{-1}|(|A||\hat{x}|+|b|) \right\|_\infty / \|\hat{x}\|_\infty \leq \mathrm{cond}(A) = \left\| |A^{-1}||A| \right\|_\infty \leq \kappa_\infty(A)$. If $\hat{x}$ is the jth column of X, then w_c is returned in **berr**$[j-1]$ and a bound on $\|x-\hat{x}\|_\infty/\|\hat{x}\|_\infty$ is returned in **ferr**$[j-1]$. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_zptsvx (f07jpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zptsvx (f07jpc) 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 number of floating-point operations required for the factorization, and for the estimation of the condition number of A is proportional to n. The number of floating-point operations required for the solution of the equations, and for the estimation of the forward and backward error is proportional to nr, where r is the number of right-hand sides.

The condition estimation is based upon Equation (15.11) of Higham (2002). For further details of the error estimation, see Section 4.4 of Anderson *et al.* (1999).

The real analogue of this function is nag_dptsvx (f07jbc).

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

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

Error estimates for the solutions and an estimate of the reciprocal of the condition number of A are also output.

10.1 Program Text

```
/* nag_zptsvx (f07jpc) 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 */
  double        rcond;
  Integer       exit_status = 0, i, j, n, nrhs, pdb, pdx;

  /* Arrays */
  Complex       *b = 0, *e = 0, *ef = 0, *x = 0;
  double        *berr = 0, *d = 0, *df = 0, *ferr = 0;

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

#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_zptsvx (f07jpc) 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)) ||
      !(x    = NAG_ALLOC(n * nrhs, Complex)) ||
      !(e    = NAG_ALLOC(n-1, Complex)) ||
      !(ef   = NAG_ALLOC(n-1, Complex)) ||
      !(d    = NAG_ALLOC(n, double)) ||
      !(df   = NAG_ALLOC(n, double)) ||
      !(berr = NAG_ALLOC(nrhs, 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 and */
  /* the right hand side b 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]");

  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]");

  /* Solve the equations AX = B for X using nag_zptsvx (f07jpc). */
  nag_zptsvx(order, Nag_NotFactored, n, nrhs, d, e, df, ef, b, pdb, x, pdx,
             &rcond, ferr, berr, &fail);
  if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
    {
      printf("Error from nag_zptsvx (f07jpc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print solution, error bounds and condition number 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;
    }

  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":" ");

  printf("\nEstimate of reciprocal condition number\n%11.1e\n", rcond);
  if (fail.code == NE_SINGULAR)
    {
      printf("Error from nag_zptsvx (f07jpc).\n%s\n", fail.message);
      exit_status = 1;
    }
 END:
  NAG_FREE(b);
  NAG_FREE(x);
  NAG_FREE(e);
  NAG_FREE(ef);
  NAG_FREE(d);
  NAG_FREE(df);
  NAG_FREE(berr);
  NAG_FREE(ferr);

  return exit_status;
}
#undef B
```

10.2 Program Data

```
nag_zptsvx (f07jpc) 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_zptsvx (f07jpc) 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
Estimate of reciprocal condition number
    1.1e-04
```