NAG Library Function Document

nag_zgtsvx (f07cpc)

1 Purpose

nag_zgtsvx (f07cpc) uses the LU factorization to compute the solution to a complex system of linear equations

$$AX = B, \quad A^{\mathrm{T}}X = B \quad \text{or} \quad A^{\mathrm{H}}X = B,$$

where A is a tridiagonal matrix of order n 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_zgtsvx (Nag_OrderType order, Nag_FactoredFormType fact,
     Nag_TransType trans, Integer n, Integer nrhs, const Complex dl[],
     const Complex d[], const Complex du[], Complex dlf[], Complex df[],
     Complex duf[], Complex du2[], Integer ipiv[], const Complex b[],
     Integer pdb, Complex x[], Integer pdx, double *rcond, double ferr[],
     double berr[], NagError *fail)
```

3 Description

nag_zgtsvx (f07cpc) performs the following steps:

1. If **fact** $=$ Nag_NotFactored, the LU decomposition is used to factor the matrix A as $A = LU$, where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.
2. If some $u_{ii} = 0$, so that U is exactly singular, then the function returns with **fail**.**errnum** $= i$. 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
dlf, **df**, **duf**, **du2** and **ipiv** contain the factorized form of the matrix A. **dlf**, **df**, **duf**, **du2** and **ipiv** will not be modified.

fact = Nag_NotFactored
The matrix A will be copied to **dlf**, **df** and **duf** and factorized.

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

3: **trans** – Nag_TransType *Input*

On entry: specifies the form of the system of equations.

trans = Nag_NoTrans
$AX = B$ (No transpose).

trans = Nag_Trans
$A^{\mathrm{T}}X = B$ (Transpose).

trans = Nag_ConjTrans
$A^{\mathrm{H}}X = B$ (Conjugate transpose).

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: **n** – Integer *Input*

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

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

5: **nrhs** – Integer *Input*

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

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

6: **dl**$[dim]$ – const Complex *Input*

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

On entry: the $(n - 1)$ subdiagonal elements of A.

7: **d**$[dim]$ – const Complex *Input*

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

On entry: the n diagonal elements of A.

8: **du**$[dim]$ – const Complex *Input*

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

On entry: the $(n - 1)$ superdiagonal elements of A.

9: **dlf**$[dim]$ – Complex *Input/Output*

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

On entry: if **fact** = Nag_Factored, **dlf** contains the $(n - 1)$ multipliers that define the matrix L from the LU factorization of A.

On exit: if **fact** = Nag_NotFactored, **dlf** contains the $(n - 1)$ multipliers that define the matrix L from the LU factorization of A.

10: **df**$[dim]$ – Complex *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** contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

On exit: if **fact** = Nag_NotFactored, **df** contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

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

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

On entry: if **fact** = Nag_Factored, **duf** contains the $(n - 1)$ elements of the first superdiagonal of U.

On exit: if **fact** = Nag_NotFactored, **duf** contains the $(n - 1)$ elements of the first superdiagonal of U.

12: **du2**$[dim]$ – Complex *Input/Output*

Note: the dimension, *dim*, of the array **du2** must be at least $\max(1, \mathbf{n} - 2)$.

On entry: if **fact** = Nag_Factored, **du2** contains the $(n - 2)$ elements of the second superdiagonal of U.

On exit: if **fact** = Nag_NotFactored, **du2** contains the $(n - 2)$ elements of the second superdiagonal of U.

13: **ipiv**$[dim]$ – Integer *Input/Output*

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

On entry: if **fact** = Nag_Factored, **ipiv** contains the pivot indices from the LU factorization of A.

On exit: if **fact** = Nag_NotFactored, **ipiv** contains the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row **ipiv**$[i - 1]$. **ipiv**$[i - 1]$ will always be either i or $i + 1$; **ipiv**$[i - 1] = i$ indicates a row interchange was not required.

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

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

15: **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** $\geq \max(1, \mathbf{n})$;
if **order** $=$ Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

16: **x**[*dim*] – Complex *Output*

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

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

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

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

On exit: if **fail.code** $=$ NE_NOERROR or NE_SINGULAR_WP, the n by r solution matrix X.

17: **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** $\geq \max(1, \mathbf{n})$;
if **order** $=$ Nag_RowMajor, **pdx** $\geq \max(1, \mathbf{nrhs})$.

18: **rcond** – double * *Output*

On exit: the estimate of the reciprocal condition number of the matrix A. If **rcond** $= 0.0$, the matrix may be exactly singular. This condition is indicated by **fail.code** $=$ NE_SINGULAR. Otherwise, if **rcond** is less than the ***machine precision***, the matrix is singular to working precision. This condition is indicated by **fail.code** $=$ NE_SINGULAR_WP.

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

On exit: if **fail.code** $=$ NE_NOERROR or NE_SINGULAR_WP, an estimate of the forward error bound for each computed solution vector, such that $\|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq$ **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 as reliable as the estimate for **rcond**, and is almost always a slight overestimate of the true error.

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

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).

21: **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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument $\langle$*value*$\rangle$ had an illegal value.

NE_INT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **nrhs** $= \langle value \rangle$.
Constraint: **nrhs** ≥ 0.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

On entry, **pdx** $= \langle value \rangle$.
Constraint: **pdx** > 0.

NE_INT_2

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

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

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

On entry, **pdx** $= \langle value \rangle$ and **nrhs** $= \langle value \rangle$.
Constraint: **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.

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_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

Element $\langle value \rangle$ of the diagonal is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. **rcond** $= 0.0$ is returned.

Element $\langle value \rangle$ of the diagonal is exactly zero. The factorization has not been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. **rcond** $= 0.0$ is returned.

NE_SINGULAR_WP

U 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

$$|E| \leq c(n)\epsilon|L||U|,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***. See Section 9.3 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

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

where $\operatorname{cond}(A, \hat{x}, b) = \left\| |A^{-1}| (|A||\hat{x}| + |b|) \right\|_\infty / \|\hat{x}\|_\infty \leq \operatorname{cond}(A) = \left\| |A^{-1}||A| \right\|_\infty \leq \kappa_\infty(A)$. 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_zgtsvx (f07cpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgtsvx (f07cpc) 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

The total number of floating-point operations required to solve the equations $AX = B$ is proportional to nr.

The condition number estimation typically requires between four and five solves and never more than eleven solves, following the factorization. The solution is then refined, and the errors estimated, using iterative refinement.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

The real analogue of this function is nag_dgtsvx (f07cbc).

10 Example

This example solves the equations

$$AX = B,$$

where A is the tridiagonal matrix

$$A = \begin{pmatrix} -1.3 + 1.3i & 2.0 - 1.0i & 0 & 0 & 0 \\ 1.0 - 2.0i & -1.3 + 1.3i & 2.0 + 1.0i & 0 & 0 \\ 0 & 1.0 + 1.0i & -1.3 + 3.3i & -1.0 + 1.0i & 0 \\ 0 & 0 & 2.0 - 3.0i & -0.3 + 4.3i & 1.0 - 1.0i \\ 0 & 0 & 0 & 1.0 + 1.0i & -3.3 + 1.3i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 2.4 - 5.0i & 2.7 + 6.9i \\ 3.4 + 18.2i & -6.9 - 5.3i \\ -14.7 + 9.7i & -6.0 - 0.6i \\ 31.9 - 7.7i & -3.9 + 9.3i \\ -1.0 + 1.6i & -3.0 + 12.2i \end{pmatrix}.$$

Estimates for the backward errors, forward errors and condition number are also output.

10.1 Program Text

```
/* nag_zgtsvx (f07cpc) Example Program.
 *
 * Copyright 2014 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, *d = 0, *df = 0, *dl = 0, *dlf = 0, *du = 0, *du2 = 0;
  Complex        *duf = 0, *x = 0;
  double         *berr = 0, *ferr = 0;
  Integer        *ipiv = 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_zgtsvx (f07cpc) Example Program Results\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &nrhs);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &nrhs);
#endif
  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)) ||
      !(d    = NAG_ALLOC(n, Complex)) ||
      !(df   = NAG_ALLOC(n, Complex)) ||
      !(dl   = NAG_ALLOC(n-1, Complex)) ||
      !(dlf  = NAG_ALLOC(n-1, Complex)) ||
      !(du   = NAG_ALLOC(n-1, Complex)) ||
      !(du2  = NAG_ALLOC(n-2, Complex)) ||
      !(duf  = NAG_ALLOC(n-1, Complex)) ||
      !(x    = NAG_ALLOC(n * nrhs, Complex)) ||
      !(berr = NAG_ALLOC(nrhs, double)) ||
```

```
      !(ferr = NAG_ALLOC(nrhs, double)) ||
      !(ipiv = NAG_ALLOC(n, Integer)))
    {
      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 tridiagonal matrix A from data file */
#ifdef _WIN32
  for (i = 0; i < n - 1; ++i) scanf_s(" ( %lf , %lf )", &du[i].re, &du[i].im);
#else
  for (i = 0; i < n - 1; ++i) scanf(" ( %lf , %lf )", &du[i].re, &du[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n; ++i) scanf_s(" ( %lf , %lf )", &d[i].re, &d[i].im);
#else
  for (i = 0; i < n; ++i) scanf(" ( %lf , %lf )", &d[i].re, &d[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n - 1; ++i) scanf_s(" ( %lf , %lf )", &dl[i].re, &dl[i].im);
#else
  for (i = 0; i < n - 1; ++i) scanf(" ( %lf , %lf )", &dl[i].re, &dl[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read the right hand matrix B */
  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("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Solve the equations AX = B using nag_zgtsvx (f07cpc). */
  nag_zgtsvx(order, Nag_NotFactored, Nag_NoTrans, n, nrhs, dl, d, du, dlf, df,
             duf, du2, ipiv, b, pdb, x, pdx, &rcond, ferr, berr,
             &fail);
  if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
    {
      printf("Error from nag_zgtsvx (f07cpc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
```

```
    }
  /* Print 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 solution, error bounds and condition number */
  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("\n\nEstimate of reciprocal condition number\n%11.1e\n", rcond);
  if (fail.code == NE_SINGULAR)
    printf("Error from nag_zgtsvx (f07cpc).\n%s\n", fail.message);
 END:
  NAG_FREE(b);
  NAG_FREE(d);
  NAG_FREE(df);
  NAG_FREE(dl);
  NAG_FREE(dlf);
  NAG_FREE(du);
  NAG_FREE(du2);
  NAG_FREE(duf);
  NAG_FREE(x);
  NAG_FREE(berr);
  NAG_FREE(ferr);
  NAG_FREE(ipiv);

  return exit_status;
}

#undef B
```

10.2 Program Data

```
nag_zgtsvx (f07cpc) Example Program Data
   5            2                                                   : n, nrhs
             (  2.0, -1.0) (  2.0,  1.0) ( -1.0,  1.0) (  1.0, -1.0) : du
 ( -1.3,  1.3) ( -1.3,  1.3) ( -1.3,  3.3) ( -0.3,  4.3) ( -3.3,  1.3) : d
 (  1.0, -2.0) (  1.0 , 1.0) (  2.0, -3.0) (  1.0,  1.0)             : dl
 (  2.4, -5.0) (  2.7,  6.9)
 (  3.4, 18.2) ( -6.9, -5.3)
 (-14.7,  9.7) ( -6.0, -0.6)
 ( 31.9, -7.7) ( -3.9,  9.3)
 ( -1.0,  1.6) ( -3.0, 12.2)                                         : B
```

10.3 Program Results

```
nag_zgtsvx (f07cpc) Example Program Results

 Solution(s)
                     1                   2
 1  ( 1.0000, 1.0000)  ( 2.0000,-1.0000)
 2  ( 3.0000,-1.0000)  ( 1.0000, 2.0000)
 3  ( 4.0000, 5.0000)  (-1.0000, 1.0000)
 4  (-1.0000,-2.0000)  ( 2.0000, 1.0000)
 5  ( 1.0000,-1.0000)  ( 2.0000,-2.0000)

Backward errors (machine-dependent)
    3.7e-17     6.7e-17

Estimated forward error bounds (machine-dependent)
    5.4e-14     7.3e-14

Estimate of reciprocal condition number
    5.4e-03
```