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

\[ AX = B, \quad A^TX = B \quad \text{or} \quad A^H X = B, \]

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

```c
#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 \( \text{fact} = \text{Nag\_NotFactored} \), 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 \( \	ext{fail.errnum} = i \). 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, \( \text{fail.code} = \text{NE\_SINGULAR\_WP} \) 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


5 Arguments

1: \texttt{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry}: the \texttt{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 \texttt{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint}: \texttt{order} = Nag_RowMajor or Nag_ColMajor.

2: \texttt{fact} – Nag_FactoredFormType \hspace{1cm} \textit{Input}

\textit{On entry}: specifies whether or not the factorized form of the matrix \( A \) has been supplied.

\texttt{fact} = Nag_Factored

\( dlf, df, duf, du2 \) and \texttt{ipiv} contain the factorized form of the matrix \( A \). \( dlf, df, duf, du2 \) and \texttt{ipiv} will not be modified.

\texttt{fact} = Nag_NotFactored

The matrix \( A \) will be copied to \( dlf, df \) and \( duf \) and factorized.

\textit{Constraint}: \texttt{fact} = Nag_Factored or Nag_NotFactored.

3: \texttt{trans} – Nag_TransType \hspace{1cm} \textit{Input}

\textit{On entry}: specifies the form of the system of equations.

\texttt{trans} = Nag_NoTrans

\( AX = B \) (No transpose).

\texttt{trans} = Nag_Trans

\( A^T X = B \) (Transpose).

\texttt{trans} = Nag_ConjTrans

\( A^H X = B \) (Conjugate transpose).

\textit{Constraint}: \texttt{trans} = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: \texttt{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \( n \), the order of the matrix \( A \).

\textit{Constraint}: \texttt{n} \geq 0.

5: \texttt{nrhs} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \( r \), the number of right-hand sides, i.e., the number of columns of the matrix \( B \).

\textit{Constraint}: \texttt{nrhs} \geq 0.

6: \texttt{dl[dim]} – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry}: the \((n-1)\) subdiagonal elements of \( A \).

7: \texttt{d[dim]} – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry}: the \( n \) diagonal elements of \( A \).

8: \texttt{du[dim]} – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry}: the \((n-1)\) superdiagonal elements of \( A \).
9: \( \text{dlf[dim]} \) – Complex \hspace{1cm} \text{Input/Output}

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

On entry: if \( \text{fact} = \text{Nag}_\text{Factored} \), \( \text{dlf} \) contains the \( (n - 1) \) multipliers that define the matrix \( L \) from the \( LU \) factorization of \( A \).

On exit: if \( \text{fact} = \text{Nag}_\text{NotFactored} \), \( \text{dlf} \) contains the \( (n - 1) \) multipliers that define the matrix \( L \) from the \( LU \) factorization of \( A \).

10: \( \text{df[dim]} \) – Complex \hspace{1cm} \text{Input/Output}

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

On entry: if \( \text{fact} = \text{Nag}_\text{Factored} \), \( \text{df} \) contains the \( n \) diagonal elements of the upper triangular matrix \( U \) from the \( LU \) factorization of \( A \).

On exit: if \( \text{fact} = \text{Nag}_\text{NotFactored} \), \( \text{df} \) contains the \( n \) diagonal elements of the upper triangular matrix \( U \) from the \( LU \) factorization of \( A \).

11: \( \text{duf[dim]} \) – Complex \hspace{1cm} \text{Input/Output}

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

On entry: if \( \text{fact} = \text{Nag}_\text{Factored} \), \( \text{duf} \) contains the \( (n - 1) \) elements of the first superdiagonal of \( U \).

On exit: if \( \text{fact} = \text{Nag}_\text{NotFactored} \), \( \text{duf} \) contains the \( (n - 1) \) elements of the first superdiagonal of \( U \).

12: \( \text{du2[dim]} \) – Complex \hspace{1cm} \text{Input/Output}

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

On entry: if \( \text{fact} = \text{Nag}_\text{Factored} \), \( \text{du2} \) contains the \( (n - 2) \) elements of the second superdiagonal of \( U \).

On exit: if \( \text{fact} = \text{Nag}_\text{NotFactored} \), \( \text{du2} \) contains the \( (n - 2) \) elements of the second superdiagonal of \( U \).

13: \( \text{ipiv[dim]} \) – Integer \hspace{1cm} \text{Input/Output}

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

On entry: if \( \text{fact} = \text{Nag}_\text{Factored} \), \( \text{ipiv} \) contains the pivot indices from the \( LU \) factorization of \( A \).

On exit: if \( \text{fact} = \text{Nag}_\text{NotFactored} \), \( \text{ipiv} \) contains the pivot indices from the \( LU \) factorization of \( A \); row \( i \) of the matrix was interchanged with row \( \text{ipiv}[i - 1] \). \( \text{ipiv}[i - 1] \) will always be either \( i \) or \( i + 1 \); \( \text{ipiv}[i - 1] = i \) indicates a row interchange was not required.

14: \( \text{b[dim]} \) – const Complex \hspace{1cm} \text{Input}

Note: the dimension, \( dim \), of the array \( \text{b} \) must be at least

\[ \max(1, \text{pdb} \times \text{nrhs}) \text{ when } \text{order} = \text{Nag}_\text{ColMajor}; \]
\[ \max(1, n \times \text{pdb}) \text{ when } \text{order} = \text{Nag}_\text{RowMajor}. \]

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

\[ \text{b}[(j - 1) \times \text{pdb} + i - 1] \text{ when } \text{order} = \text{Nag}_\text{ColMajor}; \]
\[ \text{b}[(i - 1) \times \text{pdb} + j - 1] \text{ when } \text{order} = \text{Nag}_\text{RowMajor}. \]

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

15: \( \text{pdb} \) – Integer \hspace{1cm} \text{Input}

On entry: the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( \text{b} \).
Constraints:
if order = Nag_ColMajor, pdb ≥ max(1, n);
if order = Nag_RowMajor, pdb ≥ max(1, nrhs).

16: x[dim] – Complex
Output
Note: the dimension, dim, of the array x 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 X is stored in
x[(j - 1) × pdb + i - 1] when order = Nag_ColMajor;
x[(i - 1) × pdb + 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 ≥ max(1, n);
if order = Nag_RowMajor, pdx ≥ max(1, 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 ||x̂j - xj||∞/||xj||∞ ≤ ferr[j - 1] where x̂j is
the jth column of the computed solution returned in the array x and xj 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 x̂j (i.e., the smallest relative change
in any element of A or B that makes x̂j 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 <value> had an illegal value.
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( \text{nrhs} = \langle \text{value} \rangle \).
Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

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

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

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

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

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.

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.

Element \( \langle \text{value} \rangle \) 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. \( \text{rcond} = 0.0 \) is returned.

Element \( \langle \text{value} \rangle \) 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. \( \text{rcond} = 0.0 \) is returned.

\( U \) is nonsingular, but \( \text{rcond} \) is less than \textit{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 \( \text{rcond} \) would suggest.

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
\[
|E| \leq c(n)e|L||U|,
\]
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

\[
\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq w_c \text{cond}(A, \hat{x}, b)
\]

where \( \text{cond}(A, \hat{x}, b) = \|A^{-1}\|(A\|\hat{x}\| + |b|)\|/\|\hat{x}\|_\infty \leq \text{cond}(A) = \|A^{-1}\|A\|_\infty \leq \kappa_\infty(A) \). If \( \hat{x} \) is the \( j \)th column of \( X \), then \( w_c \) is returned in \( \text{berr}[j-1] \) and a bound on \( \|x - \hat{x}\|_\infty/\|\hat{x}\|_\infty \) is returned in \( \text{ferr}[j-1] \). 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

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

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

/* zag_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;
  NagOrderType order;

#ifndef NAG_COLUMN_MAJOR
#define B(I, J) b[(J-1)*pdb + I-1]
#else
#define B(I, J) b[(I-1)*pdb+J-1]
#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

if (n < 0 || nrhs < 0)
{
  printf("Invalid n or nrhs\n");
  exit_status = 1;
  goto END;
}

/* Allocate memory */
if (!b || !d || !df || !dl || !dlf || !du || !du2 || !duf || !x || !berr || !ferr)

Mark 25
!(ferr = NAG_ALLOC(nrhs, double)) ||
!(ipiv = NAG_ALLOC(n, Integer)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
#endif 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;
go to 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
n = 5, nrhs = 2
( 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)
( 31.9, -7.7)
( -3.9, 9.3)
( -1.0, 1.6) ( -3.0, 12.2) : B

Mark 25
10.3 Program Results

nag_zgtsvx (f07cpc) Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1.0000, 1.0000)</td>
<td>(2.0000, -1.0000)</td>
</tr>
<tr>
<td>2</td>
<td>(3.0000, -1.0000)</td>
<td>(1.0000, 2.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(4.0000, 5.0000)</td>
<td>(-1.0000, 1.0000)</td>
</tr>
<tr>
<td>4</td>
<td>(-1.0000, -2.0000)</td>
<td>(2.0000, 1.0000)</td>
</tr>
<tr>
<td>5</td>
<td>(1.0000, -1.0000)</td>
<td>(2.0000, -2.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

6.7e-17

Estimated forward error bounds (machine-dependent)

7.3e-14

Estimated of reciprocal condition number

5.4e-03