NAG C Library Function Document

nag_ztrrfs (f07tvc)

1 Purpose

nag_ztrrfs (f07tvc) returns error bounds for the solution of a complex triangular system of linear equations with multiple right-hand sides, $AX = B$, $A^TX = B$ or $AHX = B$.

2 Specification

```c
void nag_ztrrfs (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
                Nag_DiagType diag, Integer n, Integer nrhs,
                const Complex a[], Integer pda,
                const Complex b[], Integer pdb, const Complex x[], Integer pdx,
                double ferr[],
                double berr[], NagError *fail)
```

3 Description

nag_ztrrfs (f07tvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex triangular system of linear equations with multiple right-hand sides $AX = B$, $A^TX = B$ or $AHX = B$. The function handles each right-hand side vector (stored as a column of the matrix $B$) independently, so we describe the function of nag_ztrrfs (f07tvc) in terms of a single right-hand side $b$ and solution $x$.

Given a computed solution $x$, the function computes the component-wise backward error $\beta$. This is the size of the smallest relative perturbation in each element of $A$ and $b$ such that $x$ is the exact solution of a perturbed system

$$(A + \delta A)x = b + \delta b$$

$$|\delta a_{ij}| \leq \beta|a_{ij}| \text{ and } |\delta b_i| \leq \beta|b_i|.$$  

Then the function estimates a bound for the component-wise forward error in the computed solution, defined by:

$$\max_i |x_i - \hat{x}_i| / \max_i |x_i|$$

where $\hat{x}$ is the true solution.

For details of the method, see the f07 Chapter Introduction.

4 References


5 Parameters

1: order – Nag_OrderType

   Input

   On entry: the order parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: uplo – Nag_UploType

   Input

   On entry: indicates whether $A$ is upper or lower triangular as follows:
if uplo = Nag_Upper, A is upper triangular;
if uplo = Nag_Lower, A is lower triangular.

Constraint: uplo = Nag_Upper or Nag_Lower.

3: trans – Nag_TransType

On entry: indicates the form of the equations as follows:
if trans = Nag_NoTrans, the equations are of the form AX = B;
if trans = Nag_Trans, the equations are of the form AT X = B;
if trans = Nag_ConjTrans, the equations are of the form AHX = B.

Constraint: trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: diag – Nag_DiagType

On entry: indicates whether A is a non-unit or unit triangular matrix as follows:
if diag = Nag_NonUnitDiag, A is a non-unit triangular matrix;
if diag = Nag_UnitDiag, A is a unit triangular matrix; the diagonal elements are not
referenced and are assumed to be 1.

Constraint: diag = Nag_NonUnitDiag or Nag_UnitDiag.

5: n – Integer

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

Constraint: n ≥ 0.

6: nrhs – Integer

On entry: r, the number of right-hand sides.

Constraint: nrhs ≥ 0.

7: a[dim] – const Complex

Note: the dimension, dim, of the array a must be at least max(1, pda × n).

On entry: the n by n triangular matrix A. If uplo = Nag_Upper, A is upper triangular and the
elements of the array below the diagonal are not referenced; if uplo = Nag_Lower, A is lower
triangular and the elements of the array above the diagonal are not referenced. If
diag = Nag_UnitDiag, the diagonal elements of A are not referenced, but are assumed to be 1.

8: pda – Integer

On entry: the stride separating row or column elements (depending on the value of order) of the
matrix A in the array a.

Constraint: pda ≥ max(1, n).

9: b[dim] – const Complex

Note: the dimension, dim, of the array b must be at least max(1, pdb × nrhs) when
order = Nag_ColMajor and at least max(1, pdb × n) when order = Nag_RowMajor.

If order = Nag_ColMajor, the (i, j)th element of the matrix B is stored in b[(j - 1) × pdb + i - 1] and
if order = Nag_RowMajor, the (i, j)th element of the matrix B is stored in b[(i - 1) × pdb + j - 1].

On entry: the n by r right-hand side matrix B.
**f07 – Linear Equations (LAPACK)**

10: **pdb** – Integer

   *Input*

   *On entry:* the stride separating matrix row or column elements (depending on the value of *order*) in the array *b*.

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

11: **x[dim]** – const Complex

   *Input*

   *Note:* the dimension, *dim*, of the array *x* must be at least $\max(1, \text{pdx} \times \text{nrhs})$ when *order* = Nag_ColMajor and at least $\max(1, \text{pdx} \times \text{n})$ when *order* = Nag_RowMajor.

   *On entry:* the $n$ by $r$ solution matrix $X$, as returned by nag_ztrtrs (f07tsc).

12: **pdx** – Integer

   *Input*

   *On entry:* the stride separating matrix row or column elements (depending on the value of *order*) in the array *x*.

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

13: **ferr[dim]** – double

   *Output*

   *Note:* the dimension, *dim*, of the array *ferr* must be at least $\max(1, \text{nrhs})$.

   *On exit:* *ferr*[j − 1] contains an estimated error bound for the *j*th solution vector, that is, the *j*th column of $X$, for $j = 1, 2, \ldots, r$.

14: **berr[dim]** – double

   *Output*

   *Note:* the dimension, *dim*, of the array *berr* must be at least $\max(1, \text{nrhs})$.

   *On exit:* *berr*[j − 1] contains the component-wise backward error bound $\beta$ for the *j*th solution vector, that is, the *j*th column of $X$, for $j = 1, 2, \ldots, r$.

15: **fail** – NagError *

   *Output*

   The NAG error parameter (see the Essential Introduction).

### 6 Error Indicators and Warnings

**NE_INT**

On entry, *n* = ⟨value⟩.

Constraint: *n* $\geq 0$.

On entry, *nrhs* = ⟨value⟩.

Constraint: *nrhs* $\geq 0$.

On entry, *pda* = ⟨value⟩.

Constraint: *pda* $> 0$.

On entry, *pdb* = ⟨value⟩.

Constraint: *pdb* $> 0$.

On entry, *pdx* = ⟨value⟩.

Constraint: *pdx* $> 0$.  

[NP3645/7] f07tvc.3
NE_INT_2
On entry, \( pda = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, n) \).

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

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

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

NE_ALLOC_FAIL
Memory allocation failed.

NE_BAD_PARAM
On entry, parameter \( \langle \text{value} \rangle \) had an illegal value.

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 consult NAG for assistance.

7  Accuracy
The bounds returned in \( ferr \) are not rigorous, because they are estimated, not computed exactly; but in
practice they almost always overestimate the actual error.

8  Further Comments
A call to \texttt{nag_ztrrfs} (f07tvc), for each right-hand side, involves solving a number of systems of linear
equations of the form \( Ax = b \) or \( A^Hx = b \); the number is usually 5 and never more than 11. Each solution
involves approximately \( 4n^2 \) real floating-point operations.
The real analogue of this function is \texttt{nag_dtrrfs} (f07thc).

9  Example
To solve the system of equations \( AX = B \) and to compute forward and backward error bounds, where
\[
A = \begin{pmatrix}
  4.78 + 4.56i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\
  2.00 - 0.30i & -4.11 + 1.25i & 0.00 + 0.00i & 0.00 + 0.00i \\
  2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.80i & 0.00 + 0.00i \\
-1.89 + 1.51i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 0.26i
\end{pmatrix}
\]
and
\[
B = \begin{pmatrix}
  -14.78 - 32.36i & -18.02 + 28.46i \\
  2.98 - 2.14i & 14.22 + 15.42i \\
  -20.96 + 17.06i & 5.62 + 35.89i \\
  9.54 + 9.91i & -16.46 - 1.73i
\end{pmatrix}.
\]
9.1 Program Text

/* nag_ztrrfs (f07tvc) Example Program. *
* Copyright 2001 Numerical Algorithms Group. *
* Mark 7, 2001. *
*/

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, j, n, nrhs, berr_len, ferr_len, pda, pdb, pdx;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char uplo[2];
    Complex *a=0, *b=0, *x=0;
    double *berr=0, *ferr=0;

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

    INIT_FAIL(fail);
    Vprintf("f07tvc Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%*[\^\n]");
    Vscanf("%ld%ld%*[\^\n]", &n, &nrhs);
    berr_len = nrhs;
    ferr_len = nrhs;
    #ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
    pdx = n;
    #else
    pda = n;
    pdb = nrhs;
    pdx = nrhs;
    #endif
    /* Allocate memory */
    if ( !(a = NAG_ALLOC(n * n, Complex)) ||
        !(b = NAG_ALLOC(n * nrhs, Complex)) ||
        !(x = NAG_ALLOC(n * nrhs, Complex)) ||
        !(berr = NAG_ALLOC(berr_len, double)) ||
        !(ferr = NAG_ALLOC(ferr_len, double)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read A and B from data file, and copy B to X */
Vscanf(" %1s \n ", uplo);
if (*((unsigned char *)uplo == 'L')
uplo_enum = Nag_Lower;
else if (*((unsigned char *)uplo == 'U')
uplo_enum = Nag_Upper;
else
{
Vprintf("Unrecognised character for Nag_UploType type\n");
exit_status = -1;
goto END;
}
if (uplo_enum == Nag_Upper)
{
for (i = 1; i <= n; ++i)
{
for (j = i; j <= n; ++j)
Vscanf("( %lf , %lf )", &A(i,j).re, &A(i,j).im);
Vscanf("%*[\n ]");
}
else
{
for (i = 1; i <= n; ++i)
{
for (j = 1; j <= i; ++j)
Vscanf("( %lf , %lf )", &A(i,j).re, &A(i,j).im);
Vscanf("%*[\n ]");
}
}
for (i = 1; i <= n; ++i)
{
for (j = 1; j <= nrhs; ++j)
Vscanf("( %lf , %lf )", &B(i,j).re, &B(i,j).im);
Vscanf("%*[\n ]");
for (i = 1; i <= n; ++i)
{
for (j = 1; j <= nrhs; ++j)
{
X(i,j).re = B(i,j).re;
X(i,j).im = B(i,j).im;
}
} /* Compute solution in the array X */
f07tsc(order, uplo_enum, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs, a, pda, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
Vprintf("Error from f07tsc.\n\n", fail.message);
exit_status = 1;
goto END;
}
} /* Compute backward errors and estimated bounds on the */
f07tvc(order, uplo_enum, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs, a, pda, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
Vprintf("Error from f07tvc.\n\n", fail.message);
exit_status = 1;
goto END;
}
/* Print solution */
Vprintf("\n");
x04dbc(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)
{
    Vprintf("Error from x04dbc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
Vprintf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e%s", berr[j-1], j%4==0 ?"\n": "");
Vprintf("\nEstimated forward error bounds "
(machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e%s", ferr[j-1], j%4==0 ?"\n": "");
Vprintf("\n");
END:
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);
if (x) NAG_FREE(x);
if (berr) NAG_FREE(berr);
if (ferr) NAG_FREE(ferr);
return exit_status;
}

9.2 Program Data

f07tvc Example Program Data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>'L'</td>
<td></td>
</tr>
<tr>
<td>( 4.78, 4.56)</td>
<td></td>
</tr>
<tr>
<td>( 2.00,-0.30) (-4.11, 1.25)</td>
<td></td>
</tr>
<tr>
<td>( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.80)</td>
<td></td>
</tr>
<tr>
<td>(-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-0.26)</td>
<td></td>
</tr>
</tbody>
</table>

End of matrix A

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-14.78,-32.36</td>
<td>(-18.02, 28.46)</td>
</tr>
<tr>
<td>-20.96, 17.06</td>
<td>( 5.62, 35.89)</td>
</tr>
<tr>
<td>9.54, 9.91</td>
<td>(-16.46, -1.73)</td>
</tr>
</tbody>
</table>

End of matrix B

9.3 Program Results

f07tvc Example Program Results

Solution(s)

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

Backward errors (machine-dependent)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2e-17</td>
<td>5.5e-17</td>
</tr>
</tbody>
</table>

Estimated forward error bounds (machine-dependent)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9e-14</td>
<td>3.3e-14</td>
</tr>
</tbody>
</table>