NAG C Library Function Document
nag_ztbrfs (f07vvc)

1 Purpose
nag_ztbrfs (f07vvc) returns error bounds for the solution of a complex triangular band system of linear
equations with multiple right-hand sides, \( AX = B \), \( A^T X = B \) or \( A^H X = B \).

2 Specification

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

3 Description
nag_ztbrfs (f07vvc) returns the backward errors and estimated bounds on the forward errors for the
solution of a complex triangular band system of linear equations with multiple right-hand sides \( AX = B \), \( A^T X = B \) or \( A^H X = B \). The function handles each right-hand side vector (stored as a column of the
matrix \( B \)) independently, so we describe the function of nag_ztbrfs (f07vvc) 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

\[
(A + \delta A)x = b + \delta b
\]

\[
|\delta a_{ij}| \leq \beta |a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta |b_i|.
\]

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
Baltimore

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:

```c
Nag_UploType Nag_UploType nag_ztbrfs (Nag_UploType uplo) { ... }
```
if $\text{uplo} = \text{Nag}_\text{Upper}$, $A$ is upper triangular;
if $\text{uplo} = \text{Nag}_\text{Lower}$, $A$ is lower triangular.

Constraint: $\text{uplo} = \text{Nag}_\text{Upper}$ or $\text{Nag}_\text{Lower}$.

3: $\text{trans} - \text{Nag}_\text{TransType}$

Input

On entry: indicates the form of the equations as follows:
if $\text{trans} = \text{Nag}_\text{NoTrans}$, the equations are of the form $AX = B$;
if $\text{trans} = \text{Nag}_\text{Trans}$, the equations are of the form $A^TX = B$;
if $\text{trans} = \text{Nag}_\text{ConjTrans}$, the equations are of the form $A^HX = B$.

Constraint: $\text{trans} = \text{Nag}_\text{NoTrans}$, $\text{Nag}_\text{Trans}$ or $\text{Nag}_\text{ConjTrans}$.

4: $\text{diag} - \text{Nag}_\text{DiagType}$

Input

On entry: indicates whether $A$ is a non-unit or unit triangular matrix as follows:
if $\text{diag} = \text{Nag}_\text{NonUnitDiag}$, $A$ is a non-unit triangular matrix;
if $\text{diag} = \text{Nag}_\text{UnitDiag}$, $A$ is a unit triangular matrix; the diagonal elements are not referenced and are assumed to be 1.

Constraint: $\text{diag} = \text{Nag}_\text{NonUnitDiag}$ or $\text{Nag}_\text{UnitDiag}$.

5: $n - \text{Integer}$

Input

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

Constraint: $n \geq 0$.

6: $\text{kd} - \text{Integer}$

Input

On entry: $k$, the number of super-diagonals of the matrix $A$ if $\text{uplo} = \text{Nag}_\text{Upper}$ or the number of sub-diagonals if $\text{uplo} = \text{Nag}_\text{Lower}$.

Constraint: $\text{kd} \geq 0$.

7: $\text{nrhs} - \text{Integer}$

Input

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

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

8: $\text{ab}[\text{dim}] - \text{const Complex}$

Input

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

On entry: the $n$ by $n$ triangular matrix $A$. This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements $a_{ij}$ depends on the $\text{order}$ and $\text{uplo}$ parameters as follows:

if $\text{order} = \text{Nag}_\text{ColMajor}$ and $\text{uplo} = \text{Nag}_\text{Upper}$,
    $a_{ij}$ is stored in $\text{ab}[k + i - j + (j - 1) \times \text{pdab}]$, for $i = 1, \ldots, n$ and
    $j = i, \ldots, \min(n, i + k)$;

if $\text{order} = \text{Nag}_\text{ColMajor}$ and $\text{uplo} = \text{Nag}_\text{Lower}$,
    $a_{ij}$ is stored in $\text{ab}[i - j + (j - 1) \times \text{pdab}]$, for $i = 1, \ldots, n$ and
    $j = \max(1, i - k), \ldots, i$;

if $\text{order} = \text{Nag}_\text{RowMajor}$ and $\text{uplo} = \text{Nag}_\text{Upper}$,
    $a_{ij}$ is stored in $\text{ab}[j - i + (i - 1) \times \text{pdab}]$, for $i = 1, \ldots, n$ and
    $j = i, \ldots, \min(n, i + k)$;
if order = Nag_RowMajor and uplo = Nag_Lower,
    \[ a_{ij} \text{ is stored in } ab[k + j - i + (i - 1) \times pdab], \]
    for \( i = 1, \ldots, n \) and \( j = \max(1, i - k), \ldots, i \).

9: \[ \text{pdab} \text{ – Integer} \]
    \[ \text{Input} \]
    \[ \text{On entry: the stride separating row or column elements (depending on the value of } \text{order} \text{) of the matrix } A \text{ in the array } ab. \]
    \[ \text{Constraint: } \text{pdab} \geq kd + 1. \]

10: \[ \text{b[dim]} \text{ – const Complex} \]
    \[ \text{Input} \]
    \[ \text{Note: the dimension, } dim, \text{ of the array } b \text{ must be at least max}(1, pdb \times nrhs) \text{ when } \text{order} = \text{Nag_ColMajor} \text{ and at least max}(1, pdb \times n) \text{ when } \text{order} = \text{Nag_RowMajor}. \]
    If \text{order} = \text{Nag_ColMajor}, the \((i, j)\)th element of the matrix \( B \) is stored in \( b[(j - 1) \times pdb + i - 1] \) and if \text{order} = \text{Nag_RowMajor}, the \((i, j)\)th element of the matrix \( B \) is stored in \( b[(i - 1) \times pdb + j - 1] \).
    \[ \text{On entry: the } n \text{ by } r \text{ right-hand side matrix } B. \]

11: \[ \text{pdb} \text{ – Integer} \]
    \[ \text{Input} \]
    \[ \text{On entry: the stride separating matrix row or column elements (depending on the value of } \text{order} \text{) in the array } b. \]
    \[ \text{Constraints:} \]
    \[ \begin{align*}
    \text{if } \text{order} = \text{Nag_ColMajor}, & \text{pdb} \geq \max(1, n); \\
    \text{if } \text{order} = \text{Nag_RowMajor}, & \text{pdb} \geq \max(1, nrhs).
    \end{align*} \]

12: \[ \text{x[dim]} \text{ – const Complex} \]
    \[ \text{Input} \]
    \[ \text{Note: the dimension, } dim, \text{ of the array } x \text{ must be at least max}(1, pdx \times nrhs) \text{ when } \text{order} = \text{Nag_ColMajor} \text{ and at least max}(1, pdx \times n) \text{ when } \text{order} = \text{Nag_RowMajor}. \]
    If \text{order} = \text{Nag_ColMajor}, the \((i, j)\)th element of the matrix \( X \) is stored in \( x[(j - 1) \times pdx + i - 1] \) and if \text{order} = \text{Nag_RowMajor}, the \((i, j)\)th element of the matrix \( X \) is stored in \( x[(i - 1) \times pdx + j - 1] \).
    \[ \text{On entry: the } n \text{ by } r \text{ solution matrix } X, \text{ as returned by } \text{nag_ztbtrs (f07vsc)}. \]

13: \[ \text{pdx} \text{ – Integer} \]
    \[ \text{Input} \]
    \[ \text{On entry: the stride separating matrix row or column elements (depending on the value of } \text{order} \text{) in the array } x. \]
    \[ \text{Constraints:} \]
    \[ \begin{align*}
    \text{if } \text{order} = \text{Nag_ColMajor}, & \text{pdx} \geq \max(1, n); \\
    \text{if } \text{order} = \text{Nag_RowMajor}, & \text{pdx} \geq \max(1, nrhs).
    \end{align*} \]

14: \[ \text{ferr[dim]} \text{ – double} \]
    \[ \text{Output} \]
    \[ \text{Note: the dimension, } dim, \text{ of the array } ferr \text{ must be at least max}(1, nrhs). \]
    \[ \text{On exit: } ferr[j - 1] \text{ contains an estimated error bound for the } j\text{th solution vector, that is, the } j\text{th column of } X, \text{ for } j = 1, 2, \ldots, r. \]

15: \[ \text{berr[dim]} \text{ – double} \]
    \[ \text{Output} \]
    \[ \text{Note: the dimension, } dim, \text{ of the array } berr \text{ must be at least max}(1, nrhs). \]
    \[ \text{On exit: } berr[j - 1] \text{ contains the component-wise backward error bound } \beta \text{ for the } j\text{th solution vector, that is, the } j\text{th column of } X, \text{ for } j = 1, 2, \ldots, r. \]

16: \[ \text{fail} \text{ – NagError *} \]
    \[ \text{Output} \]
    \[ \text{The NAG error parameter (see the Essential Introduction).} \]
6 Error Indicators and Warnings

**NE_INT**

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

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

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

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

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

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

**NE_INT_2**

On entry, \( pdab = \langle \text{value} \rangle, kd = \langle \text{value} \rangle \).
Constraint: \( pdab \geq kd + 1 \).

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

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

**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 nag_ztbrfs (f07vvc), for each right-hand side, involves solving a number of systems of linear equations of the form \( Ax = b \) or \( A^H x = b \); the number is usually 5 and never more than 11. Each solution involves approximately \( 8nk \) real floating-point operations (assuming \( n \gg k \)).

The real analogue of this function is nag_dtbrfs (f07vhc).
9 Example

To solve the system of equations $AX = B$ and to compute forward and backward error bounds, where

$$A = \begin{pmatrix}
-1.94 + 4.43i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\
-3.39 + 3.44i & 4.12 + 4.27i & 0.00 + 0.00i & 0.00 + 0.00i \\
1.62 + 3.68i & -1.84 + 5.53i & 0.43 - 2.66i & 0.00 + 0.00i \\
0.00 + 0.00i & -2.77 - 1.93i & 1.74 - 0.04i & 0.44 + 0.10i \\
\end{pmatrix}$$

and

$$B = \begin{pmatrix}
-8.86 - 3.88i & -24.09 - 5.27i \\
-15.57 - 23.41i & -57.97 + 8.14i \\
-7.63 + 22.78i & 19.09 - 29.51i \\
-14.74 - 2.40i & 19.17 + 23.31i \\
\end{pmatrix}.$$

9.1 Program Text

/* nag_ztrtrs (f07vvc) Example Program. */
* Copyright 2001 Numerical Algorithms Group.
*/
#include <stdio.h>
#include <nag.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, j, k, kd, n, nrhs, pdab, pdb, pdx;
    Integer ferr_len, berr_len;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;

    /* Arrays */
    char uplo[2];
    Complex *ab=0, *b=0, *x=0;
    double *berr=0, *ferr=0;

    INIT_FAIL(fail);
    Vprintf("f07vvc Example Program Results\n\n");

    /* Scalars */
    Integer i, j, k, kd, n, nrhs, pdab, pdb, pdx;
    Integer ferr_len, berr_len;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;

    /* Arrays */
    char uplo[2];
    Complex *ab=0, *b=0, *x=0;
    double *berr=0, *ferr=0;

    #ifdef NAG_COLUMN_MAJOR
    #define AB_UPPER(I,J) ab[(J-1)*pdab + k + I - J - 1]
    #define AB_LOWER(I,J) ab[(J-1)*pdab + I - J]
    #define B(I,J) b[(I-1)*pdb + I - 1]
    #define X(I,J) x[(I-1)*pdx + I - 1]
    order = Nag_ColMajor;
    #else
    #define AB_UPPER(I,J) ab[(I-1)*pdab + k + J - I]
    #define AB_LOWER(I,J) ab[(I-1)*pdab + J - I]
    #define B(I,J) b[(I-1)*pdb + J - I - 1]
    #define X(I,J) x[(I-1)*pdx + J - 1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    Vprintf("f07vvc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%[^\n] ");
    Vscanf("%ld%ld%ld%[^\n]", &n, &kd, &nrhs);
    pdab = kd + 1;
    #ifndef NAG_COLUMN_MAJOR
    pdb = n;
    #endif

pdx = n;
#else
pdx = nrhs;
#endif

ferr_len = nrhs;
berr_len = nrhs;

/* Allocate memory */
if ( !(berr = NAG_ALLOC(berr_len, double)) ||
    !(ferr = NAG_ALLOC(ferr_len, double)) ||
    !(ab = NAG_ALLOC((kd+1) * n, Complex)) ||
    !(b = NAG_ALLOC(n * nrhs, Complex)) ||
    !(x = NAG_ALLOC(n * nrhs, Complex)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read A and B from data file, and copy B to X */
Vscanf(" ' %ls '%*[\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;
    }
k = kd + 1;
if (uplo_enum == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= MIN(i+kd,n); ++j)
        {
            Vscanf(" ( %lf , %lf )", &AB_UPPER(i,j).re,
                     &AB_UPPER(i,j).im);
        }
    }
    Vscanf("%*[\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = MAX(1,i-kd); j <= i; ++j)
        {
            Vscanf(" ( %lf , %lf )", &AB_LOWER(i,j).re,
                     &AB_LOWER(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] ");
/* Copy B to X */
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;
}
}

f07vsc(order, uplo_enum, Nag_NoTrans, Nag_NoUnitDiag, n,
      kd, nrhs, ab, pdab, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07vsc.\n\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07vvc(order, uplo_enum, Nag_NoTrans, Nag_NoUnitDiag, n,
       kd, nrhs, ab, pdab, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07vvc.\n\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print solution */
x04dbc(order, Nag_GeneralMatrix, Nag_NoUnitDiag, 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\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%7==0 ?"\n": " ");
Vprintf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e%s", ferr[j-1], j%7==0 ?"\n": " ");
Vprintf("\n");
}
END:
if (berr) NAG_FREE(berr);
if (ferr) NAG_FREE(ferr);
if (ab) NAG_FREE(ab);
if (b) NAG_FREE(b);
if (x) NAG_FREE(x);
return exit_status;

9.2 Program Data

f07vvc Example Program Data

<table>
<thead>
<tr>
<th>N</th>
<th>KD</th>
<th>NRHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Values of N, KD and NRHS

Value of UPLO

'L'

(-1.94, 4.43)
(-3.39, 3.44) ( 4.12,-4.27)
( 1.62, 3.68) (-1.84, 5.53) ( 0.43,-2.66)
(-2.77,-1.93) ( 1.74,-0.04) ( 0.44, 0.10)
( -8.86,-3.88) (24.09, -5.27)
(-15.57,-23.41) (-57.97, 8.14)
( -7.63, 22.78) ( 19.09,-29.51)
(-14.74, -2.40) ( 19.17, 21.33)

End of matrix A

End of matrix B
9.3 Program Results

f07vvc Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 0.0000, 2.0000)</td>
<td>( 1.0000, 5.0000)</td>
</tr>
<tr>
<td>2</td>
<td>( 1.0000,-3.0000)</td>
<td>(-7.0000,-2.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(-4.0000,-5.0000)</td>
<td>( 3.0000, 4.0000)</td>
</tr>
<tr>
<td>4</td>
<td>( 2.0000,-1.0000)</td>
<td>(-6.0000,-9.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)
8.3e-18  4.2e-17

Estimated forward error bounds (machine-dependent)
1.8e-14  2.2e-14