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

```
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* β. 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

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, 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:

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 *Input*

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 $A^T X = B$;

if **trans** = **Nag_ConjTrans**, the equations are of the form $A^H X = B$.

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

4: **diag** – Nag_DiagType *Input*

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 *Input*

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

Constraint: **n** ≥ 0.

6: **kd** – Integer *Input*

On entry: k, the number of super-diagonals of the matrix A if **uplo** = **Nag_Upper** or the number of sub-diagonals if **uplo** = **Nag_Lower**.

Constraint: **kd** ≥ 0.

7: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0.

8: **ab**[dim] – const Complex *Input*

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

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 **order** and **uplo** parameters as follows:

if **order** = **Nag_ColMajor** and **uplo** = **Nag_Upper**,
a_{ij} is stored in **ab**$[k + i - j + (j - 1) \times \textbf{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k)$;

if **order** = **Nag_ColMajor** and **uplo** = **Nag_Lower**,
a_{ij} is stored in **ab**$[i - j + (j - 1) \times \textbf{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k), \ldots, i$;

if **order** = **Nag_RowMajor** and **uplo** = **Nag_Upper**,
a_{ij} is stored in **ab**$[j - i + (i - 1) \times \textbf{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k)$;

if **order** = **Nag_RowMajor** and **uplo** = **Nag_Lower**,
a_{ij} is stored in **ab**$[k + j - i + (i - 1) \times$ **pdab**$]$, for $i = 1, \dots, n$ and $j = \max(1, i - k), \dots, i$.

9: **pdab** – Integer *Input*

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

Constraint: **pdab** $\geq$ **kd** $+ 1$.

10: **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** and at least $\max(1, \mathbf{pdb} \times \mathbf{n})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix B is stored in **b**$[(j - 1) \times \mathbf{pdb} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix B is stored in **b**$[(i - 1) \times \mathbf{pdb} + j - 1]$.

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

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

12: **x**$[dim]$ – const Complex *Input*

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

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix X is stored in **x**$[(j - 1) \times \mathbf{pdx} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix X is stored in **x**$[(i - 1) \times \mathbf{pdx} + j - 1]$.

On entry: the n by r solution matrix X, as returned by nag_ztbtrs (f07vsc).

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

14: **ferr**$[dim]$ – double *Output*

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

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

15: **berr**$[dim]$ – double *Output*

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

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

16: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

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

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

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

NE_INT_2

On entry, **pdab** = $\langle value \rangle$, **kd** = $\langle value \rangle$.
Constraint: **pdab** $\geq$ **kd** $+ 1$.

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

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

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

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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

$$B = \begin{pmatrix} -8.86 - 3.88i & -24.09 - 5.27i \\ -15.57 - 23.41i & -57.97 + 8.14i \\ -7.63 + 22.78i & 19.09 - 29.51i \\ -14.74 - 2.40i & 19.17 + 21.33i \end{pmatrix}.$$

9.1 Program Text

```
/* nag_ztbrfs (f07vvc) 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, 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[(J-1)*pdb + I - 1]
#define X(I,J) x[(J-1)*pdx + I - 1]
  order = Nag_ColMajor;
#else
#define AB_UPPER(I,J) ab[(I-1)*pdab + J - I]
#define AB_LOWER(I,J) ab[(I-1)*pdab + k + J - I - 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("f07vvc Example Program Results\n\n");

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

```
  pdx = n;
#else
  pdb = nrhs;
  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(" ' %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;
    }
  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_NonUnitDiag, n,
         kd, nrhs, ab, pdab, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07vsc.\n%s\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_NonUnitDiag, n,
         kd, nrhs, ab, pdab, b, pdb, x, pdx, ferr, berr,
         &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07vvc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print solution */

  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%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
  4  2  2                                             :Values of N, KD and NRHS
  'L'                                                    :Value of UPLO
 (-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)  :End of matrix A
 ( -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 B
```

9.3 Program Results

```
f07vvc Example Program Results

 Solution(s)
                     1                   2
 1  ( 0.0000, 2.0000)  ( 1.0000, 5.0000)
 2  ( 1.0000,-3.0000)  (-7.0000,-2.0000)
 3  (-4.0000,-5.0000)  ( 3.0000, 4.0000)
 4  ( 2.0000,-1.0000)  (-6.0000,-9.0000)

Backward errors (machine-dependent)
    8.3e-18     4.2e-17
Estimated forward error bounds (machine-dependent)
    1.8e-14     2.2e-14
```