NAG 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, $AX = B$, $A^{\mathrm{T}}X = B$ or $A^{\mathrm{H}}X = B$.

2   Specification

```
#include <nag.h>
#include <nagf07.h>
```
```
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 $AX = B$, $A^{\mathrm{T}}X = B$ or $A^{\mathrm{H}}X = B$. The function handles each right-hand side vector (stored as a column of the matrix B) independently, so we describe the function of nag_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* 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

$$(A + \delta A)x = b + \delta b$$
$$\left|\delta a_{ij}\right| \leq \beta\left|a_{ij}\right| \quad \text{and} \quad |\delta b_i| \leq \beta|b_i|.$$

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   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 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2:   **uplo** – Nag_UploType   *Input*

On entry: specifies whether A is upper or lower triangular.

uplo = Nag_Upper
    A is upper triangular.

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.

trans = Nag_NoTrans
    The equations are of the form $AX = B$.

trans = Nag_Trans
    The equations are of the form $A^{\mathrm{T}}X = B$.

trans = Nag_ConjTrans
    The equations are of the form $A^{\mathrm{H}}X = B$.

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

4:   **diag** – Nag_DiagType   *Input*

On entry: indicates whether A is a nonunit or unit triangular matrix.

diag = Nag_NonUnitDiag
    A is a nonunit triangular matrix.

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

6:   **nrhs** – Integer   *Input*

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

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

7:   **a**[dim] – const Complex   *Input*

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

On entry: the n by n triangular matrix A.

If **order** = Nag_ColMajor, A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = Nag_RowMajor, A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

If **uplo** = Nag_Upper, the upper triangular part of A must be stored and the elements of the array
below the diagonal are not referenced.

If **uplo** = Nag_Lower, the lower triangular part of A must be stored and the elements of the array
above the diagonal are not referenced.

If **diag** = Nag_UnitDiag, the diagonal elements of A are assumed to be 1, and are not referenced.

8:     **pda** – Integer *Input*

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

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

9:     **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

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

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

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

11:    **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;
  $\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

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

  $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;
  $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$ 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 row or column elements (depending on the value of **order**) in the array **x**.

Constraints:

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

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

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

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

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

15:    **fail** – NagError *   *Input/Output*

  The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

  Dynamic memory allocation failed.
  See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_INT

  On entry, $\mathbf{n} = \langle value \rangle$.
  Constraint: $\mathbf{n} \geq 0$.

  On entry, $\mathbf{nrhs} = \langle value \rangle$.
  Constraint: $\mathbf{nrhs} \geq 0$.

  On entry, $\mathbf{pda} = \langle value \rangle$.
  Constraint: $\mathbf{pda} > 0$.

  On entry, $\mathbf{pdb} = \langle value \rangle$.
  Constraint: $\mathbf{pdb} > 0$.

  On entry, $\mathbf{pdx} = \langle value \rangle$.
  Constraint: $\mathbf{pdx} > 0$.

NE_INT_2

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

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

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

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

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

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 How to Use the NAG Library and its Documentation 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 How to Use the NAG Library and its Documentation for further information.

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 Parallelism and Performance

nag_ztrrfs (f07tvc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_ztrrfs (f07tvc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

A call to nag_ztrrfs (f07tvc), for each right-hand side, involves solving a number of systems of linear equations of the form $Ax = b$ or $A^{\mathrm{H}}x = b$; 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 nag_dtrrfs (f07thc).

10 Example

This example solves the system of equations $AX = B$ and to compute forward and backward error bounds, where

$$
A = \begin{pmatrix}
4.78 + 4.56i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\
2.00 - 0.30i & -4.11 + 1.25i & 0.00 + 0.00i & 0.00 + 0.00i \\
2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.80i & 0.00 + 0.00i \\
-1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 0.26i
\end{pmatrix}
$$

and

$$
B = \begin{pmatrix}
-14.78 - 32.36i & -18.02 + 28.46i \\
2.98 - 2.14i & 14.22 + 15.42i \\
-20.96 + 17.06i & 5.62 + 35.89i \\
9.54 + 9.91i & -16.46 - 1.73i
\end{pmatrix}.
$$

10.1 Program Text

```
/* nag_ztrrfs (f07tvc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  char nag_enum_arg[40];
  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);

  printf("nag_ztrrfs (f07tvc) Example Program Results\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
  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)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read A and B from data file, and copy B to X */
#ifdef _WIN32
  scanf_s(" %39s%*[^\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  if (uplo == Nag_Upper) {
```

```
      for (i = 1; i <= n; ++i) {
        for (j = i; j <= n; ++j)
#ifdef _WIN32
          scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
          scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
      }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }
    else {
      for (i = 1; i <= n; ++i) {
        for (j = 1; j <= i; ++j)
#ifdef _WIN32
          scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
          scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
      }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

    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

    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 */
    /* nag_ztrtrs (f07tsc).
     * Solution of complex triangular system of linear
     * equations, multiple right-hand sides
     */
    nag_ztrtrs(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n,
               nrhs, a, pda, x, pdx, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ztrtrs (f07tsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Compute backward errors and estimated bounds on the */
    /* forward errors */
    /* nag_ztrrfs (f07tvc).
     * Error bounds for solution of complex triangular system of
     * linear equations, multiple right-hand sides
     */
    nag_ztrrfs(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n,
               nrhs, a, pda, b, pdb, x, pdx, ferr, berr, &fail);
```

```
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztrrfs (f07tvc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print solution */
  printf("\n");
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  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;
  }
  printf("\nBackward errors (machine-dependent)\n");
  for (j = 1; j <= nrhs; ++j)

    printf("%11.1e%s", berr[j - 1], j % 4 == 0 ? "\n" : " ");
  printf("\nEstimated forward error bounds " "(machine-dependent)\n");
  for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", ferr[j - 1], j % 4 == 0 ? "\n" : " ");
  printf("\n");
END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(x);
  NAG_FREE(berr);
  NAG_FREE(ferr);

  return exit_status;
}
```

10.2 Program Data

```
nag_ztrrfs (f07tvc) Example Program Data
  4   2                                            :Values of n and nrhs
  Nag_Lower                                        :Value of uplo
 ( 4.78, 4.56)
 ( 2.00,-0.30) (-4.11, 1.25)
 ( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.80)
 (-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-0.26)  :End of matrix A
 (-14.78,-32.36) (-18.02, 28.46)
 (  2.98, -2.14) ( 14.22, 15.42)
 (-20.96, 17.06) (  5.62, 35.89)
 (  9.54,  9.91) (-16.46, -1.73)                          :End of matrix B
```

10.3 Program Results

```
nag_ztrrfs (f07tvc) Example Program Results

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

Backward errors (machine-dependent)
    6.2e-17     2.7e-17
Estimated forward error bounds (machine-dependent)
    2.9e-14     3.2e-14
```