NAG C Library Function Document

nag_ztprfs (f07uvc)

1 Purpose

nag_ztprfs (f07uvc) returns error bounds for the solution of a complex triangular system of linear equations with multiple right-hand sides, $AX = B$, $A^T X = B$ or $A^H X = B$, using packed storage.

2 Specification

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

3 Description

nag_ztprfs (f07uvc) 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 $AX = B$, $A^T X = B$ or $A^H X = B$, using packed storage. The function handles each right-hand side vector (stored as a column of the matrix $B$) independently, so we describe the function of nag_ztprfs (f07uvc) 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


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:** $n \geq 0$.

6: nrhs – Integer

*Input*

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

**Constraint:** $nrhs \geq 0$.

7: ap[dim] – const Complex

*Input*

**Note:** the dimension, dim, of the array ap must be at least max(1, $n \times (n + 1)/2$).

*On entry:* the $n$ by $n$ triangular matrix $A$, packed by rows or columns. 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 $ap[(j - 1) \times j/2 + i - 1]$, for $i \leq j$;
if order = Nag_ColMajor and uplo = Nag_Lower,
       $a_{ij}$ is stored in $ap[(2n - j) \times (j - 1)/2 + i - 1]$, for $i \geq j$;
if order = Nag_RowMajor and uplo = Nag_Upper,
       $a_{ij}$ is stored in $ap[(2n - i) \times (i - 1)/2 + j - 1]$, for $i \leq j$;
if order = Nag_RowMajor and uplo = Nag_Lower,
       $a_{ij}$ is stored in $ap[(i - 1) \times i/2 + j - 1]$, for $i \geq j$.

8: b[dim] – const Complex

*Input*

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

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

*On entry:* the $n$ by $r$ right-hand side matrix $B$. 
9: \( \text{pdb} \) – Integer

Input

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

Constraints:

\[
\begin{align*}
\text{if } & \texttt{order} = \text{Nag\_ColMajor}, & \text{pdb} \geq \max(1, n); \\
\text{if } & \texttt{order} = \text{Nag\_RowMajor}, & \text{pdb} \geq \max(1, \text{nrhs}).
\end{align*}
\]

10: \( x[dim] \) – const Complex

Input

Note: the dimension, \( dim \), of the array \( x \) must be at least \( \max(1, \text{pdx} \times \text{nrhs}) \) when \( \texttt{order} = \text{Nag\_ColMajor} \) and at least \( \max(1, \text{pdx} \times n) \) when \( \texttt{order} = \text{Nag\_RowMajor} \).

If \( \texttt{order} = \text{Nag\_ColMajor} \), the \((i, j)\)th element of the matrix \( X \) is stored in \( x[(j - 1) \times \text{pdx} + i - 1] \) and if \( \texttt{order} = \text{Nag\_RowMajor} \), the \((i, j)\)th element of the matrix \( X \) is stored in \( x[(i - 1) \times \text{pdx} + j - 1] \).

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by \( \text{nag\_ztptrs} (\text{f07usc}) \).

11: \( \text{pdx} \) – Integer

Input

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

Constraints:

\[
\begin{align*}
\text{if } & \texttt{order} = \text{Nag\_ColMajor}, & \text{pdx} \geq \max(1, n); \\
\text{if } & \texttt{order} = \text{Nag\_RowMajor}, & \text{pdx} \geq \max(1, \text{nrhs}).
\end{align*}
\]

12: \( \text{ferr}[dim] \) – double

Output

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

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

13: \( \text{berr}[dim] \) – double

Output

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

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

14: \( \text{fail} \) – NagError *

Output

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, \( \text{nrhs} = \langle \text{value} \rangle \).

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: \( \text{pdx} > 0 \).

**NE\_INT\_2**

On entry, \( \text{pdb} = \langle \text{value} \rangle, \ n = \langle \text{value} \rangle \).

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

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

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

**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 \( \text{ferr} \) 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_ztprfs (f07uvc) involves, for each right-hand side, 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 nag_dtprfs (f07uhc).

### 9 Example

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

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

and

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

using packed storage for \( A \).

### 9.1 Program Text

```c
/* nag_ztprfs (f07uvc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 * *
 */

#include <stdio.h>
#include <nag.h>
```
int main(void)
{
    /* Scalars */
    Integer ap_len, i, j, n, nrhs;
    Integer berr_len, ferr_len, pdb, pdx;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char uplo[2];
    Complex *ap=0, *b=0, *x=0;
    double *berr=0, *ferr=0;

    #ifdef NAG_COLUMN_MAJOR
    #define A_UPPER(I,J) ap[J*(J-1)/2 + I - 1]
    #define A_LOWER(I,J) ap[(2*n-J)*(J-1)/2 + 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_LOWER(I,J) ap[I*(I-1)/2 + J - 1]
    #define A_UPPER(I,J) ap[(2*n-I)*(I-1)/2 + 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("f07uvc Example Program Results\n");
    Vscanf("%*
", &n, &nrhs);

    berr_len = nrhs;
    ferr_len = nrhs;
    ap_len = n * (n + 1)/2;
    #ifdef NAG_COLUMN_MAJOR
    pdb = n;
    pdx = n;
    #else
    pdb = nrhs;
    pdx = nrhs;
    #endif

    if ( !(ap = NAG_ALLOC(ap_len, 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;
    }

    /* Allocate memory */
    if ( !(ap = NAG_ALLOC(ap_len, 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(" %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;
    }
if (uplo_enum == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &A_UPPER(i,j).re, &A_UPPER(i,j).im);
    }
    Vscanf("%*[\n ]");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &A_LOWER(i,j).re, &A_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);
}
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 */
f07usc(order, uplo_enum, Nag_NoTrans, Nag_NonUnitDiag, n,
        nrhs, ap, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07usc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute backward errors and estimated bounds on the */
/* forward errors */
f07uvc(order, uplo_enum, Nag_NoTrans, Nag_NonUnitDiag, n,
        nrhs, ap, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07uvc.\n%s\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 ")
for (j = 1; j <= nrhs; ++j)
  Vprintf("%11.1e", ferr[j-1], j%4==0 ?"\n":"");
Vprintf("\n");
END:
if (ap) NAG_FREE(ap);
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

f07uvc Example Program Data
4 2 :Values of N and NRHS
'L' :Value of UPLO
( 4.78, 4.56) (-4.11, 1.25)
( 2.00, -0.30) ( 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

9.3 Program Results

f07uvc Example Program Results

Solution(s)

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

Backward errors (machine-dependent)
6.2e-17 5.5e-17

Estimated forward error bounds (machine-dependent)
2.9e-14 3.3e-14