NAG 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
#include <nag.h>
#include <nagf07.h>

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:

\[
\max_i |x_i - \tilde{x}_i| / \max_i |x_i|
\]

where \( \tilde{x} \) is the true solution.

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

4 References


5 Arguments

1: \textbf{order} – Nag_OrderType

\textit{Input}

\textit{On entry:} the \textbf{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 \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.
2: uplo – Nag_UploType

Input

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

\( \text{uplo} = \text{Nag}_\text{Upper} \)
- \( A \) is upper triangular.

\( \text{uplo} = \text{Nag}_\text{Lower} \)
- \( A \) is lower triangular.

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

3: trans – Nag_TransType

Input

On entry: indicates the form of the equations.

\( \text{trans} = \text{Nag}_\text{NoTrans} \)
- The equations are of the form \( AX = B \).

\( \text{trans} = \text{Nag}_\text{Trans} \)
- The equations are of the form \( A^T X = B \).

\( \text{trans} = \text{Nag}_\text{ConjTrans} \)
- The equations are of the form \( A^H X = B \).

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

4: diag – Nag_DiagType

Input

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

\( \text{diag} = \text{Nag}_\text{NonUnitDiag} \)
- \( A \) is a nonunit triangular matrix.

\( \text{diag} = \text{Nag}_\text{UnitDiag} \)
- \( A \) 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 – 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 \text{order} and \text{uplo} arguments as follows:

- if \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
  \( A_{ij} \) is stored in \( \text{ap}[(j - 1) \times j/2 + i - 1] \), for \( i \leq j \);

- if \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
  \( A_{ij} \) is stored in \( \text{ap}[(2n - j) \times (j - 1)/2 + i - 1] \), for \( i \geq j \);

- if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
  \( A_{ij} \) is stored in \( \text{ap}[(2n - i) \times (i - 1)/2 + j - 1] \), for \( i \leq j \);

- if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
  \( A_{ij} \) is stored in \( \text{ap}[(i - 1) \times i/2 + j - 1] \), for \( i \geq j \).
If diag = Nag_UnitDiag, the diagonal elements of AP are assumed to be 1, and are not referenced; the same storage scheme is used whether diag = Nag.NonUnitDiag or diag = Nag_UnitDiag.

8: b[dim] – const Complex
   Input
   Note: the dimension, dim, of the array b must be at least
   max(1, pdb x nrhs) when order = Nag.ColMajor;
   max(1, n x pdb) when order = Nag.RowMajor.

   The (i, j)th element of the matrix B is stored in
   b[(j - 1) x pdb + i - 1] when order = Nag.ColMajor;
   b[(i - 1) x pdb + j - 1] when order = Nag.RowMajor.

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

9: 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, pdb ≥ max(1, n);
   if order = Nag.RowMajor, pdb ≥ max(1, nrhs).

10: x[dim] – const Complex
    Input
    Note: the dimension, dim, of the array x must be at least
    max(1, pdx x nrhs) when order = Nag.ColMajor;
    max(1, n x pdx) when order = Nag.RowMajor.

    The (i, j)th element of the matrix X is stored in
    x[(j - 1) x pdx + i - 1] when order = Nag.ColMajor;
    x[(i - 1) x pdx + j - 1] when order = Nag.RowMajor.

    On entry: the n by r solution matrix X, as returned by nag_ztptrs (f07usc).

11: 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, pdx ≥ max(1, n);
    if order = Nag.RowMajor, pdx ≥ max(1, nrhs).

12: ferr[nrhs] – double
    Output
    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, ..., r.

13: berr[nrhs] – double
    Output
    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, ..., r.

14: fail – NagError *
    Input/Output
    The NAG error argument (see Section 3.6 in the Essential Introduction).
6  Error Indicators and Warnings

**NE_ALLOCA_FAIL**
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

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

**NE_INT**
On entry, \(n = \langle\text{value}\rangle\).
Constraint: \(n \geq 0\).
On entry, \(nrhs = \langle\text{value}\rangle\).
Constraint: \(nrhs \geq 0\).
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, \(pdb = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(pdb \geq \text{max}(1, n)\).
On entry, \(pdb = \langle\text{value}\rangle\) and \(nrhs = \langle\text{value}\rangle\).
Constraint: \(pdb \geq \text{max}(1, nrhs)\).
On entry, \(pdx = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(pdx \geq \text{max}(1, n)\).
On entry, \(pdx = \langle\text{value}\rangle\) and \(nrhs = \langle\text{value}\rangle\).
Constraint: \(pdx \geq \text{max}(1, nrhs)\).

**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 the Essential Introduction 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 the Essential Introduction for further information.

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  Parallelism and Performance
nag_ztprfs (f07uvc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_ztprfs (f07uvc) 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’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

A call to nag_ztprfs (f07uvc), 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 $4n^2$ real floating-point operations.

The real analogue of this function is nag_dtprfs (f07uhc).

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

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

using packed storage for $A$.

10.1 Program Text

/* nag_ztprfs (f07uvc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
*/
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.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;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char nag_enum_arg[40];
    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_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_RowMajor;
    #endif

Mark 25
#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);

printf("nag_ztprfs (f07uvc) 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;
ap_len = n*(n+1)/2;
#endif
#endif

/* 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)))
{
  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, _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 */
if (uplo == Nag_Upper)
{
  for (i = 1; i <= n; ++i) {
    for (j = i; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#else
      scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#endif
  }
}
scanf_s("%*\n");
```c
#else
    scanf("%*[\n ] ");
#endif
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A_LOWER(i, j).re,
                   &A_LOWER(i, j).im);
        #else
            scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
                   &A_LOWER(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_ztptrs (f07usc). */
/* Solution of complex triangular system of linear */
/* equations, multiple right-hand sides, packed storage */

nag_ztptrs(order, uplo, Nag_NoTrans, Nag_NONUnitDiag, n,
            nrhs, ap, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_ztptrs (f07usc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute backward errors and estimated bounds on the */
/* forward errors */
/* nag_ztprfs (f07uvc). */
/* Error bounds for solution of complex triangular system of */
/* linear equations, multiple right-hand sides, packed */
/* storage */

nag_ztprfs(order, uplo, Nag_NoTrans, Nag_NONUnitDiag, n,
            nrhs, ap, b, pdB, x, pdbx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_ztprfs (f07uvc).\n%s\n", fail.message);
}
```
exit_status = 1;
goto END;
}

/* Print solution */
printf("\n");
/* nag_gen_complex_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_complex_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_complex_mat_print_comp (x04dbc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

printf("Backward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", berr[j-1], j%4 == 0?"\n":"");
printf("Estimated 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(ap);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(ferr);
return exit_status;
}

10.2 Program Data

nag_ztprfs (f07uvc) Example Program Data

<table>
<thead>
<tr>
<th>n</th>
<th>nrhs</th>
<th>Values of n and nrhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>(4.78, 4.56)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2.00, -0.30)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2.89, -1.34)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-1.89, 1.15)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-14.78, -32.36)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-20.96, 17.06)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(9.54, 9.91)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uplo</th>
<th>Value of uplo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_Lower</td>
<td>(-4.11, 1.25)</td>
</tr>
<tr>
<td></td>
<td>(2.36, -4.25)</td>
</tr>
<tr>
<td></td>
<td>(0.04, -3.69)</td>
</tr>
<tr>
<td></td>
<td>(-0.02, 0.46)</td>
</tr>
<tr>
<td></td>
<td>(0.33, -0.26)</td>
</tr>
<tr>
<td></td>
<td>(-18.02, 28.46)</td>
</tr>
<tr>
<td></td>
<td>(5.62, 35.89)</td>
</tr>
<tr>
<td></td>
<td>(-16.46, -1.73)</td>
</tr>
</tbody>
</table>

| End of matrix A |
| (4.15, 0.80) |
| (-0.33, -0.26) |
| (14.22, 15.42) |
| (9.54, 9.91) |

| End of matrix B |
| (-14.78, -32.36) |
| (-20.96, 17.06) |
| (9.54, 9.91) |

10.3 Program Results

nag_ztprfs (f07uvc) Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-5.0000, -2.0000)</td>
<td>(1.0000, 5.0000)</td>
</tr>
<tr>
<td>(-3.0000, -1.0000)</td>
<td>(-2.0000, -2.0000)</td>
</tr>
<tr>
<td>(2.0000, 1.0000)</td>
<td>(3.0000, 4.0000)</td>
</tr>
<tr>
<td>(4.0000, 3.0000)</td>
<td>(4.0000, -3.0000)</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>2.7e-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>3.2e-14</td>
</tr>
</tbody>
</table>