NAG C Library Function Document

nag_dtbrfs (f07vhc)

1 Purpose

nag_dtbrfs (f07vhc) returns error bounds for the solution of a real triangular band system of linear equations with multiple right-hand sides, \( AX = B \) or \( A^T X = B \).

2 Specification

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

3 Description

nag_dtbrfs (f07vhc) returns the backward errors and estimated bounds on the forward errors for the solution of a real triangular band system of linear equations with multiple right-hand sides \( AX = B \) or \( A^T X = B \). The function handles each right-hand side vector (stored as a column of the matrix \( B \)) independently, so we describe the function of nag_dtbrfs (f07vhc) in terms of a single right-hand side \( b \) and solution \( x \).

Given a computed solution \( x \), the function computes the \textit{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 \textit{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: \textbf{order} – Nag_OrderType Input

- \textit{On entry}: the \textit{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 \textit{order} = Nag_RowMajor. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

- \textit{Constraint}: \textit{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{uplo} – Nag_UploType Input

- \textit{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 or Nag_ConjTrans, the equations are of the form \(A^T X = B\).

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:  
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 \(\geq 0\).

7:  
rhs – Integer  
Input
On entry: \(r\), the number of right-hand sides.

Constraint: rhs \(\geq 0\).

8:  
ab[dim] – const double  
Input
Note: the dimension, dim, of the array ab must be at least max(1, 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 order and uplo parameters as follows:

if order = Nag_ColMajor and uplo = Nag_Upper,
\(a_{ij}\) is stored in \(ab[k+i-j+(j-1)\times pdab]\), 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 \(ab[i-j+(i-1)\times pdab]\), 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 \(ab[j-i+(i-1)\times pdab]\), 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 \(i = 1, \ldots, n\) and \(j = \max(1, i-k), \ldots, i\).
9: \( \text{pdab} \) – Integer

\( \text{Input} \)

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

Constraint: \( \text{pdab} \geq \text{kd} + 1 \).

10: \( b[\text{dim}] \) = const double

\( \text{Input} \)

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

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

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

11: \( \text{pdb} \) – Integer

\( \text{Input} \)

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

Constraints:

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

12: \( x[\text{dim}] \) = const double

\( \text{Input} \)

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

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

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by \texttt{nag\_dtbtrs (f07vec)}.

13: \( \text{pdx} \) – Integer

\( \text{Input} \)

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

Constraints:

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

14: \( \text{ferr}[\text{dim}] \) = double

\( \text{Output} \)

Note: the dimension, \( \text{dim} \), 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 \( j = 1, 2, \ldots, r \).

15: \( \text{berr}[\text{dim}] \) = double

\( \text{Output} \)

Note: the dimension, \( \text{dim} \), 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 \( j = 1, 2, \ldots, r \).

16: \( \text{fail} \) = NagError *

\( \text{Output} \)

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 ≥ kd + 1.

On entry, pdb = \langle value\rangle, n = \langle value\rangle.
Constraint: pdb ≥ max(1, n).

On entry, pdb = \langle value\rangle, nrhs = \langle value\rangle.
Constraint: pdb ≥ max(1, nrhs).

On entry, pdx = \langle value\rangle, n = \langle value\rangle.
Constraint: pdx ≥ max(1, n).

On entry, pdx = \langle value\rangle, nrhs = \langle value\rangle.
Constraint: pdx ≥ max(1, 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_dtbrfs (f07vhc), for each right-hand side, involves solving a number of systems of linear equations of the form \( Ax = b \) or \( A^T x = b \); the number is usually 4 or 5 and never more than 11. Each solution involves approximately \( 2nk \) floating-point operations (assuming \( n ≫ k \)).

The complex analogue of this function is nag_ztbrfs (f07vvc).
9 Example

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

$$A = \begin{pmatrix} -4.16 & 0.00 & 0.00 & 0.00 \\ -2.25 & 4.78 & 0.00 & 0.00 \\ 0.00 & 5.86 & 6.32 & 0.00 \\ 0.00 & 0.00 & -4.82 & 0.16 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -16.64 & -4.16 \\ -13.78 & -16.59 \\ 13.10 & -4.94 \\ -14.14 & -9.96 \end{pmatrix}.$$

9.1 Program Text

```c
/* nag_dtbrfs (f07vhc) Example Program. */
* Copyright 2001 Numerical Algorithms Group.
*/
#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];
    double *ab=0, *b=0, *berr=0, *ferr=0, *x=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]
    #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
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    Vprintf("f07vhc Example Program Results\n\n");

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

[NP3645/7]
/* Allocate memory */
if ( !(berr = NAG_ALLOC(berr_len, double)) ||
    !(ferr = NAG_ALLOC(ferr_len, double)) ||
    !(ab = NAG_ALLOC((kd+1) * n, double)) ||
    !(b = NAG_ALLOC(n * nrhs, double)) ||
    !(x = NAG_ALLOC(n * nrhs, double)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A and B from data file, and copy B to X */
Vscanf(" ' %1s '%*[\n]\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", &AB_UPPER(i,j));
        }
        Vscanf("%*[\n] ");
    }
else
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = MAX(1,i-kd); j <= i; ++j)
                Vscanf("%lf", &AB_LOWER(i,j));
        }
        Vscanf("%*[\n] ");
    }
for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            Vscanf("%lf", &B(i,j));
    }
Vscanf("%*[\n] ");
/* Copy B to X */
for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            X(i,j) = B(i,j);
    }
/* Compute solution in the array X */
f07vec(order, uplo_enum, Nag_NoTrans, Nag_NonUnitDiag, n,
    kd, nrhs, ab, pdab, x, pdx, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f07vec.\n", fail.message);
        exit_status = 1;
        goto END;
    }
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07vhc(order, uplo_enum, Nag_NoTrans, Nag_NonUnitDiag, n,
    kd, nrhs, ab, pdab, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
f07 – Linear Equations (LAPACK)

9.2 Program Data
f07vhc Example Program Data

4  1  2  :Values of N, KD and NRHS
       'L'  :Value of UPLO
-4.16
-2.25  4.78
     5.86  6.32
     -4.82  0.16  :End of matrix A
-16.64 -4.16
-13.78 -16.59
 -13.10 -4.94

9.3 Program Results
f07vhc Example Program Results

Solution(s)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.0000</td>
</tr>
<tr>
<td>2</td>
<td>-1.0000</td>
</tr>
<tr>
<td>3</td>
<td>3.0000</td>
</tr>
<tr>
<td>4</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7e-17</td>
<td>0.0e+00</td>
</tr>
</tbody>
</table>

Estimated forward error bounds (machine-dependent)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4e-14</td>
<td>5.7e-14</td>
</tr>
</tbody>
</table>