NAG C Library Function Document

nag_dgbrfs (f07bhc)

1 Purpose

nag_dgbrfs (f07bhc) returns error bounds for the solution of a real band system of linear equations with multiple right-hand sides, \( AX = B \) or \( A^TX = B \). It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

\[
\text{void nag_dgbrfs (Nag_OrderType order, Nag_TransType trans, Integer n, Integer kl, Integer ku, Integer nrhs, const double ab[], Integer pdab, const double afb[], Integer pdafb, const Integer ipiv[], const double b[], Integer pdb, double x[], Integer pdx, double ferr[], double berr[], NagError *fail)}
\]

3 Description

nag_dgbrfs (f07bhc) returns the backward errors and estimated bounds on the forward errors for the solution of a real band system of linear equations with multiple right-hand sides \( AX = B \) or \( A^TX = B \). The function handles each right-hand side vector (stored as a column of the matrix \( B \)) independently, so we describe the function of nag_dgbrfs (f07bhc) 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: \textbf{order} – Nag_OrderType \hspace{2cm} \textit{Input}

\textit{On entry:} the \textbf{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 \textbf{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:} order = Nag_RowMajor or Nag_ColMajor.

2: \textbf{trans} – Nag_TransType \hspace{2cm} \textit{Input}

\textit{On entry:} indicates the form of the linear equations for which \( X \) is the computed solution as follows:
if \( \text{trans} = \text{Nag\_NoTrans} \), then the linear equations are of the form \( AX = B \).

if \( \text{trans} = \text{Nag\_Trans} \) or \( \text{Nag\_ConjTrans} \), then the linear equations are of the form \( A^T X = B \).

\( \text{Constraint: trans} = \text{Nag\_NoTrans, Nag\_Trans or Nag\_ConjTrans.} \)

3: \( n \) – Integer

\( \text{Input} \)

\( On\ entry: \ n, \ the\ order\ of\ the\ matrix\ A. \)

\( \text{Input} \)

\( \text{Constraint:} \ n \geq 0. \)

4: \( kl \) – Integer

\( On\ entry: \ kl, \ the\ number\ of\ sub-diagonals\ within\ the\ band\ of\ A. \)

\( \text{Input} \)

\( \text{Constraint:} \ kl \geq 0. \)

5: \( ku \) – Integer

\( On\ entry: \ ku, \ the\ number\ of\ super-diagonals\ within\ the\ band\ of\ A. \)

\( \text{Input} \)

\( \text{Constraint:} \ ku \geq 0. \)

6: \( nrhs \) – Integer

\( On\ entry: \ nrhs, \ the\ number\ of\ right-hand\ sides. \)

\( \text{Input} \)

\( \text{Constraint:} \ nrhs \geq 0. \)

7: \( ab[\text{dim}] \) – const double

\( \text{Input} \)

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

\( On\ entry: \ the\ original\ n\ by\ n\ band\ matrix\ A\ as\ supplied\ to\ nag\_dgbtrf\ (f07bdc)\ but\ with\ reduced\ requirements\ since\ the\ matrix\ is\ not\ factorized.\ This\ is\ stored\ as\ a\ notional\ two-dimensional\ array\ with\ row\ elements\ or\ column\ elements\ stored\ contiguously.\ The\ storage\ of\ elements\ \text{aij, for} \ i = 1, \ldots, n \text{ and } j = \max(1, i - k_i), \ldots, \min(n, i + k_u), \text{ depends \ on \ the \ order \ parameter \ as \ follows:} \)

\( \text{if order} = \text{Nag\_ColMajor, } a_{ij} \text{ is stored as } ab[(j - 1) \times pdab + ku + i - j]; \)

\( \text{if order} = \text{Nag\_RowMajor, } a_{ij} \text{ is stored as } ab[(i - 1) \times pdab + kl + j - i]. \)

8: \( pdab \) – Integer

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

\( \text{Input} \)

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

9: \( afb[\text{dim}] \) – const double

\( \text{Input} \)

\( \text{Note:} \ the\ dimension,\ \text{dim,}\ of\ the\ array\ \text{afb}\ must\ be\ at\ least\ max(1, pdafb \times n).} \)

\( On\ entry: \ the\ \text{LU}\ factorization\ of\ A,\ as\ returned\ by\ nag\_dgbtrf\ (f07bdc). \)

10: \( pdafb \) – Integer

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

\( \text{Input} \)

\( \text{Constraint:} \ pdafb \geq 2 \times kl + ku + 1. \)

11: \( ipiv[\text{dim}] \) – const Integer

\( \text{Input} \)

\( \text{Note:} \ the\ dimension,\ \text{dim,}\ of\ the\ array\ \text{ipiv}\ must\ be\ at\ least\ max(1, n). \)
On entry: the pivot indices, as returned by nag_dgbtrf (f07bdc).

12: \( \text{b}[\text{dim}] \) – const double

**Input**

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

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

13: \( \text{pdb} \) – Integer

**Input**

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

Constraints:

if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdb} \geq \max(1, n) \);

if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdb} \geq \max(1, \text{nrhs}) \).

14: \( \text{x}[\text{dim}] \) – double

**Input/Output**

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

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by nag_dgbtrs (f07bec).

On exit: the improved solution matrix \( X \).

15: \( \text{pdx} \) – Integer

**Input**

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

Constraints:

if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdx} \geq \max(1, n) \);

if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdx} \geq \max(1, \text{nrhs}) \).

16: \( \text{ferr}[\text{dim}] \) – double

**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 \( j = 1, 2, \ldots, r \).

17: \( \text{berr}[\text{dim}] \) – double

**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 \( j = 1, 2, \ldots, r \).

18: \( \text{fail} \) – 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, \( k_i \) = \langle \text{value} \rangle.
Constraint: \( k_i \geq 0 \).

On entry, \( k_j \) = \langle \text{value} \rangle.
Constraint: \( k_j \geq 0 \).

On entry, \( n \) = \langle \text{value} \rangle.
Constraint: \( n \geq 0 \).

On entry, \( pd_{ab} \) = \langle \text{value} \rangle.
Constraint: \( pd_{ab} > 0 \).

On entry, \( pd_{afb} \) = \langle \text{value} \rangle.
Constraint: \( pd_{afb} > 0 \).

On entry, \( pd \) = \langle \text{value} \rangle.
Constraint: \( pd > 0 \).

On entry, \( pd_x \) = \langle \text{value} \rangle.
Constraint: \( pd_x > 0 \).

**NE_INT_2**

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

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

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

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

**NE_INT_3**

On entry, \( pd_{ab} \) = \langle \text{value} \rangle, \( k_l \) = \langle \text{value} \rangle, \( k_u \) = \langle \text{value} \rangle.
Constraint: \( pd_{ab} \geq k_l + k_u + 1 \).

On entry, \( pd_{afb} \) = \langle \text{value} \rangle, \( k_l \) = \langle \text{value} \rangle, \( k_u \) = \langle \text{value} \rangle.
Constraint: \( pd_{afb} \geq 2 \times k_l + k_u + 1 \).

**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 \texttt{ferr} are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

### 8 Further Comments

For each right-hand side, computation of the backward error involves a minimum of \( 4n(k_l + k_u) \) floating-point operations. Each step of iterative refinement involves an additional \( 2n(4k_l + 3k_u) \) operations. This
assumes $n \gg k_l$ and $n \gg k_u$. At most 5 steps of iterative refinement are performed, but usually only 1 or 2 steps are required.

Estimating the forward error 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 $2n(2k_l + k_u)$ operations.

The complex analogue of this function is nag_zgbrfs (f07bvc).

9 Example

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

$$A = \begin{pmatrix} -0.23 & 2.54 & -3.66 & 0.00 \\ -6.98 & 2.46 & -2.73 & -2.13 \\ 0.00 & 2.56 & 2.46 & 4.07 \\ 0.00 & 0.00 & -4.78 & -3.82 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 4.42 & -36.01 \\ 27.13 & -31.67 \\ -6.14 & -1.16 \\ 10.50 & -25.82 \end{pmatrix}.$$  

Here $A$ is nonsymmetric and is treated as a band matrix, which must first be factorized by nag_dgbtrf (f07bdc).

9.1 Program Text

```c
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, ipiv_len, j, kl, ku, n, nrhs, pdab, pdafb, pdb, pdx;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *ab=0, *afb=0, *b=0, *berr=0, *ferr=0, *x=0;
    Integer *ipiv=0;
    #ifdef NAG_COLUMN_MAJOR
        #define AB(I,J) ab[(J-1)*pdab + ku + I - J]
        #define AFB(I,J) afb[(J-1)*pdafb + kl + ku + I - J]
        #define B(I,J) b[(I-1)*pdb + I - 1]
        #define X(I,J) x[(J-1)*pdx + I - 1]
        order = Nag_ColMajor;
    #else
        #define AB(I,J) ab[(J-1)*pdab + ku + I - J]
        #define AFB(I,J) afb[(J-1)*pdafb + kl + ku + I - J]
        #define B(I,J) b[(I-1)*pdb + I - 1]
        #define X(I,J) x[(I-1)*pdx + J - 1]
        order = Nag_RowMajor;
    #endif
    INIT_FAIL(fail);
    Vprintf("f07bhc Example Program Results\n\n");
    /* Skip heading in data file */
```
Vscanf("%*[\n "");
Vscanf("%ld%ld%ld%ld%*[\n ", &n, &nrhs, &kl, &ku);
ipiv_len = n;
pdb = kl + ku + 1;
pfadb = 2*kl + ku + 1;
#endif NAG_COLUMN_MAJOR
pdb = n;
pdx = n;
#else
pdb = nrhs;
pdx = nrhs;
#endif
/* Allocate memory */
if ( !(ab = NAG_ALLOC((kl+ku+1) * n, double)) ||
    !(afb = NAG_ALLOC((2*kl+ku+1) * n, double)) ||
    !(b = NAG_ALLOC(nrhs * n, double)) ||
    !(x = NAG_ALLOC(nrhs * n, double)) ||
    !(berr = NAG_ALLOC(nrhs, double)) ||
    !(ferr = NAG_ALLOC(nrhs, double)) ||
    !(ipiv = NAG_ALLOC(ipiv_len, Integer)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Set A to zero to avoid referencing unitialized elements */
for (i = 0; i < n*(kl+ku+1); ++i)
ab[i] = 0.0;
/* Read A from data file */
for (i = 1; i <= n; ++i)
{
    for (j = MAX(i-kl,1); j <= MIN(i+ku,n); ++j)
        Vscanf("%lf", &AB(i,j));
} Vscanf("%*[\n ");
/* Read B from data file */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf("%lf", &B(i,j));
} Vscanf("%*[\n ");
/* Copy A to AFB and B to X */
for (i = 1; i <= n; ++i)
{
    for (j = MAX(i-kl,1); j <= MIN(i+ku,n); ++j)
        AFB(i,j) = AB(i,j);
} for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        X(i,j) = B(i,j);
} /* Factorize A in the array AFB */
f07bdc(order, n, n, kl, ku, afb, pdafb, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07bdc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
} /* Compute solution in the array X */
f07bec(order, Nag_NoTrans, n, kl, ku, nrhs, afb, pdafb, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07bec.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07bhc(order, Nag_NoTrans, n, kl, ku, nrhs, ab, pdab, afb, pdafb,
ipiv, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07bhc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print solution */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
"Solution(s)", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print forward and backward errors */
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 (ab) NAG_FREE(ab);
    if (afb) NAG_FREE(afb);
    if (b) NAG_FREE(b);
    if (x) NAG_FREE(x);
    if (berr) NAG_FREE(berr);
    if (ferr) NAG_FREE(ferr);
    if (ipiv) NAG_FREE(ipiv);
return exit_status;

9.2 Program Data
f07bhc Example Program Data
4 2 1 2 :Values of N, NRHS, KL and KU
-0.23 2.54 -3.66
-6.98 2.46 -2.73 -2.13
 2.56 2.46 4.07
 -4.78 -3.82 :End of matrix A
4.42 -36.01
27.13 -31.67
-6.14 -1.16
10.50 -25.82 :End of matrix B

9.3 Program Results
f07bhc Example Program Results
Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>3.0000</td>
<td>-4.0000</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>7.0000</td>
</tr>
<tr>
<td>4</td>
<td>-4.0000</td>
<td>-2.0000</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)
1.0e-16 8.2e-17
Estimated forward error bounds (machine-dependent)
1.5e-14 1.8e-14