

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 $A^T X = B$. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```
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 $A^T X = B$. The function handles each right-hand side vector (stored as a column of the matrix B) independently, so we describe the function of nag_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* 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

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

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

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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: **trans** – Nag_TransType *Input*

On entry: indicates the form of the linear equations for which X is the computed solution as follows:

if **trans** = **Nag_NoTrans**, then the linear equations are of the form $AX = B$.

if **trans** = **Nag_Trans** or **Nag_ConjTrans**, then the linear equations are of the form $A^T X = B$.

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

3: **n** – Integer *Input*

On entry: n , the order of the matrix A .

Constraint: $n \geq 0$.

4: **kl** – Integer *Input*

On entry: k_l , the number of sub-diagonals within the band of A .

Constraint: $kl \geq 0$.

5: **ku** – Integer *Input*

On entry: k_u , the number of super-diagonals within the band of A .

Constraint: $ku \geq 0$.

6: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0 .

7: **ab**[*dim*] – const double *Input*

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

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 a_{ij} , for $i = 1, \dots, n$ and $j = \max(1, i - k_l), \dots, \min(n, i + k_u)$, depends on the **order** parameter as follows:

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

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

8: **pdab** – Integer *Input*

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

Constraint: **pdab** $\geq \mathbf{kl} + \mathbf{ku} + 1$.

9: **afb**[*dim*] – const double *Input*

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

On entry: the LU factorization of A , as returned by nag_dgbtrf (f07bdc).

10: **pdafb** – Integer *Input*

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

Constraint: **pdafb** $\geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

11: **ipiv**[*dim*] – const Integer *Input*

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

On entry: the pivot indices, as returned by nag_dgbtrf (f07bdc).

- 12: **b**[*dim*] – const double Input

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

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

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

- 13: **pdb** – Integer Input

On entry: the stride separating matrix 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**).

- 14: **x**[*dim*] – double Input/Output

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

If **order** = **Nag_ColMajor**, the (*i*, *j*)th element of the matrix *X* is stored in **x**[(*j* – 1) × **pdx** + *i* – 1] and if **order** = **Nag_RowMajor**, the (*i*, *j*)th element of the matrix *X* is stored in **x**[(*i* – 1) × **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: **pdx** – Integer Input

On entry: the stride separating matrix 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**).

- 16: **ferr**[*dim*] – double Output

Note: the dimension, *dim*, of the array **ferr** must be at least max(1, **nrhs**).

On exit: **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, ..., *r*.

- 17: **berr**[*dim*] – double Output

Note: the dimension, *dim*, of the array **berr** must be at least max(1, **nrhs**).

On exit: **berr**[*j* – 1] contains the component-wise backward error bound β for the *j*th solution vector, that is, the *j*th column of *X*, for *j* = 1, 2, ..., *r*.

- 18: **fail** – NagError * Output

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = *<value>*.

Constraint: **n** ≥ 0.

On entry, **kl** = $\langle value \rangle$.

Constraint: **kl** ≥ 0 .

On entry, **ku** = $\langle value \rangle$.

Constraint: **ku** ≥ 0 .

On entry, **nrhs** = $\langle value \rangle$.

Constraint: **nrhs** ≥ 0 .

On entry, **pdab** = $\langle value \rangle$.

Constraint: **pdab** > 0 .

On entry, **pdafb** = $\langle value \rangle$.

Constraint: **pdafb** > 0 .

On entry, **pdb** = $\langle value \rangle$.

Constraint: **pdb** > 0 .

On entry, **pdx** = $\langle value \rangle$.

Constraint: **pdx** > 0 .

NE_INT_2

On entry, **pdb** = $\langle value \rangle$, **n** = $\langle value \rangle$.

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

On entry, **pdb** = $\langle value \rangle$, **nrhs** = $\langle value \rangle$.

Constraint: **pdb** $\geq \max(1, \mathbf{nrhs})$.

On entry, **pdx** = $\langle value \rangle$, **n** = $\langle value \rangle$.

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

On entry, **pdx** = $\langle value \rangle$, **nrhs** = $\langle value \rangle$.

Constraint: **pdx** $\geq \max(1, \mathbf{nrhs})$.

NE_INT_3

On entry, **pdab** = $\langle value \rangle$, **kl** = $\langle value \rangle$, **ku** = $\langle value \rangle$.

Constraint: **pdab** $\geq \mathbf{kl} + \mathbf{ku} + 1$.

On entry, **pdafb** = $\langle value \rangle$, **kl** = $\langle value \rangle$, **ku** = $\langle value \rangle$.

Constraint: **pdafb** $\geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

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

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

```
/* nag_dgbrfs (f07bhc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#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[(J-1)*pdb + I - 1]
#define X(I,J) x[(J-1)*pdx + I - 1]
    order = Nag_ColMajor;
#else
#define AB(I,J) ab[(I-1)*pdab + kl + J - I]
#define AFB(I,J) afb[(I-1)*pdafb + kl + J - I]
#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("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;
pdab = kl + ku + 1;
pdafb = 2*kl + ku + 1;
#ifdef 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 uninitialized 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)
          1          2
1      -2.0000      1.0000
2       3.0000     -4.0000
3       1.0000      7.0000
4      -4.0000     -2.0000

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

```