

NAG C Library Function Document

nag_dpbrfs (f07hhc)

1 Purpose

nag_dpbrfs (f07hhc) returns error bounds for the solution of a real symmetric positive-definite band system of linear equations with multiple right-hand sides, $AX = B$. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```
void nag_dpbrfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer kd,
                Integer nrhs, const double ab[], Integer pdab, const double afb[],
                Integer pdafb, const double b[], Integer pdb, double x[], Integer pdx,
                double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dpbrfs (f07hhc) returns the backward errors and estimated bounds on the forward errors for the solution of a real symmetric positive-definite band system of linear equations with multiple right-hand sides $AX = B$. The function handles each right-hand side vector (stored as a column of the matrix B) independently, so we describe the function of nag_dpbrfs (f07hhc) 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, 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: **uplo** – Nag_UploType *Input*
On entry: indicates whether the upper or lower triangular part of A is stored and how A has been factorized, as follows:
 if **uplo** = **Nag_Upper**, the upper triangular part of A is stored and A is factorized as $U^T U$, where U is upper triangular;
 if **uplo** = **Nag_Lower**, the lower triangular part of A is stored and A is factorized as LL^T , where L is lower triangular.
Constraint: **uplo** = **Nag_Upper** or **Nag_Lower**.
- 3: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.
- 4: **kd** – Integer *Input*
On entry: k , the number of super-diagonals or sub-diagonals of the matrix A .
Constraint: $kd \geq 0$.
- 5: **nrhs** – Integer *Input*
On entry: r , the number of right-hand sides.
Constraint: **nrhs** ≥ 0 .
- 6: **ab**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **ab** must be at least $\max(1, \mathbf{pdab} \times \mathbf{n})$.
 If **order** = **Nag_ColMajor**, the (i, j) th element of the matrix is stored in **ab**[($j - 1$) $\times$ **pdab** + $i - 1$] and if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix is stored in **ab**[($i - 1$) $\times$ **pdab** + $j - 1$].
On entry: the n by n original symmetric band matrix A as supplied to nag_dpbtrf (f07hdc).
- 7: **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** $\geq kd + 1$.
- 8: **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 Cholesky factor of A , as returned by nag_dpbtrf (f07hdc).
- 9: **pdafb** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix in the array **afb**.
Constraint: **pdafb** $\geq kd + 1$.
- 10: **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**[($j - 1$) $\times$ **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 .

- 11: **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** $\geq \max(1, n)$;
 if **order** = **Nag_RowMajor**, **pdb** $\geq \max(1, nrhs)$.
- 12: **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 $\max(1, \mathbf{pdx} \times n)$ when **order** = **Nag_RowMajor**.
 If **order** = **Nag_ColMajor**, the (i, j) th element of the matrix X is stored in $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix X is stored in $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$.
On entry: the n by r solution matrix X , as returned by nag_dpbtrs (f07hec).
On exit: the improved solution matrix X .
- 13: **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** $\geq \max(1, n)$;
 if **order** = **Nag_RowMajor**, **pdx** $\geq \max(1, nrhs)$.
- 14: **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 $j = 1, 2, \dots, r$.
- 15: **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 $j = 1, 2, \dots, r$.
- 16: **fail** – NagError * *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, **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, **pdab** = $\langle value \rangle$, **kd** = $\langle value \rangle$.

Constraint: **pdab** $\geq$ **kd** + 1.

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

Constraint: **pdafb** $\geq$ **kd** + 1.

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

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

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

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

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

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

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

Constraint: **pdx** $\geq$ 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

For each right-hand side, computation of the backward error involves a minimum of $8nk$ floating-point operations. Each step of iterative refinement involves an additional $12nk$ operations. This assumes $n \gg k$. 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 $Ax = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $4nk$ operations.

The complex analogue of this function is nag_zpbrfs (f07hvc).

9 Example

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

$$A = \begin{pmatrix} 5.49 & 2.68 & 0.00 & 0.00 \\ 2.68 & 5.63 & -2.39 & 0.00 \\ 0.00 & -2.39 & 2.60 & -2.22 \\ 0.00 & 0.00 & -2.22 & 5.17 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 22.09 & 5.10 \\ 9.31 & 30.81 \\ -5.24 & -25.82 \\ 11.83 & 22.90 \end{pmatrix}.$$

Here A is symmetric and positive-definite, and is treated as a band matrix, which must first be factorized by `nag_dpbtrf` (f07hdc).

9.1 Program Text

```

/* nag_dpbrfs (f07hhc) 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, j, k, kd, n, nrhs, pdab, pdafb, 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, *afb=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 AFB_UPPER(I,J) afb[(J-1)*pdafb + k + I - J - 1]
#define AFB_LOWER(I,J) afb[(J-1)*pdafb + 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_UPPER(I,J) ab[(I-1)*pdab + J - I]
#define AB_LOWER(I,J) ab[(I-1)*pdab + k + J - I - 1]
#define AFB_UPPER(I,J) afb[(I-1)*pdafb + J - I]
#define AFB_LOWER(I,J) afb[(I-1)*pdafb + k + J - I - 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("f07hhc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^\\n] ");
    Vscanf("%ld%ld%ld%*[^\\n] ", &n, &kd, &nrhs);
    pdab = kd + 1;
    pdafb = kd + 1;
#ifdef NAG_COLUMN_MAJOR
    pdb = n;
    pdx = n;
#else

```

```

    pdb = nrhs;
    pdx = nrhs;
#endif

    ferr_len = nrhs;
    berr_len = nrhs;

    /* Allocate memory */
    if ( !(berr = NAG_ALLOC(berr_len, double)) ||
        !(ferr = NAG_ALLOC(ferr_len, double)) ||
        !(ab = NAG_ALLOC((kd+1) * n, double)) ||
        !(afb = 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 from data file */
    Vscanf(" ' %1s '%*[\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] ");
    }
    /* 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 AF and B to X */
    if (uplo_enum == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = i; j <= MIN(i+kd,n); ++j)
                AFB_UPPER(i,j) = AB_UPPER(i,j);
        }
    }
    else
    {
        for (i = 1; i <= n; ++i)
        {

```

```

        for (j = MAX(1,i-kd); j <= i; ++j)
            AFB_LOWER(i,j) = AB_LOWER(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 AFP */
f07hdc(order, uplo_enum, n, kd, afb, pdafb, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07hdc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution in the array X */
f07hec(order, uplo_enum, n, kd, nrhs, afb, pdafb, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07hec.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07hhc(order, uplo_enum, n, kd, nrhs, ab, pdab, afb, pdafb,
        b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07hhc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print details of 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;
}
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 (berr) NAG_FREE(berr);
if (ferr) NAG_FREE(ferr);
if (ab) NAG_FREE(ab);
if (afb) NAG_FREE(afb);
if (b) NAG_FREE(b);
if (x) NAG_FREE(x);
return exit_status;
}

```

9.2 Program Data

f07hhc Example Program Data

4	1	2		:Values of N, KD and NRHS
'L'				:Value of UPLO
5.49				
2.68	5.63			
-2.39	2.60			

```
      -2.22   5.17   :End of matrix A
22.09   5.10
 9.31  30.81
-5.24 -25.82
11.83  22.90           :End of matrix B
```

9.3 Program Results

f07hhc Example Program Results

Solution(s)

	1	2
1	5.0000	-2.0000
2	-2.0000	6.0000
3	-3.0000	-1.0000
4	1.0000	4.0000

Backward errors (machine-dependent)

6.4e-17	6.3e-17
---------	---------

Estimated forward error bounds (machine-dependent)

2.0e-14	2.9e-14
---------	---------
