

NAG C Library Function Document

nag_zpbrfs (f07hvc)

1 Purpose

nag_zpbrfs (f07hvc) returns error bounds for the solution of a complex Hermitian 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_zpbrfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer kd,
                 Integer nrhs, const Complex ab[], Integer pdab, const Complex afb[],
                 Integer pdafb, const Complex b[], Integer pdb, Complex x[], Integer pdx,
                 double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zpbrfs (f07hvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex Hermitian 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_zpbrfs (f07hvc) 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: **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 $U^H U$, where U is upper triangular;
 if **uplo** = **Nag_Lower**, the lower triangular part of A is stored and A is factorized as LL^H , 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** ≥ 0 .
- 4: **kd** – Integer *Input*
On entry: k , the number of super-diagonals or sub-diagonals of the matrix A .
Constraint: **kd** ≥ 0 .
- 5: **nrhs** – Integer *Input*
On entry: r , the number of right-hand sides.
Constraint: **nrhs** ≥ 0 .
- 6: **ab**[dim] – const Complex *Input*
Note: the dimension, dim , of the array **ab** must be at least $\max(1, \mathbf{pdab} \times \mathbf{n})$.
On entry: the n by n original Hermitian positive-definite band matrix A as supplied to nag_zpbtrf (f07hrc).
- 7: **pdab** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix in the array **ab**.
Constraint: **pdab** $\geq \mathbf{kd} + 1$.
- 8: **afb**[dim] – const Complex *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_zpbtrf (f07hrc).
- 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** $\geq \mathbf{kd} + 1$.
- 10: **b**[dim] – const Complex *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*] – Complex *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 **x**[(*j* – 1) $\times$ **pdx** + *i* – 1] and if **order** = **Nag_RowMajor**, the (*i*, *j*)th element of the matrix *X* is stored in **x**[(*i* – 1) $\times$ **pdx** + *j* – 1].
On entry: the *n* by *r* solution matrix *X*, as returned by nag_zpbtrs (f07hsc).
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, ..., *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, ..., *r*.
- 16: **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, **kd** = *<value>*.

Constraint: **kd** ≥ 0 .

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

Constraint: **nrhs** ≥ 0 .

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

Constraint: **pdab** > 0 .

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

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 $32nk$ real floating-point operations. Each step of iterative refinement involves an additional $48nk$ real 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 5 and never more than 11. Each solution involves approximately $16nk$ real operations.

The real analogue of this function is nag_dpbrfs (f07hhc).

9 Example

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

$$A = \begin{pmatrix} 9.39 + 0.00i & 1.08 - 1.73i & 0.00 + 0.00i & 0.00 + 0.00i \\ 1.08 + 1.73i & 1.69 + 0.00i & -0.04 + 0.29i & 0.00 + 0.00i \\ 0.00 + 0.00i & -0.04 - 0.29i & 2.65 + 0.00i & -0.33 + 2.24i \\ 0.00 + 0.00i & 0.00 + 0.00i & -0.33 - 2.24i & 2.17 + 0.00i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -12.42 & + & 68.42i & 54.30 & - & 56.56i \\ -9.93 & + & 0.88i & 18.32 & + & 4.76i \\ -27.30 & - & 0.01i & -4.40 & + & 9.97i \\ 5.31 & + & 23.63i & 9.43 & + & 1.41i \end{pmatrix}.$$

Here A is Hermitian positive-definite, and is treated as a band matrix, which must first be factorized by `nag_zpbtrf` (f07hrc).

9.1 Program Text

```
/* nag_zpbrfs (f07hvc) 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];
    Complex *ab=0, *afb=0, *b=0, *x=0;
    double *berr=0, *ferr=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("f07hvc 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, Complex)) ||
    !(afb = NAG_ALLOC((kd+1) * n, Complex)) ||
    !(b = NAG_ALLOC(n * nrhs, Complex)) ||
    !(x = NAG_ALLOC(n * nrhs, Complex)) )
{
    Vprintf("Allocation failure\\n");
    exit_status = -1;
    goto END;
}

/* Read A from data file */
Vscanf(" ' %ls '%*[^\\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 , %lf )", &AB_UPPER(i,j).re,
                &AB_UPPER(i,j).im);
        }
    }
    Vscanf("%*[^\\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = MAX(1,i-kd); j <= i; ++j)
        {
            Vscanf(" ( %lf , %lf )", &AB_LOWER(i,j).re,
                &AB_LOWER(i,j).im);
        }
    }
    Vscanf("%*[^\\n] ");
}

/* Read B from data file */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
}

```

```

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).re = AB_UPPER(i,j).re;
            AFB_UPPER(i,j).im = AB_UPPER(i,j).im;
        }
    }
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = MAX(1,i-kd); j <= i; ++j)
        {
            AFB_LOWER(i,j).re = AB_LOWER(i,j).re;
            AFB_LOWER(i,j).im = AB_LOWER(i,j).im;
        }
    }
}
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;
    }
}
/* Factorize A in the array AFP */
f07hrc(order, uplo_enum, n, kd, afb, pdafb, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07hrc.\\n%s\\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution in the array X */
f07hsc(order, uplo_enum, n, kd, nrhs, afb, pdafb, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07hsc.\\n%s\\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07hvc(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 f07hvc.\\n%s\\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print details of solution */
x04dbc(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)
{
    Vprintf("Error from x04dbc.\\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

f07hvc Example Program Data

```

    4 1 2                                     :Values of N, KD and NRHS
    'L'                                       :Value of UPLO
    ( 9.39, 0.00)
    ( 1.08, 1.73) ( 1.69, 0.00)
                      (-0.04,-0.29) ( 2.65, 0.00)
                                  (-0.33,-2.24) ( 2.17, 0.00) :End of matrix A
    (-12.42,68.42) (54.30,-56.56)
    ( -9.93, 0.88) (18.32, 4.76)
    (-27.30,-0.01) (-4.40, 9.97)
    ( 5.31,23.63) ( 9.43, 1.41)               :End of matrix B

```

9.3 Program Results

f07hvc Example Program Results

```

Solution(s)
           1           2
1  (-1.0000, 8.0000) ( 5.0000,-6.0000)
2  ( 2.0000,-3.0000) ( 2.0000, 3.0000)
3  (-4.0000,-5.0000) (-8.0000, 4.0000)
4  ( 7.0000, 6.0000) (-1.0000,-7.0000)

Backward errors (machine-dependent)
    3.2e-17    3.3e-17
Estimated forward error bounds (machine-dependent)
    3.2e-14    3.0e-14

```
