

# NAG 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

```
#include <nag.h>
#include <nagf07.h>
```

```
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*  $\beta$ . 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

$$|\delta a_{ij}| \leq \beta |a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta |b_i|.$$

$$(A + \delta A)x = b + \delta b$$

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 Arguments

1: **order** – Nag\_OrderType

*Input*

*On entry:* the **order** argument 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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

*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  $A$  is to be factorized.  
**uplo** = Nag\_Upper  
 The upper triangular part of  $A$  is stored and  $A$  is factorized as  $U^H U$ , where  $U$  is upper triangular.  
**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**  $\geq$  0.
- 4: **kd** – Integer *Input*  
*On entry:*  $k_d$ , the number of superdiagonals or subdiagonals of the matrix  $A$ .  
*Constraint:* **kd**  $\geq$  0.
- 5: **nrhs** – Integer *Input*  
*On entry:*  $r$ , the number of right-hand sides.  
*Constraint:* **nrhs**  $\geq$  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**  $\geq$  **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**  $\geq$  **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;  
 $\max(1, \mathbf{pdb} \times \mathbf{n})$  when **order** = Nag\_RowMajor.  
 The  $(i, j)$ th element of the matrix  $B$  is stored in

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$  when **order** = Nag\_ColMajor;

$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$  when **order** = Nag\_RowMajor.

*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 **b**.

*Constraints:*

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

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

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;

$\max(1, \mathbf{pdx} \times \mathbf{n})$  when **order** = Nag\_RowMajor.

The  $(i, j)$ th element of the matrix  $X$  is stored in

$\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$  when **order** = Nag\_ColMajor;

$\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$  when **order** = Nag\_RowMajor.

*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 **x**.

*Constraints:*

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

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

14: **ferr**[**nrhs**] – double *Output*

*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**[**nrhs**] – double *Output*

*On exit:* **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  $j = 1, 2, \dots, r$ .

16: **fail** – NagError \* *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

**NE\_INT**

On entry, **kd** =  $\langle value \rangle$ .  
 Constraint: **kd**  $\geq 0$ .

On entry, **n** =  $\langle value \rangle$ .  
 Constraint: **n**  $\geq 0$ .

On entry, **nrhs** =  $\langle value \rangle$ .  
 Constraint: **nrhs**  $\geq 0$ .

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**  $\geq \mathbf{kd} + 1$ .

On entry, **pdafb** =  $\langle value \rangle$ , **kd** =  $\langle value \rangle$ .  
 Constraint: **pdafb**  $\geq \mathbf{kd} + 1$ .

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\_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 contact 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 five steps of iterative refinement are performed, but usually only one or two 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

This example solves 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(int argc, char *argv[])
{
    FILE          *fpin, *fpout;
    char          *outfile = 0;
    /* Scalars */
    Integer       i, j, k, kd, n, nrhs, pdab, pdafb, pdb, pdx;
    Integer       ferr_len, berr_len;
    Integer       exit_status = 0;
    Nag_UploType  uplo;
    NagError      fail;
    Nag_OrderType order;

    /* Arrays */
    char          nag_enum_arg[40];
    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
}

```

```

#endif

INIT_FAIL(fail);

/* Check for command-line IO options */
fpin = nag_example_file_io(argc, argv, "-data", NULL);
fpout = nag_example_file_io(argc, argv, "-results", NULL);
(void) nag_example_file_io(argc, argv, "-nag_write", &outfile);
fprintf(fpout, "nag_zpbrfs (f07hvc) Example Program Results\n\n");

/* Skip heading in data file */
fscanf(fpin, "%*[\n] ");
fscanf(fpin, "%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)))
{
    fprintf(fpout, "Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A from data file */
fscanf(fpin, "%s*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value(x04nac).
 * Converts NAG enum member name to value
 */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

k = kd + 1;
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= MIN(i+kd, n); ++j)
        {
            fscanf(fpin, " ( %lf , %lf )", &AB_UPPER(i, j).re,
                &AB_UPPER(i, j).im);
        }
        fscanf(fpin, "%*[\n] ");
    }
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = MAX(1, i-kd); j <= i; ++j)
        {
            fscanf(fpin, " ( %lf , %lf )", &AB_LOWER(i, j).re,
                &AB_LOWER(i, j).im);
        }
        fscanf(fpin, "%*[\n] ");
    }
}

```

```

    }
    /* Read B from data file */
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            fscanf(fpin, " ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }
    fscanf(fpin, "%*[\n] ");
    /* Copy A to AF and B to X */
    if (uplo == 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 */
    /* nag_zpbtrf (f07hrc).
    * Cholesky factorization of complex Hermitian
    * positive-definite band matrix
    */
    nag_zpbtrf(order, uplo, n, kd, afb, pdafb, &fail);
    if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_zpbtrf (f07hrc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Compute solution in the array X */
    /* nag_zpbtrs (f07hsc).
    * Solution of complex Hermitian positive-definite band
    * system of linear equations, multiple right-hand sides,
    * matrix already factorized by nag_zpbtrf (f07hrc)
    */
    nag_zpbtrs(order, uplo, n, kd, nrhs, afb, pdafb, x, pdx, &fail);
    if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_zpbtrs (f07hsc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Improve solution, and compute backward errors and */
    /* estimated bounds on the forward errors */
    /* nag_zpbrfs (f07hvc).
    * Refined solution with error bounds of complex Hermitian
    * positive-definite band system of linear equations,
    * multiple right-hand sides
    */

```

```

nag_zpbrfs(order, uplo, n, kd, nrhs, ab, pdab, afb, pdafb,
           b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_zpbrfs (f07hvc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print details of solution */

/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
if (outfile) fclose(fpout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                             nrhs, x, pdx, Nag_BracketForm, "%7.4f",
                             "Solution(s)", Nag_IntegerLabels,
                             0, Nag_IntegerLabels, 0, 80, 0, outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout,
           "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}
fprintf(fpout, "\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    fprintf(fpout, "%11.1e%s", berr[j-1], j%7 == 0?"\n":" ");
fprintf(fpout, "\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    fprintf(fpout, "%11.1e%s", ferr[j-1], j%7 == 0?"\n":" ");
fprintf(fpout, "\n");
END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
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

```

nag_zpbrfs (f07hvc) Example Program Data
  4  1  2                               :Values of n, kd and nrhs
  Nag_Lower                             :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

nag\_zpbrfs (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)

1.0e-16      7.8e-17

Estimated forward error bounds (machine-dependent)

3.8e-14      3.3e-14

---