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

```c
#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

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

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

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 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`. 

Mark 25
2:  uplo – Nag_UploType

   Input

   On entry: specifies 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

   On entry: the $n$ by $n$ original Hermitian positive definite band matrix $A$ as supplied to nag_zpbtrf
   (f07hrc).

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

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:  a_fb[dim] – const Complex

   Input

   On entry: the $n$ by $n$ original Hermitian positive definite band matrix $A$ as supplied to nag_zpbtrf
   (f07hrc).

   Note: the dimension, $dim$, of the array $a_fb$ must be at least max(1, $pdafb \times n$).

   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 $a_fb$.

   Constraint: $pdafb \geq kd + 1$.

10: b[dim] – const Complex

    Input

    Note: the dimension, $dim$, of the array $b$ must be at least

    $\max(1, \text{pdb} \times \text{nrhs})$ when $\text{order} = \text{Nag_ColMajor}$;

    $\max(1, n \times \text{pdb})$ when $\text{order} = \text{Nag_RowMajor}$. 
The $i,j$th element of the matrix $B$ is stored in
\[
\begin{align*}
\text{when } \text{order} &= \text{Nag\_ColMajor; } \\
\text{when } \text{order} &= \text{Nag\_RowMajor}.
\end{align*}
\]

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

11: $\text{pdb}$ – Integer  

Input

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

Constraints:
\[
\begin{align*}
\text{if } \text{order} &= \text{Nag\_ColMajor}, \text{pdb} \geq \max(1,n); \\
\text{if } \text{order} &= \text{Nag\_RowMajor}, \text{pdb} \geq \max(1,nrhs).
\end{align*}
\]

12: $x[\text{dim}]$ – Complex  

Input/Output

Note: the dimension, $\text{dim}$, of the array $x$ must be at least
\[
\begin{align*}
\max(1, \text{pdb} \times \text{nrhs}) \text{ when } \text{order} &= \text{Nag\_ColMajor}; \\
\max(1, n \times \text{pdb}) \text{ when } \text{order} &= \text{Nag\_RowMajor}.
\end{align*}
\]

The $i,j$th element of the matrix $X$ is stored in
\[
\begin{align*}
\text{when } \text{order} &= \text{Nag\_ColMajor; } \\
\text{when } \text{order} &= \text{Nag\_RowMajor}.
\end{align*}
\]

On entry: the $n$ by $r$ solution matrix $X$, as returned by nag\_zpbtrs (f07hsc).

On exit: the improved solution matrix $X$.

13: $\text{pdx}$ – Integer  

Input

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

Constraints:
\[
\begin{align*}
\text{if } \text{order} &= \text{Nag\_ColMajor}, \text{pdx} \geq \max(1,n); \\
\text{if } \text{order} &= \text{Nag\_RowMajor}, \text{pdx} \geq \max(1,nrhs).
\end{align*}
\]

14: $\text{ferr[nrhs]}$ – double  

Output

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 $j = 1,2,\ldots,r$.

15: $\text{berr[nrhs]}$ – double  

Output

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 $j = 1,2,\ldots,r$.

16: $\text{fail}$ – NagError*  

Input/Output

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

6   Error Indicators and Warnings

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}

On entry, argument (value) had an illegal value.
NE_INT

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

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

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

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

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

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

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

NE_INT_2

On entry, \( pdab = \langle \text{value} \rangle \) and \( kd = \langle \text{value} \rangle. \)
Constraint: \( pdab \geq kd + 1. \)

On entry, \( pdafb = \langle \text{value} \rangle \) and \( kd = \langle \text{value} \rangle. \)
Constraint: \( pdafb \geq kd + 1. \)

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

On entry, \( pdb = \langle \text{value} \rangle \) and \( nrhs = \langle \text{value} \rangle. \)
Constraint: \( pdb \geq \max(1, nrhs). \)

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

On entry, \( pdx = \langle \text{value} \rangle \) and \( nrhs = \langle \text{value} \rangle. \)
Constraint: \( pdx \geq \max(1, nrhs). \)

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.

An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

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 Parallelism and Performance

\texttt{nag_zpbrfs (f07hvc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
nag_zpbrfs (f07hvc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 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).

10 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).

10.1 Program Text

```c
/* nag_zpbrfs (f07hvc) Example Program.  
 * Copyright 2014 Numerical Algorithms Group.  
 */

#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;
    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;

#define AB_UPPER(I, J) ab[(J-1)*pdab + k + I + J - 1]
#define AB_LOWER(I, J) ab[(J-1)*pdab + I + J - 1]
#define AFB_UPPER(I, J) afb[(J-1)*pdafb + k + I + J - 1]
#define AFB_LOWER(I, J) afb[(J-1)*pdafb + I + J - 1]
#define B(I, J) b[(J-1)*pdb + J + I - 1]
#define X(I, J) x[(J-1)*pdx + J + I - 1]

order = Nag_ColMajor;
#else
#define AB_UPPER(I, J) ab[(I-1)*pdab + J + I - 1]
#define AB_LOWER(I, J) ab[(I-1)*pdab + J + I - 1]
#define AFB_UPPER(I, J) afb[(I-1)*pdafb + J + I - 1]
#define AFB_LOWER(I, J) afb[(I-1)*pdafb + J + I - 1]
#define B(I, J) b[(I-1)*pdb + J + I - 1]
#define X(I, J) x[(I-1)*pdx + J + I - 1]

order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
printf("nag_zpbrfs (f07hvc) Example Program Results\n\n"); /* Skip heading in data file */
#endif _WIN32
scanf_s("%*\[\n\] ");
#else
scanf("%*\[\n\] ");
#endif _WIN32
	scanf_s("%NAG_IFMT%NAG_IFMT%NAG_IFMT%*\[\n\] ", &n, &kd, &nrhs);
#endif _WIN32
	pdb = n;
pdx = n;
#else
pdb = nrhs;
pdx = nrhs;
#endif

temp = berr_len;
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))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

/* Read A from data file */
#endif _WIN32
scanf_s( "%39s*[\`n] ", nag_enum_arg, _countof(nag_enum_arg));
#endif
/* 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)
        {
            #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &AB_UPPER(i, j).re,
                     &AB_UPPER(i, j).im);
            #else
            scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re,
                     &AB_UPPER(i, j).im);
            #endif
        }
    }
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif

    for (i = 1; i <= n; ++i)
    {
        for (j = MAX(1, i-kd); j <= i; ++j)
        {
            #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &AB_LOWER(i, j).re,
                     &AB_LOWER(i, j).im);
            #else
            scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re,
                     &AB_LOWER(i, j).im);
            #endif
        }
    }
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
}

else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
        {
            #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
            #else
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
            #endif
        }
    }
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
    if (uplo == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = i; j <= MIN(i+kd, n); ++j)
            {
                #ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &AB_UPPER(i, j).re,
                         &AB_UPPER(i, j).im);
                #else
                scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re,
                         &AB_UPPER(i, j).im);
                #endif
            }
        }
        #ifdef _WIN32
        scanf_s("%*[\n]");
        #else
        scanf("%*[\n]");
        #endif
    }
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 AFB */
/* nag_zpbtrf (f07hrc).*/
* Cholesky factorization of complex Hermitian
* positive-definite band matrix
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zpbtrf (f07hrc).\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)
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zpbtrs (f07hsc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
/* nag_zpbbrfs (f07hvc).*/
* Refined solution with error bounds of complex Hermitian
* positive-definite band system of linear equations,
* multiple right-hand sides
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zpbbrfs (f07hvc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print details of solution */
/* nag_gen_complx_mat_print_comp (x04dbc).*/
* Print complex general matrix (comprehensive)
*/
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
nrhs, x, pdx, Nag BracketForm, "%.7f",
"Solution(s)", Nag IntegerLabels,
0, Nag IntegerLabels, 0, 80, 0, 0, &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complex_mat_print_comp (x04dbc)\n",
          fail.message);
    exit_status = 1;
    goto END;
}

printf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%.11le\n", berr[j-1], j%7 == 0?
          "%n\": "");
printf("Estimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%.11le\n", ferr[j-1], j%7 == 0?
          "%n\": "");
printf("\n");

END:
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(ab);
NAG_FREE(afb);
NAG_FREE(b);
NAG_FREE(x);
return exit_status;

10.2 Program Data

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

10.3 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)

8.2e-17 5.4e-17
Estimated forward error bounds (machine-dependent)

3.6e-14 3.0e-14