NAG Library Function Document
nag_zgbrfs (f07bvc)

1 Purpose
nag_zgbrfs (f07bvc) returns error bounds for the solution of a complex band system of linear equations with multiple right-hand sides, $AX = B$, $A^T X = B$ or $A^H X = B$. 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_zgbrfs (Nag_OrderType order, Nag_TransType trans, Integer n,
                 Integer kl, Integer ku, Integer nrhs, const Complex ab[],
                 Integer pdab, const Complex afb[], Integer pdafb,
                 const Integer ipiv[],
                 const Complex b[], Integer pdb, Complex x[], Integer pdx,
                 double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zgbrfs (f07bvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex band system of linear equations with multiple right-hand sides $AX = B$, $A^T X = B$ or $A^H X = B$. The function handles each right-hand side vector (stored as a column of the matrix $B$) independently, so we describe the function of nag_zgbrfs (f07bvc) 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

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

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

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


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:  
   trans – Nag_TransType  
   On entry: indicates the form of the linear equations for which X is the computed solution as follows:
   
   trans = Nag_NoTrans
   The linear equations are of the form AX = B.
   
   trans = Nag_Trans
   The linear equations are of the form ATX = B.
   
   trans = Nag_ConjTrans
   The linear equations are of the form AHX = B.

Constraint: trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

3:  
   n – Integer  
   On entry: n, the order of the matrix A.

Constraint: n ≥ 0.

4:  
   kl – Integer  
   On entry: kl, the number of subdiagonals within the band of the matrix A.

Constraint: kl ≥ 0.

5:  
   ku – Integer  
   On entry: ku, the number of superdiagonals within the band of the matrix A.

Constraint: ku ≥ 0.

6:  
   nrhs – Integer  
   On entry: nrhs, the number of right-hand sides.

Constraint: nrhs ≥ 0.

7:  
   ab[dim] – const Complex  
   Note: the dimension, dim, of the array ab must be at least max(1, pdab × n).

On entry: the original n by n band matrix A as supplied to nag_zgbtrf (f07brc) 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 Aij, for row i = 1, ..., n and column j = max(1, i - kl), ..., min(n, i + ku), depends on the order argument as follows:

   if order = Nag_ColMajor, Aij is stored as ab[(j - 1) × pdab + ku + i - j];
   
   if order = Nag_RowMajor, Aij is stored as ab[(i - 1) × pdab + kl + j - i].

See Section 9 in nag_zgbsv (f07bnc) for further details.

8:  
   pdab – Integer  
   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 ≥ kl + ku + 1.
9: \texttt{afb[dim]} – const Complex

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

On entry: the LU factorization of \(A\), as returned by \texttt{nag_zgbtrf (f07brc)}.

10: \texttt{pdafb} – Integer

On entry: the stride separating row or column elements (depending on the value of \texttt{order}) of the matrix in the array \texttt{afb}.

Constraint: \(\texttt{pdafb} \geq 2 \times \text{kl} + \text{ku} + 1\).

11: \texttt{ipiv[dim]} – const Integer

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

On entry: the pivot indices, as returned by \texttt{nag_zgbtrf (f07brc)}.

12: \texttt{b[dim]} – const Complex

Note: the dimension, \texttt{dim}, of the array \texttt{b} must be at least

\[
\max(1, \texttt{pdb} \times \texttt{nrhs}) \text{ when } \texttt{order} = \texttt{Nag\_ColMajor};
\max(1, n \times \texttt{pdb}) \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\]

The \((i,j)\)th element of the matrix \(B\) is stored in

\[
\texttt{b[(j - 1) x pdb + i - 1]} \text{ when } \texttt{order} = \texttt{Nag\_ColMajor};
\]

\[
\texttt{b[(i - 1) x pdb + j - 1]} \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\]

On entry: the \(n\) by \(r\) right-hand side matrix \(B\).

13: \texttt{pdb} – Integer

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

Constraints:

\[
\text{if } \texttt{order} = \texttt{Nag\_ColMajor}, \texttt{pdb} \geq \max(1, n);
\]
\[
\text{if } \texttt{order} = \texttt{Nag\_RowMajor}, \texttt{pdb} \geq \max(1, \texttt{nrhs}).
\]

14: \texttt{x[dim]} – Complex

Note: the dimension, \texttt{dim}, of the array \texttt{x} must be at least

\[
\max(1, \texttt{pdx} \times \texttt{nrhs}) \text{ when } \texttt{order} = \texttt{Nag\_ColMajor};
\max(1, n \times \texttt{pdx}) \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\]

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

\[
\texttt{x[(j - 1) x pdx + i - 1]} \text{ when } \texttt{order} = \texttt{Nag\_ColMajor};
\]

\[
\texttt{x[(i - 1) x pdx + j - 1]} \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\]

On entry: the \(n\) by \(r\) solution matrix \(X\), as returned by \texttt{nag_zgbtrs (f07bsc)}.

On exit: the improved solution matrix \(X\).

15: \texttt{pdx} – Integer

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

Constraints:

\[
\text{if } \texttt{order} = \texttt{Nag\_ColMajor}, \texttt{pdx} \geq \max(1, n);
\]
\[
\text{if } \texttt{order} = \texttt{Nag\_RowMajor}, \texttt{pdx} \geq \max(1, \texttt{nrhs}).
\]
16:  \texttt{ferr[nrhs]} – double

\textit{Output}

\textit{On exit:} \texttt{ferr}[j - 1] contains an estimated error bound for the \textit{j}th solution vector, that is, the \textit{j}th column of \textit{X}, for \textit{j} = 1, 2, \ldots, \textit{r}.

17:  \texttt{berr[nrhs]} – double

\textit{Output}

\textit{On exit:} \texttt{berr}[j - 1] contains the component-wise backward error bound \textit{\beta} for the \textit{j}th solution vector, that is, the \textit{j}th column of \textit{X}, for \textit{j} = 1, 2, \ldots, \textit{r}.

18:  \texttt{fail} – \texttt{NagError*}  

\textit{Input/Output}

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

6  \textbf{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 \textit{\langle value\rangle} had an illegal value.

\textbf{NE_INT}

On entry, \texttt{kl} = \textit{\langle value\rangle}.

Constraint: \texttt{kl} \geq 0.

On entry, \texttt{ku} = \textit{\langle value\rangle}.

Constraint: \texttt{ku} \geq 0.

On entry, \texttt{n} = \textit{\langle value\rangle}.

Constraint: \texttt{n} \geq 0.

On entry, \texttt{nrhs} = \textit{\langle value\rangle}.

Constraint: \texttt{nrhs} \geq 0.

On entry, \texttt{pdab} = \textit{\langle value\rangle}.

Constraint: \texttt{pdab} > 0.

On entry, \texttt{pdafb} = \textit{\langle value\rangle}.

Constraint: \texttt{pdafb} > 0.

On entry, \texttt{pdb} = \textit{\langle value\rangle}.

Constraint: \texttt{pdb} > 0.

On entry, \texttt{pdx} = \textit{\langle value\rangle}.

Constraint: \texttt{pdx} > 0.

\textbf{NE_INT_2}

On entry, \texttt{pdb} = \textit{\langle value\rangle} and \texttt{n} = \textit{\langle value\rangle}.

Constraint: \texttt{pdb} \geq \texttt{max(1, n)}.

On entry, \texttt{pdb} = \textit{\langle value\rangle} and \texttt{nrhs} = \textit{\langle value\rangle}.

Constraint: \texttt{pdb} \geq \texttt{max(1, nrhs)}.

On entry, \texttt{pdx} = \textit{\langle value\rangle} and \texttt{n} = \textit{\langle value\rangle}.

Constraint: \texttt{pdx} \geq \texttt{max(1, n)}.

On entry, \texttt{pdx} = \textit{\langle value\rangle} and \texttt{nrhs} = \textit{\langle value\rangle}.

Constraint: \texttt{pdx} \geq \texttt{max(1, nrhs)}. 
NE_INT_3

On entry, \( \text{pdab} = (\text{value}) \), \( \text{kl} = (\text{value}) \) and \( \text{ku} = (\text{value}) \).
Constraint: \( \text{pdab} \geq \text{kl} + \text{ku} + 1. \)

On entry, \( \text{pdafb} = (\text{value}) \), \( \text{kl} = (\text{value}) \) and \( \text{ku} = (\text{value}) \).
Constraint: \( \text{pdafb} \geq 2 \times \text{kl} + \text{ku} + 1. \)

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 \( \text{ferr} \) are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

8 Parallelism and Performance

\text{nag_zgbrfs (f07bvc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\text{nag_zgbrfs (f07bvc)} 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 \( 16n(k_l + k_u) \) real floating-point operations. Each step of iterative refinement involves an additional \( 8n(4k_l + 3k_u) \) real operations. This assumes \( n \gg k_l \) and \( n \gg k_u \). 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 \( A x = b \) or \( A^T x = b \); the number is usually 5 and never more than 11. Each solution involves approximately \( 8n(2k_l + k_u) \) real operations.

The real analogue of this function is \text{nag_dgbrfs (f07bhc)}.

10 Example

This example solves the system of equations \( AX = B \) using iterative refinement and to compute the forward and backward error bounds, where

\[
A = \begin{pmatrix}
-1.65 + 2.26i & -2.05 - 0.85i & 0.97 - 2.84i & 0.00 + 0.00i \\
0.00 + 6.30i & -1.48 + 1.48i & -3.99 + 4.01i & 0.59 - 0.48i \\
0.00 + 0.00i & -0.77 + 2.83i & -1.06 + 1.94i & 3.33 - 1.04i \\
0.00 + 0.00i & 0.00 + 0.00i & 4.48 - 1.09i & -0.46 - 1.72i
\end{pmatrix}
\]

and
Here \( A \) is nonsymmetric and is treated as a band matrix, which must first be factorized by \texttt{nag_zgbtrf} (f07brc).

### 10.1 Program Text

```c
/* nag_zgbrfs (f07bvc) Example Program. */
* Copyright 2014 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 */
  Complex *ab = 0, *afb = 0, *b = 0, *x = 0;
  double *berr = 0, *ferr = 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);
  printf("nag_zgbrfs (f07bvc) Example Program Results\n\n");

  /* Skip heading in data file */
  #ifdef _WIN32
  scanf_s("%*[\n"]);
  #else
  scanf("%*[\n"]);
  #endif
  #ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n"] , &n, &nrhs, &kl, &ku);
  #else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n"] , &n, &nrhs, &kl, &ku);
  #endif
  ipiv_len = n;
```
pdab = k1 + ku + 1;
pdafb = 2*k1 + ku + 1;
#ifdef NAG_COLUMN_MAJOR
pdb = n;
pdx = n;
#else
pdb = nrhs;
pdx = nrhs;
#endif

/* Allocate memory */
if (!(!ab = NAG_ALLOC((k1+ku+1) * n, Complex)) ||
!(!afb = NAG_ALLOC((2*k1+ku+1) * n, Complex)) ||
!(!b = NAG_ALLOC(nrhs * n, Complex)) ||
!(!x = NAG_ALLOC(nrhs * n, Complex)) ||
!(!berr = NAG_ALLOC(nrhs, double)) ||
!(!ferr = NAG_ALLOC(nrhs, double)) ||
!(!ipiv = NAG_ALLOC(ipiv_len, Integer)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
/* Set A to zero to avoid referencing uninitialized elements */
for (i = 0; i < n*(k1+ku+1); ++i)
{
ab[i].re = 0.0;
ab[i].im = 0.0;
}
/* Read A from data file */
for (i = 1; i <= n; ++i)
{
for (j = MAX(i-k1, 1); j <= MIN(i+ku, n); ++j)
#ifdef _WIN32
scanf_s(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
#else
scanf(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
#endif
}
#ifdef _WIN32
scanf_s("%*[^
] ");
#else
scanf("%*[^
] ");
#endif
/* Read B from data file */
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
/* Copy A to AFB and B to X */
for (i = 1; i <= n; ++i)
{
for (j = MAX(i-k1, 1); j <= MIN(i+ku, n); ++j)
{
AFB(i, j).re = AB(i, j).re;
AFB(i, j).im = AB(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_zgbtrf (f07brc). */
{ 
    nag_zgbtrf(order, n, n, kl, ku, afb, pdafb, ipiv, &fail);
    if (fail.code != NE_NOERROR) 
    { 
        printf("Error from nag_zgbtrf (f07brc).\n%s\n", fail.message); 
        exit_status = 1; 
        goto END; 
    }
}

/* Compute solution in the array X */
/* nag_zgbtrs (f07bsc). */
{ 
    nag_zgbtrs(order, Nag_NoTrans, n, kl, ku, nrhs, afb, pdafb, ipiv, 
                x, pdx, &fail);
    if (fail.code != NE_NOERROR) 
    { 
        printf("Error from nag_zgbtrs (f07bsc).\n%s\n", fail.message); 
        exit_status = 1; 
        goto END; 
    }
}

/* Improve solution, and compute backward errors and */)/* estimated bounds on the forward errors */
/* nag_zgbrfs (f07bvc). */
{ 
    nag_zgbrfs(order, Nag_NoTrans, n, kl, ku, nrhs, ab, pdab, afb, pdafb, 
                ipiv, b, pdb, x, pdx, ferr, berr, &fail);
    if (fail.code != NE_NOERROR) 
    { 
        printf("Error from nag_zgbrfs (f07bvc).\n%s\n", fail.message); 
        exit_status = 1; 
        goto END; 
    }
}

/* Print solution */
/* nag_gen_complx_mat_print_comp (x04dbc). */
{ 
    nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NoUnitDiag, n, 
                               nrhs, x, pdx, NagBracketForm, "%7.4f", 
                               "Solution(s)", Nag_IntegerLabels, 
                               0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
    if (fail.code != NE_NOERROR) 
    { 
        printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n", fail.message); 
        exit_status = 1; 
        goto END; 
    }
}

/* Print forward and backward errors */
printf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j) 
    printf("%11.1e%s", berr[j-1], j%7 == 0?"\n": "");
printf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j) 
    printf("%11.1e%s", ferr[j-1], j%7 == 0?"\n": "");
printf("\n");
END:
NAG_FREE(ab);
NAG_FREE(afb);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(ipiv);
return exit_status;
}

## 10.2 Program Data

*nag_zgbrfs (f07bvc)* Example Program Data

<table>
<thead>
<tr>
<th>N</th>
<th>NRHS</th>
<th>KL</th>
<th>KU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Values of N, NRHS, KL and KU

\[
\begin{align*}
(-1.65, 2.26) & (-2.05,-0.85) ( 0.97,-2.84) \\
( 0.00, 6.30) & (-1.48, 1.75) ( 3.99, 4.01) ( 0.59,-0.48) \\
(-0.77, 2.83) & (-1.06, 1.94) ( 3.33,-1.04) \\
( 4.48,-1.09) & (-0.46,-1.72) \\
\end{align*}
\]

:End of matrix A

\[
\begin{align*}
(-1.06, 21.50) & ( 12.85, 2.84) \\
(-22.72,-53.90) & (-70.22, 21.57) \\
( 28.24,-38.60) & (-20.73, -1.23) \\
(-34.56, 16.73) & ( 26.01, 31.97) \\
\end{align*}
\]

:End of matrix B

## 10.3 Program Results

*nag_zgbrfs (f07bvc)* Example Program Results

### Solution(s)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-3.0000, 2.0000)</td>
<td>(1.0000, 6.0000)</td>
</tr>
<tr>
<td>2</td>
<td>( 1.0000, -7.0000)</td>
<td>(-7.0000, 4.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(-5.0000, 4.0000)</td>
<td>( 3.0000, 5.0000)</td>
</tr>
<tr>
<td>4</td>
<td>( 6.0000, -8.0000)</td>
<td>(-8.0000, 2.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8e-17</td>
<td>6.7e-17</td>
</tr>
</tbody>
</table>

Estimated forward error bounds (machine-dependent)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5e-14</td>
<td>4.3e-14</td>
</tr>
</tbody>
</table>