NAG C 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, $AX = B$, $A^TX = B$ or $AHX = B$. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```c
void nag_zgbrfs (Nag_OrderType order, Nag_TransType trans, Integer n, Integer kl, Integer ku, Integer nrhs, const Complex ab[], Integer pdab, const Complexafb[], 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 $AX = B$, $A^TX = B$ or $AHX = B$. 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

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

$$|\delta a_{ij}| \leq \beta |a_{ij}|$$
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 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.
trans – Nag_TransType  

On entry: indicates the form of the linear equations for which \( X \) is the computed solution as follows:

- if \( \text{trans} = \text{Nag\_NoTrans} \), the linear equations are of the form \( AX = B \);
- if \( \text{trans} = \text{Nag\_Trans} \), the linear equations are of the form \( A^T X = B \);
- if \( \text{trans} = \text{Nag\_ConjTrans} \), the linear equations are of the form \( A^H X = B \).

Constraint: \( \text{trans} = \text{Nag\_NoTrans}, \text{Nag\_Trans} \) or \( \text{Nag\_ConjTrans} \).

n – Integer

On entry: \( n \), the order of the matrix \( A \).

Constraint: \( n \geq 0 \).

kl – Integer

On entry: \( k_l \), the number of sub-diagonals within the band of \( A \).

Constraint: \( kl \geq 0 \).

ku – Integer

On entry: \( k_u \), the number of super-diagonals within the band of \( A \).

Constraint: \( ku \geq 0 \).

nrhs – Integer

On entry: \( r \), the number of right-hand sides.

Constraint: \( nrhs \geq 0 \).

\( \text{ab}[\text{dim}] \) – const Complex

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

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 \( a_{ij} \), for \( i = 1, \ldots, n \) and \( j = \max(1, i - k_l), \ldots, \min(n, i + k_u) \), depends on the order parameter as follows:

- if \( \text{order} = \text{Nag\_ColMajor} \), \( a_{ij} \) is stored as \( \text{ab}[(j - 1) \times \text{pdab} + ku + i - j] \);
- if \( \text{order} = \text{Nag\_RowMajor} \), \( a_{ij} \) is stored as \( \text{ab}[(i - 1) \times \text{pdab} + kl + j - i] \).

pdab – Integer

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

Constraint: \( \text{pdab} \geq kl + ku + 1 \).

afb[\text{dim}] – const Complex

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

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

pdafb – Integer

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

Constraint: \( \text{pdafb} \geq 2 \times kl + ku + 1 \).
The NAG error parameter (see the Essential Introduction).
6 Error Indicators and Warnings

NE_INT

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

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

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

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

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

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

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

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

NE_INT_2

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

On entry, \( pda = \langle \text{value} \rangle \), \( n_{rhs} = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, n_{rhs}) \).

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

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

NE_INT_3

On entry, \( pda_b = \langle \text{value} \rangle \), \( k_l = \langle \text{value} \rangle \), \( k_u = \langle \text{value} \rangle \).
Constraint: \( pda_b \geq k_l + k_u + 1 \).

On entry, \( pdaf_b = \langle \text{value} \rangle \), \( k_l = \langle \text{value} \rangle \), \( k_u = \langle \text{value} \rangle \).
Constraint: \( pdaf_b \geq 2 \times k_l + k_u + 1 \).

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter \( \langle \text{value} \rangle \) 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 \texttt{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 $16n(k_l + k_u)$ 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 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 $Ax = b$ or $A^Tx = b$; 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 nag_dgbrefs (f07bhc).

9 Example

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

$$A = \begin{pmatrix} -1.65 + 2.26i & -2.05 - 0.85i & 0.97 - 2.84i & 0.00 + 0.00i \\ 0.00 + 6.30i & -1.48 - 1.75i & -3.99 + 4.01i & 0.59 - 0.48i \\ 0.00 + 0.00i & -0.77 + 2.83i & -1.06 + 1.94i & 3.33 - 1.04i \\ 0.00 + 0.00i & 0.00 + 0.00i & 4.48 - 1.09i & -0.46 - 1.72i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -1.06 + 21.50i & 12.85 + 2.84i \\ -22.72 - 53.90i & -70.22 + 21.57i \\ 28.24 - 38.60i & -20.73 - 1.23i \\ -34.56 + 16.73i & 26.01 + 31.97i \end{pmatrix}.$$ 

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

9.1 Program Text

/* nag_zgbrfs (f07bvc) Example Program. */
* Copyright 2001 Numerical Algorithms Group.
*/
#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]
#endif

INIT_FAIL(fail);
Vprintf("f07bvc Example Program Results\n\n");
/* Skip heading in data file */
Vscanf("%*[\n"]);
Vscanf("%ld%ld%ld%ld%*[\n"] , &n, &nrhs, &kl, &ku);
ipiv_len = n;
pdab = kl + ku + 1;
pdafb = 2*kl + ku + 1;
#else
#define pdb = n;
#endif

dpdb = nrhs;
#endif

/* Allocate memory */
if ( !(ab = NAG_ALLOC((kl+ku+1) * n, Complex)) ||
     !(afb = NAG_ALLOC((2*kl+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))   )
{
  Vprintf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Set A to zero to avoid referencing uninitialized elements */
for (i = 0; i < n*(kl+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-kl,1); j <= MIN(i+ku,n); ++j)
    Vscanf(" ( %lf , %lf )", &AB(i,j).re, &AB(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 AFB and B to X */
for (i = 1; i <= n; ++i)
{
  for (j = MAX(i-kl,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 */
f07brc(order, n, n, kl, ku, afb, pdafb, ipiv, &fail);
if (fail.code != NE_NOERROR)
    { 
        Vprintf("Error from f07brc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

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

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

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

/* Print forward and backward errors */
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 (ab) NAG_FREE(ab);
if (afb) NAG_FREE(afb);
if (b) NAG_FREE(b);
if (x) NAG_FREE(x);
if (berr) NAG_FREE(berr);
if (ferr) NAG_FREE(ferr);
if (ipiv) NAG_FREE(ipiv);
return exit_status;
}
9.2 Program Data

Example Program Data

Values of N, NRHS, KL and KU

( 4.00, -0.06 ) ( -1.00, -2.00 ) ( 1.00, 0.00 ) ( 0.00, 0.00 )
( 0.00, 0.00 ) ( 1.00, 2.00 ) ( -1.00, 0.00 ) ( 0.00, 0.00 )
( 0.00, 0.00 ) ( 0.00, 0.00 ) ( 1.00, 2.00 ) ( -1.00, 0.00 )
( 0.00, 0.00 ) ( 0.00, 0.00 ) ( 0.00, 0.00 ) ( 1.00, 2.00 )

End of matrix A

9.3 Program Results

Example Program Results

Solution(s)

1
2

Backward errors (machine-dependent)

9.8e-17 7.2e-17

Estimated forward error bounds (machine-dependent)

3.7e-14 4.4e-14