NAG C Library Function Document
nag_zgbtrs (f07bsc)

1 Purpose
nag_zgbtrs (f07bsc) solves a complex band system of linear equations with multiple right-hand sides, 
\[ AX = B, \quad A^T X = B \text{ or } A^H X = B, \]
where \( A \) has been factorized by nag_zgbtrf (f07brc).

2 Specification

```c
void nag_zgbtrs (Nag_OrderType order, Nag_TransType trans, Integer n, Integer kl,
                Integer ku, Integer nrhs, const Complex ab[], Integer pdab,
                const Integer ipiv[], Complex b[], Integer pdb, NagError *fail)
```

3 Description
To solve a complex band system of linear equations \( AX = B \), \( A^T X = B \) or \( A^H X = B \), this function must
be preceded by a call to nag_zgbtrf (f07brc) which computes the \( LU \) factorization of \( A \) as \( A = PLU \).
The solution is computed by forward and backward substitution.

If \( trans = \text{Nag\_NoTrans} \), the solution is computed by solving \( PLY = B \) and then \( UX = Y \).
If \( trans = \text{Nag\_Trans} \), the solution is computed by solving \( U^T Y = B \) and then \( L^T P^T X = Y \).
If \( trans = \text{Nag\_ConjTrans} \), the solution is computed by solving \( U^H Y = B \) and then \( L^H P^T X = Y \).

4 References

5 Parameters
1: order – Nag_OrderType
   
   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: trans – Nag_TransType
   
   On entry: indicates the form of the equations as follows:
      
      if trans = Nag_NoTrans, \( AX = B \) is solved for \( X \);
      
      if trans = Nag_Trans, \( A^T X = B \) is solved for \( X \);
      
      if trans = Nag_ConjTrans, \( A^H X = B \) is solved for \( X \).
   
   Constraint: trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

3: n – Integer
   
   On entry: \( n \), the order of the matrix \( A \).
   
   Constraint: \( n \geq 0 \).
4: \( k_l \) – Integer \hspace{2cm} Input

On entry: \( k_l \), the number of sub-diagonals within the band of \( A \).
Constraint: \( k_l \geq 0 \).

5: \( k_u \) – Integer \hspace{2cm} Input

On entry: \( k_u \), the number of super-diagonals within the band of \( A \).
Constraint: \( k_u \geq 0 \).

6: \( n_{rhs} \) – Integer \hspace{2cm} Input

On entry: \( r \) the number of right-hand sides.
Constraint: \( n_{rhs} \geq 0 \).

7: \( ab[\text{dim}] \) – const Complex \hspace{2cm} Input

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

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

8: \( pdab \) – Integer \hspace{2cm} Input

On entry: the stride separating row or column elements (depending on the value of \text{order}) of the matrix in the array \( ab \).
Constraint: \( pdab \geq 2 \times k_l + k_u + 1 \).

9: \( ipiv[\text{dim}] \) – const Integer \hspace{2cm} Input

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

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

10: \( b[\text{dim}] \) – Complex \hspace{2cm} Input/Output

Note: the dimension, \( \text{dim} \), of the array \( b \) must be at least \( \text{max}(1,pdab \times n_{rhs}) \) when \text{order} = \text{Nag_ColMajor} and at least \( \text{max}(1,pdab \times n) \) when \text{order} = \text{Nag_RowMajor}.

If \text{order} = \text{Nag_ColMajor}, the \((i,j)\)th element of the matrix \( B \) is stored in \( b[(j - 1) \times pdab + i - 1] \) and if \text{order} = \text{Nag_RowMajor}, the \((i,j)\)th element of the matrix \( B \) is stored in \( b[(i - 1) \times pdab + j - 1] \).

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

11: \( pdb \) – Integer \hspace{2cm} Input

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

Constraints:
- if \text{order} = \text{Nag_ColMajor}, \( pdb \geq \text{max}(1,n) \);
- if \text{order} = \text{Nag_RowMajor}, \( pdb \geq \text{max}(1,n_{rhs}) \).

12: \( fail \) – NagError * \hspace{2cm} Output

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

\text{NE_INT}

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

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

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

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

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

\textbf{NE_INT_2}

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

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

\textbf{NE_INT_3}

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

\textbf{NE_ALLOC_FAIL}
Memory allocation failed.

\textbf{NE_BAD_PARAM}
On entry, parameter \( \langle \text{value} \rangle \) had an illegal value.

\textbf{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

For each right-hand side vector \( b \), the computed solution \( x \) is the exact solution of a perturbed system of equations \((A + E)x = b\), where

\[ |E| \leq c(k)\epsilon|L||U|, \]

\( c(k) \) is a modest linear function of \( k = kl + ku + 1 \), and \( \epsilon \) is the \textit{machine precision}. This assumes \( k \ll n \).

If \( \hat{x} \) is the true solution, then the computed solution \( x \) satisfies a forward error bound of the form

\[ \frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq c(k) \text{cond}(A, x)\epsilon \]

where \( \text{cond}(A, x) = \|A^{-1}\|A\|x\|_\infty / \|x\|_\infty \leq \text{cond}(A) = \|A^{-1}\|A\|\leq \kappa_\infty(A) \). Note that \( \text{cond}(A, x) \) can be much smaller than \( \text{cond}(A) \), and \( \text{cond}(A^T) \) (which is the same as \( \text{cond}(A^T) \)) can be much larger (or smaller) than \( \text{cond}(A) \).

Forward and backward error bounds can be computed by calling \texttt{nag_zgbrfs (f07bvc)}, and an estimate for \( \kappa_\infty(A) \) can be obtained by calling \texttt{nag_zgbcon (f07buc)} with \( \text{norm} = \text{Nag_InfNorm} \).

8 Further Comments

The total number of real floating-point operations is approximately \( 8n(2kl + ku)r \), assuming \( n \gg kl \) and \( n \gg ku \).
This function may be followed by a call to nag_zgbrfs (f07bvc) to refine the solution and return an error estimate.

The real analogue of this function is nag_dgbtrs (f07bec).

9 Example

To solve the system of equations $AX = B$, 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

$$B = \begin{pmatrix}
-1.06 + 21.50i & 12.85 + 2.84i \\
-22.72 - 53.90i & -70.22 + 21.57i \\
28.24 - 38.60i & -20.7 - 31.23i \\
-34.56 + 16.73i & 26.01 + 31.97i
\end{pmatrix}.$$ 

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_zgbtrs (f07bsc) 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, ipiv_len, j, kl, ku, n, nrhs, pdab, pdb;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Complex *ab=0, *b=0;
    Integer *ipiv=0;

    #ifdef NAG_COLUMN_MAJOR
    #define AB(I,J) ab[(J-1)*pdab + kl + ku +I-J]
    #define B(I,J) b[(J-1)*pdb +I-1]
    #else
    #define AB(I,J) ab[(I-1)*pdab + kl + J - I]
    #define B(I,J) b[(I-1)*pdb +J-1]
    #endif

    INIT_FAIL(fail);
    Vprintf("f07bsc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[\n "]);
    Vscanf("%ld%ld%ld%ld%*[\n ]", &n, &nrhs, &kl, &ku);

    /* Other code... */

    return exit_status;
}
ipiv_len = n;
pdab = 2*kl + ku + 1;
#ifdef NAG_COLUMN_MAJOR
pdb = n;
#else
pdb = nrhs;
#endif

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

/* 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 ]");

/* Factorize A */
f07brc(order, n, n, kl, ku, ab, pdab, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f07brc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Compute solution */
f07bsc(order, Nag_NoTrans, n, kl, ku, nrhs, ab, pdab, ipiv,
        b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f07bsc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print solution */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb,
        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;
}

END:
if (ab) NAG_FREE(ab);
if (b) NAG_FREE(b);
if (ipiv) NAG_FREE(ipiv);
return exit_status;

9.2 Program Data

f07bsc Example Program Data

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1.65, 2.26)</td>
<td>(-2.05, -0.85)</td>
<td>(0.97, -2.84)</td>
<td></td>
</tr>
<tr>
<td>(0.00, 6.30)</td>
<td>(-1.48, -1.75)</td>
<td>(-3.99, 4.01)</td>
<td>(0.59, -0.48)</td>
</tr>
<tr>
<td>(-0.77, 2.83)</td>
<td>(-1.06, 1.94)</td>
<td>(3.33, -1.04)</td>
<td></td>
</tr>
<tr>
<td>(4.48, -1.09)</td>
<td>(-0.46, -1.72)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

:Values of N, NRHS, KL and KU

| (-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 of matrix A

9.3 Program Results

f07bsc Example Program Results

Solution(s)

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

:End of matrix B