NAG Library Function Document

nag_zgbtrs (f07bsc)

1 Purpose

nag_zgbtrs (f07bsc) solves a complex band system of linear equations with multiple right-hand sides,

$$AX = B, \quad A^{\mathrm{T}}X = B \quad \text{or} \quad A^{\mathrm{H}}X = B,$$

where A has been factorized by nag_zgbtrf (f07brc).

2 Specification

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

```
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

nag_zgbtrs (f07bsc) is used to solve a complex band system of linear equations $AX = B$, $A^{\mathrm{T}}X = B$ or $A^{\mathrm{H}}X = B$, the 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** = Nag_NoTrans, the solution is computed by solving $PLY = B$ and then $UX = Y$.

If **trans** = Nag_Trans, the solution is computed by solving $U^{\mathrm{T}}Y = B$ and then $L^{\mathrm{T}}P^{\mathrm{T}}X = Y$.

If **trans** = Nag_ConjTrans, the solution is computed by solving $U^{\mathrm{H}}Y = B$ and then $L^{\mathrm{H}}P^{\mathrm{T}}X = Y$.

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: **trans** – Nag_TransType *Input*

On entry: indicates the form of the equations.

trans = Nag_NoTrans
$AX = B$ is solved for X.

trans = Nag_Trans
$A^{\mathrm{T}}X = B$ is solved for X.

trans = Nag_ConjTrans
$A^{\mathrm{H}}X = B$ is solved for X.

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

3: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: **n** ≥ 0.

4: **kl** – Integer *Input*

On entry: k_l, the number of subdiagonals within the band of the matrix A.

Constraint: **kl** ≥ 0.

5: **ku** – Integer *Input*

On entry: k_u, the number of superdiagonals within the band of the matrix A.

Constraint: **ku** ≥ 0.

6: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0.

7: **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 LU factorization of A, as returned by nag_zgbtrf (f07brc).

8: **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: $\mathbf{pdab} \geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

9: **ipiv**[*dim*] – const Integer *Input*

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

On entry: the pivot indices, as returned by nag_zgbtrf (f07brc).

10: **b**[*dim*] – Complex *Input/Output*

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

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ 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.

On exit: the n by r solution matrix X.

11: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdb** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

12: **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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **kl** = ⟨*value*⟩.
Constraint: **kl** ≥ 0.

On entry, **ku** = ⟨*value*⟩.
Constraint: **ku** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **nrhs** = ⟨*value*⟩.
Constraint: **nrhs** ≥ 0.

On entry, **pdab** = ⟨*value*⟩.
Constraint: **pdab** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

NE_INT_2

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = ⟨*value*⟩ and **nrhs** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{nrhs})$.

NE_INT_3

On entry, **pdab** = ⟨*value*⟩, **kl** = ⟨*value*⟩ and **ku** = ⟨*value*⟩.
Constraint: **pdab** $\geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

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.

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 $k = k_l + k_u + 1$, and ϵ is the ***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

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq c(k)\,\mathrm{cond}(A,x)\epsilon$$

where $\mathrm{cond}(A,x) = \left\||A^{-1}||A||x|\right\|_\infty / \|x\|_\infty \leq \mathrm{cond}(A) = \left\||A^{-1}||A|\right\|_\infty \leq \kappa_\infty(A)$.

Note that $\mathrm{cond}(A,x)$ can be much smaller than $\mathrm{cond}(A)$, and $\mathrm{cond}(A^{\mathrm{H}})$ (which is the same as $\mathrm{cond}(A^{\mathrm{T}})$) can be much larger (or smaller) than $\mathrm{cond}(A)$.

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

8 Parallelism and Performance

nag_zgbtrs (f07bsc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgbtrs (f07bsc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of real floating-point operations is approximately $8n(2k_l + k_u)r$, assuming $n \gg k_l$ and $n \gg k_u$.

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).

10 Example

This example solves 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).

10.1 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]
  order = Nag_ColMajor;
#else
#define AB(I, J) ab[(I-1)*pdab + kl + J - I]
#define B(I, J)  b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zgbtrs (f07bsc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%ld%*[^\n] ", &n, &nrhs,
         &kl, &ku);
  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)))
    {
      printf("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)
        scanf(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
    }
  scanf("%*[^\n] ");
  /* Read B from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }
  scanf("%*[^\n] ");

  /* Factorize A */
  /* nag_zgbtrf (f07brc).
   * LU factorization of complex m by n band matrix
   */
```

```
  nag_zgbtrf(order, n, n, kl, ku, ab, pdab, 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 */
  /* nag_zgbtrs (f07bsc).
   * Solution of complex band system of linear equations,
   * multiple right-hand sides, matrix already factorized by
   * nag_zgbtrf (f07brc)
   */
  nag_zgbtrs(order, Nag_NoTrans, n, kl, ku, nrhs, ab, pdab, ipiv,
             b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgbtrs (f07bsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print 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, b, pdb, Nag_BracketForm, "%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;
    }
 END:
  NAG_FREE(ab);
  NAG_FREE(b);
  NAG_FREE(ipiv);
  return exit_status;
}
```

10.2 Program Data

```
nag_zgbtrs (f07bsc) Example Program Data
  4  2  1  2                                       :Values of N, NRHS, KL and KU
 (-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 of matrix A
 ( -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 B
```

10.3 Program Results

```
nag_zgbtrs (f07bsc) Example Program Results

 Solution(s)
                    1                   2
 1  (-3.0000, 2.0000)  ( 1.0000, 6.0000)
 2  ( 1.0000,-7.0000)  (-7.0000,-4.0000)
 3  (-5.0000, 4.0000)  ( 3.0000, 5.0000)
 4  ( 6.0000,-8.0000)  (-8.0000, 2.0000)
```