NAG Library Function Document

nag_zgbsv (f07bnc)

1 Purpose

nag_zgbsv (f07bnc) computes the solution to a complex system of linear equations

$$AX = B,$$

where A is an n by n band matrix, with k_l subdiagonals and k_u superdiagonals, and X and B are n by r matrices.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_zgbsv (Nag_OrderType order, Integer n, Integer kl, Integer ku,
     Integer nrhs, Complex ab[], Integer pdab, Integer ipiv[], Complex b[],
     Integer pdb, NagError *fail)
```

3 Description

nag_zgbsv (f07bnc) uses the LU decomposition with partial pivoting and row interchanges to factor A as $A = PLU$, where P is a permutation matrix, L is a product of permutation and unit lower triangular matrices with k_l subdiagonals, and U is upper triangular with $(k_l + k_u)$ superdiagonals. The factored form of A is then used to solve the system of equations $AX = B$.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

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: **n** – Integer *Input*

On entry: n, the number of linear equations, i.e., the order of the matrix A.

Constraint: **n** ≥ 0.

3: **kl** – Integer *Input*

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

Constraint: **kl** ≥ 0.

4: **ku** – Integer *Input*

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

Constraint: **ku** ≥ 0.

5: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

Constraint: **nrhs** ≥ 0.

6: **ab**[dim] – Complex *Input/Output*

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

On entry: the n by n coefficient matrix A.

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements A_{ij}, for row $i = 1, \ldots, n$ and column $j = \max(1, i - k_l), \ldots, \min(n, i + k_u)$, depends on the **order** argument as follows:

if **order** = Nag_ColMajor, A_{ij} is stored as **ab**$[(j-1) \times \textbf{pdab} + \textbf{kl} + \textbf{ku} + i - j]$;

if **order** = Nag_RowMajor, A_{ij} is stored as **ab**$[(i-1) \times \textbf{pdab} + \textbf{kl} + j - i]$.

See Section 9 for further details.

On exit: **ab** is overwritten by details of the factorization.

The elements, u_{ij}, of the upper triangular band factor U with $k_l + k_u$ super-diagonals, and the multipliers, l_{ij}, used to form the lower triangular factor L are stored. The elements u_{ij}, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_l + k_u)$, and l_{ij}, for $i = 1, \ldots, n$ and $j = \max(1, i - k_l), \ldots, i$, are stored where A_{ij} is stored on entry.

7: **pdab** – Integer *Input*

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** $\geq 2 \times \textbf{kl} + \textbf{ku} + 1$.

8: **ipiv**[**n**] – Integer *Output*

On exit: if no constraints are violated, the pivot indices that define the permutation matrix P; at the ith step row i of the matrix was interchanged with row **ipiv**$[i-1]$. **ipiv**$[i-1] = i$ indicates a row interchange was not required.

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

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

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

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

b$[(j-1) \times \textbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
b$[(i-1) \times \textbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

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

On exit: if **fail.code** = NE_NOERROR, the n by r solution matrix X.

10: **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** $\geq \max(1, \mathbf{nrhs})$.

11: **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.
See Section 3.2.1.2 in the Essential Introduction for further information.

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.

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.

NE_SINGULAR

Element ⟨*value*⟩ of the diagonal is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

7 Accuracy

The computed solution for a single right-hand side, $\hat{x}$, satisfies an equation of the form

$$(A + E)\hat{x} = b,$$

where

$$\|E\|_1 = O(\epsilon)\|A\|_1$$

and ϵ is the ***machine precision***. An approximate error bound for the computed solution is given by

$$\frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A)\frac{\|E\|_1}{\|A\|_1},$$

where $\kappa(A) = \left\|A^{-1}\right\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. See Section 4.4 of Anderson *et al.* (1999) for further details.

Following the use of nag_zgbsv (f07bnc), nag_zgbcon (f07buc) can be used to estimate the condition number of A and nag_zgbrfs (f07bvc) can be used to obtain approximate error bounds. Alternatives to nag_zgbsv (f07bnc), which return condition and error estimates directly are nag_complex_band_lin_solve (f04cbc) and nag_zgbsvx (f07bpc).

8 Parallelism and Performance

nag_zgbsv (f07bnc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgbsv (f07bnc) 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

The band storage scheme for the array **ab** is illustrated by the following example, when $n = 6$, $k_l = 1$, and $k_u = 2$. Storage of the band matrix A in the array **ab**:

order = Nag_ColMajor

$$\begin{array}{cccccc} * & * & * & + & + & + \\ * & * & a_{13} & a_{24} & a_{35} & a_{46} \\ * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \end{array}$$

order = Nag_RowMajor

$$\begin{array}{ccccc} * & a_{11} & a_{12} & a_{13} & + \\ a_{21} & a_{22} & a_{23} & a_{24} & + \\ a_{32} & a_{33} & a_{34} & a_{35} & + \\ a_{43} & a_{44} & a_{45} & a_{46} & * \\ a_{54} & a_{55} & a_{56} & * & * \\ a_{65} & a_{66} & * & * & * \end{array}$$

Array elements marked $*$ need not be set and are not referenced by the function. Array elements marked $+$ need not be set, but are defined on exit from the function and contain the elements u_{14}, u_{25} and u_{36}.

The total number of floating-point operations required to solve the equations $AX = B$ depends upon the pivoting required, but if $n \gg k_l + k_u$ then it is approximately bounded by $O(nk_l(k_l + k_u))$ for the factorization and $O(n(2k_l + k_u)r)$ for the solution following the factorization.

The real analogue of this function is nag_dgbsv (f07bac).

10 Example

This example solves the equations

$$Ax = b,$$

where A is the band matrix

$$A = \begin{pmatrix} -1.65+2.26i & -2.05-0.85i & 0.97-2.84i & 0 \\ 6.30i & -1.48-1.75i & -3.99+4.01i & 0.59-0.48i \\ 0 & -0.77+2.83i & -1.06+1.94i & 3.33-1.04i \\ 0 & 0 & 4.48-1.09i & -0.46-1.72i \end{pmatrix}$$

and

$$b = \begin{pmatrix} -1.06+21.50i \\ -22.72-53.90i \\ 28.24-38.60i \\ -34.56+16.73i \end{pmatrix}.$$

Details of the LU factorization of A are also output.

10.1 Program Text

```
/* nag_zgbsv (f07bnc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>

int main(void)
{

  /* Scalars */
  Integer       exit_status = 0, i, j, kl, ku, n, nrhs, pdab, pdb;

  /* Arrays */
  Complex       *ab = 0, *b = 0;
  Integer       *ipiv = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

#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_zgbsv (f07bnc) 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"%*[^\n]", &n, &kl, &ku);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &kl, &ku);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &nrhs);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &nrhs);
#endif
  if (n < 0 || kl < 0 || ku < 0 || nrhs < 0)
    {
      printf("Invalid n, kl, ku or nrhs\n");
      exit_status = 1;
      goto END;
    }
  /* Allocate memory */
  if (!(ab   = NAG_ALLOC((2*kl+ku+1) * n, Complex)) ||
      !(b    = NAG_ALLOC(n*nrhs, Complex)) ||
      !(ipiv = NAG_ALLOC(n, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  pdab = 2*kl+ku+1;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif
  /* Read the band matrix A and the right hand side b from data file */
  for (i = 1; i <= n; ++i)
    for (j = MAX(i - kl, 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("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  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

  /* Solve the equations Ax = b for x using nag_zgbsv (f07bnc). */
  nag_zgbsv(order, n, kl, ku, nrhs, ab, pdab, ipiv, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgbsv (f07bnc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  printf("Solution\n");
```

```
  for (i = 1; i <= n; ++i) {
    for (j = 1; j <= nrhs; ++j)
      printf("(%7.4f, %7.4f)%s", B(i, j).re, B(i, j).im, j%4 == 0?"\n":" ");
    printf("\n");
  }
  printf("\n");

  /* Print details of the factorization using
   * nag_band_complx_mat_print_comp (x04dfc).
   */
  fflush(stdout);
  nag_band_complx_mat_print_comp(order, n, n, kl, kl+ku, ab, pdab,
                                 Nag_BracketForm, "%7.4f",
                                 "Details of factorization", Nag_IntegerLabels,
                                 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_band_complx_mat_print_comp (x04dfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print pivot indices' */
  printf("\nPivot indices\n");
  for (i = 1; i <= n; ++i) printf("%11"NAG_IFMT"", ipiv[i-1]);
  printf("\n");
 END:
  NAG_FREE(ab);
  NAG_FREE(b);
  NAG_FREE(ipiv);
  return exit_status;
}
#undef AB
#undef B
```

10.2 Program Data

```
nag_zgbsv (f07bnc) Example Program Data
    4             1              2                            : n, kl, ku
    1                                                         : nrhs
 ( -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) : matrix A
 ( -1.06, 21.50)
 (-22.72,-53.90)
 ( 28.24,-38.60)
 (-34.56, 16.73)                                              : vector b
```

10.3 Program Results

```
nag_zgbsv (f07bnc) Example Program Results

Solution
(-3.0000,  2.0000)
( 1.0000, -7.0000)
(-5.0000,  4.0000)
( 6.0000, -8.0000)

 Details of factorization
                    1                  2                  3                  4
 1  ( 0.0000, 6.3000)  (-1.4800,-1.7500)  (-3.9900, 4.0100)  ( 0.5900,-0.4800)
 2  ( 0.3587, 0.2619)  (-0.7700, 2.8300)  (-1.0600, 1.9400)  ( 3.3300,-1.0400)
```

```
 3                    ( 0.2314, 0.6358)  ( 4.9303,-3.0086)  (-1.7692,-1.8587)
 4                                       ( 0.7604, 0.2429)  ( 0.4338, 0.1233)

Pivot indices
          2          3          3          4
```