NAG Library Function Document

nag_complex_band_lin_solve (f04cbc)

1 Purpose

nag_complex_band_lin_solve (f04cbc) 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. An estimate of the condition number of A and an error bound for the computed solution are also returned.

2 Specification

```
#include <nag.h>
#include <nagf04.h>
```

```
void nag_complex_band_lin_solve (Nag_OrderType order, Integer n, Integer kl,
     Integer ku, Integer nrhs, Complex ab[], Integer pdab, Integer ipiv[],
     Complex b[], Integer pdb, double *rcond, double *errbnd, NagError *fail)
```

3 Description

The LU decomposition with partial pivoting and row interchanges is used to factor A as $A = PLU$, where P is a permutation matrix, L is the product of permutation matrices 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

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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

Constraint: **n** ≥ 0.

3: **kl** – Integer *Input*

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

Constraint: **kl** ≥ 0.

4: **ku** – Integer *Input*

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

Constraint: **ku** ≥ 0.

5: **nrhs** – Integer *Input*

On entry: the number of right-hand sides r, 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 $\max(1, \mathbf{pdab} \times \mathbf{n})$.

On entry: the n by n 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 $\mathbf{ab}[(j-1) \times \mathbf{pdab} + \mathbf{kl} + \mathbf{ku} + i - j]$;

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

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

8: **ipiv**[**n**] – Integer *Output*

On exit: if **fail.code** = NE_NOERROR, the pivot indices that define the permutation matrix P; at the ith step row i of the matrix was interchanged with row $\mathbf{ipiv}[i-1]$. $\mathbf{ipiv}[i-1] = i$ 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

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ 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 matrix of right-hand sides B.

On exit: if **fail.code** = NE_NOERROR or NE_RCOND, 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: **rcond** – double * *Output*

On exit: if **fail.code** $=$ NE_NOERROR, an estimate of the reciprocal of the condition number of the matrix A, computed as **rcond** $= \left(\|A\|_1 \|A^{-1}\|_1\right)$.

12: **errbnd** – double * *Output*

On exit: if **fail.code** $=$ NE_NOERROR or NE_RCOND, an estimate of the forward error bound for a computed solution $\hat{x}$, such that $\|\hat{x} - x\|_1 / \|x\|_1 \leq$ **errbnd**, where $\hat{x}$ is a column of the computed solution returned in the array **b** and x is the corresponding column of the exact solution X. If **rcond** is less than ***machine precision***, then **errbnd** is returned as unity.

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

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

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.

NE_RCOND

A solution has been computed, but **rcond** is less than ***machine precision*** so that the matrix A is numerically singular.

NE_SINGULAR

Diagonal element $\langle value \rangle$ of the upper triangular factor is zero. The factorization has been completed, but 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

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

where $\kappa(A) = \|A^{-1}\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. nag_complex_band_lin_solve (f04cbc) uses the approximation $\|E\|_1 = \epsilon\|A\|_1$ to estimate **errbnd**. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_complex_band_lin_solve (f04cbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_complex_band_lin_solve (f04cbc) 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 band storage scheme for the array **ab** is illustrated by the following example, when $n = 5$, $k_l = 2$, and $k_u = 1$. Storage of the band matrix A in the array **ab**:

Band matrix A	**Band storage in array ab**	
	order = Nag_ColMajor	**order** = Nag_RowMajor
a_{11} a_{12}	$*$ $*$ $*$ $+$ $+$	$*$ $*$ a_{11} a_{12} $+$ $+$
a_{21} a_{22} a_{23}	$*$ $*$ $+$ $+$ $+$	$*$ a_{21} a_{22} a_{23} $+$ $+$
a_{31} a_{32} a_{33} a_{34}	$*$ a_{12} a_{23} a_{34} a_{45}	a_{31} a_{32} a_{33} a_{34} $+$ $*$
a_{42} a_{43} a_{44} a_{45}	a_{11} a_{22} a_{33} a_{44} a_{55}	a_{42} a_{43} a_{44} a_{45} $*$ $*$
a_{53} a_{54} a_{55}	a_{21} a_{32} a_{43} a_{54} $*$	a_{53} a_{54} a_{55} $*$ $*$ $*$
	a_{31} a_{42} a_{53} $*$ $*$	

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_{13}, u_{14}, u_{24}, u_{25} and u_{35}. In this example when **order** = Nag_ColMajor the first referenced element of **ab** is **ab**[3] $= a_{11}$; while for **order** = Nag_RowMajor the first referenced element is **ab**[2] $= a_{11}$.

In general, elements a_{ij} are stored as follows:

if **order** $=$ Nag_ColMajor, a_{ij} are stored in $\mathbf{ab}[(j-1)\times\mathbf{pdab}+\mathbf{kl}+\mathbf{ku}+i-j]$

if **order** $=$ Nag_RowMajor, a_{ij} are stored in $\mathbf{ab}[(i-1)\times\mathbf{pdab}+\mathbf{kl}+j-i]$

where $\max(1,i-\mathbf{kl})\leq j\leq\min(\mathbf{n},i+\mathbf{ku})$.

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 condition number estimation typically requires between four and five solves and never more than eleven solves, following the factorization.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

The real analogue of nag_complex_band_lin_solve (f04cbc) is nag_real_band_lin_solve (f04bbc).

10 Example

This example solves the equations

$$AX=B,$$

where A is the band matrix

$$A=\begin{pmatrix} -1.65+2.26i & -2.05-0.85i & 0.97-2.84i & 0 \\ 0.00+6.30i & -1.48-1.75i & -3.99+4.01i & 0.59-0.48i \\ 0 & -0.77+2.83i & -1.06+1.94i & 3.33-1.04i \\ 0 & 0 & 4.48-1.09i & -0.46-1.72i \end{pmatrix}$$

and

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

An estimate of the condition number of A and an approximate error bound for the computed solutions are also printed.

10.1 Program Text

```
/* nag_complex_band_lin_solve (f04cbc) Example Program.
 *
 * Copyright 2004 Numerical Algorithms Group.
 *
 * Mark 8, 2004.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf04.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double        errbnd, rcond;
  Integer       exit_status, i, j, kl, ku, n, nrhs, pdab, pdb;

  /* Arrays */
  char          *clabs = 0, *rlabs = 0;
  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

  exit_status = 0;
  INIT_FAIL(fail);

  printf("nag_complex_band_lin_solve (f04cbc)"
          " Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%ld%*[^\n] ",
         &n, &kl, &ku, &nrhs);
  if (n > 0 && kl > 0 && ku > 0 && nrhs > 0)
    {
      /* Allocate memory */
      if (!(clabs = NAG_ALLOC(2, char)) ||
          !(rlabs = NAG_ALLOC(2, char)) ||
          !(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
    }
  else
    {
      printf("%s\n", "One or more of NMAX, KLMAX, KUMAX or NRHSMX is"
              " too small");
      exit_status = 1;
      return exit_status;
    }

  /* Read A and B 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] ");

  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] ");
```

```
/* Solve the equations AX = B for X */
/* nag_complex_band_lin_solve (f04cbc).
 * Computes the solution and error-bound to a complex banded
 * system of linear equations
 */
nag_complex_band_lin_solve(order, n, kl, ku, nrhs, ab, pdab, ipiv, b,
                           pdb, &rcond, &errbnd, &fail);
if (fail.code == NE_NOERROR)
  {
    /* Print solution, estimate of condition number and approximate */
    /* error bound */
    /* 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", 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;
      }

    printf("\n%s\n%4s%10.1e\n\n", "Estimate of condition number", "",
            1.0/rcond);
    printf("\n%s\n%4s%10.1e\n\n",
            "Estimate of error bound for computed solutions", "",
            errbnd);
  }
else if (fail.code == NE_RCOND)
  {
    /* Matrix A is numerically singular.  Print estimate of */
    /* reciprocal of condition number and solution */
    printf("\n%s\n%4s%10.1e\n\n\n",
            "Estimate of reciprocal of condition number", "", rcond);
    /* nag_gen_complx_mat_print_comp (x04dbc), see above. */
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix,
                                  Nag_NonUnitDiag, n, nrhs, b, pdb,
                                  Nag_BracketForm, "%7.4f",
                                  "Solution", 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;
      }
  }
else if (fail.code == NE_SINGULAR)
  {
    /* The upper triangular matrix U is exactly singular.  Print */
    /* details of factorization */

    printf("\n");
    /* nag_band_complx_mat_print_comp (x04dfc).
     * Print complex packed banded matrix (comprehensive)
     */
    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("\n%s\n", "Pivot indices");
      for (i = 1; i <= n; ++i)
        {
          printf("%11ld%s", ipiv[i - 1],
                  i%7 == 0 || i == n?"\n":" ");
        }
      printf("\n");
    }
  else
    {
      printf("Error from nag_complex_band_lin_solve (f04cbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(clabs);
  NAG_FREE(rlabs);
  NAG_FREE(ab);
  NAG_FREE(b);
  NAG_FREE(ipiv);

  return exit_status;
}
```

10.2 Program Data

```
nag_complex_band_lin_solve (f04cbc) Example Program Data

    4               1               2               2      :N, KL, KU and 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) :End 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 matrix B
```

10.3 Program Results

```
nag_complex_band_lin_solve (f04cbc) Example Program Results

 Solution
                   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)

Estimate of condition number
       1.0e+02


Estimate of error bound for computed solutions
       1.2e-14
```