NAG Library Function Document

nag_complex_lu_solve_mult_rhs (f04akc)

1 Purpose

nag_complex_lu_solve_mult_rhs (f04akc) calculates the approximate solution of a set of complex linear equations with multiple right-hand sides $AX = B$, where A has been factorized by nag_complex_lu (f03ahc).

2 Specification

```
#include <nag.h>
#include <nagf04.h>
void nag_complex_lu_solve_mult_rhs (Integer n, Integer nrhs,
     const Complex a[], Integer tda, const Integer pivot[], Complex b[],
     Integer tdb, NagError *fail)
```

3 Description

To solve a set of complex linear equations $AX = B$, the function must be preceded by a call to nag_complex_lu (f03ahc) which computes an LU factorization of A with partial pivoting, $PA = LU$, where P is a permutation matrix, L is lower triangular and U is unit upper triangular. The columns x of the solution X are found by forward and backward substitution in $Ly = Pb$ and $Ux = y$, where b is a column of the right-hand side.

4 References

Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer–Verlag

5 Arguments

1: **n** – Integer *Input*

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

Constraint: **n** ≥ 1.

2: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 1.

3: **a**[**n** $\times$ **tda**] – const Complex *Input*

Note: the (i, j)th element of the matrix A is stored in **a**$[(i-1) \times$ **tda** $+ j - 1]$.

On entry: details of the LU factorization, as returned by nag_complex_lu (f03ahc).

4: **tda** – Integer *Input*

On entry: the stride separating matrix column elements in the array **a**.

Constraint: **tda** $\geq$ **n**.

5: **pivot**[**n**] – const Integer *Input*

On entry: details of the row interchanges as returned by nag_complex_lu (f03ahc).

6: **b**[**n** × **tdb**] – Complex *Input/Output*

Note: the (i,j)th element of the matrix B is stored in **b**$[(i-1) \times$ **tdb** $+ j - 1]$.

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

On exit: B is overwritten by the solution matrix X.

7: **tdb** – Integer *Input*

On entry: the stride separating matrix column elements in the array **b**.

Constraint: **tdb** $\geq$ **nrhs**.

8: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **tda** $= \langle value \rangle$ while **n** $= \langle value \rangle$. These arguments must satisfy **tda** $\geq$ **n**.

On entry, **tdb** $= \langle value \rangle$ while **nrhs** $= \langle value \rangle$. These arguments must satisfy **tdb** $\geq$ **nrhs**.

NE_INT_ARG_LT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 1.

On entry, **nrhs** $= \langle value \rangle$.
Constraint: **nrhs** ≥ 1.

7 Accuracy

The accuracy of the computed solution depends on the conditioning of the original matrix. For a detailed error analysis see page 106 of Wilkinson and Reinsch (1971).

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_complex_lu_solve_mult_rhs (f04akc) is approximately proportional to $n^2 r$.

10 Example

To solve the set of linear equations $AX = B$ where

$$A = \begin{pmatrix} 1 & 1+2i & 2+10i \\ 1+i & 3i & -5+14i \\ 1+i & 5i & -8+20i \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_complex_lu_solve_mult_rhs (f04akc) Example Program.
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 1A revised, (Oct 1990).
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagf03.h>
#include <nagf04.h>

#define A(I, J) a[(I) *tda + J]
#define B(I, J) b[(I) *tdb + J]
int main(void)
{

  Complex  *a = 0, *b = 0, det;
  Integer  dete, exit_status = 0, i, j, n, nrhs, *pivot = 0, tda, tdb;
  NagError fail;

  INIT_FAIL(fail);

  printf(
          "nag_complex_lu_solve_mult_rhs (f04akc) Example Program Results\n");
  scanf("%*[^\n]");  /* Skip heading in data file */
  scanf("%ld", &n);
  nrhs = 1;
  if (n >= 1)
    {
      if (!(pivot = NAG_ALLOC(n, Integer)) ||
          !(a = NAG_ALLOC(n*n, Complex)) ||
          !(b = NAG_ALLOC(n*nrhs, Complex)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tda = n;
      tdb = nrhs;
    }
  else
    {
      printf("Invalid n.\n");
      exit_status = 1;
      return exit_status;
    }
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      scanf(" ( %lf,  %lf ) ", &A(i, j).re, &A(i, j).im);
  for (i = 0; i < n; i++)
    for (j = 0; j < nrhs; j++)
      scanf(" ( %lf,  %lf ) ", &B(i, j).re, &B(i, j).im);
  /* nag_complex_lu (f03ahc).
   * LU factorization and determinant of complex matrix
   */
  nag_complex_lu(n, a, tda, pivot, &det, &dete, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_lu (f03ahc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* nag_complex_lu_solve_mult_rhs (f04akc).
   * Approximate solution of complex simultaneous linear
   * equations (coefficient matrix already factorized by
   * nag_complex_lu (f03ahc))
```

```
   */
  nag_complex_lu_solve_mult_rhs(n, nrhs, a, tda, pivot, b, tdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_complex_lu_solve_mult_rhs (f04akc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("Solution\n");
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < nrhs; j++)
        printf("(%7.3f, %7.3f) ", B(i, j).re, B(i, j).im);
      printf("\n");
    }
 END:
  NAG_FREE(pivot);
  NAG_FREE(a);
  NAG_FREE(b);
  return exit_status;
}
```

10.2 Program Data

```
nag_complex_lu_solve_mult_rhs (f04akc) Example Program Data
  3
( 1.0, 0.0 ) ( 1.0, 2.0 ) ( 2.0,10.0)
( 1.0, 1.0 ) ( 0.0, 3.0 ) (-5.0,14.0)
( 1.0, 1.0 ) ( 0.0, 5.0 ) (-8.0,20.0)
( 1.0, 0.0 ) ( 0.0, 0.0 ) ( 0.0, 0.0)
```

10.3 Program Results

```
nag_complex_lu_solve_mult_rhs (f04akc) Example Program Results
Solution
( 10.000,   1.000)
(  9.000,  -3.000)
( -2.000,   2.000)
```