

## 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_lin_eqn_mult_rhs(Integer n, Integer nrhs, Complex a[],
    Integer tda, 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. Parameters

**n**

Input:  $n$ , the order of the matrix  $A$ .  
Constraint:  $n \geq 1$ .

**rhs**

Input:  $r$ , the number of right-hand sides.  
Constraint:  $nrhs \geq 1$ .

**a[n][tda]**

Input: details of the  $LU$  factorization, as returned by **nag\_complex\_lu (f03ahc)**.

**tda**

Input: the second dimension of the array **a** as declared in the function from which **nag\_complex\_lu\_solve\_mult\_rhs** is called.  
Constraint:  $tda \geq n$ .

**pivot[n]**

Input: details of the row interchanges as returned by **nag\_complex\_lu (f03ahc)**.

**b[n][tdb]**

Input: the  $n$  by  $r$  right-hand side matrix  $B$ .  
Output:  $B$  is overwritten by the solution matrix  $X$ .

**tdb**

Input: the second dimension of the array **b** as declared in the function from which **nag\_complex\_lu\_solve\_mult\_rhs** is called.  
Constraint:  $tdb \geq nrhs$ .

**fail**

The NAG error parameter, see the Essential Introduction to the NAG C Library.

### 5. Error Indications and Warnings

**NE\_INT\_ARG\_LT**

On entry, **n** must not be less than 1:  $n = \langle value \rangle$ .  
On entry, **nrhs** must not be less than 1:  $nrhs = \langle value \rangle$ .

**NE\_2\_INT\_ARG\_LT**

On entry, **tda** = *<value>* while **n** = *<value>*. These parameters must satisfy **tda** ≥ **n**.

On entry, **tdb** = *<value>* while **nrhs** = *<value>*. These parameters must satisfy **tdb** ≥ **nrhs**.

**6. Further Comments**

The time taken by the function is approximately proportional to  $n^2r$ .

**6.1. Accuracy**

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

**6.2. References**

Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation (Vol II, Linear Algebra)* Springer-Verlag pp 93–110.

**7. See Also**

nag\_complex\_lu (f03ahc)

nag\_complex\_lin\_eqn\_mult\_rhs (f04adc)

**8. Example**

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

$$A = \begin{pmatrix} 1 & 1 + 2i & 2 + 10i \\ 1 + i & 3i & -5 + 14i \\ 1 + i & 5i & -8 + 20i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

**8.1. Program Text**

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

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

#define NMAX 5
#define TDA NMAX
#define TDB NMAX

main()
{
    Complex det;
    Complex a[NMAX][TDA], b[NMAX][TDB];
    Integer i, j, n, dete, nrhs = 1;
    Integer pivot[NMAX];
    static NagError fail;

    fail.print = TRUE;
    Vprintf("f04akc Example Program Results\n");
```

```

Vscanf("%*[\n]"); /* Skip heading in data file */
Vscanf("%ld",&n);
if (n > 0 && n <= NMAX)
{
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      Vscanf(" ( %lf , %lf ) ", &a[i][j].re, &a[i][j].im);
  for (i = 0; i < n; i++)
    for (j = 0; j < nrhs; j++)
      Vscanf(" ( %lf , %lf ) ", &b[i][j].re, &b[i][j].im);
  f03ahc(n, (Complex *)a, (Integer)TDA, pivot, &det, &dete, &fail);
  if (fail.code!=NE_NOERROR)
    exit(EXIT_FAILURE);
  else
  {
    f04akc(n, nrhs, (Complex *)a, (Integer)TDA, pivot,
           (Complex *)b, (Integer)TDB, &fail);
    if (fail.code!=NE_NOERROR)
      exit(EXIT_FAILURE);
    Vprintf("Solution\n");
    for (i=0; i<n; i++)
      {
        for (j=0; j<nrhs; j++)
          Vprintf("(%7.3f, %7.3f) ", b[i][j].re, b[i][j].im);
        Vprintf("\n");
      }
  }
}
else
{
  Vfprintf(stderr, "Error: n is out of range: n = %3ld\n", n);
  exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}

```

## 8.2. Program Data

```

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)

```

## 8.3. Program Results

```

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

```

---