NAG C Library Function Document

nag_zgetrf (f07arc)

1 Purpose

nag_zgetrf (f07arc) computes the LU factorization of a complex m by n matrix.

2 Specification

```
void nag_zgetrf (Nag_OrderType order, Integer m, Integer n, Complex a[],
      Integer pda, Integer ipiv[], NagError *fail)
```

3 Description

nag_zgetrf (f07arc) forms the LU factorization of a complex m by n matrix A as $A = PLU$, where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$) and U is upper triangular (upper trapezoidal if $m < n$). Usually A is square $(m = n)$, and both L and U are triangular. The function uses partial pivoting, with row interchanges.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **m** – Integer *Input*

On entry: m, the number of rows of the matrix A.

Constraint: **m** ≥ 0.

3: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

Constraint: **n** ≥ 0.

4: **a**[dim] – Complex *Input/Output*

Note: the dimension, dim, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i,j)th element of the matrix A is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i,j)th element of the matrix A is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

On entry: the m by n matrix A.

On exit: A is overwritten by the factors L and U; the unit diagonal elements of L are not stored.

5: **pda** – Integer *Input*

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

Constraints:

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

6: **ipiv**[dim] – Integer *Output*

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

On exit: the pivot indices. Row i of the matrix A was interchanged with row **ipiv**[$i-1$], for $i = 1, 2, \ldots, \min(m, n)$.

7: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

NE_INT_2

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

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

NE_SINGULAR

u(⟨*value*⟩, ⟨*value*⟩) is exactly zero. The factorization has been completed but the factor U is exactly singular, and division by zero will occur if it is subsequently used to solve a system of linear equations or to invert A.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ had an illegal value.

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 consult NAG for assistance.

7 Accuracy

The computed factors L and U are the exact factors of a perturbed matrix $A + E$, where

$$|E| \leq c(\min(m, n))\epsilon P|L|\,|U|,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***.

8 Further Comments

The total number of real floating-point operations is approximately $\frac{8}{3}n^3$ if $m = n$ (the usual case), $\frac{4}{3}n^2(3m - n)$ if $m > n$ and $\frac{4}{3}m^2(3n - m)$ if $m < n$.

A call to this function with $m = n$ may be followed by calls to the functions:

nag_zgetrs (f07asc) to solve $AX = B$, $A^TX = B$ or $A^HX = B$;

nag_zgecon (f07auc) to estimate the condition number of A;

nag_zgetri (f07awc) to compute the inverse of A.

The real analogue of this function is nag_dgetrf (f07adc).

9 Example

To compute the LU factorization of the matrix A, where

$$A = \begin{pmatrix} -1.34 + 2.55i & 0.28 + 3.17i & -6.39 - 2.20i & 0.72 - 0.92i \\ -0.17 - 1.41i & 3.31 - 0.15i & -0.15 + 1.34i & 1.29 + 1.38i \\ -3.29 - 2.39i & -1.91 + 4.42i & -0.14 - 1.35i & 1.72 + 1.35i \\ 2.41 + 0.39i & -0.56 + 1.47i & -0.83 - 0.69i & -1.96 + 0.67i \end{pmatrix}.$$

9.1 Program Text

```
/* nag_zgetrf (f07arc) 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, j, m, n, pda, ipiv_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex  *a=0;
  Integer  *ipiv=0;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
  order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
```

```
  Vprintf("f07arc Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%ld%*[^\n] ", &m, &n);
#ifdef NAG_COLUMN_MAJOR
  pda = m;
#else
  pda = n;
#endif
  ipiv_len = MIN(m,n);
  /* Allocate memory */
  if ( !(a = NAG_ALLOC(m * n, Complex)) ||
       !(ipiv = NAG_ALLOC(ipiv_len, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A from data file */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
        Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
    }
  Vscanf("%*[^\n] ");

  /* Factorize A */
  f07arc(order, m, n, a, pda, ipiv, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07arc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print details of factorization */
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n, a, pda,
         Nag_BracketForm, "%7.4f", "Details of factorization",
         Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print pivot indices */
  Vprintf("\nIPIV\n");
  for (i = 1; i <= MIN(m,n); ++i)
    Vprintf("%12ld%s", ipiv[i - 1], i%4 == 0 ?"\n":" ");
  Vprintf("\n");

 END:
  if (a) NAG_FREE(a);
  if (ipiv) NAG_FREE(ipiv);
  return exit_status;
}
```

9.2 Program Data

```
f07arc Example Program Data
  4  4                                                      :Values of M and N
 (-1.34, 2.55) ( 0.28, 3.17) (-6.39,-2.20) ( 0.72,-0.92)
 (-0.17,-1.41) ( 3.31,-0.15) (-0.15, 1.34) ( 1.29, 1.38)
 (-3.29,-2.39) (-1.91, 4.42) (-0.14,-1.35) ( 1.72, 1.35)
 ( 2.41, 0.39) (-0.56, 1.47) (-0.83,-0.69) (-1.96, 0.67)  :End of matrix A
```

9.3 Program Results

```
f07arc Example Program Results

 Details of factorization
                    1                  2                  3                  4
 1  (-3.2900,-2.3900)  (-1.9100, 4.4200)  (-0.1400,-1.3500)  ( 1.7200, 1.3500)
 2  ( 0.2376, 0.2560)  ( 4.8952,-0.7114)  (-0.4623, 1.6966)  ( 1.2269, 0.6190)
 3  (-0.1020,-0.7010)  (-0.6691, 0.3689)  (-5.1414,-1.1300)  ( 0.9983, 0.3850)
 4  (-0.5359, 0.2707)  (-0.2040, 0.8601)  ( 0.0082, 0.1211)  ( 0.1482,-0.1252)

IPIV
           3             2             3             4
```