Title

stata.com

lud( ) — LU decomposition

Description

`lud(`*A*`,` *L*`,` *U*`,` *p*`)` returns the LU decomposition (with partial pivoting) of *A* in *L* and *U* along with a permutation vector *p*. The returned results are such that *A*=*L*`[`*p*`,.]*`*U* up to roundoff error.

`_lud(`*L*`,` *U*`,` *p*`)` is similar to `lud()`, except that it conserves memory. The matrix to be decomposed is passed in *L*, and the same storage location is overwritten with the calculated *L* matrix.

`_lud_la(`*A*`,` *q*`)` is the [M-1] **LAPACK** routine that the above functions use to calculate the LU decomposition. See *LAPACK routine* below.

Syntax

void `lud(`*numeric matrix A*`,` *L*`,` *U*`,` *p*`)`

void `_lud(`*numeric matrix L*`,` *U*`,` *p*`)`

void `_lud_la(`*numeric matrix A*`,` *q*`)`

where

1. *A* may be real or complex and need not be square.
2. The types of *L*, *U*, *p*, and *q* are irrelevant; results are returned there.

Remarks and examples

stata.com

Remarks are presented under the following headings:

LU decomposition
LAPACK routine

LU decomposition

The LU decomposition of matrix *A* can be written as

$$P'A = LU$$

where P' is a permutation matrix that permutes the rows of *A*. *L* is lower triangular and *U* is upper triangular. The decomposition can also be written as

$$A = PLU$$

because, given that *P* is a permutation matrix, $P^{-1} = P'$.

Rather than returning P directly, returned is p corresponding to P. Lowercase p is a column vector that contains the subscripts of the rows in the desired order. That is,

$$PL = L[p,.]$$

The advantage of this is that p requires less memory than P and the reorganization, should it be desired, can be performed more quickly; see [M-1] **Permutation**. In any case, the formula defining the LU decomposition can be written as

$$A = L[p,.]*U$$

One can also write

$$B = LU, \text{ where } B[p,.] = A$$

LAPACK routine

`_lud_la(A, q)` is the interface into the [M-1] **LAPACK** routines that the above functions use to calculate the LU decomposition. You may use it directly if you wish.

Matrix A is decomposed, and the decomposition is placed back in A. U is stored in the upper triangle (including the diagonal) of A. L is stored in the lower triangle of A, and it is understood that L is supposed to have ones on its diagonal. q is a column vector recording the row swaps that account for the pivoting. This is the same information as stored in p, but in a different format.

q records the row swaps to be made. For instance, `q = (1\2\2)` means that (start at the end) the third row is to be swapped with the second row, then the second row is to stay where it is, and finally the first row is to stay where it is. q can be converted into p by the following logic:

```
p = 1::rows(q)
for (i=rows(q); i>=1; i--) {
     hold = p[i]
     p[i] = p[q[i]]
     p[q[i]] = hold
}
```

Conformability

`lud(`*A*`, `*L*`, `*U*`, `*p*`)`:

input:

	A:	$r \times c$

output:

	L:	$r \times m, \quad m = \min(r, c)$
	U:	$m \times c$
	p:	$r \times 1$

`_lud(`*L*`, `*U*`, `*p*`)`:

input:

	L:	$r \times c$

output:

	L:	$r \times m, \quad m = \min(r, c)$
	U:	$m \times c$
	p:	$r \times 1$

`_lud_la(`*A*`, `*q*`)`:

input:

	A:	$r \times c$

output:

	A:	$r \times c$
	q:	$r \times 1$

Diagnostics

`lud(`*A*`, `*L*`, `*U*`, `*p*`)` returns missing results if *A* contains missing values; *L* will have missing values below the diagonal, 1s on the diagonal, and 0s above the diagonal; *U* will have missing values on and above the diagonal and 0s below. Thus if there are missing values, *U*`[1,1]` will contain missing.

`_lud(`*L*`, `*U*`, `*p*`)` sets *L* and *U* as described above if *A* contains missing values.

`_lud_la(`*A*`, `*q*`)` aborts with error if *A* is a view.

Also see

[M-5] **det( )** — Determinant of matrix

[M-5] **lusolve( )** — Solve AX=B for X using LU decomposition

[M-4] **Matrix** — Matrix functions