

**lusolve()** — Solve  $AX=B$  for  $X$  using LU decomposition

Description Diagnostics	Syntax Also see	Remarks and examples	Conformability
----------------------------	--------------------	----------------------	----------------

## Description

`lusolve(A, B)` solves  $AX=B$  and returns  $X$ . `lusolve()` returns a matrix of missing values if  $A$  is singular.

`lusolve(A, B, tol)` does the same thing but allows you to specify the tolerance for declaring that  $A$  is singular; see [Tolerance](#) under *Remarks and examples* below.

`_lusolve(A, B)` and `_lusolve(A, B, tol)` do the same thing except that, rather than returning the solution  $X$ , they overwrite  $B$  with the solution and, in the process of making the calculation, they destroy the contents of  $A$ .

`_lusolve_la(A, B)` and `_lusolve_la(A, B, tol)` are the interfaces to the [\[M-1\] LAPACK](#) routines that do the work. They solve  $AX=B$  for  $X$ , returning the solution in  $B$  and, in the process, using as workspace (overwriting)  $A$ . The routines return 1 if  $A$  was singular and 0 otherwise. If  $A$  was singular,  $B$  is overwritten with a matrix of missing values.

## Syntax

<i>numeric matrix</i>	<code>lusolve(<i>numeric matrix A, numeric matrix B</i>)</code>
<i>numeric matrix</i>	<code>lusolve(<i>numeric matrix A, numeric matrix B, real scalar tol</i>)</code>
<i>void</i>	<code>_lusolve(<i>numeric matrix A, numeric matrix B</i>)</code>
<i>void</i>	<code>_lusolve(<i>numeric matrix A, numeric matrix B, real scalar tol</i>)</code>
<i>real scalar</i>	<code>_lusolve_la(<i>numeric matrix A, numeric matrix B</i>)</code>
<i>real scalar</i>	<code>_lusolve_la(<i>numeric matrix A, numeric matrix B, real scalar tol</i>)</code>

## Remarks and examples

The above functions solve  $AX=B$  via LU decomposition and are accurate. An alternative is `qrsolve()` (see [\[M-5\] qrsolve\(\)](#)), which uses QR decomposition. The difference between the two solutions is not, practically speaking, accuracy. When  $A$  is of full rank, both routines return equivalent results, and the LU approach is quicker, using approximately  $O(2/3n^3)$  operations rather than  $O(4/3n^3)$ , where  $A$  is  $n \times n$ .

The difference arises when  $A$  is singular. Then the LU-based routines documented here return missing values. The QR-based routines documented in [\[M-5\] qrsolve\(\)](#) return a generalized (least squares) solution.

For more information on LU and QR decomposition, see [\[M-5\] lud\(\)](#) and see [\[M-5\] qrd\(\)](#).

Remarks are presented under the following headings:

*Derivation*  
*Relationship to inversion*  
*Tolerance*

### Derivation

We wish to solve for  $X$

$$AX = B \tag{1}$$

Perform LU decomposition on  $A$  so that we have  $A = PLU$ . Then (1) can be written as

$$PLUX = B$$

or, premultiplying by  $P'$  and remembering that  $P'P = I$ ,

$$LUX = P'B \tag{2}$$

Define

$$Z = UX \tag{3}$$

Then (2) can be rewritten as

$$LZ = P'B \tag{4}$$

It is easy to solve (4) for  $Z$  because  $L$  is a lower-triangular matrix. Once  $Z$  is known, it is easy to solve (3) for  $X$  because  $U$  is upper triangular.

### Relationship to inversion

Another way to solve

$$AX = B$$

is to obtain  $A^{-1}$  and then calculate

$$X = A^{-1}B$$

It is, however, better to solve  $AX = B$  directly because fewer numerical operations are required, and the result is therefore more accurate and obtained in less computer time.

Indeed, rather than thinking about how solving a system of equations can be implemented via inversion, it is more productive to think about how inversion can be implemented via solving a system of equations. Obtaining  $A^{-1}$  amounts to solving

$$AX = I$$

Thus `lusolve()` (or any other solve routine) can be used to obtain inverses. The inverse of  $A$  can be obtained by coding

```
: Ainv = lusolve(A, I(rows(A)))
```

In fact, we provide `luinv()` (see [M-5] [luinv\(\)](#)) for obtaining inverses via LU decomposition, but `luinv()` amounts to making the above calculation, although a little memory is saved because the matrix  $I$  is never constructed.

Hence, everything said about `lusolve()` applies equally to `luinv()`.

## Tolerance

The default tolerance used is

$$\text{eta} = (1\text{e-}13) * \text{trace}(\text{abs}(U)) / n$$

where  $U$  is the upper-triangular matrix of the LU decomposition of  $A$ :  $n \times n$ .  $A$  is declared to be singular if any diagonal element of  $U$  is less than or equal to  $\text{eta}$ .

If you specify  $\text{tol} > 0$ , the value you specify is used to multiply  $\text{eta}$ . You may instead specify  $\text{tol} \leq 0$ , and then the negative of the value you specify is used in place of  $\text{eta}$ ; see [M-1] **Tolerance**.

So why not specify  $\text{tol} = 0$ ? You do not want to do that because, as matrices become close to being singular, results can become inaccurate. Here is an example:

```

: rseed(12345)
: A = lowertriangle(runiform(4,4))
: A[3,3] = 1e-15
: trux = runiform(4,1)
: b = A*trux
: /* the above created an Ax=b problem, and we have placed the true
> value of x in trux. We now obtain the solution via lusolve()
> and compare trux with the value obtained:
> */
: x = lusolve(A, b, 0)
: trux, x

```

1	.260768733	.260768733
2	.0267289389	.0267289389
3	.1079423963	.0989119749
4	.3666839808	.3863636364

← The discussed numerical instability can cause this output to vary a little across different computers

We would like to see the second column being nearly equal to the first—the estimated  $x$  being nearly equal to the true  $x$ —but there are substantial differences.

Even though the difference between  $x$  and  $\text{trux}$  is substantial, the difference between them is small in the prediction space:

```

: A*trux-b, A*x-b

```

	1	2
1	0	0
2	0	0
3	0	-2.77556e-17
4	0	0

What made this problem so difficult was the line  $A[3,3] = 1\text{e-}15$ . Remove that and you would find that the maximum absolute difference between  $x$  and  $\text{trux}$  would be  $-2.44249\text{e-}15$ .

The degree to which the residuals  $A*x-b$  are a reliable measure of the accuracy of  $x$  depends on the condition number of the matrix, which can be obtained by [M-5] **cond()**, which for  $A$ , is  $4.47684\text{e+}15$ . If the matrix is well conditioned, small residuals imply an accurate solution for  $x$ . If the matrix is ill conditioned, small residuals are not a reliable indicator of accuracy.

Another way to check the accuracy of  $x$  is to set  $tol = 0$  and to see how well  $x$  could be obtained were  $b = A*x$ :

```
: x = lusolve(A, b, 0)
: x2 = lusolve(A, A*x, 0)
```

If  $x$  and  $x2$  are virtually the same, then you can safely assume that  $x$  is the result of a numerically accurate calculation. You might compare  $x$  and  $x2$  with `mreldif(x2,x)`; see [M-5] `reldif()`. In our example, `mreldif(x2,x)` is `.03`, a large difference.

If  $A$  is ill conditioned, then small changes in  $A$  or  $B$  can lead to radical differences in the solution for  $X$ .

## Conformability

`lusolve(A, B, tol)`:

*input:*

```
A:    n × n
B:    n × k
tol:   1 × 1 (optional)
```

*output:*

```
result: n × k
```

`_lusolve(A, B, tol)`:

*input:*

```
A:    n × n
B:    n × k
tol:   1 × 1 (optional)
```

*output:*

```
A:    0 × 0
B:    n × k
```

`_lusolve_la(A, B, tol)`:

*input:*

```
A:    n × n
B:    n × k
tol:   1 × 1 (optional)
```

*output:*

```
A:    0 × 0
B:    n × k
result: 1 × 1
```

## Diagnostics

`lusolve(A, B, ...)`, `_lusolve(A, B, ...)`, and `_lusolve_la(A, B, ...)` return a result containing missing if  $A$  or  $B$  contain missing values. The functions return a result containing all missing values if  $A$  is singular.

`_lusolve(A, B, ...)` and `_lusolve_la(A, B, ...)` abort with error if  $A$  or  $B$  is a view.

`_lusolve_la(A, B, ...)` should not be used directly; use `_lusolve()`.

## Also see

[M-5] **cholsolve()** — Solve  $AX=B$  for  $X$  using Cholesky decomposition

[M-5] **lud()** — LU decomposition

[M-5] **luinv()** — Square matrix inversion

[M-5] **qrsolve()** — Solve  $AX=B$  for  $X$  using QR decomposition

[M-5] **solverlower()** — Solve  $AX=B$  for  $X$ ,  $A$  triangular

[M-5] **svsolve()** — Solve  $AX=B$  for  $X$  using singular value decomposition

[M-4] **Matrix** — Matrix functions

[M-4] **Solvers** — Functions to solve  $AX=B$  and to obtain  $A$  inverse

Stata, Stata Press, and Mata are registered trademarks of StataCorp LLC. Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations. Other brand and product names are registered trademarks or trademarks of their respective companies. Copyright © 1985–2023 StataCorp LLC, College Station, TX, USA. All rights reserved.

