

Description

`lssolve(A, B)` finds the minimum-norm least-squares solution for $\min \|AX - B\|_2$ and returns X . A can be real or complex and can also be rank deficient. A does not have to be a square matrix.

`lssolve(A, B, rank)` does the same thing but also returns the effective rank in *rank*.

`lssolve(A, B, rank, favorspeed)` does the same thing but allows you to specify the computation method; see [Computation methods and tolerance](#) under *Remarks and examples* below.

`lssolve(A, B, rank, favorspeed, tol)` does the same thing but allows you to specify the tolerance for declaring the effective rank of A ; see [Computation methods and tolerance](#) under *Remarks and examples* below.

`_lssolve(A, B)`, `_lssolve(A, B, favorspeed)`, and `_lssolve(A, B, favorspeed, tol)` do the same thing except that rather than returning the solution X , they overwrite B with the solution and return the effective rank. In the process of performing the calculation, they destroy the contents of A .

`_leastsquare_lapacke(A, B, rank)`, `_leastsquare_lapacke(A, B, rank, favorspeed)`, and `_leastsquare_lapacke(A, B, rank, favorspeed, tol)` are the interfaces to the LAPACK routines that do the work. They find the minimum-norm solution for the least-squares problem $\|AX - B\|_2$, returning the solution in B and, in the process, using as workspace (overwriting) A . The routines return 0 if a solution was found and 1 otherwise. If 1 is returned, B is overwritten with a matrix of missing values.

Note that these functions can be used only when set `lapack_mkl` on is in effect on Windows or Linux or when set `lapack_openblas` on is in effect on Mac; see [M-1] [LAPACK](#).

Syntax

```

numeric matrix  lssolve(A, B)
numeric matrix  lssolve(A, B, rank)
numeric matrix  lssolve(A, B, rank, favorspeed)
numeric matrix  lssolve(A, B, rank, favorspeed, tol)

real scalar     _lssolve(A, B)
real scalar     _lssolve(A, B, favorspeed)
real scalar     _lssolve(A, B, favorspeed, tol)

real scalar     _leastsquare_lapacke(A, B, rank)
real scalar     _leastsquare_lapacke(A, B, rank, favorspeed)
real scalar     _leastsquare_lapacke(A, B, rank, favorspeed, tol)

```

where inputs are

```

A:  numeric matrix
B:  numeric matrix
favorspeed:  real scalar
tol:  real scalar

```

and outputs are

```

B:  numeric matrix (solution of  $AX = B$  overwritten in B)
rank:  real scalar
result:  real scalar

```

Remarks and examples

Remarks are presented under the following headings:

```

Introduction
Computation methods and tolerance
Examples

```

Introduction

The above functions solve $AX = B$ via the least-squares method. A does not have to be square and can be rank deficient.

The least-squares method tries to find the minimum-norm solution of the following problem:

$$\min \|AX - B\|_2$$

When A is square and of full rank, the computed solution is the same as

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

When A is not square and not rank deficient, the computed solution is

$$X = (A'A)^{-1}A'B$$

when the number of rows of A is greater than the number of columns of A , and

$$X = A'(AA')^{-1}B$$

when the number of rows of A is less than the number of columns of A .

When A is rank deficient, the inverses in the above formulas are replaced by the generalized Moore–Penrose pseudoinverse. See [M-5] `pinv()` for more details about the pseudoinverse.

Computation methods and tolerance

When *favorspeed* is missing or 0, singular value decomposition is used. This is also the default method when *favorspeed* is not specified. It works with the optional argument *tol* to decide whether the matrix A is rank deficient.

When *favorspeed* is not missing and not 0, the QR or LQ factorization method is used. This method is faster but assumes A has full rank, so the optional argument *tol* is irrelevant in this case.

The default tolerance used is

$$\eta = \frac{(1e-13)*\text{trace}(\text{abs}(A))}{l}$$

where A is $m \times n$ and l is the minimum of m and n . A singular value of A is considered 0 if it is less than or equal to $\text{tol} \times$ the largest singular value of A .

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

See [M-5] `lusolve()` for a detailed discussion of the issues surrounding solving nearly singular systems. The main point is that if A is ill conditioned, then small changes in A or B can lead to radically large differences in the solution for X .

Examples

► Example 1: Square matrix

If A is square and has full rank, the minimum-norm least-squares solution computed by `lssolve()` is the same as $X = A^{-1}B$.

```
: A = (3, 2, 5 \ 2, 3, 1 \ 1, 2, 10)
```

```
: b = (1 \ 1.5 \ 2)
```

```
: A
```

	1	2	3
1	3	2	5
2	2	3	1
3	1	2	10

```

: b
      1
  1  1
  2  1.5
  3  2

: X = lssolve(A, b)
: X
      1
  1  -.2549019608
  2  .637254902
  3  .0980392157

: mreldif(A * X, b)
3.70074e-16

```

We can also check the effective rank by typing

```

: X = lssolve(A, b, rank = .)
: rank
3

```

◀

▷ Example 2: Nonsquare matrix

When A is not square and is not rank deficient, the solution of the least-squares problem provided by `lssolve()`,

$$\min \|AX - B\|_2$$

is the same as $X = (A'A)^{-1}A'B$. We can confirm that we get the same result with both methods with the example below.

```

: A = (3, 2 \ 2, 1 \ 1, 20)
: b = (1 \ 1.5 \ 2)
: A
      1  2
  1  3  2
  2  2  1
  3  1  20

: b
      1
  1  1
  2  1.5
  3  2

: X = lssolve(A, b)

```

```

: X
      1
1  .4138354482
2  .0787965616

: mreldif(luinv(A'A) * A' * b, X)
1.28641e-17

```

Now the effective rank is

```

: X = lssolve(A, b, rank = .)
: rank
2

```



▷ Example 3: Rank-deficient matrix

When A is not square but is rank deficient, we can find the solution to the least-squares problem

$$\min \|AX - B\|_2$$

with `lssolve()` as well. Note that we cannot specify the parameter *favorspeed* this time.

```

: A = (3, 3 \ 2, 2 \ 20, 20)
: b = (1 \ 1.5 \ 2)
: A
      1  2
1  3  3
2  2  2
3  20 20

: b
      1
1  1
2  1.5
3  2

: X = lssolve(A, b)
: X
      1
1  .0556900726
2  .0556900726

: mreldif(pinv(A'A) * A' * b, X)
1.97186e-17

```

Now the effective rank is

```
: X = lssolve(A, b, rank = .)
: rank
: 1
```



Conformability

`lssolve(A, B, rank, favorspeed, tol):`

input:

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

output:

```
rank:      1 × 1 (optional)
result:    n × k
```

`_lssolve(A, B, favorspeed, tol):`

input:

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

output:

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

`_leastsquare_lapacke(A, B, rank, favorspeed, tol) :`

input:

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

output:

```
A:      0 × 0
B:      n × k
rank:    1 × 1 (optional)
result:  1 × 1
```

Diagnostics

`lssolve(A, B, ...)`, `_lssolve(A, B, ...)`, and `_leastsquare_lapacke(A, B, ...)` return a result containing missing if A or B contains missing values. The functions with a nonzero *favorspeed* return a result containing all missing values if A is singular. The functions abort with error if set `lapack_mkl` on is not in effect on Windows or Linux or when set `lapack_openblas` on is not in effect on Mac.

`_lssolve(A, B, ...)` and `_leastsquare_lapacke(A, B, ...)` abort with error if A or B is a view.

`_leastsquare_lapacke(A, B, ...)` should not be used directly; use `_lssolve()`.

Also see

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

[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

