Title

solvelower( ) — Solve AX=B for X, A triangular

Syntax

numeric matrix `solvelower(`*A*`,` *B* [`,` *rank* [`,` *tol* [`,` *d*]]]`)`

numeric matrix `solveupper(`*A*`,` *B* [`,` *rank* [`,` *tol* [`,` *d*]]]`)`

real scalar `_solvelower(`*A*`,` *B* [`,` *tol* [`,` *d*]]`)`

real scalar `_solveupper(`*A*`,` *B* [`,` *tol* [`,` *d*]]`)`

where

A:	*numeric matrix*
B:	*numeric matrix*
rank:	irrelevant; *real scalar* returned
tol:	*real scalar*
d:	*numeric scalar*

Description

These functions are used in the implementation of the other solve functions; see [M-5] **lusolve( )**, [M-5] **qrsolve( )**, and [M-5] **svsolve( )**.

`solvelower(`*A*`,` *B*`, ...)` and `_solvelower(`*A*`,` *B*`, ...)` solve lower-triangular systems.

`solveupper(`*A*`,` *B*`, ...)` and `_solveupper(`*A*`,` *B*`, ...)` solve upper-triangular systems.

Functions without a leading underscore—`solvelower()` and `solveupper()`—return the solution; *A* and *B* are unchanged.

Functions with a leading underscore—`_solvelower()` and `_solveupper()`—return the solution in *B*.

All four functions produce a generalized solution if *A* is singular. The functions without an underscore place the rank of *A* in *rank*, if the argument is specified. The underscore functions return the rank.

Determination of singularity is made via *tol*. *tol* is interpreted in the standard way—as a multiplier for the default if $tol > 0$ is specified and as an absolute quantity to use in place of the default if $tol \leq 0$ is specified.

All four functions allow *d* to be optionally specified. Specifying $d = .$ is equivalent to not specifying *d*.

If $d \neq$. is specified, that value is used as if it appeared on the diagonal of A. The four functions do not in fact require that A be triangular; they merely look at the lower or upper triangle and pretend that the opposite triangle contains zeros. This feature is useful when a decomposition utility has stored both the lower and upper triangles in one matrix, because one need not take apart the combined matrix. In such cases, it sometimes happens that the diagonal of the matrix corresponds to one matrix but not the other, and that for the other matrix, one merely knows that the diagonal elements are, say, 1. Then you can specify $d = 1$.

Remarks and examples

stata.com

The triangular-solve functions documented here exploit the triangular structure in A and solve for X by recursive substitution.

When A is of full rank, these functions provide the same solution as the other solve functions, such as [M-5] **lusolve( )**, [M-5] **qrsolve( )**, and [M-5] **svsolve( )**. The `solvelower()` and `solveupper()` functions, however, will produce the answer more quickly because of the large computational savings.

When A is singular, however, you may wish to consider whether you want to use these triangular-solve functions. The triangular-solve functions documented here reach a generalized solution by setting $B_{ij} = 0$, for all j, when A_{ij} is zero or too small (as determined by *tol*). The method produces a generalized inverse, but there are many generalized inverses, and this one may not have the other properties you want.

Remarks are presented under the following headings:

Derivation
Tolerance

Derivation

We wish to solve

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

when A is triangular. Let us consider the lower-triangular case first. `solvelower()` is up to handling full matrices for B and X, but let us assume X: $n \times 1$ and B: $m \times 1$:

$$\begin{bmatrix} a_{11} & 0 & 0\ldots & 0 \\ a_{21} & 0 & 0\ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

The first equation to be solved is

$$a_{11}x_1 = b_1$$

and the solution is simply

$$x_1 = \frac{b_1}{a_{11}} \tag{2}$$

The second equation to be solved is

$$a_{21}x_1 + a_{22}x_2 = b_2$$

and because we have already solved for x_1, the solution is simply

$$x_2 = \frac{b_2 - a_{21}x_1}{a_{22}} \tag{3}$$

We proceed similarly for the remaining rows of A. If there are additional columns in B and X, we can then proceed to handling each remaining column just as we handled the first column above.

In the upper-triangular case, the formulas are similar except that you start with the last row of A.

Tolerance

In (2) and (3), we divide by the diagonal elements of A. If element a_{ii} is less than *eta* in absolute value, the corresponding x_i is set to zero. *eta* is given by

$$eta = \texttt{1e-13} * \texttt{trace(abs(}A\texttt{))/rows(}A\texttt{)}$$

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

Conformability

solvelower(*A*, *B*, *rank*, *tol*, *d*), solveupper(*A*, *B*, *rank*, *tol*, *d*):

input:

A:	$n \times n$	
B:	$n \times k$	
tol:	1×1	(optional)
d:	1×1	(optional)

output:

rank:	1×1	(optional)
result:	$n \times k$	

_solvelower(*A*, *B*, *tol*, *d*), _solveupper(*A*, *B*, *tol*, *d*):

input:

A:	$n \times n$	
B:	$n \times k$	
tol:	1×1	(optional)
d:	1×1	(optional)

output:

B:	$n \times k$	
result:	1×1	(contains rank)

Diagnostics

solvelower(*A*, *B*, ...), _solvelower(*A*, *B*, ...), solveupper(*A*, *B*, ...), and _solveupper(*A*, *B*, ...) do not verify that the upper (lower) triangle of *A* contains zeros; they just use the lower (upper) triangle of *A*.

_solvelower(*A*, *B*, ...) and _solveupper(*A*, *B*, ...) do not abort with error if *B* is a view but can produce results subject to considerable roundoff error.

Also see

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

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

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

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

[M-5] **solve_tol( )** — Tolerance used by solvers and inverters

[M-4] **matrix** — Matrix functions