_solvemat( ) — Solve AX=B for X

Description Syntax Remarks and examples Conformability
Diagnostics Also see

Description

`_solvemat(`*A*`,` *B*`)` solves *AX*=*B*. Rather than returning the solution *X*, it overwrites *B* with the solution, and, in the process of making the calculation, it destroys the contents of *A*. *B* is overwritten with a matrix of missing values if *A* is singular.

`_solvemat(`*A*`,` *B*`,` *mattype*`)` does the same thing but allows you to specify the type of the matrix *A*. The corresponding solver is used based on the type of the matrix *A*. *mattype* may be `"general"`, `"lowertriangular"`, `"uppertriangular"`, or `"symposdef"`; the default is `"general"`.

`_solvemat(`*A*`,` *B*`,` *mattype*`,` *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.

The above routines return `0` if the functions successfully find the solution and `1` otherwise.

Syntax

real scalar `_solvemat(`*A*`,` *B*`)`

real scalar `_solvemat(`*A*`,` *B*`,` *mattype*`)`

real scalar `_solvemat(`*A*`,` *B*`,` *mattype*`,` *tol*`)`

where inputs are

A: *numeric matrix*
B: *numeric matrix*
mattype: *string scalar*
tol: *real scalar*

and outputs are

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

and where *mattype*, optionally specified, is one of the following:

mattype	Description
`"general"`	general matrix
`"lowertriangular"`	lower-triangular matrix
`"uppertriangular"`	upper-triangular matrix
`"symposdef"`	symmetric (or Hermitian for complex) and positive definite matrix

The default is `"general"` if not set.

Remarks and examples

Remarks are presented under the following headings:

Introduction

`_solvemat(`*A*`,` *B*`)` solves *AX*=*B*; which solver is used depends on the type of matrix *A*. By default, `_solvemat(`*A*`,` *B*`)` uses LU decomposition. However, with `_solvemat(`*A*`,` *B*`,` *mattype*`)` you can specify the matrix type for *A*. Additionally, you can specify the tolerance for declaring that *A* is singular with `_solvemat(`*A*`,` *B*`,` *mattype*`,` *tol*`)`.

If matrix type (*mattype*) is not specified or if it is specified as `"general"`, the functions solve the linear system using LU decomposition.

If *mattype* is specified as `"lowertriangular"` (lower triangular matrix) or `"uppertriangular"` (upper triangular matrix), the functions solve the linear system using corresponding triangular systems solvers. Please note that the functions use the elements from the lower or upper triangle of *A* without checking whether *A* is a lower or upper triangular matrix.

If *mattype* is specified as `"symposdef"`, that is, symmetric (or Hermitian in the complex case) and positive definite matrix, the functions solve the linear system using Cholesky decomposition. Please note that the functions use the elements from the lower triangle of *A* without checking whether *A* is symmetric or, in the complex case, Hermitian.

Tolerance

The default tolerance for declaring that *A* is singular is the same for all the solvers implemented in `_solvemat()`; the only exception is that the tolerance is not applicable for the triangular systems solvers when Intel MKL LAPACK routines are used.

The default tolerance used is

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

where n is the number of rows for the matrix *A*. 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**.

Examples

▷ Example 1: General matrix

If *A* has full rank and no *mattype* is specified, `_solvemat()` computes the solution of *AX*=*B* using LU decomposition and returns 0.

```
: A
         1     2     3
    +-------------------+
  1 |    1     2     3  |
  2 |  2.5     5     1  |
  3 |    3     2     1  |
    +-------------------+

: b
         1
    +--------+
  1 |    .5  |
  2 |   1.5  |
  3 |   2.5  |
    +--------+

: rc = _solvemat(A2 = A, x = b)

: rc
  0

: x
                  1
    +----------------+
  1 |   .9615384615  |
  2 |  -.1730769231  |
  3 |  -.0384615385  |
    +----------------+

: mreldif(A * x, b)
  8.88178e-17
```

◁

▷ Example 2: Symmetric matrix

If A is a symmetric and positive definite matrix, we can specify *mattype* as `"symposdef"`; `_solvemat()` will compute the solution of AX=B using Cholesky decomposition and return `0`.

```
: A
[symmetric]
          1      2      3
    +----------------------+
  1 |     1                |
  2 |    .5      2         |
  3 |   .25    .15      3  |
    +----------------------+

: b
          1
    +---------+
  1 |    .5   |
  2 |   1.5   |
  3 |   2.5   |
    +---------+

: rc = _solvemat(A2 = A, x = b, "symposdef")

: rc
  0

: x
                   1
    +-----------------+
  1 |  -.0520428016   |
  2 |   .7028210117   |
  3 |   .8025291829   |
    +-----------------+

: mreldif(A * x, b)
  0
```

◁

Conformability

`_solvemat(`A`,` B`,` *mattype*`,` *tol*`)`:

input:

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

output:

A:	0×0	
B:	$n \times k$	
result:	1×1	

Diagnostics

_solvemat(*A*, *B*, ...) returns a result containing all missing values if *A* or *B* contains missing values or if *A* is singular.

_solvemat(*A*, *B*, ...) aborts with error if *A* or *B* is a view.

Also see

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

[M-5] **_invmat()** — Inverse and pseudoinverse of a square matrix

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

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

[M-5] **solvelower()** — 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

For suggested citations, see the FAQ on citing Stata documentation.