

cholsolve() — Solve $AX=B$ for X using Cholesky decomposition

Description Syntax Remarks and examples Conformability
 Diagnostics Also see

Description

`cholsolve(A, B)` solves $AX = B$ and returns X for symmetric ([Hermitian](#)), positive-definite A . `cholsolve()` returns a matrix of missing values if A is not positive definite or if A is singular.

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

`_cholsolve(A, B)` and `_cholsolve(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 .

`cholsolve_lapacke(A)`, `cholsolve_lapacke(A, tol)`, `_cholsolve_lapacke(A)`, and `_cholsolve_lapacke(A, tol)` are similar to their correspondent functions without `lapacke` endings, but instead they use interfaces to the [LAPACK](#) routines to compute the solutions.

Syntax

<i>numeric matrix</i>	<code>cholsolve(<i>numeric matrix A</i>, <i>numeric matrix B</i>)</code>
<i>numeric matrix</i>	<code>cholsolve(<i>numeric matrix A</i>, <i>numeric matrix B</i>, <i>real scalar tol</i>)</code>
<i>void</i>	<code>_cholsolve(<i>numeric matrix A</i>, <i>numeric matrix B</i>)</code>
<i>void</i>	<code>_cholsolve(<i>numeric matrix A</i>, <i>numeric matrix B</i>, <i>real scalar tol</i>)</code>
<i>numeric matrix</i>	<code>cholsolve_lapacke(<i>numeric matrix A</i>, <i>numeric matrix B</i>)</code>
<i>numeric matrix</i>	<code>cholsolve_lapacke(<i>numeric matrix A</i>, <i>numeric matrix B</i>, <i>real scalar tol</i>)</code>
<i>void</i>	<code>_cholsolve_lapacke(<i>numeric matrix A</i>, <i>numeric matrix B</i>)</code>
<i>void</i>	<code>_cholsolve_lapacke(<i>numeric matrix A</i>, <i>numeric matrix B</i>, <i>real scalar tol</i>)</code>

Remarks and examples

stata.com

The above functions solve $AX = B$ via Cholesky decomposition and are accurate. When A is not symmetric and positive definite, [\[M-5\] lusolve\(\)](#), [\[M-5\] qrsolve\(\)](#), and [\[M-5\] svsolve\(\)](#) are alternatives based on the LU decomposition, the QR decomposition, and the singular value decomposition (SVD). The alternatives differ in how they handle singular A . Then, the LU-based routines return missing values, whereas the QR-based and SVD-based routines return generalized (least-squares) solutions.

Remarks are presented under the following headings:

[Derivation](#)
[Relationship to inversion](#)
[Tolerance](#)

Derivation

We wish to solve for X

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

when A is symmetric and positive definite. Perform the Cholesky decomposition of A so that we have $A = GG'$. Then, (1) can be written as

$$GG'X = B \tag{2}$$

Define

$$Z = G'X \tag{3}$$

Then, (2) can be rewritten as

$$GZ = B \tag{4}$$

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

Relationship to inversion

See [Relationship to inversion](#) in [M-5] `lusolve()` for a discussion of the relationship between solving the linear system and matrix inversion.

Tolerance

The default tolerance used is

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

where G is the lower-triangular Cholesky factor of A : $n \times n$. A is declared to be singular if `cholesky()` (see [M-5] `cholesky()`) finds that A is not positive definite or, if A is found to be positive definite, if any diagonal element of G is less than or equal to η . Mathematically, positive definiteness implies that the matrix is not singular. In the numerical method used, two checks are made: `cholesky()` makes one, and then the η rule is applied to ensure numerical stability in the use of the result `cholesky()` returns.

If you specify $tol > 0$, the value you specify is used to multiply η . You may instead specify $tol \leq 0$, 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 to keep in mind 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 .

Conformability

`cholsolve(A, B, tol):`

input:

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

`_cholsolve(A, B, tol):`

input:

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

output:

$A:$ 0×0
 $B:$ $n \times k$

`cholsvelapacke(A, B, tol):`

input:

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

`_cholsvelapacke(A, B, tol):`

input:

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

output:

$A:$ 0×0
 $B:$ $n \times k$

Diagnostics

`cholsolve(A, B, ...)` and `_cholsolve(A, B, ...)` return a result of all missing values if A is not positive definite or if A contains missing values.

`_cholsolve(A, B, ...)` also aborts with error if A or B is a view.

`cholsvelapacke(A, B, ...)` and `_cholsvelapacke(A, B, ...)` return a result of all missing values if A is not positive definite or if A contains missing values.

`_cholsvelapacke(A, B, ...)` also aborts with error if A or B is a view.

All functions use the elements from the lower triangle of A without checking whether A is symmetric or, in the complex case, Hermitian.

Also see

[M-5] **cholesky()** — Cholesky square-root decomposition

[M-5] **cholinv()** — Symmetric, positive-definite matrix inversion

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

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

[M-4] **Matrix** — Matrix functions

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