

# Package ‘quadform’

May 9, 2026

**Type** Package

**Title** Efficient Evaluation of Quadratic Forms

**Version** 0.0-4

**Depends** R (>= 3.0.1)

**Suggests** testthat

**Maintainer** Robin K. S. Hankin <hankin.robin@gmail.com>

**Description** A range of quadratic forms are evaluated, using efficient methods. Unnecessary transposes are not performed. Complex values are handled consistently.

**License** GPL

**URL** <https://github.com/RobinHankin/quadform>

**BugReports** <https://github.com/RobinHankin/quadform/issues>

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Robin K. S. Hankin [aut, cre] (ORCID:  
<<https://orcid.org/0000-0001-5982-0415>>)

**Repository** CRAN

**Date/Publication** 2025-11-11 09:50:02 UTC

## Contents

quad.form . . . . .	2
<b>Index</b>	<b>5</b>

quad.form

*Evaluate a quadratic form efficiently***Description**

Given a square matrix  $M$  of size  $n \times n$ , and a matrix  $x$  of size  $n \times p$  (or a vector of length  $n$ ), evaluate various quadratic forms.

The archetype is `quad.form(M,x)` for real or complex square matrix  $M$  and vector or matrix  $x$ . This evaluates `Conj(t(x)) %% M %% x` but using `crossprod(crossprod(M,Conj(x)),x)` thus avoiding taking a needless transpose.

**Usage**

```
quad.form(M, x)
quad.form.inv(M, x)
quad.form.chol(chol, x)
quad.tform(M, x)
quad3.form(M, left, right)
quad3.tform(M, left, right)
quad.tform.inv(M, x)
quad.diag(M, x)
quad.tdiag(M, x)
quad3.diag(M, left, right)
quad3.tdiag(M, left, right)
cprod(x, y)
tcprod(x, y)
ht(x)
```

**Arguments**

<code>M</code>	Square matrix of size $n \times n$
<code>x, y</code>	Matrix of size $n \times p$ , or vector of length $n$
<code>chol</code>	Lower triangular Cholesky decomposition of the quadratic form, see details
<code>left, right</code>	In function <code>quad3.form()</code> , matrices with $n$ rows and arbitrary number of columns

**Details**

<code>ht(x)</code>	$\mathbf{x}^* = \overline{\mathbf{x}^T}$	<code>Conj(t(x))</code>	<code>ht(x)</code>
<code>cprod(x,y)</code>	$\mathbf{x}^* \mathbf{y}$	<code>ht(x) %% y</code>	<code>cp()</code>
<code>tcprod(x,y)</code>	$\mathbf{xy}^*$	<code>x %% ht(y)</code>	<code>tcp()</code>
<code>quad.form(M,x)</code>	$\mathbf{x}^* \mathbf{M} \mathbf{x}$	<code>ht(x) %% M %% x</code>	<code>qf()</code>
<code>quad.form.inv(M,x)</code>	$\mathbf{x}^* \mathbf{M}^{-1} \mathbf{x}$	<code>ht(x) %% solve(M) %% x</code>	<code>qfi()</code>
<code>quad.tform(M,x)</code>	$\mathbf{x} \mathbf{M} \mathbf{x}^*$	<code>x %% A %% ht(x)</code>	<code>qt()</code>
<code>quad.tform.inv(M,x)</code>	$\mathbf{x} \mathbf{M}^{-1} \mathbf{x}^*$	<code>x %% solve(M) %% ht(x)</code>	<code>qti()</code>
<code>quad3.form(M,l,r)</code>	$\mathbf{l}^* \mathbf{M} \mathbf{r}$	<code>t(l) %% M %% r</code>	<code>q3()</code>

quad3.form.inv(M,l,r)	$\mathbf{I}^* M^{-1} \mathbf{r}$	t(l) %% solve(M) %% r	q3i()
quad3.tform(M,l,r)	$\mathbf{I} M \mathbf{r}^*$	l %% M %% t(r)	q3t()
quad.diag(M,x)	diag( $\mathbf{x}^* M \mathbf{x}$ )	diag(quad.form(M,x))	qd()
quad.tdiag(M,x)	diag( $\mathbf{x} M \mathbf{x}^*$ )	diag(quad.tform(M,x))	qtd()
quad3.diag(M,l,r)	diag( $\mathbf{I}^* M \mathbf{r}$ )	diag(quad3.form(M,l,r))	q3d()
quad3.tdiag(M,l,r)	diag( $\mathbf{I} M \mathbf{r}^*$ )	diag(quad3.tform(M,l,r))	q3td()
quad.trace(M,x)	tr( $\mathbf{x}^* M \mathbf{x}$ )	tr(quad.form(M,x))	qt()
quad.ttrace(M,x)	tr( $\mathbf{x} M \mathbf{x}^*$ )	tr(quad.tform(M,x))	qtt()

In the above,  $\mathbf{x}^*$  denotes the *complex conjugate of the transpose*, also known as the *Hermitian transpose* (this only matters when considering complex numbers).

These various functions generally avoid taking needless expensive transposes in favour of using nested crossprod() and tcrossprod() calls. For example, the “meat” of quad.form() is just crossprod(crossprod(M,Conj(x)),x).

Functions such as quad.form.inv() avoid taking a matrix inverse. The meat of quad.form.inv(), for example, is cprod(x,solve(M,x)). Many people have stated things like “Never invert a matrix unless absolutely necessary”. But I have *never* seen a case where quad.form.inv(M,x) is faster than quad.form(solve(M),x).

One motivation for the package is to return consistent results with complex arguments. Note, for example, that base::crossprod(x,y) evaluates t(x) %% y and not, as one would almost always want, Conj(t(x)) %% y. Function cprod(), unlike crossprod(), is consistent and returns Conj(t(x)) %% y [or ht(x) %% y]; internally it is essentially crossprod(Conj(x),y).

Function quad.form.chol() interprets argument chol as the lower triangular Cholesky decomposition of the quadratic form. Remember that M.lower %% M.upper == M, and chol() returns the upper triangular matrix, so one needs to use the transpose t(chol(M)) in calls. If the Cholesky decomposition of M is available, then using quad.form.chol() and supplying chol(M) should generally be faster (for large matrices) than calling quad.form() and using M directly. The time saving is negligible for matrices smaller than about  $50 \times 50$ , even if the overhead of computing the decomposition is ignored.

Functions quad3.foo() take three arguments: a matrix M and two other vectors l and r [or left and right]. For these functions, M is not necessarily square although of course the matrices have to be compatible.

Functions quad3.form\_ab() and quad3.form\_bc() are helper functions not really intended for the end-user. They return mathematically identical results but differ in the bracketing order of their operations: quad3.form\_ab(M,l,r) returns  $(\mathbf{I}^* M) \mathbf{r}$  and quad3.form\_bc(M,l,r) returns  $\mathbf{I}^* (M \mathbf{r})$ . The mnemonic for their names is derived from the first multiplication when calculating  $(ab)c$  and  $a(bc)$ . Note that quad3.form\_ab(M,l,r) returns crossprod(crossprod(M,Conj(l)),r) rather than the mathematically equivalent cprod(cprod(M,l),r) on efficiency grounds (only a single conjugate is taken).

Function quad3.form() dispatches to either quad3.form\_ab() or quad3.form\_bc() depending on the dimensions of its argument as per the efficiency discussion at inst/quadform3test.Rmd. Similar considerations apply to quad3.tform(), quad3.tform\_ab(), and quad3.tform\_bc().

Terse forms [qf() for quad.form(), qti() for quad.tform.inv(), etc] are provided for the perl golfers among us.

**Value**

Generally, return a (dropped) matrix, real or complex as appropriate

**Note**

These functions are used extensively in the **emulator** and **calibrator** packages, primarily in the interests of elegant code, but also speed. For the problems I usually consider, the speedup (of `quad.form(M, x)` over `t(x) %*% M %*% x`, say) is marginal at best.

**Author(s)**

Robin K. S. Hankin

**Examples**

```

jj <- matrix(rnorm(80), 20, 4)
M <- crossprod(jj, jj)
M.lower <- t(chol(M))
x <- matrix(rnorm(8), 4, 2)

jj.1 <- t(x) %*% M %*% x
jj.2 <- quad.form(M, x)
jj.3 <- quad.form.chol(M.lower, x)
print(jj.1)
print(jj.2)
print(jj.3)

## Make two Hermitian positive-definite matrices:
L <- matrix(c(1, 0.1i, -0.1i, 1), 2, 2)
LL <- diag(11)
LL[2,1] <- -(LL[1,2] <- 0.1i)

z <- matrix(rnorm(22) + 1i*rnorm(22), 2, 11)

quad.diag(L, z)      # elements real because L is HPD
quad.tdiag(LL, z)   # ditto

## Now consider accuracy:
quad.form(solve(M), x) - quad.form.inv(M, x) # should be zero
quad.form(M, x) - quad.tform(M, t(x))       # should be zero
quad.diag(M, x) - diag(quad.form(M, x))     # should be zero
diag(quad.form(L, z)) - quad.diag(L, z)     # should be zero
diag(quad.tform(LL, z)) - quad.tdiag(LL, z) # should be zero

```

# Index

## \* **array**

- quad. form, 2
  
- cp (quad. form), 2
- cprod (quad. form), 2
  
- ht (quad. form), 2
  
- q3 (quad. form), 2
- q3d (quad. form), 2
- q3i (quad. form), 2
- q3t (quad. form), 2
- q3td (quad. form), 2
- qd (quad. form), 2
- qf (quad. form), 2
- qfi (quad. form), 2
- qt (quad. form), 2
- qtd (quad. form), 2
- qti (quad. form), 2
- qtr (quad. form), 2
- qttr (quad. form), 2
- quad.diag (quad. form), 2
- quad. form, 2
- quad.tdiag (quad. form), 2
- quad.tform (quad. form), 2
- quad.trace (quad. form), 2
- quad.ttrace (quad. form), 2
- quad3.diag (quad. form), 2
- quad3.form (quad. form), 2
- quad3.form\_ab (quad. form), 2
- quad3.form\_bc (quad. form), 2
- quad3.tdiag (quad. form), 2
- quad3.tform (quad. form), 2
- quad3.tform\_ab (quad. form), 2
- quad3.tform\_bc (quad. form), 2
- quadform (quad. form), 2
  
- tcp (quad. form), 2
- tcprod (quad. form), 2