Package ‘quadform’

May 14, 2024

Type Package

Title Efficient Evaluation of Quadratic Forms

Version 0.0-2

Depends R (>= 3.0.1)

Imports mathjaxr

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

RdMacros mathjaxr

R topics documented:

`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

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

Details

`ht(x)`	$\mathbf{x}^* = \overline{\mathbf{x}^T}$	`Conj(t(x))`	`ht(x)`
`cprod(x,y)`	$\mathbf{x}^*\mathbf{y}$	`ht(x) %*% y`	`cp()`
`tcprod(x,y)`	$\mathbf{x}\mathbf{y}^*$	`x %*% ht(y)`	`tcp()`
`quad.form(M,x)`	$\mathbf{x}^*M\mathbf{x}$	`ht(x) %*% M %*% x`	`qf()`
`quad.form.inv(M,x)`	$\mathbf{x}^*M^{-1}\mathbf{x}$	`ht(x) %*% solve(M) %*% x`	`qfi()`
`quad.tform(M,x)`	$\mathbf{x}M\mathbf{x}^*$	`x %*% A %*% ht(x)`	`qt()`
`quad.tform.inv(M,x)`	$\mathbf{x}M^{-1}\mathbf{x}^*$	`x %*% solve(M) %*% ht(x)`	`qti()`
`quad3.form(M,l,r)`	$\mathbf{l}^*M\mathbf{r}$	`t(l) %*% M %*% r`	`q3()`
`quad3.form.inv(M,l,r)`	$\mathbf{l}^*M^{-1}\mathbf{r}$	`t(l) %*% solve(M) %*% r`	`q3i()`
`quad3.tform(M,l,r)`	$\mathbf{l}M\mathbf{r}^*$	`l %*% M %*% t(r)`	`q3t()`
`quad.diag(M,x)`	$\mathrm{diag}(\mathbf{x}^*M\mathbf{x})$	`diag(quad.form(M,x))`	`qd()`
`quad.tdiag(M,x)`	$\mathrm{diag}(\mathbf{x}M\mathbf{x}^*)$	`diag(quad.tform(M,x))`	`qtd()`
`quad3.diag(M,l,r)`	$\mathrm{diag}(\mathbf{l}^*M\mathbf{r})$	`diag(quad3.form(M,l,r))`	`q3d()`
`quad3.tdiag(M,l,r)`	$\mathrm{diag}(\mathbf{l}M\mathbf{r}^*)$	`diag(quad3.tform(M,l,r))`	`q3td()`
`quad.trace(M,x)`	$\mathrm{tr}(\mathbf{x}^*M\mathbf{x})$	`tr(quad.form(M,x))`	`qt()`
`quad.ttrace(M,x)`	$\mathrm{tr}(\mathbf{x}M\mathbf{x}^*)$	`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 50×50, 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 $(\mathbf{l}^*M)\,\mathbf{r}$ and `quad3.form_bc(M,l,r)` returns $\mathbf{l}^*\,(M\mathbf{r})$. 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