Package ‘quadform’

April 9, 2024

Type Package

Title Efficient Evaluation of Quadratic Forms

Version 0.0-1

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.

(in the following, x^T denotes the complex conjugate of the transpose, also known as the Hermitian transpose. This only matters when considering complex numbers).

Usage

```
quad.form(M, x)
quad.form.inv(M, x)
quad.form.chol(chol,x)
quad.tform(M, x)
quad.3form(M,left,right)
quad.3tform(M,left,right)
quad.tform.inv(M,x)
quad.diag(M,x)
quad.tdiag(M,x)
quad.3diag(M,left,right)
quad.3tdiag(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 `quad.3form()`, matrices with n rows and arbitrary number of columns

Details

- Function `quad.form(M,x)` evaluates $x^T Mx$ in an efficient manner [terse form `qf()`]
- Function `quad.form.inv(M,x)` returns $x^T M^{-1} x$ using an efficient method that avoids inverting M [terse form `qfi()`]
- Function `quad.tform(M,x)` returns xMx^T using `tcrossprod()` without taking a transpose [`qt()`]
- Function `quad.tform.inv(M,x)` returns $xM^{-1}x^T$, although a single transpose is needed [`qti()`]
- Function `quad.3form(M,l,r)` returns $l^T Mr$ using nested calls to `crossprod()`. It's no faster than calling `crossprod()` directly, but makes code neater and less error-prone (IMHO) [`q3()`]
- Function `quad.3form.inv(M,l,r)` returns $l^T M^{-1} r$ [`q3i()`]
- Function `quad.3tform(M,l,r)` returns lMr^T using nested calls to `tcrossprod()`. Again, this is to make for neater code [`q3t()`]
- Function `quad.diag(M,x)` returns the *diagonal* of the (potentially very large) square matrix `quad.form(M,x)` without calculating the off diagonal elements [`qd()`]
- Function `quad.tdiag(M,x)` similarly returns the diagonal of `quad.tform(M,x)` [`qtd()`]
- Function `quad.3diag(M,l,r)` returns the diagonal of `quad.3form(M,l,r)` [`q3d()`]
- Function `quad.3tdiag(M,l,r)` returns the diagonal of `quad.3tform(M,l,r)` [`q3td()`]
- 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.

These functions invoke the following lower-level calls:

- Function `ht(x)` returns the Hermitian transpose, that is, the complex conjugate of the transpose, sometimes written x^*

- Function `cprod(x,y)` returns $x^T y$, equivalent to `crossprod(Conj(x),y)` `[cp()]`
- Function `tcprod(x,y)` returns xy^T, equivalent to `crossprod(x,Conj(y))` `[tcp()]`

Note again that in the calls above, "transpose" [that is, x^T] means "Conjugate transpose", or the Hermitian transpose.

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)`.

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.

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

See Also

`optimize`

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