Gram Schmidt with R

Asked 9 years, 5 months ago Modified 9 years, 5 months ago Viewed 7k times

8

4

Here is a MATLAB code for performing Gram Schmidt in page 1
http://web.mit.edu/18.06/www/Essays/gramschmidtmat.pdf

I am trying for hours and hours to perform this with R since I don't have MATLAB Here is my R

```
f=function(x){
m=nrow(x);
n=ncol(x);
Q=matrix(0,m,n);
R=matrix(0,n,n);

for(j in 1:n){
v=x[,j,drop=FALSE];

for(i in 1:j-1){
R[i,j]=t(Q[,i,drop=FALSE])%*%x[,j,drop=FALSE];
v=v-R[i,j]%*%Q[,i,drop=FALSE]
}

R[j,j]=max(svd(v)$d);
Q[,j,,drop=FALSE]=v/R[j,j]}

return(list(Q,R))}
```

It keeps on saying there is errors in either:

```
v=v-R[i,j]%*%Q[,i,drop=FALSE]
```

or

```
R[j,j]=max(svd(v)$d);
```

What is it that I am doing wrong translating MATLAB code to R???

r matlab Edit tags

Share Edit Follow Close Flag

edited Mar 27, 2013 at 2:07

Asad Saeeduddin
45.3k 6 87 135

asked Mar 23, 2013 at 6:57

user2201675
105 1 5

2 You may use Octave which is pretty close in functionalities to Matlab. You can download it from internet for free. You need slight or no modifications at all to run your Matlab programs in Octave. But unlike Matlab, Octave has no native GUI and only terminal-like command execution. So, you might need a little time to get used to Octave. – RAM Mar 23, 2013 at 7:09

I do not know whether the answers to this post might be helpful. I did not try using your data with

their approach: stackoverflow.com/questions/3238242/... – Mark Miller Mar 23, 2013 at 7:13

Without having yet a deeper look, but at least your line `for(i in 1:j-1)` should be `for(i in 1:(j-1))` and then probably you have to use brackets also here `v=v-R[i,j]%*%Q[,i,drop=FALSE]` . – Daniel Fischer Mar 23, 2013 at 7:20

What is the expected result? – Roman Luštrik Mar 23, 2013 at 8:08

math.ucla.edu/~yanovsky/Teaching/Math151B/handouts/... Using matrix A from example 2, I am trying to get the same answer for Q and R – user2201675 Mar 23, 2013 at 14:41

4 Answers

Sorted by:
Reset to default

Date modified (newest first)

4

You could simply use Hans W. Borchers' pracma package, which provides many Octave/Matlab functions translated in R.

```
> library(pracma)
> gramSchmidt
function (A, tol = .Machine$double.eps^0.5)
{
    stopifnot(is.numeric(A), is.matrix(A))
    m <- nrow(A)
    n <- ncol(A)
    if (m < n)
        stop("No. of rows of 'A' must be greater or equal no. of colums.")
    Q <- matrix(0, m, n)
    R <- matrix(0, n, n)
    for (k in 1:n) {
        Q[, k] <- A[, k]
        if (k > 1) {
            for (i in 1:(k - 1)) {
                R[i, k] <- t(Q[, i]) %*% Q[, k]
                Q[, k] <- Q[, k] - R[i, k] * Q[, i]
            }
        }
        R[k, k] <- Norm(Q[, k])
        if (abs(R[k, k]) <= tol)
            stop("Matrix 'A' does not have full rank.")
        Q[, k] <- Q[, k]/R[k, k]
    }
    return(list(Q = Q, R = R))
}
<environment: namespace:pracma>
```

Share Edit Follow Flag

answered Mar 24, 2013 at 10:25

Stéphane Laurent
62.6k 14 103 202

I didn't about this package thanks, but it removes all the fun of programming it yourself. If you want to be efficient, just stick to the `qr` function in R which is fast enough (see the benchmark). Along this line, I was actually thinking about a Julia version too (but posting someJulia code in the R section of SO can put you into trouble...) – dickoa Mar 24, 2013 at 14:15

1 @dickoa There are sufficiently other fun things to program with R :) – Stéphane Laurent Mar 24, 2013 at 14:55

4

Here a version very similar to yours but without the use of the extra variabale v. I use directly the Q matrix. So no need to use `drop` . Of course since you have `j-1` in the index you need to add the condition `j>1` .

```
f=function(x){
  m <- nrow(x)
  n <- ncol(x)
  Q <- matrix(0, m, n)
  R <- matrix(0, n, n)
  for (j in 1:n) {
    Q[, j] <- x[, j]
    if (j > 1) {
      for (i in 1:(j - 1)) {
        R[i, j] <- t(Q[, i]) %*% Q[, j]
        Q[, j] <- Q[, j] - R[i, j] * Q[, i]
      }
    }
    R[j, j] <- max(svd(Q[, j])$d)
    Q[, j] <- Q[, j]/R[j, j]
  }
  return(list(Q = Q, R = R))
}
```

EDIT add some benchmarking:

To get some real case I use the `Hilbert` matrix from the `Matrix` package.

```
library(microbenchmark)
library(Matrix)
A <- as.matrix(Hilbert(100))
microbenchmark(grahm_schimdtR(A),
               grahm_schimdtCpp(A),times = 100L)

Unit: milliseconds
expr       min         lq     median        uq        max neval
grahm_schimdtR(A) 330.77424 335.648063 337.443273 343.72888 601.793201   100
grahm_schimdtCpp(A)   1.45445   1.510768   1.615255   1.66816   2.062018   100
```

As expected **CPP** solution is really fster.

Share Edit Follow Flag

edited Mar 23, 2013 at 22:18

answered Mar 23, 2013 at 9:04

agstudy
118k 17 189 253

I want to vote up but I don't have enough reputation to do it yet. I didn't try this yet but thank you so much for your time! – user2201675 Mar 23, 2013 at 15:03

If you are translating code in Matlab into R, then code semantics (code logic) should remain same. For example, in your code, you are transposing Q in `t(Q[,i,drop=FALSE])` as per the

7

given Matlab code. But `Q[,i,drop=FALSE]` does not return the column in column vector. So, we can make it a column vector by using the statement:

```
matrix(Q[,i],n,1); # n is the number of rows.
```

There is no error in `R[j,j]=max(svd(v)$d)` if `v` is a vector (row or column).

Yes, there is an error in

```
v=v-R[i,j]%*%Q[,i,drop=FALSE]
```

because you are using a matrix multiplication. Instead you should use a normal multiplication:

```
v=v-R[i,j] * Q[,i,drop=FALSE]
```

Here `R[i,j]` is a number, whereas `Q[,i,drop=FALSE]` is a vector. So, dimension mismatch arises here.

One more thing, if `j` is 3 , then `1:j-1` returns [0,1,2]. So, it should be changed to `1:(j-1)` , which returns [1,2] for the same value for `j` . But there is a catch. If `j` is 2, then `1:(j-1)` returns [1,0]. So, 0th index is undefined for a vector or a matrix. So, we can bypass `0` value by putting a conditional expression.

Here is a working code for Gram Schmidt algorithm:

```
A = matrix(c(4,3,-2,1),2,2)
m = nrow(A)
n = ncol(A)
Q = matrix(0,m,n)
R = matrix(0,n,n)

for(j in 1:n)
{
    v = matrix(A[,j],n,1)
    for(i in 1:(j-1))
    {
        if(i!=0)
        {
            R[i,j] = t(matrix(Q[,i],n,1))%*%matrix(A[,j],n,1)
            v = v - (R[i,j] * matrix(Q[,i],n,1))
        }
    }
    R[j,j] = svd(v)$d
    Q[,j] = v/R[j,j]
}
```

If you need to wrap the code into a function, you can do so as per your convenience.

Share Edit Follow Flag

edited Mar 23, 2013 at 17:00

answered Mar 23, 2013 at 8:43

RAM

2,366 1 20 32

R should be an `n * n` square matrix. – Simon O'Hanlon Mar 23, 2013 at 13:19

I want to vote up but I don't have enough reputation to do it yet. Thank you for explanations along with the code! – user2201675 Mar 23, 2013 at 15:03

11

Just for fun I added an Armadillo version of this code and benchmark it

Armadillo code :

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

using namespace Rcpp;

//[[Rcpp::export]]
List grahm_schimdtCpp(arma::mat A) {
    int n = A.n_cols;
    int m = A.n_rows;
    arma::mat Q(m, n);
    Q.fill(0);
    arma::mat R(n, n);
    R.fill(0);
    for (int j = 0; j < n; j++) {
    arma::vec v = A.col(j);
    if (j > 0) {
        for(int i = 0; i < j; i++) {
        R(i, j) = arma::as_scalar(Q.col(i).t() *  A.col(j));
        v = v - R(i, j) * Q.col(i);
        }
    }
    R(j, j) = arma::norm(v, 2);
    Q.col(j) = v / R(j, j);
    }
    return List::create(_["Q"] = Q,
                        _["R"] = R
    );
    }
```

R code not optimized (directly based on algorithm)

```
grahm_schimdtR <- function(A) {
    m <- nrow(A)
    n <- ncol(A)
    Q <- matrix(0, nrow = m, ncol = n)
    R <- matrix(0, nrow = n, ncol = n)
    for (j in 1:n) {
    v <- A[ , j, drop = FALSE]
        if (j > 1) {
    for(i in 1:(j-1)) {
            R[i, j] <- t(Q[,i,drop = FALSE]) %*% A[ , j, drop = FALSE]
            v <- v - R[i, j] * Q[ ,i]
    }
    }
    R[j, j] = norm(v, type = "2")
    Q[ ,j] = v / R[j, j]
    }

    list("Q" = Q, "R" = R)
```

```
}
```

Native QR decomposition in R

```
qrNative <- function(A) {
    qrdec <- qr(A)
    list(Q = qr.R(qrdec), R = qr.Q(qrdec))
}
```

We will test it with the same matrix as in original document (link in the post above)

```
A <- matrix(c(4, 3, -2, 1), ncol = 2)

all.equal(grahm_schimdtR(A)$Q %*% grahm_schimdtR(A)$R, A)
## [1] TRUE

all.equal(grahm_schimdtCpp(A)$Q %*% grahm_schimdtCpp(A)$R, A)
## [1] TRUE

all.equal(qrNative(A)$Q %*% qrNative(A)$R, A)
## [1] TRUE
```

Now let's benchmark it

```
require(rbenchmark)
set.seed(123)
A <- matrix(rnorm(10000), 100, 100)
benchmark(qrNative(A),
          grahm_schimdtR(A),
          grahm_schimdtCpp(A),
          order = "elapsed")
##                  test replications elapsed relative user.self
## 3 grahm_schimdtCpp(A)          100   0.272    1.000     0.272
## 1         qrNative(A)          100   1.013    3.724     1.144
## 2   grahm_schimdtR(A)          100  84.279  309.849    95.042
##   sys.self user.child sys.child
## 3    0.000          0         0
## 1    0.872          0         0
## 2   72.577          0         0
```

I really love how easy to port code into Rcpp....

Share Edit Follow Flag

answered Mar 23, 2013 at 16:51

dickoa
17.9k 3 37 50

+1! funny and excellent initiative to translate this to an Rcpp code. – agstudy Mar 23, 2013 at 22:19