STATS607B HW4

GAO Zheng

February 8, 2017

Load some benchmarking tools

```
if (!require(microbenchmark)) {
  install.packages("microbenchmark")
  library(microbenchmark)
}
```

```
## Loading required package: microbenchmark
```

Initialize data and try R default regression

```
m <- 100
n <- 15
i <- 1:m
j <- 1:n
X <- outer(i, j, function(i,j){((i-1)/(m-1))^(j-1)})
Y = exp(sin(4*(i-1)/(m-1)))/2006.787453080206

lm(Y~X-1)$coefficients[15]
```

```
##        X15
## -0.1958184
```

R default runs terribly.

QR decomposition using standard Gram-Schmitd

```
QRstdGS <- function(){
  Q <- matrix(0, m, n)
  R <- matrix(0, n, n)
  for (j in 1:n) {
    #cat("j = ", j)
    v <- X[,j]
    if (j>1) {
      for (i in 1:(j-1)) {
        #cat("i = ",i)
        R[i,j] <- sum(X[,j]*Q[,i])
        v <- v - R[i,j]*Q[,i]
      }
    }
    R[j,j] <- sqrt(sum(v^2))
    Q[,j] <- v/R[j,j]
  }
  # max(abs(Q%*%R-X))
  # solve Rb=Q'Y=y
  b <- numeric(n)
```

```
  y <- t(Q)%*%Y
  b[n] <- y[n]/R[n,n]
  for (j in (n-1):1) {
    b[j] <- (y[j] - sum(b[(j+1):n]*R[j,(j+1):n]))/R[j,j]
  }
  (b15QRstdGE <- b[15])
}

QRstdGS()
```

```
## [1] 0.0005245056
```

QR decomposition using modified Gram-Schmitd

```
QRmodGS <- function(){
  Q <- matrix(0, m, n)
  R <- matrix(0, n, n)
  for (j in 1:n) {
    #cat("j = ", j)
    v <- X[,j]
    if (j>1) {
      for (i in 1:(j-1)) {
        #cat("i = ",i)
        R[i,j] <- sum(v*Q[,i])
        v <- v - R[i,j]*Q[,i]
      }
    }
    R[j,j] <- sqrt(sum(v^2))
    Q[,j] <- v/R[j,j]
  }
  # max(abs(Q%*%R-X))
  # solve Rb=Q'Y=y
  b <- numeric(n)
  y <- t(Q)%*%Y
  b[n] <- y[n]/R[n,n]
  for (j in (n-1):1) {
    b[j] <- (y[j] - sum(b[(j+1):n]*R[j,(j+1):n]))/R[j,j]
  }
  (b15QRmodGE <- b[15])
}

QRmodGS()
```

```
## [1] 0.9914539
```

QR decomposition using House Holder approach

```
QRHH <- function(){
  R <- X
  y <- Y
  for (j in 1:n) {
```

```
    x <- R[j:m,j]
    v <- x + sign(x[1]) * sqrt(sum(x^2)) * c(1,rep(0,m-j))
    R[j:m,j:n] <- (diag(m-j+1) - 2*outer(v,v)/sum(v^2)) %*% R[j:m,j:n]
    y[j:m] <- (diag(m-j+1) - 2*outer(v,v)/sum(v^2)) %*% y[j:m]
  }
  # max(Q%*%R - X)
  # solve Rb=Q'Y=y
  b <- numeric(n)
  b[n] <- y[n]/R[n,n]
  for (j in (n-1):1) {
    b[j] <- (y[j] - sum(b[(j+1):n]*R[j,(j+1):n]))/R[j,j]
  }
  (b15QRHouseHolder <- b[15])
}

QRHH()
```

```
## [1] 0.9999998
```

Cholesky decomposition and solve normal equation

```
XX <- t(X)%*%X
w <- t(X)%*%Y
L <- matrix(0,n,n)
for (j in 1:n) {
  L[j,j] <- sqrt(XX[j,j] - sum(L[j,1:(j-1)]^2))
  for (i in (j+1):n) {
    L[i,j] <- (XX[i,j] - sum(L[j,1:(j-1)]*L[i,1:(j-1)])) / L[j,j]
  }
}
max(abs(L%*%t(L)-XX),na.rm = T)
```

The Cholesky decomposition breaks down due to rounding in R! Last entry cannot be computed, although LL' is close to X'X .

Perhaps I should do it in MATLAB

Compare speed and accuracy

```
microbenchmark(list = c(QRstdGS(), QRmodGS(), QRHH()),times = 1e6)
```

```
## Unit: nanoseconds
##                  expr min lq     mean median uq   max neval
##  0.000524505634284717   3  5 8.384768      8 10 12668 1e+06
##     0.991453927119202   3  5 8.347794      8 10 12534 1e+06
##     0.999999830007055   3  5 8.580755      8 10 15648 1e+06
```

Householder decomposition is most accurate in terms of finding the regression coefficient for b_15.

The three methods have comparable speed, which is expected given that the computational cost is identical from the analysis of the algorithms.

Question 2

Standard GS

```
QRstdGS <- function(X){
  Q <- matrix(0, n, n)
  R <- matrix(0, n, n)
  for (j in 1:n) {
    v <- X[,j]
    if (j>1) {
      for (i in 1:(j-1)) {
        #cat("i = ",i)
        R[i,j] <- sum(X[,j]*Q[,i])
        v <- v - R[i,j]*Q[,i]
      }
    }
    R[j,j] <- sqrt(sum(v^2))
    Q[,j] <- v/R[j,j]
  }
  Q
}
```

Modified GS

```
QRmodGS <- function(X){
  Q <- matrix(0, n, n)
  R <- matrix(0, n, n)
  for (j in 1:n) {
    v <- X[,j]
    if (j>1) {
      for (i in 1:(j-1)) {
        #cat("i = ",i)
        R[i,j] <- sum(v*Q[,i])
        v <- v - R[i,j]*Q[,i]
      }
    }
    R[j,j] <- sqrt(sum(v^2))
    Q[,j] <- v/R[j,j]
  }
  Q
}
```

We compare Standard Gram-Schmitd and modified version under 2 setting considered by John R. Rice.[1]

1. First is for randomly generated matrices, sized ranging from 5x5 to 30x30
2. Then matrices generated by functions that induce colinearity among columns

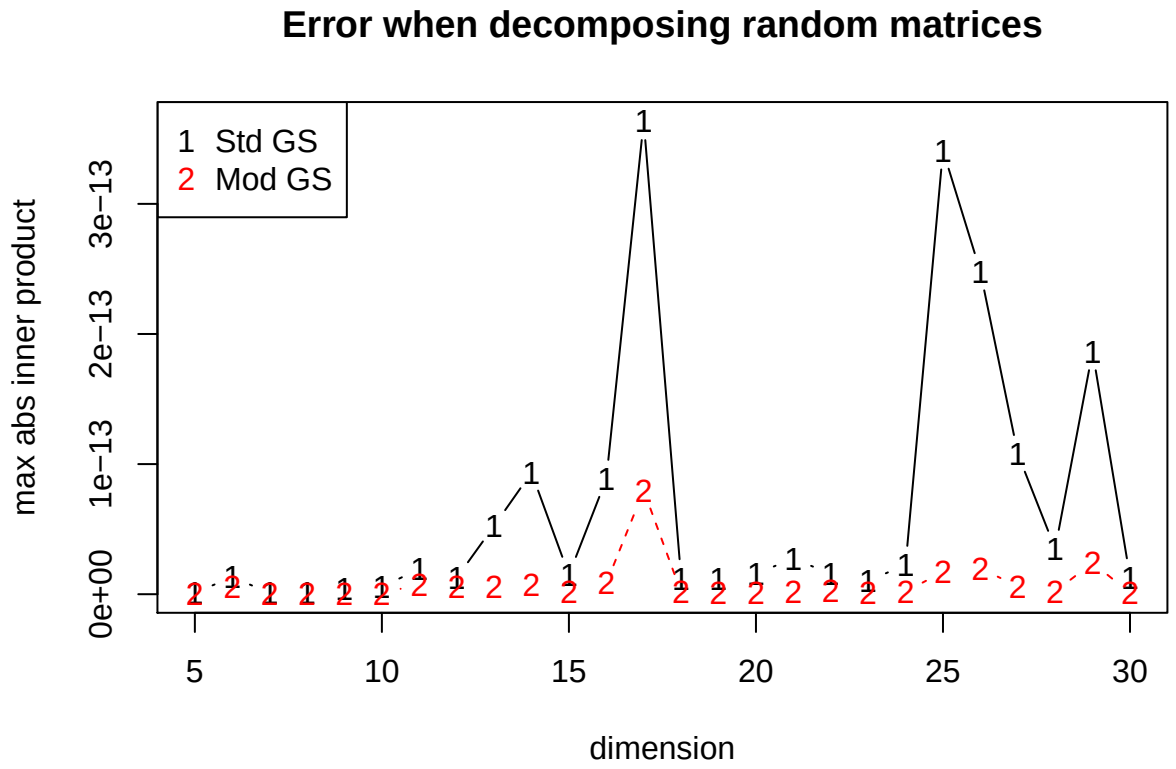
Evaluation of the accuracies will be based on the maximum absolute value of the inner products among the orthogonalized matrix Q

```
evaluate_Q <- function(Q) {
  max_inner_prod <- 0
  for (i in 1:(n-1)) {
    for (j in (i+1):n) {
      max_inner_prod <- max(abs(sum(Q[,i]*Q[,j])), max_inner_prod)
```

[1]Rice, John R. "Experiments on gram-schmidt orthogonalization." Mathematics of Computation 20.94 (1966): 325-328.

```
    }
  }
  max_inner_prod
}
```

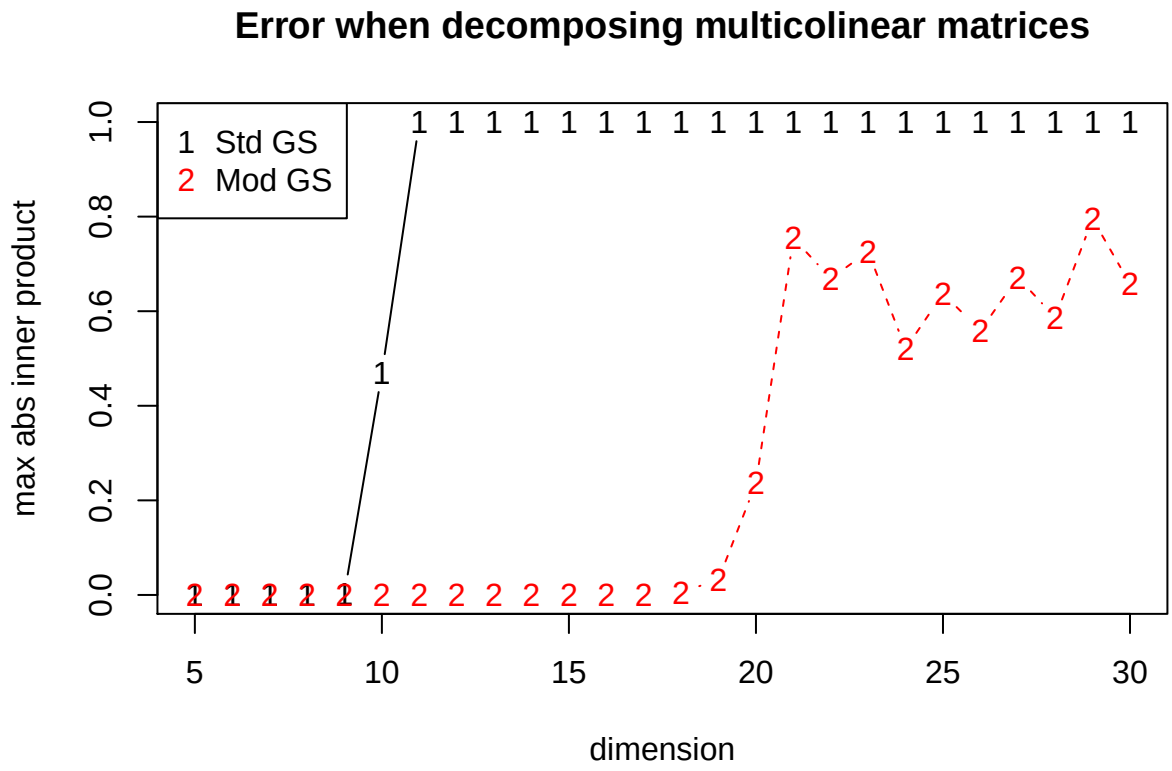
```
# Case 1: random matrices
dimension <- 5:30
result_stdGS <- numeric(length(dimension))
result_modGS <- numeric(length(dimension))
for (i in 1:length(dimension)) {
  n <- dimension[i]
  X <- matrix(runif(n^2),n,n)
  result_stdGS[i] <- evaluate_Q(QRstdGS(X))
  result_modGS[i] <- evaluate_Q(QRmodGS(X))
}
matplot(x = dimension, y = cbind(result_stdGS,result_modGS),
        ylab = "max abs inner product", type = 'b')
title("Error when decomposing random matrices")
legend("topleft",pch = c("1","2"), legend = c("Std GS","Mod GS"), col = 1:2)
```



Although the modified Gram-Schmitd performs consistently better than the standard Gram-Schmitd, the error is tiny ($\sim 10^{-13}$) for both algorithms.

```
# Case 2: polynomials
dimension <- 5:30
result_stdGS <- numeric(length(dimension))
result_modGS <- numeric(length(dimension))
for (i in 1:length(dimension)) {
  n <- dimension[i]
  #X <- matrix(runif(n^2),n,n)
```

```
  X <- outer(1:n,1:n-1,function(x,y){x^y})
  result_stdGS[i] <- evaluate_Q(QRstdGS(X))
  result_modGS[i] <- evaluate_Q(QRmodGS(X))
}
matplot(x = dimension, y = cbind(result_stdGS,result_modGS),
        ylab = "max abs inner product", type = 'b')
title("Error when decomposing multicolinear matrices")
legend("topleft",pch = c("1","2"), legend = c("Std GS","Mod GS"), col = 1:2)
```



In this case the modified Gram-Schmitd performs a lot better than the standard Gram-Schmitd. When dimension is large, the columns of X are highly dependent and multicolinear, the standard GS almost always produce identical columns. While the modified GS, although also performs poorly, is more stable.

Section 3 of John's paper gave a reasonable explanation of this phenomena.

In the standatd Gram-Schmitd the errors in taking the inner products can accumulate. When the independent element in the vector is small compared to the errors accumulated, the remainder will be a linear combination of the previous vectors still. This error is further blown up by the normalization step.

In the modified Gram-Schmitd, the latest vector is at least orthogonal to the immediate preceeding one within machine accuracy, thus preserving orthogonality better. (Although I believe this is no guarantee that the latest vector is orthogonal to even earlier ones, hence the errors are still large for vectors further apart.)