---

title: "Lab 5"

author: "Feiyi Ma"

output: pdf_document

date: "11:59PM March 18, 2021"

Create a 2x2 matrix with the first column 1's and the next column iid normals. Find the absolute value of the angle (in degrees, not radians) between the two columns.

```{r}
x = matrix(1:1, nrow =2, ncol=2)
x[,2]=rnorm(2)
norm_vec = function (v){
  sqrt(sum(v^2))
}

#x[,2]%*%t(x[,1])
cos_theta = t(x[,1])%*%x[,2]/(norm_vec(x[,1])%*%norm_vec(x[,2]))
cos_theta

acos(cos_theta)*180/pi
```

Repeat this exercise `Nsim = 1e5` times and report the average absolute angle.

```{r}
Nsim = 1e5 #100,000
```

```
angles = array(NA,Nsim)
for(i in 1:Nsim)  {
  x = matrix(1:1, nrow =2, ncol=2)
x[,2]=rnorm(2)
cos_theta = t(x[,1])%*%x[,2]/(norm_vec(x[,1])%*%norm_vec(x[,2]))
angles[i]=abs(90-acos(cos_theta)*180/pi)
}
mean(angles)
```

Create a nx2 matrix with the first column 1's and the next column iid normals. Find the absolute value of the angle (in degrees, not radians) between the two columns. For n = 10, 50, 100, 200, 500, 1000, report the average absolute angle over `Nsim = 1e5` simulations.

```{r}
#return 0-360 or -180 to 180
N=c(2,5,10,50,100, 200, 500, 1000)
Nsim = 1e5
angles = matrix(NA,nrow=Nsim,ncol =length(N))
for(j in 1:length(N)){
 for(i in 1:Nsim)  {
  x = matrix(1, nrow =N[j], ncol=2)
  x[,2]=rnorm(N[j])
  cos_theta = t(x[,1])%*%x[,2]/(norm_vec(x[,1])%*%norm_vec(x[,2]))
  angles[i,j]=abs(90-acos(cos_theta)*180/pi)
 }
}
colMeans(angles) #1000 damnation is 1.44 degree from orthogonal
```

What is this absolute angle converging to? Why does this make sense?

absolute angles form 90 to it converging to 0. This make sense because in high Dim orthogonal.

Create a vector y by simulating n = 100 standard iid normals. Create a matrix of size 100 x 2 and populate the first column by all ones (for the intercept) and the second column by 100 standard iid normals. Find the R^2 of an OLS regression of `y ~ X`. Use matrix algebra.

```{r}
n=100
X = cbind(1,rnorm(n))
y = rnorm(n)
H = X%*%solve(t(X)%*%X)%*% t(X)
y_hat = H %*%y
y_bar  = mean(y)
SSR =sum((y_hat-y_bar)^2)
SST = sum((y-y_bar)^2)

Rsq = (SSR/SST)
Rsq
```

Write a for loop to each time bind a new column of 100 standard iid normals to the matrix X and find the R^2 each time until the number of columns is 100. Create a vector to save all R^2's. What happened??

```{r}
#need to run the above function in order to run this one
Rsq_s = array(NA,dim = n-2)
```

```
for (j in 1:(n-2)){
  X=cbind(X,rnorm(n))
  H = X%*%solve(t(X)%*%X)%*% t(X)
y_hat = H %*%y
y_bar  = mean(y)
SSR =sum((y_hat-y_bar)^2)
SST = sum((y-y_bar)^2)

Rsq_s[j] = (SSR/SST)
}
Rsq_s
diff(Rsq_s)
```

Test that the projection matrix onto this X is the same as I_n. You may have to vectorize the matrices in the `expect_equal` function for the test to work.

```{r}
pacman::p_load(testthat)
dim(x) # y_hat = h_hat
H = X%*%solve(t(X)%*%X)%*% t(X) #identity matrix
H[1:10,1:10]  # 0 for all of them
I = diag(n)
expect_equal(I,H) # it passed after 10^(-7) or 10^(-8)
```

Add one final column to X to bring the number of columns to 101. Then try to compute R^2. What happens?

```{r}
 X=cbind(X,rnorm(n))
  H = X%*%solve(t(X)%*%X)%*% t(X)
  #solve will fail by 1 column,can invert the 101 by 101 in rank 100
y_hat = H %*%y
y_bar  = mean(y)
SSR =sum((y_hat-y_bar)^2)
SST = sum((y-y_bar)^2)

rsq = (SSR/SST)
rsq
```

Why does this make sense?

x_ inverse & x can not invert due to rank different. solve will fail by 1 column,can invert the 101 by 101 in rank 100

Write a function spec'd as follows:

```{r}
#' Orthogonal Projection
#'
#' Projects vector a onto v.
#'
#' @param a   the vector to project
#' @param v   the vector projected onto
#'
#' @returns   a list of two vectors, the orthogonal projection parallel to v named a_parallel,
```

```
#'          and the orthogonal error orthogonal to v called a_perpendicular
orthogonal_projection = function(a, v){
  H = v%*%t(v) / norm_vec(v)^2  #dimension is
  a_parallel = H%*%a
  a_perpendicular = a - a_parallel


  list(a_parallel = a_parallel, a_perpendicular = a_perpendicular)
}
```

Provide predictions for each of these computations and then run them to make sure you're correct.

```{r}
orthogonal_projection(c(1,2,3,4), c(1,2,3,4))
#prediction:
orthogonal_projection(c(1, 2, 3, 4), c(0, 2, 0, -1))
#prediction:
result = orthogonal_projection(c(2, 6, 7, 3), c(1, 3, 5, 7))
# v = c(1, 3, 5, 7) the direction of projection
t(result$a_parallel) %*% result$a_perpendicular
#prediction:
result$a_parallel + result$a_perpendicular
#prediction: give the original vector
result$a_parallel / c(1, 3, 5 ,7)
#prediction: % of the orthogonal line present from the original line
```

Let's use the Boston Housing Data for the following exercises

```{r}

#model to be worked on

y = MASS::Boston$medv

X = model.matrix(medv ~ ., MASS::Boston)

p_plus_one = ncol(X)

n = nrow(X)

head(X)

```

Using your function `orthogonal_projection` orthogonally project onto the column space of X by projecting y on each vector of X individually and adding up the projections and call the sum `yhat_naive`.

```{r}

yhat_naive = rep(0,n)

 for(j in 1:p_plus_one){  #col use j


   yhat_naive = yhat_naive + orthogonal_projection(y,X[,j])$a_parallel

   # like += itself   second is a matrix need to pull the data out


 }




```

How much double counting occurred? Measure the magnitude relative to the true LS orthogonal projection.

```{r}
yhat = X %*% solve(t(X) %*% X) %*% t(X) %*% y
  sqrt(sum(yhat_naive^2)) / sqrt(sum(yhat^2))
  #yhat_naive not y_hat, many doube double counting
```

Is this ratio expected? Why or why not?

#TO-DO yhat_naive not y_hat, many double double counting. It's expected to be differnt from 1.

Convert X into V where V has the same column space as X but has orthogonal columns. You can use the function `orthogonal_projection`. This is the Gram-Schmidt orthogonalization algorithm.

```{r}
V = matrix(NA, nrow = n, ncol = p_plus_one)
V[ , 1] = X[ , 1]
for (j in 2:p_plus_one){
  V[,j]= X[,j]
  for (k in 1:(j-1)){
    V[,j]=V[,j] - orthogonal_projection(X[,j], V[,k])$a_parallel
  }
}
head(X)
V[,7]%*%V[,9]
```

Convert V into Q whose columns are the same except normalized

```{r}
```

```
Q = matrix(NA, nrow = n, ncol = p_plus_one)
for (j in 1:p_plus_one){
  Q[,j] = V[, j] / norm_vec(V[,j])
}
#TO-DO
```

Verify Q^T Q is I_{p+1} i.e. Q is an orthonormal matrix.

```{r}
expect_equal(t(Q)%*%Q, diag(p_plus_one))
```

Is your Q the same as what results from R's built-in QR-decomposition function?

```{r}
Q_from_Rs_builtin = qr.Q(qr(x))
#qr.Q is the get function
expect_equal(Q,Q_from_Rs_builtin)  # 7084 =506*18 matrix,
```

Is this expected? Why did this happen?

#To-do Yes, the Q matric is 506 by 18 = 7084,the original matirx, the qr of Q matix is the orthognal projection of the matix. It showsonly part of the matix is orthognal to the orgianal one.

Project y onto colsp[Q] and verify it is the same as the OLS fit. You may have to use the function `unname` to compare the vectors since they the entries will likely have different names.

```{r}
yhat_naive = rep(0,ncol(Q))
 for(j in 1:ncol(Q)){  #col use j

   yhat_naive = yhat_naive + orthogonal_projection(y,Q[,j])$a_parallel
   # like += itself   second is a matrix need to pull the data out

 }
expect_equal(yhat_naive,orthogonal_projection(y,Q)$a_parallel)
```

Project y onto colsp[Q] one by one and verify it sums to be the projection onto the whole space.

```{r}
yhat_naive = rep(0,ncol(Q))
 for(j in 1:ncol(Q)){  #col use j

   yhat_naive = yhat_naive + orthogonal_projection(y,Q[,j])$a_parallel
   # like += itself   second is a matrix need to pull the data out
 }
expect_equal(sum(yhat_naive),sum(orthogonal_projection(y,Q)$a_parallel))
```

Split the Boston Housing Data into a training set and a test set where the training set is 80% of the observations. Do so at random.

```{r}
K = 5
n_test= round(n* 1/K)
```

```
n_train = n- n_test

test_indices = sample(1:n, n * 1 / K)

train_indices = setdiff(1:n , test_indices)

y=MASS::Boston$medv

y_bar = mean(y)

SST = sum((y-y_bar)^2)

x = as.matrix((cbind(1,MASS::Boston[,1:13])))

n = nrow(x)


x_train = x[train_indices,]

x_test = x[test_indices,]

y_train = y[train_indices]

y_test = y[test_indices]


dim(x_train)

dim(x_test)

length(y_train)

length(y_test)
```

Fit an OLS model. Find the s_e in sample and out of sample. Which one is greater? Note: we are now using s_e and not RMSE since RMSE has the n-(p + 1) in the denominator not n-1 which attempts to de-bias the error estimate by inflating the estimate when overfitting in high p. Again, we're just using `sd(e)`, the sample standard deviation of the residuals.

```{r}
mod = lm(y_train ~ ., data.frame(x_train))

summary(mod)$r.squared

sd(mod$residuals)
```

```

Do these two exercises `Nsim = 1000` times and find the average difference between s_e and ooss_e.

```{r}
K= 5
n_test= round(n* 1/K)
n_train = n- n_test
Nsim = 1000
ooss_e = array(NA, dim = Nsim)
s_e = array(NA, dim = Nsim)

for (i in 1:Nsim){

test_indices = sample(1:n, 1 / K * n)
train_indices = setdiff(1:n , test_indices)
#y=MASS::Boston$medv
#y_bar = mean(y)
#SST = sum((y-y_bar)^2)
#x = as.matrix((cbind(1,MASS::Boston[,1:13])))
#n = nrow(x)

x_train = x[train_indices,]
x_test = x[test_indices,]
y_train = y[train_indices]
y_test = y[test_indices]

N_matrix = lm(y_train~.+0, data.frame(x_train))
y_hat_oos  = predict(N_matrix, data.frame(x_test))
```
```

```
ooss_e[i] = sd(y_test - y_hat_oos)

s_e[i]  = sd(N_matrix$residuals)

}

abs(mean(s_e - ooss_e))
```

```
```

We'll now add random junk to the data so that `p_plus_one = n_train` and create a new data matrix `X_with_junk.`

```{r}
#p_plus_one = n_train

x_with_junk = cbind(x, matrix(rnorm(n * (n_train - p_plus_one)), nrow = n))

dim(x)

dim(x_with_junk)
```

```
```

Repeat the exercise above measuring the average s_e and ooss_e but this time record these metrics by number of features used. That is, do it for the first column of `X_with_junk` (the intercept column), then do it for the first and second columns, then the first three columns, etc until you do it for all columns of `X_with_junk`. Save these in `s_e_by_p` and `ooss_e_by_p`.

```{r}
K= 5

n_test= round(n* 1/K)

n_train = n- n_test

ooss_e_by_p = array(NA, dim = ncol(x_with_junk))

s_e_by_p = array(NA, dim = ncol(x_with_junk))

Nsim = 10
```

```
for (i in 1:ncol(x_with_junk)){
      ooss_e = array(NA, dim = Nsim)
      s_e = array(NA, dim = Nsim)
  for(j in 1:Nsim){

test_indices = sample(1:n, 1 / K * n)
train_indices = setdiff(1:n , test_indices)



x_train = x_with_junk[train_indices, 1:j, drop= FALSE]
x_test = x_with_junk[test_indices, 1:j, drop= FALSE]
y_train = y[train_indices]
y_test = y[test_indices]

N_matrix = lm(y_train~.+0, data.frame(x_train))
y_hat_oos  = predict(N_matrix, data.frame(x_test))
ooss_e[i] = sd(y_test - y_hat_oos)
s_e[i]  = sd(N_matrix$residuals)
  }
  ooss_e_by_p[j] = mean(ooss_e)
  s_e_by_p[j] = mean(s_e)
}
ooss_e_by_p
s_e_by_p
```

You can graph them here:

```{r}
pacman::p_load(ggplot2)
ggplot(
  rbind(
    data.frame(s_e = s_e_by_p, p = 1 : n_train, series = "in-sample"),
    data.frame(s_e = ooss_e_by_p, p = 1 : n_train, series = "out-of-sample")
  )) +
  geom_line(aes(x = p, y = s_e, col = series))
```

Is this shape expected? Explain.

#TO-DO Should be, as we add more junk, there are small part orthognal to the regression as overfitting all the regression.