Lab 5

Jaime Lin

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.

```
norm_vec = function (v){
  sqrt(sum(v^2))
}

X<-matrix(1, nrow = 2, ncol = 2)
X[,2] = rnorm(2)
cos_Theta = t(X[,1]) %*% X[,2]/ (norm_vec(X[,1]) *norm_vec(X[,2]))
cos_Theta
```

```
##            [,1]
## [1,] -0.4886468
```

```
abs(90 - acos(cos_Theta) * 180 / pi)
```

```
##          [,1]
## [1,] 29.25168
```

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

```
Nsim = 1e5
angles = array(NA, Nsim)

for (i in 1:Nsim) {
matrix(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)
```

```
## [1] 45.18397
```

Create a 2xn 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.

```
n_s = c(2, 5, 10, 50, 100, 200, 500, 1000)
Nsim = 1e5
angles = matrix(NA, nrow = Nsim, ncol = length(n_s))
```

```
for (j in 1:length(n_s)) {
  for (i in 1:Nsim) {
  X= matrix(1, nrow = n_s[j], ncol = 2)
  X[,2] = rnorm(n_s[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)
```

```
## [1] 44.959304 23.160798 15.317266  6.509503  4.608693  3.247427  2.041592
## [8]  1.446941
```

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

The absolute angle difference from ninety is converging to zero. This make sense because in a high dimensional space random directions are 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.

```
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
```

```
## [1] 0.02873395
```

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??

```
Rsq_s = array(NA, dim = n-2)

for (i 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[i] = SSR / SST
}

Rsq_s
```

```
##  [1] 0.02874106 0.03428715 0.03448227 0.03574930 0.03789131 0.06387732
##  [7] 0.06397891 0.07439874 0.07446485 0.07853353 0.08465336 0.08548007
## [13] 0.10653485 0.12369037 0.12462757 0.12577153 0.13160898 0.13229897
## [19] 0.14181199 0.14494531 0.15657935 0.20818364 0.20899509 0.20922115
## [25] 0.21350130 0.23499337 0.25230412 0.25475207 0.25712014 0.25756732
## [31] 0.26007777 0.26059945 0.26292045 0.26292989 0.26405571 0.26436192
## [37] 0.27777574 0.28778709 0.29172717 0.29279651 0.30643679 0.34338930
## [43] 0.35080472 0.35298600 0.36943934 0.38468876 0.40372143 0.41217566
## [49] 0.41363777 0.45309519 0.45522390 0.45726929 0.46771310 0.48218649
## [55] 0.49141920 0.50456999 0.50472014 0.51073564 0.53794197 0.56196363
## [61] 0.57031129 0.57064324 0.58795487 0.58800855 0.59169560 0.60736141
## [67] 0.60988419 0.62141014 0.62141091 0.66709916 0.68352520 0.68649885
## [73] 0.71862447 0.74740256 0.75896331 0.76567835 0.76855976 0.78320514
## [79] 0.78444531 0.78745441 0.78777279 0.80660580 0.80660632 0.80666217
## [85] 0.80977707 0.80987335 0.84334217 0.86244879 0.88350787 0.88842764
## [91] 0.88891809 0.89628546 0.92856209 0.93928861 0.93939400 0.93977935
## [97] 0.99992250 1.00000000
```

```
diff(Rsq_s)
```

```
##  [1] 5.546091e-03 1.951181e-04 1.267026e-03 2.142015e-03 2.598601e-02
##  [6] 1.015855e-04 1.041983e-02 6.610474e-05 4.068683e-03 6.119833e-03
## [11] 8.267018e-04 2.105478e-02 1.715552e-02 9.372052e-04 1.143957e-03
## [16] 5.837455e-03 6.899844e-04 9.513020e-03 3.133323e-03 1.163404e-02
## [21] 5.160429e-02 8.114438e-04 2.260611e-04 4.280153e-03 2.149207e-02
## [26] 1.731075e-02 2.447947e-03 2.368070e-03 4.471803e-04 2.510447e-03
## [31] 5.216874e-04 2.320994e-03 9.442291e-06 1.125823e-03 3.062049e-04
## [36] 1.341382e-02 1.001136e-02 3.940081e-03 1.069341e-03 1.364027e-02
## [41] 3.695252e-02 7.415414e-03 2.181281e-03 1.645334e-02 1.524942e-02
## [46] 1.903267e-02 8.454234e-03 1.462108e-03 3.945742e-02 2.128704e-03
## [51] 2.045399e-03 1.044380e-02 1.447339e-02 9.232712e-03 1.315078e-02
## [56] 1.501500e-04 6.015506e-03 2.720633e-02 2.402166e-02 8.347659e-03
## [61] 3.319542e-04 1.731163e-02 5.367609e-05 3.687051e-03 1.566581e-02
## [66] 2.522779e-03 1.152596e-02 7.710263e-07 4.568825e-02 1.642604e-02
## [71] 2.973656e-03 3.212562e-02 2.877809e-02 1.156075e-02 6.715040e-03
## [76] 2.881411e-03 1.464538e-02 1.240171e-03 3.009102e-03 3.183764e-04
## [81] 1.883301e-02 5.149395e-07 5.585016e-05 3.114903e-03 9.628348e-05
## [86] 3.346881e-02 1.910662e-02 2.105907e-02 4.919774e-03 4.904524e-04
## [91] 7.367372e-03 3.227663e-02 1.072651e-02 1.053936e-04 3.853475e-04
## [96] 6.014315e-02 7.749796e-05
```

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.

```
pacman::p_load(testthat)
dim(X)
```

```
## [1] 100 100

H = X %*% solve((t(X) %*% X)) %*% t(X)
H[1:10, 1:10]

##                [,1]          [,2]          [,3]          [,4]
[,5]
##  [1,]  1.000000e+00  1.595946e-15  3.365364e-15  2.994133e-14  4.987677e-
14
##  [2,]  1.705905e-14  1.000000e+00  4.507050e-14  1.527337e-14  5.443042e-
14
##  [3,]  3.731043e-14 -1.619191e-14  1.000000e+00 -8.690097e-14  1.803419e-
14
##  [4,] -4.718448e-15 -4.735101e-14 -1.839848e-14  1.000000e+00  1.726397e-
14
##  [5,] -7.687861e-15  3.740064e-14  2.017960e-14 -3.239379e-14
1.000000e+00
##  [6,] -1.503485e-14 -2.065362e-14  3.906857e-14 -2.710158e-14 -1.378064e-
14
##  [7,] -5.433848e-14 -9.499346e-14  5.042494e-14 -5.287437e-14  2.720046e-
15
##  [8,]  6.170303e-14 -3.190330e-14 -3.311370e-15  3.320911e-14  1.705407e-
14
##  [9,] -5.449113e-14  7.539802e-14  6.391068e-14  2.955969e-14 -9.492407e-
15
## [10,] -1.269124e-14 -4.497791e-14 -2.305794e-14 -1.026956e-15 -3.941292e-
14
##                [,6]          [,7]          [,8]          [,9]
[,10]
##  [1,] -1.665335e-15 -1.418310e-14 -7.147061e-15 -3.110012e-14  2.280121e-
14
##  [2,]  1.112235e-13  8.798344e-14 -4.428923e-14  4.021870e-14  3.590270e-
14
##  [3,]  6.869505e-15 -6.711992e-14  4.315992e-15 -9.641593e-15 -1.719111e-
15
##  [4,]  1.010303e-14 -1.282308e-14  1.107447e-14  1.798561e-14  3.509693e-
14
##  [5,] -5.934142e-14 -7.840950e-16 -1.640701e-14  2.872529e-14 -1.554746e-
14
##  [6,]  1.000000e+00  6.297046e-14  8.472389e-15  2.436246e-14 -2.295386e-
14
##  [7,]  3.363976e-14  1.000000e+00  2.682576e-14  4.038436e-15  3.247402e-
14
##  [8,]  3.727400e-14  4.138183e-15  1.000000e+00  6.246826e-14  2.452335e-
14
##  [9,] -3.097522e-14  9.645063e-14  5.132006e-14  1.000000e+00  1.354472e-
14
## [10,] -4.607426e-15 -1.317557e-13  2.778333e-14 -3.230749e-14
1.000000e+00
```

```
I = diag(n)
expect_equal(H, I)
```

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

```
  X = cbind(X, rnorm(n))
 # H = X %*% solve((t(X) %*% X)) %*% t(X) #you get a error because you get a
small number
 # y_hat = H %*% y
 # y_bar = mean(y)
 # SSR = sum((y_hat - y_bar)^2)
 # SST = sum((y - y_bar)^2)
 # Rsq = SSR / SST
```

Why does this make sense?

Line 113 fails because you cannot invert a rank defficient matrix.

Write a function spec'd as follows:

```
#' 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

  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.

```
orthogonal_projection(c(1,2,3,4), c(1,2,3,4))
```

```
## $a_parallel
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```
##
## $a_perpendicular
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
```

```
#prediction: this is going to show a parallel line.
orthogonal_projection(c(1, 2, 3, 4), c(0, 2, 0, -1))
```

```
## $a_parallel
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
##
## $a_perpendicular
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```
#prediction: This is going to show a perpendicular line.
result = orthogonal_projection(c(2, 6, 7, 3), c(1, 3, 5, 7))
t(result$a_parallel) %*% result$a_perpendicular
```

```
##               [,1]
## [1,] -3.552714e-15
```

```
#prediction: this going to show a matrix
result$a_parallel + result$a_perpendicular
```

```
##      [,1]
## [1,]    2
## [2,]    6
## [3,]    7
## [4,]    3
```

```
#prediction: this going to show the sum
result$a_parallel / c(1, 3, 5 ,7)
```

```
##           [,1]
## [1,] 0.9047619
## [2,] 0.9047619
## [3,] 0.9047619
## [4,] 0.9047619
```

```
#prediction: this going to show itself
```

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

```
y = MASS::Boston$medv
X = model.matrix(medv ~ ., MASS::Boston)
p_plus_one = ncol(X)
n = nrow(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`.

```
yhat_naive = rep(0,n)
  for (i in 1:p_plus_one) {
    yhat_naive = yhat_naive +orthogonal_projection(y,X[,i])$a_parallel

}
```

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

```
yhat = X %*% solve((t(X) %*% X)) %*% t(X) %*% y
sqrt(sum(yhat_naive^2)) / sqrt(sum(yhat^2))
```

```
## [1] 8.997118
```

Is this ratio expected? Why or why not?

#TO-DO It 's expected to be different 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.

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

Convert V into Q whose columns are the same except normalized

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

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

```
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?

```
Q_from_Rs_builtin = qr.Q(qr(X))
#expect_equal(Q, Q_from_Rs_builtin) # It gives us an error. It found 2024
mismatches. This means that there are 2024 values that are not equal.
```

Is this expected? Why did this happen?

#TO-DO Many orthonormal basis of

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.

```
colsp = array(NA)
for (i in 1:length(y)) {
  colsp[i] = y[i]
}

unname(colsp)
```

```
##   [1] 24.0 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.0 18.9 21.7
20.4 18.2
##  [16] 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
18.4 21.0
##  [31] 12.7 14.5 13.2 13.1 13.5 18.9 20.0 21.0 24.7 30.8 34.9 26.6 25.3
24.7 21.2
##  [46] 19.3 20.0 16.6 14.4 19.4 19.7 20.5 25.0 23.4 18.9 35.4 24.7 31.6
23.3 19.6
##  [61] 18.7 16.0 22.2 25.0 33.0 23.5 19.4 22.0 17.4 20.9 24.2 21.7 22.8
23.4 24.1
##  [76] 21.4 20.0 20.8 21.2 20.3 28.0 23.9 24.8 22.9 23.9 26.6 22.5 22.2
23.6 28.7
##  [91] 22.6 22.0 22.9 25.0 20.6 28.4 21.4 38.7 43.8 33.2 27.5 26.5 18.6
19.3 20.1
## [106] 19.5 19.5 20.4 19.8 19.4 21.7 22.8 18.8 18.7 18.5 18.3 21.2 19.2
20.4 19.3
## [121] 22.0 20.3 20.5 17.3 18.8 21.4 15.7 16.2 18.0 14.3 19.2 19.6 23.0
18.4 15.6
## [136] 18.1 17.4 17.1 13.3 17.8 14.0 14.4 13.4 15.6 11.8 13.8 15.6 14.6
17.8 15.4
## [151] 21.5 19.6 15.3 19.4 17.0 15.6 13.1 41.3 24.3 23.3 27.0 50.0 50.0
50.0 22.7
## [166] 25.0 50.0 23.8 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6
29.9 37.2
## [181] 39.8 36.2 37.9 32.5 26.4 29.6 50.0 32.0 29.8 34.9 37.0 30.5 36.4
31.1 29.1
## [196] 50.0 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.0 22.6 24.4 22.5
24.4 20.0
## [211] 21.7 19.3 22.4 28.1 23.7 25.0 23.3 28.7 21.5 23.0 26.7 21.7 27.5
30.1 44.8
```

```
## [226] 50.0 37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.0 24.0 25.1 31.5
23.7 23.3
## [241] 22.0 20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8 29.6
42.8 21.9
## [256] 20.9 44.0 50.0 36.0 30.1 33.8 43.1 48.8 31.0 36.5 22.8 30.7 50.0
43.5 20.7
## [271] 21.1 25.2 24.4 35.2 32.4 32.0 33.2 33.1 29.1 35.1 45.4 35.4 46.0
50.0 32.2
## [286] 22.0 20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9 21.7 28.6 27.1 20.3
22.5 29.0
## [301] 24.8 22.0 26.4 33.1 36.1 28.4 33.4 28.2 22.8 20.3 16.1 22.1 19.4
21.6 23.8
## [316] 16.2 17.8 19.8 23.1 21.0 23.8 23.1 20.4 18.5 25.0 24.6 23.0 22.2
19.3 22.6
## [331] 19.8 17.1 19.4 22.2 20.7 21.1 19.5 18.5 20.6 19.0 18.7 32.7 16.5
23.9 31.2
## [346] 17.5 17.2 23.1 24.5 26.6 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7
22.7 22.6
## [361] 25.0 19.9 20.8 16.8 21.9 27.5 21.9 23.1 50.0 50.0 50.0 50.0 50.0
13.8 13.8
## [376] 15.0 13.9 13.3 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4
10.2 11.5
## [391] 15.1 23.2  9.7 13.8 12.7 13.1 12.5  8.5  5.0  6.3  5.6  7.2 12.1
8.3  8.5
## [406]  5.0 11.9 27.9 17.2 27.5 15.0 17.2 17.9 16.3  7.0  7.2  7.5 10.4
8.8  8.4
## [421] 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.0  9.5 14.5 14.1 16.1
14.3 11.7
## [436] 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
14.1 13.0
## [451] 13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.0 16.4 17.7 19.5
20.2 21.4
## [466] 19.9 19.0 19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3 16.7 12.0
14.6 21.4
## [481] 23.0 23.7 25.0 21.8 20.6 21.2 19.1 20.6 15.2  7.0  8.1 13.6 20.1
21.8 24.5
## [496] 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9 22.0 11.9
```

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

```
#yhat_naive = colSums(colsp)
```

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.

```
K = 5
n_test = round(n * 1 / K)
n_train = n - n_test
#TO-DO
```

```
Training_set = sample(y, n_test)
Test_set = sample(y, n_train)
```

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.

```
in_sample= array(y[1:length(y)/2], dim = length(y)/2)
out_of_sample = array(y[length(y)/2 : length(y)], dim = length(y)/2)

s_e_in = sd(in_sample)
s_e_out = sd(out_of_sample)
# The s_e_in is greater that s_e_out. Also, we can see that both of them are
relative close if in and out are only half each other.

s_e_in

## [1] 5.426467

s_e_out

## [1] 6.816378
```

Do these two exercises `Nsim = 100` times and find the average difference between s_e and ooss_e.

```
Nsim = 100

H = X %*% solve(t(X) %*% X) %*% t(X)
y_hat = H %*% y

s_e = (y-y_hat)
ooss_e = sum(s_e^2)

mean(s_e)

## [1] -4.626056e-12

mean(ooss_e)

## [1] 11078.78
```

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`.

```
X_with_junk = cbind(X, matrix(rnorm(n * (n_train - p_plus_one)), nrow = n))
dim(X)

## [1] 506  14

dim(X_with_junk)
```

```
## [1] 506 405
```

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`.

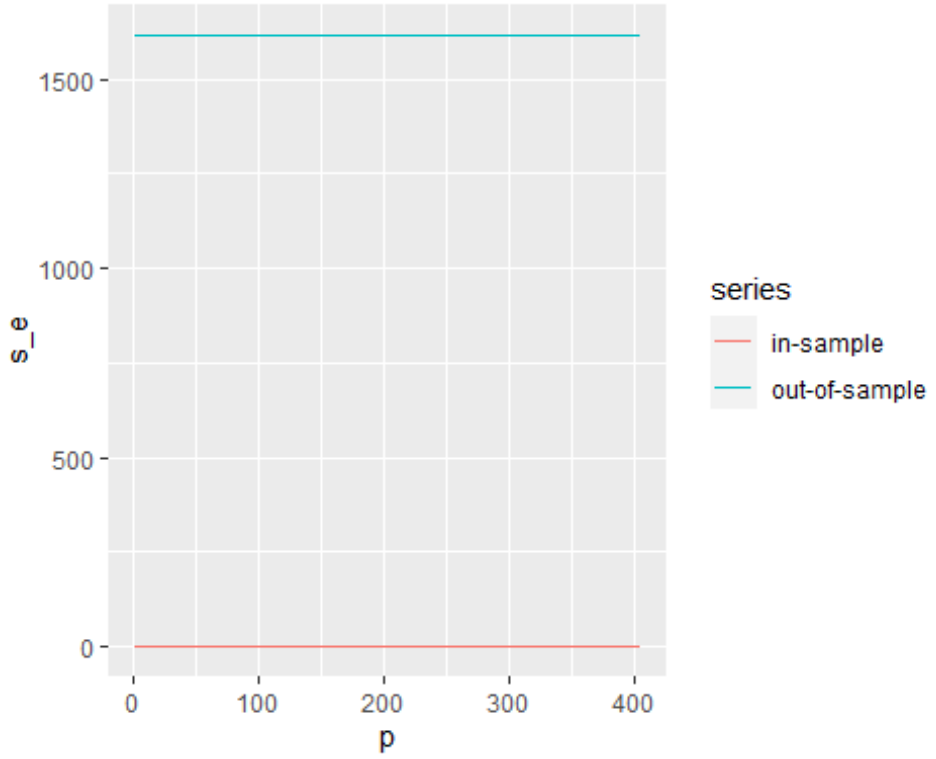
```
#TODO
s_e_by_p = matrix(1,405)
for (j in 2: ncol(X_with_junk)) {

  s_e_by_p[j,] = X_with_junk[j-1] + X_with_junk[j]
}

ooss_e_by_p = colSums(s_e_by_p^2)
```

You can graph them here:

```
#comment
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 Note: make a mistake with s_e_by_p