Exploration 3: Matrices make life easier.

Jake Bowers

September 17, 2016

So far you have received no new communication from the UN. However, you did find an envelope full of cash with a cheery note saying "Thanks from the United Nations!" when you looked in your backpack. After you stopped wondering how someone had put the envelope into your bag, you receive a WhatsApp from another old friend. He is from one of the new political analytics firms that started to grow during the Obama campaign in the USA and he has a prediction problem. He has a small dataset of 8 cities and would like to predict current voting turnout in those cities based on a complex model. He says, "My last analyst provided the following code to fit my model but then stopped. I think that the best model of voting turnout uses median household income, median age, racial diversity (here, percent african american), and the number of candidates running for this city council office. I told the analyst this model and he provided the following code. Can you help me?"

```
news.df<-read.csv("http://jakebowers.org/Data/news.df.csv")
news.df$sF<-factor(news.df$s)
```

"I really don't understand the lsCriterion function. Can you write out the criterion using math and explain it to me in plain language? I'm always especially interested in understanding why these stats types are doing this stuff, and I'm so grateful that you can explain it simply and plainly to me."

```
lsCriterion<-function(b,y,X){
  yhat<-b[1]*X[,1]+b[2]*X[,2]+b[3]*X[,3]+b[4]*X[,4]+b[5]*X[,5]
  ehat<-y-yhat
  thessr<-sum(ehat^2)
  return(thessr)
}

X<-as.matrix(cbind(constant=1,news.df[,c("medhhi1000","medage","blkpct","cands")]))
y<-news.df$rpre
```

"He said to 'try some different vectors' and I think that this meant that I was to guess about the values for the coefficients in the model and that, after trying a bunch, I would choose the vector that I liked best. So, for example, I tried a model with all zeros:"

```
lsCriterion(c(0,0,0,0,0,0),y=y,X=X)
```

```
[1] 6329
```

And then tried to see a bunch of other models.

```
set.seed(12345)
lsCriterion(c(1,0,4,0,0,0),y=y,X=X) ## hmm..ok... can I do better?
```

```
[1] 90693
```

```
lsCriterion(runif(5,-100,100),y=y,X=X) ## bad
```

```
[1] 218105731
```

“After trying a bunch of models, however, I started to get tired. Can you help me do this faster? I asked the analyst (who is really a construction engineer and not a statistician) if I could use a loop but he said it would be better to ‘try to optimize the objective function’ and this is as far as I got.”

```
lsSolution<-optim(fn=lsCriterion,par=c(0,0,0,0,0),X=X,y=y,
          method="BFGS",control=list(trace=1,REPORT=1))
```

```
initial  value 6329.000000
iter   2 value 2427.613241
iter   3 value 2299.647197
iter   4 value 1996.222734
iter   5 value 1988.986737
iter   6 value 1079.199017
iter   7 value 908.053870
iter   8 value 888.270807
iter   9 value 872.716838
iter  10 value 872.526305
iter  11 value 872.522390
iter  11 value 872.522388
iter  11 value 872.522388
final  value 872.522388
converged
```

“Is this the best solution? How well does this model predict the actual outcome (r) (this model uses baseline turnout or rpre). Can you give me some quantitative measure of how well our model predicted the outcome? I think that you can use the code from the lsCriterion function to develop predictions and compare our predictions to the actual turnout observed (the variable r), right?”

“Now, I wanted to add another variable to see how well that new model fit. For example, maybe blkpct has a curvilinear relationship with the outcome. I complained to the analyst that I would have to re-write the function every time that I had a new model. So, the analyst said, ‘Use matrices.’ and he sent this:”

```
bVector<-solve(t(X) %*% X) %*% t(X) %*% y
yhat<- X %*% bVector
ehat<-y-yhat
summary(ehat)
```

```
       V1
 Min.   :-15.063
 1st Qu.: -8.575
 Median :  0.199
 Mean   :  0.000
 3rd Qu.:  5.656
 Max.   : 17.094
```

“Now, I’m very impressed at the speed and conciseness of this! I mean, he got the same vector of coefficients in like 1/1000 the time that it took me to search for a solution — even using a fast optimizer. Also, he got the predictions very quickly too! But I'm confused about how this works. I understand the idea of proposing different values for the different coefficients and then asking how well they do — in a sum of squared error sense. But the three lines that create `bVector` and `yhat` and even `ehat` are a mystery and I worry that they are not actually comparing my predictions of past turnout to observed future turnout (`rpre` versus `r`). Maybe you can help? I asked the analyst to provide a little guidance to help you get practice with these ideas."

So, you need to be able to explain what is happening when we tell R to do least squares with the matrix $\boldsymbol{X}$ and vector $\boldsymbol{Y}$ via $(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$ using the command `solve(t(X)%*%X)%*%(t(X)%*%y)`. Where `t(X)`$\equiv \boldsymbol{X}^T$ (meaning the transpose of $\boldsymbol{X}$) and `solve(X)`$\equiv \boldsymbol{X}^{-1}$ (meaning the inverse of $\boldsymbol{X}$).

Here are some steps you might take to produce this explanation.

1. First, let's create a $\boldsymbol{X}$ matrix using the newspapers data to do a regression of the form `baseline.turnoutx~income+median age` (where baseline.turnout is `rpre` and income is `medhhi1000` and median age of the city is `medage`). Here is how one might do this in R for both $\boldsymbol{X}$ and $\boldsymbol{Y}$ (where, bold represents matrices or vectors):

```
X<-as.matrix(cbind(1,news.df$medhhi1000,news.df$medage))  # "bind" together columns from the data and a
y<-matrix(news.df$rpre,ncol=1) # not strictly necessary, y<-news.df$rpre would also work

# Look at the objects
X
y

# Structure of the objects
str(X)
str(y)

#Look at the dimensions of the objects: number rows by number columns
dim(X)
dim(y)
```

- Explain how we created the $\boldsymbol{X}$ matrix Hint: The column of 1s has to do with the intercept or constant term.
- What do the columns of $\boldsymbol{X}$ represent?
- What do the rows represent?

2. First, addition and subtraction: Try each of the following lines of math in R and explain to yourself (or your colleagues) what happened, and what this means about matrix math. I did the first one as an example.

Explain what is happening with each of the following

```
X+2
```

```
     [,1]  [,2] [,3]
[1,]    3 28.48 32.7
[2,]    3 39.43 35.4
[3,]    3 37.49 36.7
[4,]    3 50.44 38.2
[5,]    3 27.16 23.3
[6,]    3 41.19 33.4
[7,]    3 31.48 33.4
[8,]    3 55.09 39.7
```

"When you add a single number (aka a scalar) to a matrix, the scalar is added to each entry in the matrix."

Notice: If we didn't have matrix math, here is what we'd have to do to add a scalar to a matrix

```
Xplus2 <- matrix(NA, nrow = 8, ncol = 3)  # Initialize an empty matrix for results
# Loop over rows and then over columns
for (row.entry in 1:8) {
    for (col.entry in 1:3) {
        # Add each element to 2 and record in the Xplus2 matrix
        Xplus2[row.entry, col.entry] <- X[row.entry, col.entry] + 2
    }
}

(X + 2) == Xplus2  # Same entries
```

```
     [,1] [,2] [,3]
[1,] TRUE TRUE TRUE
[2,] TRUE TRUE TRUE
[3,] TRUE TRUE TRUE
[4,] TRUE TRUE TRUE
[5,] TRUE TRUE TRUE
[6,] TRUE TRUE TRUE
[7,] TRUE TRUE TRUE
[8,] TRUE TRUE TRUE
```

```
# An easier check on whether two objects are the same (except for names and such)
all.equal((X + 2), Xplus2, check.attributes = FALSE)
```

```
[1] TRUE
```

```
X - 2  # Subtraction and addition of a scalar with a matrix are the same: they operate on each element
```

```
     [,1]  [,2] [,3]
[1,]   -1 24.48 28.7
[2,]   -1 35.43 31.4
[3,]   -1 33.49 32.7
[4,]   -1 46.44 34.2
[5,]   -1 23.16 19.3
[6,]   -1 37.19 29.4
[7,]   -1 27.48 29.4
[8,]   -1 51.09 35.7
```

```
twovec<-matrix(c(2,2,2),nrow=1,ncol=3)  # make a vector of 2s
twomat<-matrix(2,nrow=8,ncol=3)  # make a matrix of 2s
```

You'll see some errors appear below. Your job is to explain why R failed or made an error. I had to surround these lines in `try()` to prevent R from stopping at the error.

```
try(X+twovec)
try( X+t(twovec) )  # Here, you need to explain what t(twovec) does.
```

```
X+twomat
# Addition/Subtraction is elementwise so two matrices/vectors
# of the same dimensions can be added/subtracted easily.
(X+twomat)==(X+2)
# Adding a matrix full of 2s and adding a scalar 2
# is the same thing.
all.equal((X+twomat),(X+2),check.attributes=FALSE)
all.equal((X+twomat),(twomat+X),check.attributes=FALSE)
```

```
X+X
# X can be added to itself since it amounts to an operation with
# two matrices of the same size
```

3. Second, multiplication. Notice that the symbols for scalar multiplication and matrix multiplication are not the same. Try each of the following lines of math in R and explain to yourself (or your colleagues) what happened, and what this means about matrix math.

```
X*2  # Multiplying a scalar by a matrix works elementwise just like addition
all.equal((X*2),(2*X))  # it is also commutative

X^2  # Exponents work the same way: each element is squared

X^.5  # or X^{1/2}, the square roots of each element

sqrt(X)  # this operator is also elementwise
```

Now, let's get a vector of coefficients to make matrix math link even more tightly with what we've already done fitting models to data:

```
b<-solve(t(X) %*% X) %*% t(X) %*% y
dim(b)
```

```
[1] 3 1
```

Now, let's do matrix multiplication the tedious way. What does this function tell us about the rule for doing matrix multiplication?

```
X.times.b <- matrix(NA, nrow = 8, ncol = 1)
for (row.entry in 1:8) {
    temp <- vector(length = 3)
    for (col.entry in 1:3) {
        temp[col.entry] <- X[row.entry, col.entry] * b[col.entry, ]
    }
    X.times.b[row.entry, ] <- sum(temp)
}
X.times.b
```

Now, doing part of it by hand:

```
(X[1,1] * b[1])+(X[1,2] * b[2])+(X[1,3] * b[3])  # Matrix multiplication is sum of row-times-column mul

(X[2,1] * b[1])+(X[2,2] * b[2])+(X[2,3] * b[3])

(X[3,1] * b[1])+(X[3,2] * b[2])+(X[3,3] * b[3])
## etc.... for each row in X
```

And now a little faster (multiplying vectors rather than scalars and summing): You can break matrix multiplication into separate vector multiplication tasks since vector multiplication also goes sum-of-row-times-column.

```
X[1,] %*% b # First row of X by b
X[2,] %*% b # Second row of X by b [b is a single column]
```

And doing it very fast: This is direct matrix multiplication. So nice and clean compared to the previous! Don't we love matrix multiplication?

```
X %*% b
```

How does `fitted(thelm)` relate to `X %*% b`? What is `%*%` in `X %*% b` (often written $\boldsymbol{Xb}$ or $\boldsymbol{X}\hat{\boldsymbol{\beta}}$)?

```
thelm<-lm(rpre~medhhi1000+medage,data=news.df)
fitted(thelm)
```

```
    1     2     3     4     5     6     7     8 
23.94 24.97 24.09 25.98 27.40 26.10 24.23 26.28 
```

4. How would you use matrix addition/subtraction to get the residuals once you had $\boldsymbol{Xb}$ (aka `X %*% b`)?
5. Now, let's meet another important matrix:

```
# Now another important matrix:
try(X %*% X)  # why doesn't this work.
```

```
t(X)  # Transpose of X

XtX<-t(X) %*% X  # Aha! t(x) is 3x8 and X is 8x3, so XtX is 3x3

XtX
```

To make our lives easier, let's mean deviate or center or align all of the variables (i.e. set it up so that all of them have mean=0). Now the `XtX` matrix will be easier to understand:

```
colmeansvec<-colMeans(X)  # get the means of the columsn of X

colmeansmat<-matrix(rep(colmeansvec,8),ncol=3,byrow=TRUE)
# Fill a matrix with those means: repeat each row 8 times and stack them
```

Why would we want a matrix like the following?

```
X-colmeansmat
# Subtract the column means from each element of X
# That is, we are mean-deviating or centering the columns of X

t(apply(X,1,function(x){x-colmeansvec}))
# This is another way to do the mean-deviating, apply() repeats the
# same vector subtraction on each row of X, and t(apply()) transposes
# the result so that it looks like the other results.

# And here is another way to do it:
sweep(X,2,colmeansvec)  # See the help page for sweep
```

```
X.md<-X-colmeansmat
y.md<-y-mean(y)
XtX.md<-t(X.md) %*% X.md
XtX.md
```

Explain what each entry in `XtX.md` is: if you can, relate those numbers to quantities like variances, covariances, sums (of squares or not), sample sizes, do so.

Here another representation of `XtX` that might help you explain and reproduce the entries above where x_{1i} and x_{2i} represent the two covariates in our prediction model.

$$\boldsymbol{X}^T\boldsymbol{X} = \begin{bmatrix} n & \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{2i} \\ \sum_{i=1}^n x_{1i} & \sum_{i=1}^n {x_{1i}}^2 & \sum_{i=1}^n x_{1i}x_{2i} \\ \sum_{i=1}^n x_{2i} & \sum_{i=1}^n x_{1i}x_{2i} & \sum_{i=1}^n {x_{2i}}^2 \end{bmatrix}$$

Try some of the following commands to get some other help in understanding what XtX is:

```
sum(X.md[,1])

sum(X.md[,2]^2)
sum((X.md[,2]-mean(X.md[,2]))^2)  # sum of squared deviations: same as previous because mean(X.md[,2])=
sum((X.md[,2]-mean(X.md[,2]))^2)/(8-1)  # The variance of z
var(X.md[,2])  # Another way to get variance of z

sum(X.md[,2]*X.md[,3])  # why not use %*%? (ans: we want the cross-product for the covariance)
sum(X.md[,2]*X.md[,3])/(8-1)  # the covariance of z and rpre

cov(X.md)  # see the help file on cov(): The variance-covariance matrix

XtX.md/(8-1)  # The variance-covariance matrix of the x's
```

What about $\boldsymbol{X}^T\boldsymbol{Y}$? Explain the entries in `Xty.md`

```
t(X.md)  # Transpose of mean-deviated X

y.md  # mean deviated y

Xty.md<-t(X.md) %*% y.md
Xty.md  # Looks like covariances between the different columns in X and y, with no denominator

cov(cbind(X.md,y.md))  # Verified
```

```
Xty.md/7  # Verifies that Xty.md/7 contains the covariances between X and y.
```

The following is a verbal formula for a covariance: deviations of x from its mean times deviations of y from its mean divided by n-1 (i.e. roughly the average of the product of the deviations)

```
sum((X[,2]-mean(X[,2]))*(y-mean(y)))
sum((X[,2]-mean(X[,2]))*(y-mean(y)))/(8-1)

# Same as above because we've removed the means from X.md and y.md
sum((X.md[,2])*(y.md))/(8-1)

Xty<-t(X) %*% y
Xty
Xty/(8-1)
```

4. And finally division: Try each of the following lines of math in R and explain to yourself (or your colleagues) what happened (ideally relate what happened to ideas about variances, covariances, sums, etc..)

```
X/2  # divides each element in X by 2

1/X  # the inverse of each element in X (notice the Infinities for the 0s in z)

X^(-1)  # Same as above: 1/X==(X^(-1))

1/X==(X^(-1))

# Now matrix inversion using solve()
try(solve(X))  # Doesn't work --- requires a square matrix

solve(XtX)  # This works, XtX is square.

dim(XtX)
dim(Xty)

solve(XtX)%*%Xty  # Least squares!

try(solve(XtX.md))  # Problem with the zeros from the intercept with mean-deviated variables.
XtX.md<-t(X.md[,2:3])%*%X.md[,2:3]  # So, exclude the intercept (only use columns 2 and 3)
try(solve(XtX.md))

Xty.md<-t(X.md[,2:3]) %*% y.md

solve(XtX.md) %*% Xty.md
```

```
  # Notice that this is not the same as
(1/XtX) %*% Xty  # Matrix inversion is different from scalar division
```

```
       [,1]
[1,] 76.147
[2,]  2.005
[3,]  2.333
```

```
# But it is the same as the regressions with mean deviated variables
lm(I(rpre-mean(rpre))~I(medhhi1000-mean(medhhi1000))+I(medage-mean(medage))-1,data=news.df)
```

```

Call:
lm(formula = I(rpre - mean(rpre)) ~ I(medhhi1000 - mean(medhhi1000)) + 
    I(medage - mean(medage)) - 1, data = news.df)

Coefficients:
I(medhhi1000 - mean(medhhi1000))          I(medage - mean(medage))  
                           0.192                            -0.395  
```

```
# orpre
lm(scale(rpre,scale=FALSE)~scale(medhhi1000,scale=FALSE)+scale(medage,scale=FALSE)-1,data=news.df)
```

```

Call:
lm(formula = scale(rpre, scale = FALSE) ~ scale(medhhi1000, scale = FALSE) + 
    scale(medage, scale = FALSE) - 1, data = news.df)

Coefficients:
scale(medhhi1000, scale = FALSE)      scale(medage, scale = FALSE)  
                           0.192                            -0.395  
```

```
# or
lm(y.md~X.md[,2]+X.md[,3]-1)
```

```

Call:
lm(formula = y.md ~ X.md[, 2] + X.md[, 3] - 1)

Coefficients:
X.md[, 2]  X.md[, 3]  
    0.192     -0.395  
```

```
# And the slopes are the same as the regression with an intercept (and variables on their original scal
coef(lm(rpre~medhhi1000+medage,data=news.df))
```

```
(Intercept)  medhhi1000      medage 
    30.9737      0.1922     -0.3950 
```

6. So, the vector of least squares coefficients is the result of dividing what kind of matrix by what kind of matrix? (what kind of information by what kind of information)? Hint: This perspective of "accounting for covariation" is another valid way to think about what least squares is doing [in addition to smoothing conditional means]. They are mathematically equivalent.

Why should covariances divided by variances amount to differences of means, let alone adjusted differences of means?

Here are some definitions of covariance and variance:

$$\text{Cov}(X,Y) = \frac{\sum_i^n (X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

$$\text{Var}(X) = \text{Cov}(X,X) = \frac{\sum_i^n (X_i - \bar{X})(X_i - \bar{X})}{n-1} = \frac{\sum_i^n (X_i - \bar{X})^2}{n-1}$$

So, first,

$$\frac{\text{Cov}(X,Y)}{\text{Var}(X)} = \frac{\sum_i^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_i^n (X_i - \bar{X})^2}$$

because the (n-1) cancels out. (Thus, we had to divide $X^T X$ and $X^T y$ by n-1 in the sections above to get the analogous covariances/variances). So, this is the bivariate case with $y = \beta_0 + \beta_1 x_1$. What about $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$?

This becomes notationally messy fast. Already, however, you can get a sense for the idea of deviations from the mean being a key ingredient in these calculations.

7. Why might $\boldsymbol{X\beta}$ be useful? Let's get back to the question of prediction. So far we have the a model that predicts future turnout with the following squared error:

```
X<-as.matrix(cbind(constant=1,news.df[,c("medhhi1000","medage","blkpct","cands")]))
y<-news.df$rpre
bVector<-solve(t(X) %*% X) %*% t(X) %*% y
yhat<- X %*% bVector
errors<-news.df$r-yhat
summary(errors)
```

```
       V1
 Min.   :-20.06
 1st Qu.: -7.58
 Median : -1.06
 Mean   :  3.12
 3rd Qu.:  8.54
 Max.   : 37.09
```

```
mseOLS<-mean(errors^2) ## mean squared errors, MSE
```

Now, we suspect that we could do better than this from our reading in James et al. (2013). Here is an example using the lasso. What do you think? Did we do a better job than OLS in this small dataset? What is going on in this code?

```
lassoCriterion<-function(lambda,b,y,X){
  ## Assumes that the first column of X is all 1s
  yhat<-X %*% b
  ehat<-y-yhat
  l1.penalty<-sum(abs(b[-1])) ## no penalty on intercept
  thessr<-sum(ehat^2)
  lassocrit<-thessr+lambda*l1.penalty
  return(lassocrit)
}

lassoCriterion(lambda=.5,b=rep(0,ncol(X)),y=y,X=X)
```

```
[1] 6329
```

```
## Best fit for lambda=.5
lassoSol<-optim(par=c(0,0,0,0,0),
             fn=lassoCriterion,
             method="BFGS",
             X=X,
             y=y,
         lambda=.5,
             control=list(trace=1,REPORT=1))
```

```
initial  value 6329.000000
iter   2 value 2427.837751
iter   3 value 2299.928566
iter   4 value 1997.942064
iter   5 value 1990.580450
iter   6 value 1084.900229
iter   7 value 909.338480
iter   8 value 889.537671
iter   9 value 873.869383
iter  10 value 873.686183
iter  11 value 873.682497
iter  11 value 873.682495
iter  11 value 873.682495
final  value 873.682495
converged
```

```
yhatL1<-X %*% lassoSol$par
errorsL1<-news.df$r-yhatL1
mseL1<-mean(errorsL1^2)

## Now see if we can find the best value of lambda
nlambdas<-100
##results<-matrix(NA,nrow=nlambdas,ncol=nrow(X)+1+1)
somelambdas<-seq(.001,100,length=nlambdas)

## A function to get lasso criterion coefs and MSE

getlassoMSE<-function(lambda,X,y){
  lassoSol<-optim(par=rep(0,ncol(X)),
          fn=lassoCriterion,
          method="BFGS",
          X=X,
          y=y,
          lambda=lambda)
          #,control=list(trace=1,REPORT=1))

  yhatL1<-X %*% lassoSol$par
  errorsL1<-news.df$r-yhatL1
  mseL1<-mean(errorsL1^2)
  return(c(par=lassoSol$par,mse=mseL1,lambda=lambda))
}
```

```
results<-sapply(somelambdas,function(l){ getlassoMSE(l,X=X,y=y) })
## apply(results,1,summary)
min(results["mse",])>mseOLS
```

```
[1] TRUE
```

```
results["lambda",results["mse",]==min(results["mse",])]
```

```
lambda
 0.001
```

```
## To do this even faster:
library(glmnet)
lassoFits<-glmnet(x=X[,-1],y=y,alpha=.5) ## using an elastic net fit rather than strict lasso
par(mar=c(6,2,3,1),mgp=c(1.5,.5,0))
plot(lassoFits,xvar="lambda",label=TRUE)
## This next line add the raw lambdas since the default plot shows only the log transformed lambdas
axis(1,line=2,at=log(lassoFits$lambda),labels=round(lassoFits$lambda,3))
abline(h=0,col="gray",lwd=.5)
```

```
getMSEs<-function(B,X,obsy){
  yhats <- apply(B,2,function(b){ X %*% b })
  ehats <- apply(yhats,2,function(y){ obsy - y })
  apply(ehats,2,function(e){ mean(e^2) })
}

getMSEs(B=coef(lassoFits),X=X,obsy=y)
```

Hmmm. . . . should the MSE always just go down? I wonder what James et al. (2013) has to say about this.[1]

Now, more benefits of penalized models. Imagine this model:

```
newmodel<-rpre ~ blkpct*medhhi1000*medage*cands
lm4<-lm(newmodel,news.df)
X<-model.matrix(newmodel,data=news.df)

try(b<-solve(t(X) %*% X) %*% X %*% y)

lsCriterion2<-function(b,y,X){
  yhat<-X %*% b
  ehat<-y-yhat
  thessr<-sum(ehat^2)
  return(thessr)
}

lsSolution1<-optim(fn=lsCriterion2,par=rep(0,ncol(X)),X=X,y=y,method="BFGS",control=list(trace=1,REPORT=
```

```
initial  value 6329.000000
iter   2 value 5656.480212
```

[1] I suspect that James et al. (2013) would recommend cross-validation — but we only have 8 observations here, so that would be difficult.

```
iter   3 value 4047.012198
iter   4 value 3482.751538
iter   5 value 3452.997505
iter   6 value 2939.600824
iter   7 value 2852.644316
iter   8 value 2359.121263
iter   9 value 2354.120891
iter  10 value 13.540365
iter  11 value 0.000007
iter  12 value 0.000000
iter  13 value 0.000000
iter  13 value 0.000000
iter  13 value 0.000000
final  value 0.000000
converged
```

```
lsSolution1
```

```
$par
 [1]  0.015053  0.051740  0.185690  0.190425  0.058299  0.718082  0.580475  0.028731  0.127707  0.213224
[13] -0.213189  0.071521 -0.008270  0.004771

$value
[1] 5.426e-18

$counts
function gradient
     104       13

$convergence
[1] 0

$message
NULL
```

```
lsSolution2<-optim(fn=lsCriterion2,par=rep(100,ncol(X)),X=X,y=y,method="BFGS",control=list(trace=1,REPO
```

```
initial  value 7441391090677444.000000
iter   2 value 1356076409941235.500000
iter   3 value 1350491960224328.000000
iter   4 value 1349666718569429.500000
iter   5 value 1349665271356833.250000
iter   6 value 1349551090723999.000000
iter   7 value 1347760180862289.750000
iter   8 value 1131556848321706.250000
iter   9 value 149897448372970.750000
iter  10 value 147564160960367.593750
iter  11 value 9411889700922.917969
iter  12 value 162233.132360
iter  13 value 0.082982
iter  14 value 0.000000
iter  15 value 0.000000
iter  16 value 0.000000
iter  17 value 0.000000
iter  17 value 0.000000
iter  17 value 0.000000
final  value 0.000000
converged
```

```
lsSolution2
```

```
$par
 [1]  2501.091  2662.019  2693.273  1578.838  -319.924  -551.454  -221.663  -100.712 -1061.509 -1043.17
[13]   313.841   -52.662    27.193    -6.422

$value
[1] 6.99e-15

$counts
function gradient
     145       17

$convergence
[1] 0

$message
NULL
```

```
lsSolution3<-optim(fn=lsCriterion2,par=rep(-100,ncol(X)),X=X,y=y,method="BFGS",control=list(trace=1,REP
```

```
initial  value 7441401704329883.000000
iter   2 value 1356079027123320.500000
iter   3 value 1350494194517561.250000
iter   4 value 1349668866831628.750000
iter   5 value 1349667419301707.000000
iter   6 value 1349553308310410.500000
iter   7 value 1347768090571115.500000
iter   8 value 1132598803304008.250000
iter   9 value 150606021046742.625000
iter  10 value 148263839837378.593750
iter  11 value 10907826566701.357422
iter  12 value 111708.219110
iter  13 value 0.045310
iter  14 value 0.000000
iter  15 value 0.000000
iter  15 value 0.000000
final  value 0.000000
converged
```

```
lsSolution3
```

```
$par
 [1] -1427.455 -3072.428  3574.534  2109.512 -2949.417 -1131.721   605.658  -124.494   637.673 -1047.22
[13]   360.732  -170.080    29.066    -5.742
```

```
$value
[1] 2.688e-15

$counts
function gradient
     109       15

$convergence
[1] 0

$message
NULL
```

```
cbind(lsSolution1$par,lsSolution2$par,lsSolution3$par)
```

```
           [,1]      [,2]      [,3]
 [1,]  0.015053  2501.091 -1427.455
 [2,]  0.051740  2662.019 -3072.428
 [3,]  0.185690  2693.273  3574.534
 [4,]  0.190425  1578.838  2109.512
 [5,]  0.058299  -319.924 -2949.417
 [6,]  0.718082  -551.454 -1131.721
 [7,]  0.580475  -221.663   605.658
 [8,]  0.028731  -100.712  -124.494
 [9,]  0.127707 -1061.509   637.673
[10,]  0.213224 -1043.176 -1047.221
[11,]  0.228549     4.614   -66.649
[12,] -0.059538    16.384    14.362
[13,] -0.213189   313.841   360.732
[14,]  0.071521   -52.662  -170.080
[15,] -0.008270    27.193    29.066
[16,]  0.004771    -6.422    -5.742
```

```
## Notice that we are not standardizing the columns of X here. Should do that for real use.
ridgeCriterion<-function(lambda,b,y,X){
  ## Assumes that the first column of X is all 1s
  yhat<-X %*% b
  ehat<-y-yhat
  l2.penalty<-sum(b[-1]^2) ## no penalty on intercept
  thessr<-sum(ehat^2)
  lassocrit<-thessr+lambda*l2.penalty
  return(lassocrit)
}


ridgeSolution1<-optim(fn=ridgeCriterion,par=rep(0,ncol(X)),lambda=.5,X=X,y=y,method="BFGS",control=list
```

```
initial  value 6329.000000
iter   2 value 5656.480212
iter   3 value 4047.012202
iter   4 value 3482.751544
iter   5 value 3452.997565
iter   6 value 2939.607925
iter   7 value 2852.686174
iter   8 value 2359.249616
iter   9 value 2356.101926
iter  10 value 14.092219
iter  11 value 0.547131
iter  12 value 0.546899
iter  13 value 0.541653
iter  14 value 0.532499
iter  15 value 0.505338
iter  16 value 0.445721
iter  17 value 0.325127
iter  18 value 0.161774
iter  19 value 0.046056
iter  20 value 0.014910
iter  21 value 0.012250
iter  22 value 0.012185
iter  23 value 0.012185
iter  24 value 0.012185
iter  24 value 0.012185
iter  24 value 0.012185
final  value 0.012185
converged
```

```
ridgeSolution1
```

```
$par
 [1] 71.3736705 -0.0011064 -0.0039195  0.0085869 -0.0003061 -0.0261919  0.0058879 -0.0123911 -0.0054513
[12] -0.0215740 -0.1074402  0.0878351  0.0018812  0.0014181

$value
[1] 0.01218

$counts
function gradient
     120       24

$convergence
[1] 0

$message
NULL
```

```
ridgeSolution2<-optim(fn=ridgeCriterion,par=rep(100,ncol(X)),lambda=.5,X=X,y=y,method="BFGS",control=li
```

```
initial  value 7441391090752444.000000
iter   2 value 1356076410010608.500000
iter   3 value 1350491960306302.250000
iter   4 value 1349666718642526.500000
iter   5 value 1349665271458166.250000
iter   6 value 1349551083440734.250000
iter   7 value 1347760322768737.000000
iter   8 value 1131560842811048.000000
iter   9 value 149873538022417.062500
iter  10 value 147544281813811.500000
iter  11 value 9501783822758.935547
iter  12 value 477787.957506
iter  13 value 4.162805
iter  14 value 2.457498
iter  15 value 2.456468
iter  16 value 1.907530
iter  17 value 1.300097
iter  18 value 0.458548
iter  19 value 0.085397
iter  20 value 0.015646
iter  21 value 0.012237
iter  22 value 0.012185
iter  23 value 0.012185
iter  24 value 0.012185
iter  24 value 0.012185
iter  24 value 0.012185
final  value 0.012185
converged
```

```
ridgeSolution2
```

```
$par
 [1] 71.3736713 -0.0011064 -0.0039195  0.0085869 -0.0003061 -0.0261919  0.0058879 -0.0123911 -0.0054513
[12] -0.0215740 -0.1074402  0.0878351  0.0018812  0.0014181

$value
[1] 0.01218

$counts
function gradient
     121       24

$convergence
[1] 0

$message
NULL
```

```
ridgeSolution3<-optim(fn=ridgeCriterion,par=rep(-100,ncol(X)),lambda=.5,X=X,y=y,method="BFGS",control=li
```

```
initial  value 7441401704404883.000000
iter   2 value 1356079027192693.500000
iter   3 value 1350494194599539.250000
iter   4 value 1349668866903427.500000
iter   5 value 1349667419377215.250000
iter   6 value 1349553307118946.750000
iter   7 value 1347768841400444.500000
iter   8 value 1132690100101400.500000
iter   9 value 150654430316838.593750
iter  10 value 148316452125697.093750
iter  11 value 11912015510059.597656
iter  12 value 163774.834192
iter  13 value 1677.541867
iter  14 value 1676.600446
iter  15 value 1674.320896
iter  16 value 1668.179649
iter  17 value 1652.425259
iter  18 value 1611.876365
iter  19 value 1511.681234
iter  20 value 1284.041044
iter  21 value 863.035858
iter  22 value 356.103886
iter  23 value 64.603326
iter  24 value 3.947251
iter  25 value 0.081469
iter  26 value 0.012512
iter  27 value 0.012185
iter  28 value 0.012185
iter  28 value 0.012185
iter  28 value 0.012185
final  value 0.012185
converged
```

```r
ridgeSolution3
```

```
$par
 [1] 71.3736614 -0.0011064 -0.0039194  0.0085869 -0.0003061 -0.0261918  0.0058880 -0.0123911 -0.0054513
[12] -0.0215740 -0.1074402  0.0878351  0.0018812  0.0014181

$value
[1] 0.01218

$counts
function gradient
     126       28

$convergence
[1] 0

$message
NULL
```

```r
cbind(ridgeSolution1$par,ridgeSolution2$par,ridgeSolution3$par)
```

```
            [,1]       [,2]       [,3]
 [1,] 71.3736705 71.3736713 71.3736614
 [2,] -0.0011064 -0.0011064 -0.0011064
 [3,] -0.0039195 -0.0039195 -0.0039194
 [4,]  0.0085869  0.0085869  0.0085869
 [5,] -0.0003061 -0.0003061 -0.0003061
 [6,] -0.0261919 -0.0261919 -0.0261918
 [7,]  0.0058879  0.0058879  0.0058880
 [8,] -0.0123911 -0.0123911 -0.0123911
 [9,] -0.0054513 -0.0054513 -0.0054513
[10,] -0.0173382 -0.0173382 -0.0173381
```

```
[11,]  0.0578158  0.0578158  0.0578158
[12,] -0.0215740 -0.0215740 -0.0215740
[13,] -0.1074402 -0.1074402 -0.1074402
[14,]  0.0878351  0.0878351  0.0878351
[15,]  0.0018812  0.0018812  0.0018812
[16,]  0.0014181  0.0014181  0.0014181
```

```
## A sketch of Cross-validation to choose lambda: here using 3-fold because of the small dataset
cvfn<-function(){
  testids<-sample(1:nrow(X),nrow(X)/3)
  trainingids <- (1:nrow(X))[-testids]
  ## Fit
  ## Predict yhat for testids
  ## MSE for y_test versus yhat_test
}

## Average of the MSE across folds is CV MSE
```

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. An Introduction to Statistical Learning. Springer.