Why We Love OLS

Joel Ferguson

8/29/2022

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

I'll use R Markdown documents to demonstrate principles of statistical programming as well as to present additional information. I've forgotten most of my Stata knowledge at this point and I would recommend trying R if you're a devout Stata user for a few reasons:

- R is free. You'll never have to pay for a license to a new version of R, unlike Stata
- R is an object oriented programming language. It's not so important to know what this literally means, but what it effectively means is that R is much more flexible than Stata in terms of how it can manipulate datasets, values, functions, etc.
- Nowadays, packages for the hottest new econometric techniques tend to come out for R first. So unless you want to spend hours coding up a new estimator in Mata (probably a useful exercise, but also a painful one), learning R now may be beneficial in the long run.

Why We Love OLS Part I: Consistent Estimates of Linear CEFs

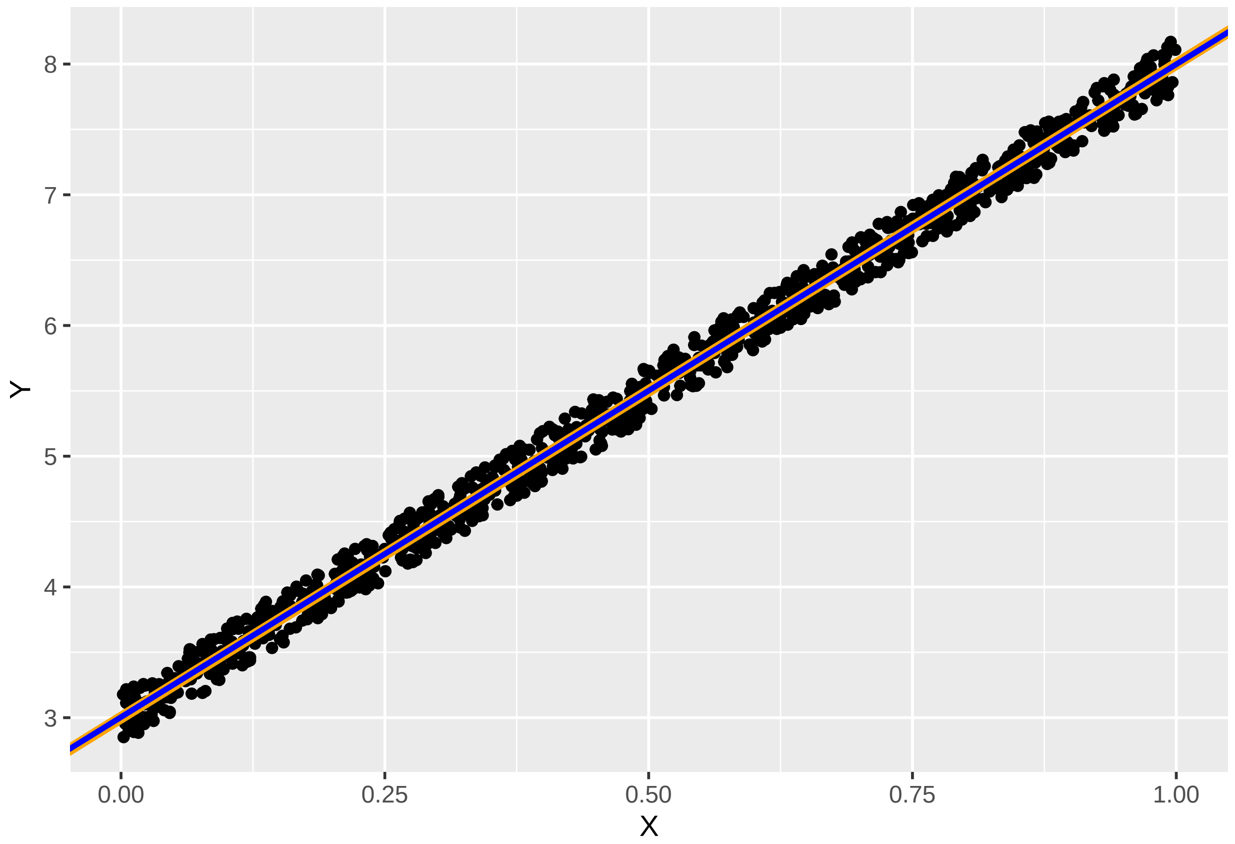
Unlike most sections, this section will mostly be going over the material from class. However, I'm going to try to do so a bit more visually. If you recall from the lecture notes, the first reason we love OLS is because if the true CEF underlying the data we observe is linear, OLS consistently estimates it. That is, if we have

$$\mathbb{E}[Y|X] = \alpha + X\beta$$

Running $OLS(Y, X)$ will give us consistent estimates of α, β. Let's simulate some data to see this.

```r
# Set alpha and beta
a = 3
b = 5

# Choose N
N = 1000

# Choose N Xs in [0,1]
X = runif(N)
# Simulate Ys with linear CEF
Y = a+b*X + runif(N,-0.2,0.2) # Uniform needs to be centered on 0 now
```

```
# Run ols
X_ols = cbind(matrix(1,nrow=N,ncol=1),X)
ab_hat = solve(t(X_ols)%*%X_ols)%*%(t(X_ols)%*%Y)

df = data.frame("Y"=Y,
                "X"=X)

ggplot(data=df)+
  geom_point(aes(x=X,y=Y))+
  geom_abline(intercept=a,slope=b,size=2,color="orange")+ # True CEF is orange
  geom_abline(intercept=ab_hat[1],slope=ab_hat[2],size=1,color="blue") # OLS estimate is BLUE (lol)
```



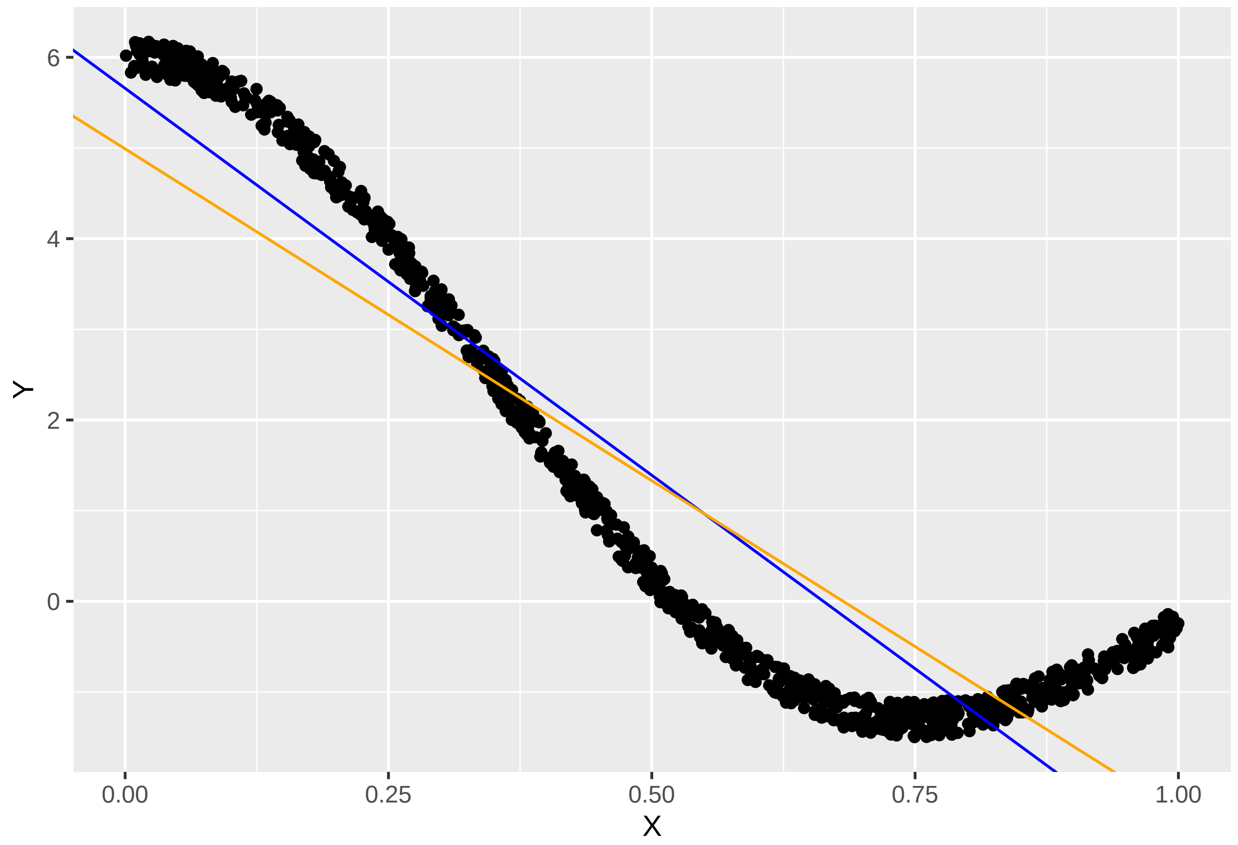
Hey, pretty good! This example used iid data, but that doesn't need to be the case to get the result.

Why We Love OLS Part II: Best Linear Predictor (in MSE sense)

The second reason we love OLS so much is because it is the *minimum mean squared error linear predictor*. This should make intuitive sense: OLS is estimated by minimizing the sum (or equivalently the mean) of squared errors.

One way to think of this property is that if I let you form whatever linear estimator you want and estimate its parameters with a data set, it will predict worse (in a mean squared error sense) on a new data set from the same DGP than OLS (in expectation). Let's see this in action.

```r
# Choose N
N = 1000

# Choose N Xs in [0,1]
X = runif(N)

# Now CEF is going to be nonlinear
Y = sin(X*6)/X + runif(N,-0.2,0.2)

# Run ols
X_ols = cbind(matrix(1,nrow=N,ncol=1),X)
ab_hat = solve(t(X_ols)%*%X_ols)%*%(t(X_ols)%*%Y)

# Now lets try a different linear predictor, e.g. ridge regression
lambda = matrix(0,nrow=2,ncol=2)
diag(lambda)=10
ridge_ab = solve(t(X_ols)%*%X_ols+lambda)%*%(t(X_ols)%*%Y)

df = data.frame("Y"=Y,
                "X"=X)
ggplot(data=df)+
  geom_point(aes(x=X,y=Y))+
  geom_abline(intercept=ab_hat[1],slope=ab_hat[2],color="blue")+
  geom_abline(intercept=ridge_ab[1],slope=ridge_ab[2],color="orange")
```



```r
# Let's calculate the MSE on a new sample
X_oos = runif(N)
Y_oos = sin(X_oos*6)/X_oos + runif(N,-0.2,0.2)

Y_hat_ols = ab_hat[1]+ab_hat[2]*X_oos
Y_hat_ridge = ridge_ab[1] + ridge_ab[2]*X_oos

MSE_ols = mean((Y_oos-Y_hat_ols)^2)
MSE_ridge = mean((Y_oos-Y_hat_ridge)^2)

print(paste("OLS out of sample MSE:",round(MSE_ols,2)))
```

```
## [1] "OLS out of sample MSE: 0.96"
```

```r
print(paste("Ridge out of sample MSE:",round(MSE_ridge,2)))
```

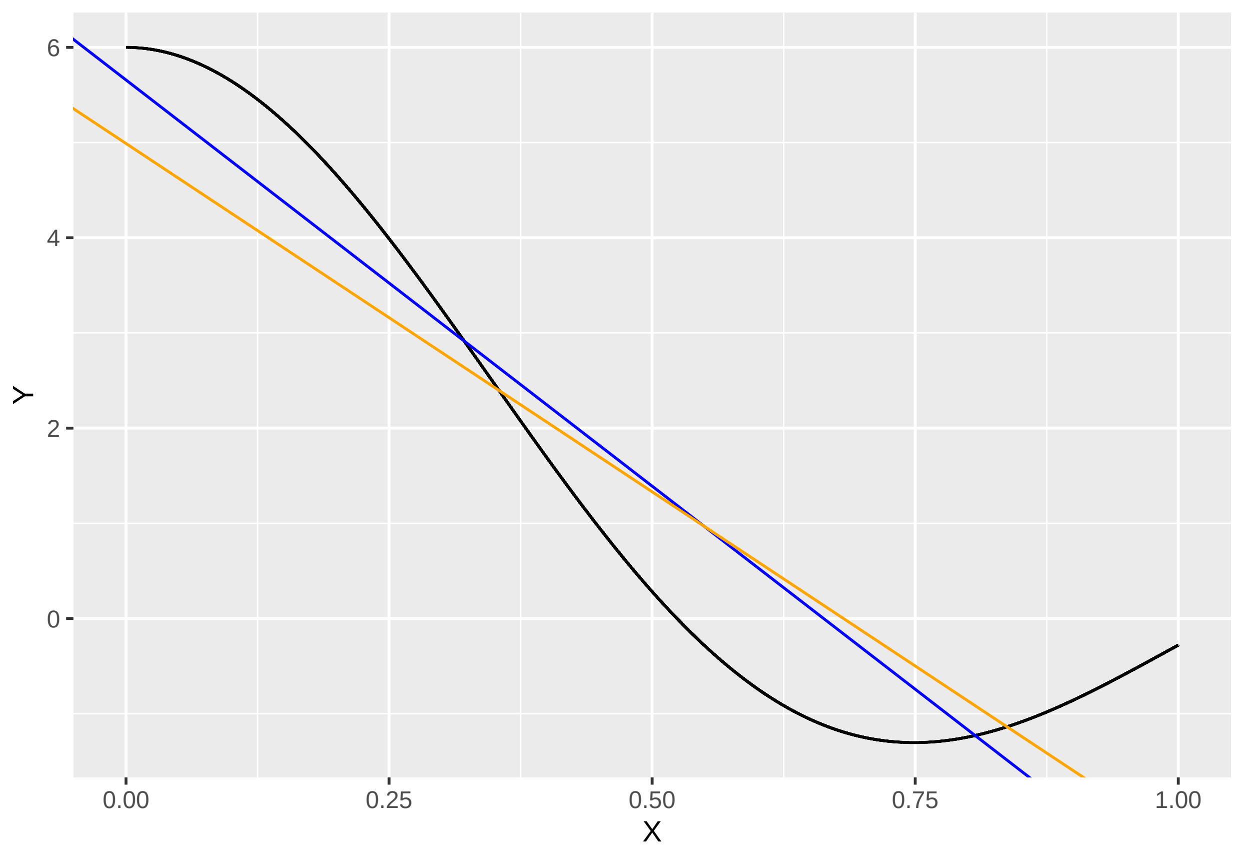
```
## [1] "Ridge out of sample MSE: 1.07"
```

There's nothing special about that ridge estimator I used. OLS will outperform *any* linear predictor (in expectation).

Why We Love OLS Part III: Best Linear CEF Approximator (in MSE sense)

Sometimes it can be a bit tricky to get what's different between this and Part II. The difference is that in Part II, we were trying to find linear transformations of X that were close to Y. In this part, we're trying to find a linear function $a + bX$ that is close to $\mathbb{E}[Y|X]$ in a mean squared error sense. You can see this depicted below.

```r
X_line = runif(100000)
Y_line = sin(X_line*6)/X_line # No error for CEF

Y_hat_ols = ab_hat[1]+ab_hat[2]*X_line
Y_hat_ridge = ridge_ab[1] + ridge_ab[2]*X_line

cef_df = data.frame("Y"=Y_line,
                    "X"=X_line)

ggplot(data=cef_df)+
  geom_line(aes(x=X,y=Y))+
  geom_abline(intercept=ab_hat[1],slope=ab_hat[2],color="blue")+
  geom_abline(intercept=ridge_ab[1],slope=ridge_ab[2],color="orange")
```



```
MSE_ols = mean((Y_line-Y_hat_ols)^2)
MSE_ridge = mean((Y_line-Y_hat_ridge)^2)

print(paste("OLS MSE:",round(MSE_ols,2)))
```

```
## [1] "OLS MSE: 0.95"
```

```
print(paste("Ridge MSE:",round(MSE_ridge,2)))
```

```
## [1] "Ridge MSE: 1.06"
```