STAT GU4206/GR5206 Homework 4 (Practice)
Not Due

Goals: writing functions and using them as larger parts of code, more practice with the bootstrap, fitting models by optimizing loss functions.

In lecture we studied

$$Y = \beta_0 X^{\beta_1} + \epsilon, \tag{1}$$

as a model for the relationship between the per-capita gross metropolitan product of a city and the city's population, with the idea that bigger cities tend to produce more economically per capita. We saw how to estimate the parameter β_1 in the model (assuming β_0 was known) using gradient descent to minimize the mean squared error (MSE)

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \beta_0 X_i^{\beta_1})^2.$$

For the gradient descent algorithm, we approximated the derivative of the MSE, and adjusted our estimate β_1 by an amount proportional (and opposite) to that approximation. We stopped the algorithm when the derivative became small (assuming, then, that we were near a minimum). For this homework we will use a built-in `R` optimization function to estimate both β_0 and β_1 at once, which essentially does a fancier version of what we did in class.

Because the model in (1) is non-linear, there is no simple formula to calculate the parameter estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ in terms of the data, like there was, for example, when studying multiple linear regression models. And also unlike linear models, there is no simple way to calculate the variance of the parameter estimates, so we will use the bootstrap to get approximate variances of the estimates.

We estimate the model in (1), and the uncertainty in our estimates, using the same data from class found in the `gmp.txt` file.

```
gmp     <- read.table("gmp.txt", header = TRUE)
gmp$pop <- round(gmp$gmp/gmp$pcgmp)
```

For this assignment, perform the following tasks:

i. Plot the data as in lecture, with per-capita GMP on the y-axis and population on the x-axis. Using the `curve()` function to add the model (1) using the default values provided in lecture. Add two more curves using $\beta_1 = 0.1$ and $\beta_1 = 0.15$ with the same value of β_0 from class. Make all three curves different colors using the `col` option. **Note: you can also use `ggplot` for the graphs.**

ii. Write a function called `mse()` which calculates the mean squared error of the model on a given dataset. `mse()` should have three arguments: (1) a numeric vector of length two, with the first component corresponding to β_0 and the second to β_1, (2) a numeric vector containing the population values (X), and (3) a numeric vector containing the values of the per-capita GMP (Y). The output of your function should be a single numeric value (the mean squared error). The second and third arguments should have as default values the variables `pop` and `pcgmp`, respectively, from the `gmp` dataset. **Your function may not use a loop (neither `for` nor `which`).** Check that using the default data your function returns the following values:

```
> mse(c(6611, 0.15))
[1] 207057513
> mse(c(5000, 0.10))
[1] 298459914
```

iii. `R` has several built-in functions for optimization which we'll talk about later on in the course. One of the simplest, which we use today, is `nlm()`, or non-linear minimization. `nlm()` takes two required arguments, a function to minimize and a starting value for that function. Run `nlm()` three times with your function `mse()` and three starting pairs for β_0 and β_1 as in

```
nlm(mse, c(beta0 = 6611, beta1 = 1/8))
```

What do the output quantities `minmum` and `estimate` represent? Check `?nlm` for help. What values does the call return for these? Note that you can access these elements using the dollar sign '$' as in the following:

```
nlm(mse, c(beta0 = 6611, beta1 = 1/8))$estimate
```

Hint: In the minimization you may return an error message about replacing NA/Inf values. That's ok as long as the minimization still worked. **When you write your solutions in Markdown, surround your code chunks with the following:**

```
'''{r, warning=FALSE}
'''
```

instead of the usual

```
'''{r}
'''
```

There should be no printed warnings in your `.html` file! In my minimization, it was rare that the β_0 value moved much from its initial guess.

iv. Using `nlm()` and the `mse()` function you wrote, write a function `plm()` which estimates the parameters β_0 and β_1 of the model by minimizing the mean squared error. The function `plm()` should take the following four arguments: (1) an initial guess for β_0, (2) an initial guess for β_1, (3) a vector containing the population values (X), and (4) a vector containing the per-capita GMP values (Y). All arguments except for the initial guesses should have suitable default values. It should return a list with the following three components: (1) the final guess for β_0, (2) the final guess for β_1, and (3) the final value of the MSE. Your function must call those you wrote in earlier questions (as opposed to simply repeating the code) and the appropriate arguments to `plm()` should be passed on to them.

    What parameter estimate do you get when starting from $\beta_0 = 6611$ and $\beta_1 = 0.15$? From $\beta_0 = 5000$ and $\beta_1 = 0.10$? If these are not the same, why do they differ? Which estimate has a lower MSE?

v. Let's practice the bootstrap in a simple example to convince ourselves, again, that it will work.

   (a) Calculate the mean and standard deviation of the per-capita GMP values in your dataset using built-in function `mean()` and `sd()`. Using these values calculate the standard error of the mean.

   (b) Write a function which takes in a vector `indices` of length n, where n is the number of per-capita GMP values in our dataset, and calculates the mean per-capita GMP for the cities indicated by the `indices`. For example if `indices = c(1, 5, 1, 3)`, then your function should calculate the mean `c(24490, 37657, 24490, 24269)`, the per-capita GMP for the first, fifth, first again, and third cities.

   (c) Using the function in (b) create a vector `bootstrap.means`, which has the mean per-capita GMP for one hundred bootstrap samples. Here you'll want to create a loop where each iteration performs a new bootstrap sample. In each iteration you use `sample()` to create an `indices` vector containing the indices of your bootstrap sample and then using `indices` calculate the mean using your function from (b).

   (d) Calculate the standard deviation of the `bootstrap.means` to approximate the standard error from part (a). How well does this estimate match the value form part (a)?

vi. Write a function `plm.bootstrap()`, the calculate a bootstrap estimates of the standard errors of β_0 and β_1. It should take the same arguments as `plm()` along with an argument `B` for the number of bootstrap samples to draw. Let the default be `B = 100`. `plm.bootstrap()` should return standard errors for both parameters. This function should call your `plm()` function repeatedly. What standard errors do you get for the two parameters using initial conditions $\beta_0 = 6611$ and $\beta_1 = 0.15$? Initial conditions $\beta_0 = 5000$ and $\beta_1 = 0.10$

vii. The file `gmp-2013.txt` contains measurements for 2013 (in contrast to measurements from 2006 in `gmp.txt`). Load it and use `plm()` and `plm.bootstrap()` to estimate the parameters for the model for 2013 and their standard errors. Have the parameters of the model changed significantly?

viii. Add one more output to your function `plm.bootstrap()` which also computes the non-parametric bootstrap intervals of β_0 and β_1. Also include one additional argument in the `plm.bootstrap()` function that allows you to change the confidence level with default set at 95%. You can use the regular or percentile based bootstrap intervals. Apply this procedure to both the `gmp.txt` and `gmp-2013.txt` datasets.