Homework 3: The Death and Life of Great American City Scaling Laws

**Background**: In the previous lectures and lab, we began to look at user-written functions. For this assignment we will continue with a look at fitting models by optimizing error functions, and making user-written functions parts of larger pieces of code.

In lecture, we saw how to estimate the parameter in a nonlinear model,

by minimizing the mean squared error

We did this by approximating the derivative of the MSE, and adjusting by an amount proportional to that, stopping when the derivative became small. Our procedure assumed we knew . In this assignment, we will use a built-in R function to estimate both parameters at once; it uses a fancier version of the same idea.

Because the model is nonlinear, there is no simple formula for the parameter estimates in terms of the data. Also unlike linear models, there is no simple formula for the *standard errors* of the parameter estimates. We will therefore use a technique called **the jackknife** to get approximate standard errors.

Here is how the jackknife works:

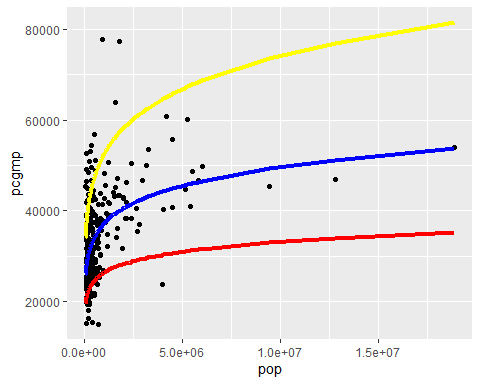
* Get a set of data points and get an estimate for the parameter of interest .
* For each data point , remove from the data set, and get an estimate from the remaining data points. The are sometimes called the “jackknife estimates”.
* Find the mean of the values of
* The jackknife variance of is
* where stands for the sample variance. (*Challenge*: can you explain the factor of ? *Hint*: think about what happens when is large so .)
* The jackknife standard error of is the square root of the jackknife variance.

You will estimate the power-law scaling model, and its uncertainty, using the data alluded to in lecture, available in the file gmp.dat from lecture, which contains data for 2006.

gmp <- read.table("data/gmp.dat")  
gmp$pop <- round(gmp$gmp/gmp$pcgmp)

1. First, plot the data as in lecture, with per capita GMP on the y-axis and population on the x-axis. Add the curve function with the default values provided in lecture. Add two more curves corresponding to and ; use the col option to give each curve a different color (of your choice).

gmp <- gmp %>% mutate(pop = gmp/pcgmp, nlmfit = 6611\*(gmp/pcgmp)^(1/8),nlmfit1=6611\*(gmp/pcgmp)^(0.1),nlmfit15=6611\*(gmp/pcgmp)^(0.15))   
gmp %>% ggplot() + geom\_point(aes(x = pop, y = pcgmp))+  
 geom\_line(aes(x = pop, y = nlmfit), col = 'blue', size = 1.5)+  
 geom\_line(aes(x =pop, y = nlmfit1),col='red',size=1.5)+  
 geom\_line(aes(x =pop, y = nlmfit15),col='yellow',size=1.5)



1. Write a function, called mse(), which calculates the mean squared error of the model on a given data set. mse() should take three arguments: a numeric vector of length two, the first component standing for and the second for ; a numerical vector containing the values of ; and a numerical vector containing the values of . The function should return a single numerical value. The latter two arguments should have as the default values the columns pop and pcgmp (respectively) from the gmp data frame from lecture. Your function may not use for() or any other loop. Check that, with the default data, you get the following

> mse(c(6611,0.15))  
[1] 207057513  
> mse(c(5000,0.10))  
[1] 298459915

mse=function(x){  
 a=gmp$pcgmp  
 b=gmp$pop  
 return(mean((a-x[1]\*b^x[2])^2))  
}  
mse(c(6611,0.15))

## [1] 207057513

mse(c(5000,0.10))

## [1] 298459915

1. R has several built-in functions for optimization, which we will meet as we go through the course. One of the simplest is nlm(), or non-linear minimization. nlm() takes two required arguments: a function, and a starting value for that function. Run nlm() three times with your function mse() and three starting value pairs for and as in

nlm(mse, c(y0=7000,a=0.2))

## $minimum  
## [1] 1168662933  
##   
## $estimate  
## [1] 6999.9983 -153.6302  
##   
## $gradient  
## [1] 0 0  
##   
## $code  
## [1] 1  
##   
## $iterations  
## [1] 1

nlm(mse, c(y0=6611,a=0.2))

## $minimum  
## [1] 1168662933  
##   
## $estimate  
## [1] 6610.9983 -145.0805  
##   
## $gradient  
## [1] 0 0  
##   
## $code  
## [1] 1  
##   
## $iterations  
## [1] 1

nlm(mse, c(y0=6611,a=0.15))

## $minimum  
## [1] 61857061  
##   
## $estimate  
## [1] 6610.9999997 0.1263182  
##   
## $gradient  
## [1] 51.76342 -210.18948  
##   
## $code  
## [1] 2  
##   
## $iterations  
## [1] 7

What do the quantities minimum and estimate represent? What values does it return for these? answer:minimum 代表在所给初值下，均方误差所能达到的最小值，estiamte表示在所给初值下，均方误差最小时参数的估计值

1. Using nlm(), and the mse() function you wrote, write a function, plm(), which estimates the parameters and of the model by minimizing the mean squared error. It should take the following arguments: an initial guess for ; an initial guess for ; a vector containing the values; a vector containing the values. All arguments except the initial guesses should have suitable default values. It should return a list with the following components: the final guess for ; the final guess for ; the final value of the MSE. Your function must call those you wrote in earlier questions (it should not repeat their code), and the appropriate arguments to plm() should be passed on to them.

plm=function(x,Y=gmp$pcgmp,N=gmp$pop){  
 result=nlm(mse,x)  
 b=list(y0=result$estimate[1],a=result$estimate[2],final\_value=result$minimum)  
 return(b)  
}

What parameter estimate do you get when starting from and From and ? If these are not the same, why do they differ? Which estimate has the lower MSE?  
answer:因为迭代的初值不同，所以最后收敛的结果也不同， and MSE更低

plm(c(6611,0.15))

## $y0  
## [1] 6611  
##   
## $a  
## [1] 0.1263182  
##   
## $final\_value  
## [1] 61857061

plm(c(5000,0.1))

## Warning in nlm(mse, x): NA/Inf被换成最大的正值  
  
## Warning in nlm(mse, x): NA/Inf被换成最大的正值  
  
## Warning in nlm(mse, x): NA/Inf被换成最大的正值  
  
## Warning in nlm(mse, x): NA/Inf被换成最大的正值  
  
## Warning in nlm(mse, x): NA/Inf被换成最大的正值  
  
## Warning in nlm(mse, x): NA/Inf被换成最大的正值

## $y0  
## [1] 5000  
##   
## $a  
## [1] 0.1475913  
##   
## $final\_value  
## [1] 62521485

1. *Convince yourself the jackknife can work*.
   1. Calculate the mean per-capita GMP across cities, and the standard error of this mean, using the built-in functions mean() and sd(), and the formula for the standard error of the mean you learned in your intro. stats. class (or looked up on Wikipedia…).

mean(gmp$pcgmp)

## [1] 32922.53

sd(gmp$pcgmp)/sqrt(length(gmp$pcgmp))

## [1] 481.9195

b. Write a function which takes in an integer `i`, and calculate the mean per-capita GMP for every city \_except\_ city number `i`.

mean\_i=function(i,Y=gmp$pcgmp){  
 return(mean(Y[-i]))  
}

c. Using this function, create a vector, `jackknifed.means`, which has the mean per-capita GMP where every city is held out in turn. (You may use a `for` loop or `sapply()`.)

n=length(gmp$pcgmp)  
jackknifed.means=vector(length = n)  
for (i in 1:n){  
 jackknifed.means[i]=mean\_i(i)  
}

d. Using the vector `jackknifed.means`, calculate the jack-knife approximation to the standard error of the mean. How well does it match your answer from part (a)?  
answer:和（a）中结果一样

sqrt((n-1)^2/n\*var(jackknifed.means))

## [1] 481.9195

1. Write a function, plm.jackknife(), to calculate jackknife standard errors for the parameters and . It should take the same arguments as plm(), and return standard errors for both parameters. This function should call your plm() function repeatedly. What standard errors do you get for the two parameters?

plm.jackknife=function(x,Y=gmp$pcgmp){  
 vec\_y0=vector(length = n)  
 vec\_a=vector(length = n)  
 for (i in 1:n){  
 vec\_y0[i]=plm(x,Y[-i])$y0[1]  
 vec\_a[i]=plm(x,Y[-i])$a[1]  
 }  
 result=list(y0\_sd=sqrt((n-1)^2/n\*var(vec\_y0)),a\_sd=sqrt((n-1)^2/n\*var(vec\_a)))  
 return(result)  
}

1. The file gmp-2013.dat contains measurements for for 2013. Load it, and use plm() and plm.jackknife to estimate the parameters of the model for 2013, and their standard errors. Have the parameters of the model changed significantly?

gmp\_2013=read.table("data/gmp-2013.dat")  
gmp\_2013=gmp\_2013%>%mutate(pop=round(gmp/pcgmp))   
plm(c(6611,0.15),Y=gmp\_2013$pcgmp,N=gmp\_2013$pop)

## $y0  
## [1] 6611  
##   
## $a  
## [1] 0.1263182  
##   
## $final\_value  
## [1] 61857061

plm.jackknife(c(6611,0.15),Y=gmp\_2013$pcgmp)

## $y0\_sd  
## [1] 0  
##   
## $a\_sd  
## [1] 0