Problem Set 1

Applied Stats II

Due: February 14, 2022

Instructions

- Please show your work! You may lose points by simply writing in the answer. If the problem requires you to execute commands in `R`, please include the code you used to get your answers. Please also include the `.R` file that contains your code. If you are not sure if work needs to be shown for a particular problem, please ask.
- Your homework should be submitted electronically on GitHub in `.pdf` form.
- This problem set is due before class on Monday February 14, 2022. No late assignments will be accepted.
- Total available points for this homework is 80.

Question 1

The Kolmogorov-Smirnov test uses cumulative distribution statistics test the similarity of the empirical distribution of some observed data and a specified PDF, and serves as a goodness of fit test. The test statistic is created by:

$$D = \max_{i=1:n} \left\{ \frac{i}{n} - F_{(i)}, F_{(i)} - \frac{i-1}{n} \right\}$$

where F is the theoretical cumulative distribution of the distribution being tested and $F_{(i)}$ is the ith ordered value. Intuitively, the statistic takes the largest absolute difference between the two distribution functions across all x values. Large values indicate dissimilarity and the rejection of the hypothesis that the empirical distribution matches the queried theoretical distribution. The p-value is calculated from the Kolmogorov- Smirnoff CDF:

$$p(D \leq x) \frac{\sqrt{2\pi}}{x} \sum_{k=1}^{\infty} e^{-(2k-1)^2 \pi^2 / (8x^2)}$$

which generally requires approximation methods (see Marsaglia, Tsang, and Wang 2003). This so-called non-parametric test (this label comes from the fact that the distribution of

the test statistic does not depend on the distribution of the data being tested) performs poorly in small samples, but works well in a simulation environment. Write an `R` function that implements this test where the reference distribution is normal. Using `R` generate 1,000 Cauchy random variables (`rcauchy(1000, location = 0, scale = 1)`) and perform the test (remember, use the same seed, something like `set.seed(123)`, whenever you're generating your own data).

Write an `R` function that implements this test where the reference distribution is normal. As a hint, you can create the empirical distribution and theoretical CDF using this code:

```
  # create empirical distribution of observed data
  ECDF <- ecdf(data)
  empiricalCDF <- ECDF(data)
  # generate test statistic
  D <- max(abs(empiricalCDF - pnorm(empirical)))
```

My solution to this question was as follows:

```
set.seed(123)
```

First I generated 1,000 random cauchy variables to perform the test on, and I called the vector data2:

```
data2 <- rcauchy(1000, location = 0, scale = 1)
```

Next, I created the empirical distribution of observed data using empirical cumulative distribution function.

```
ECDF <- ecdf(data2)
empiricalCDF <- ECDF(data2)
```

I then generated the test statistic, D, and did my best to put the p value formula we were given into R:

```
D <- max(abs(empiricalCDF - pnorm(data2)))

p_value <- sqrt(2*3.141593)*sum((1)^2)*(3.141593)^2/(8*(x)^2))
```

After that, I wrote a function with these two formulae incorporated into it:

```
KS_function <- function(x){
  ECDF <- ecdf(x)
  empiricalCDF <- ECDF(x)
  #generate test statistic
  D <- max(abs(empiricalCDF - pnorm(x)))
  #get p value
  square_root<- sqrt(2*3.141593)
  power_value <- ((1)^2)*((3.141593)^2)/(8*(x)^2)
  p_value <- square_root*sum(exp(power_value)) #somehow need to get k to
    increase with each iteration
  #telling the function what the output is
  output <- list(D, p_value)
  return(output)
}
```

```
14
15 KS_function(data2)
```

The outputs this formula gave me were D= 0.141021 and p value = Inf, meaning infinity.

I know k is meant to be increasing with each iteration, but I couldn't figure out how to get my function to do that, so I just put in k=1 because that's what it says under the "sum" symbol on the formula on the question. I would be grateful for an explanation of what I was supposed to do here.

I then checked my outputs by comparing them using the outputs given by the built-in Komolgorov-Smirnov function.

```
1 ks.test(data2, "pnorm")
```

The outputs given by the built-in function were D= 0.14202 and p value = 2.2e-16

As you can see, something obviously went wrong with my inputting of the p value formula into R, because the value that the function I wrote returned for that is wrong. The values for D returned by both functions were very similar. I currently can't find my mistake outside of the fact that I didn't know how to get k to increase with each iteration, but I would be very grateful if you could point it out to me in your feedback, and hopefully I will still get some attempt marks for what I've managed to do here.

Question 2

Estimate an OLS regression in `R` that uses the Newton-Raphson algorithm (specifically `BFGS`, which is a quasi-Newton method), and show that you get the equivalent results to using `lm`. Use the code below to create your data.

```
1 set.seed(123)
2 data <- data.frame(x = runif(200, 1, 10))
3 data$y <- 0 + 2.75*data$x + rnorm(200, 0, 1.5)
```

I used the week 3 tutorial to inform my approach to this question, and like that tutorial I tried out two different ways of doing the log likelihood function so that I can learn through practising that there are often multiple ways of doing things in R.
First (using the initial code you gave us above, which I won't write out again to avoid repe-

tition), I created my first version of the log likelihood function using the normal distribution, which was as follows:

```
norm_likelihood <- function(outcome, input, parameter) {
  n      <- nrow(input)
  k      <- ncol(input)
  beta   <- parameter[1:k]
  sigma2 <- parameter[k+1]^2
  e      <- outcome - input%*%beta
  logl   <- -.5*n*log(2*pi)-.5*n*log(sigma2) - ( (t(e) %*% e)/ (2*sigma2) )
  return(-logl)
}
```

Next, I made the second option for creating a log likelihood function with a normal distribution (as I said, I know this wasn't necessary, but I did it anyway for the purpose of my own learning), and it looked like this:

```
norm_likelihood2 <- function(outcome, input, parameter) {
  n <- ncol(input)
  beta <- parameter[1:n]
  sigma <- sqrt(parameter[1+n])
  -sum(dnorm(outcome, input %*% beta, sigma, log=TRUE))
}
```

I then printed out the estimated coefficients (intercept and beta1) associated with the above:

```
results_norm <- optim(fn=norm_likelihood, outcome=data$y, input=cbind(1, data$
    x), par=c(1,1,1), hessian=T, method="BFGS")
results_norm2 <- optim(fn=norm_likelihood, outcome=data$y, input=cbind(1, data
    $x), par=c(1,1,1), hessian=T, method="BFGS")
```

The result I got was an intercept of 0.1398324 and a beta1 value of 2.7265559 from both log likelihood functions.

After that, I illustrated for myself that we get the same results no matter what log likelihood function we use:

```
results_norm$par; results_norm2$par
```

Finally, I completed the exercise by confirming that we get the same thing with lm():

```
coef(lm(data$y~data$x))
```

The y intercept the lm() function returned was 0.1391874, and the beta1 x value for the data that was returned was 2.7266985, which are extremely similar to the values returned by the log likelihood functions above, so I'm guessing there was perhaps a difference in rounding

off the decimals or a slight difference in the method used by the built-in function, but the results of those functions are essentially correct and the same? Please clarify this for me in your feedback, thank you!