Applied Stats II PS1

Luke Duggan

14th February 2022

1 Question One

As a preliminary note, I did not get the correct output in this question: the answer merely shows my workings.

Firstly, we generate the data:

```
set.seed(123)
data <- rcauchy(1000, location=0, scale=1)
data <- sort(data)
```

Next, we create the empirical CDF and calculate the test statistic:

```
empirical_cdf <- ecdf(data)

D <- max(abs(empirical_cdf(data) - pnorm(data)))
D
```

The test statistic is roughly 0.135. For some reason, this is close but not exactly equal to the value from R's built-in Kolmogorov-Smirnov function.

```
ks.test(data, pnorm)$statistic
```

The outputs are:

```
> D
[1] 0.1347281
> ks.test(data, pnorm)$statistic
        D
0.1357281
```

I'm not sure why these outputs aren't equal, since we're just subtracting elements from a list.

Next, we write the formula for the Kolmogorov-Smirnov CDF. Here, x is the argument and n is the summand:

```
KS_CDF <- function(x, n) {

  sum <- 0

  for (k in (1:n)) {
    sum <- sum + (sqrt(2*pi)/(x))*exp(-(pi^2)*((2*k-1)^2)*(8*(x)^2)^-1)
  }

  return(sum)
}
```

Unfortunately, this function doesn't seem to give the correct answer for either of the values of the test statistic we've seen.

```
> KS_CDF(0.1347281,100)
[1] 5.65274e-29
> KS_CDF(0.1357281, 100)
[1] 1.521927e-28
> ks.test(data, pnorm)

        One-sample Kolmogorov-Smirnov test

data:  data
D = 0.13573, p-value = 2.22e-16
alternative hypothesis: two-sided
```

I really don't know why the output is wrong: having checked small values of n with a scientific calculator, I don't think the function is written incorrectly. Since the values are very small floats and the function is implemented as a for loop with repeated addition and multiplication, at least some of the error may be due to rounding.

Finally, on the question of approximation. The above function requires the user to supply the summand n as an argument. One could instead use a convergence condition, e.g., if one wanted an approximation to the 3rd decimal place, one would instruct the function to halt upon reaching a summand n such that KS_CDF(x, n) and KS_CDF(x, n+1) agree in their third or fourth decimal place (depending on whether one was rounding or not).

This is a well-defined stopping rule: because a series of positive terms has increasing partial sums, all values of CDF_(x, n) for larger n will agree in this way too.

2 Question Two

Firstly, we generate the data:

```
set.seed(123)

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

Next, we estimate a linear model by OLS. Note: in the true data-generating process, there is no intercept term. The question didn't specify whether or not to include an intercept in the model: in line with the idea that we don't know the true parameters and are trying to estimate them, I decided to include an intercept in the model.

```
OLS <- lm(data$y ~ data$x, data = data)

summary(OLS)
OLS$coefficients
```

Upon estimating this model, we find that the estimated coefficients and their standard errors are:

```
Coefficients:
            Estimate Std. Error
(Intercept)  0.13919    0.25276
data$x       2.72670    0.04159
```

(Notice how the intercept term is not statistically significantly different from 0.)

Next, we want to estimate the same linear model: this time, however, using maximum likelihood and the BFGS variant of Newton-Raphson to numerically approximate our estimates. First of all, we have to write the function we want to optimize: which, for maximum likelihood with normal errors, is the following log-likelihood function (the reason for the minuses is that R's built-in optimization function automatically minimizes, whereas we want to maximize).

```
linear.lik <- function(theta, y, X) {
  n = nrow(X)
  k = ncol(X)

  beta = theta[1:k]
  sigma2 = (theta[k+1])^2

  e = y - X%*%beta

  logli <- -(0.5)*n*log(2*pi) - (0.5)*n*log(sigma2) - (t(e)%*%e)/(2*sigma2)

  return(-logli)
}
```

Then, we use R's built-in optimization function to numerically approximate the maximum likelihood estimates. We arbitrarily choose the initialization values (1,1,1), which are not too far from the true values (0, 2.75, 1.5).

```
linear.MLE <- optim(fn = linear.lik, par = c(1,1,1), hessian = TRUE, y = data$y,
                    X = cbind(1, data$x), method = "BFGS")
```

The output is:

```
> linear.MLE$par
[1]  0.1398324  2.7265559 -1.4390716
```

The coefficient estimates are almost exactly identical to the OLS estimates. For some reason (probably a sign error somewhere), the estimate of the population standard deviation is negative, which makes no sense: however, the absolute value of the estimate is pretty close to the true value of 1.5.