Problem Set 1 - Answers (Applied Stats II)

Sarah Magdihs

10.02, 2026

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:

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

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).

As a hint, you can create the empirical distribution and theoretical CDF using this code:

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

Answer to Question 1:

To implement the Kolmogorov-Smirnov test (KS test), I first created the empirical distribution that I used for this question. As outlined by the task, I generated 1,000 Cauchy random variables. I used the *set.seed*() function to ensure that the results are reproducible.

```
#Always the same numbers
set.seed(123)

#create data
data <- rcauchy(1000, location = 0, scale = 1)
```

Then, I wrote an R function that implements the KS test. As shown below, I first made sure that the data is sorted. I constructed the empirical CDF using the *ecdf*() function, as was suggested in the hint above. While constructing the thereotical CDF was not strictly necessary, I decided to explicitly include this step because it helped me make sense of the formula as a whole. Lastly, the function computes the KS test statistic by calculating the maximum absolute difference between the empirical CDF and the theoretical CDF across all observed values.

```
####Option 2: Assignment asks for a FUNCTION

ks_test <- function(data) {
  data_sorted <- sort(data) ##still need sorted data
  ECDF_func <- ecdf(data_sorted) ##create empirical distribution of data
  empiricalCDF_func <- ECDF_func(data_sorted)
  theoreticalCDF_func <- pnorm(data_sorted) # theoretical distribution (normal
    )
  D_func <- max(abs(empiricalCDF_func - theoreticalCDF_func)) ##KS test stat
  return(D_func)
}
```

This produced the following test statistic:

```
D_stat <- ks_test(data)
print(D_stat)
```

```
[1] 0.1347281
```

Since the last step of a hypothesis test is to compare and interpret the produced TS, I calculated the p-value. For this, I created an R function that uses the equation which is outlined above. The approximation uses 1,000 terms.

```
ks_pvalue <- function(D_stat, terms = 1000) { ##approx. infinity with 1000
  k <- 1:terms ### take the first X terms
  sum_calc <- sum(exp(-((2*k - 1)^2 * pi^2) / (8 * D_stat^2)))
  p <- (sqrt(2 * pi) / D_stat) * sum_calc
  return(p)
}

```

```r
p_value <- ks_pvalue(D_stat)
print(p_value)
if (p_value < 0.05) {
  print("Reject H0: Evidence suggests that data does not follow a normal
    distribution")
} else {
  print("Fail to reject H0")
}
```

Based on this, R provided the following output:

```
[1] 5.652523e-29
[1] "Reject H0: Evidence suggests that data does not follow a normal
     distribution"
```

Since H_0 is that the sample follows a specified distribution (in this case: a normal distribution), the evidence suggest that H_0 must be rejected ($p < 0.05$). Hence, the test leads to the conclusion that the sample does not follow a normal distribution.

Additionally, I compared the manually computed results from above with the output from R's *ks.test*. While these results somewhat differ (which I assume to be due to different scaling?), the KS test statistic remains similar and the p-value is below 0.05 in both cases. This suggests to me that the manual approximation is doing a good enough job.

```r
# check
ks_check <- ks.test(data, "pnorm", mean = 0, sd = 1)

print(D_stat)
print(ks_check$statistic)

print(p_value)
print(ks_check$p.value)
```

For a better overview, I manually put the results into a table:

Table 1: Overview: Comparison

	Manual Calculation	ks.test
KS test statistic	0.1347281	0.1357281
p-value	5.652523e-29	1.994304e-16

Lastly: Please note that the R-file includes some additional steps (line 68-82) as I first took a step-by-step approach to fully understand what I was doing before wrapping all steps into one function.

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.

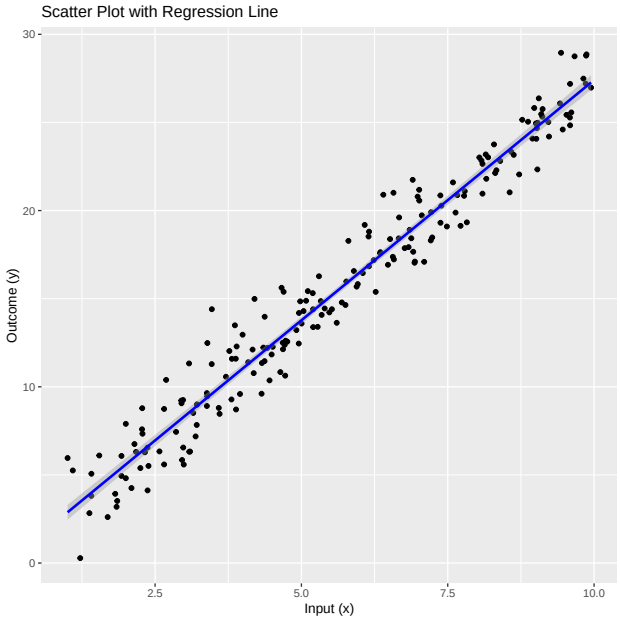
```
data <- data.frame(x = runif(200, 1, 10))
data$y <- 0 + 2.75*data$x + rnorm(200, 0, 1.5)
```

Answer to Question 2:

To show that both approaches produce the same result, I created the data as requested above. Following this, I plotted the data in order to get a better understanding for the data that I am working with. The code that I used is outlined below.

```
#plot data
pdf("plot.pdf")
ggplot(data=data_2, aes(x=x, y= y)) +
  geom_point() +
  geom_smooth(method = "lm", color="blue") +
  labs(title = "Scatter Plot with Regression Line",
       x = "Input (x)",
       y = "Outcome (y)")
dev.off()
```

This produced the following graph:



As a next step, I estimated a linear regression using three approaches: To begin, I used the *lm*() function. Then, I estimated the OLS model by using the Newton-Raphson algorithm

(specifically `BFGS`). To do so, I built on the models that were presented during the lecture and tutorial. Lastly, I used the *glm*() function as an alternative method to verify that the second estimation was done correctly. Lastly, I used *stargazer* to export the main coefficients from all three estimations.

```r
#Linear Model using lm (reference)
lm_model <- lm(y~x, data=data_2)
summary(lm_model)
print(lm_model)


#Estimate an OLS regression by hand; optimisation with BFGS
lin_likelihood <- function(y, X, parameter) {
  n      <- nrow(X)                    # number of observations
  k      <- ncol(X)                    # number of coefficients (including intercept)
  beta   <- parameter[1:k]             # first of k parameters is beta
  sigma2 <- parameter[k+1]^2           # last parameter is sigma (variance squared)
  e      <- y - X%*%beta
  logl   <- -.5*n*log(2*pi)-.5*n*log(sigma2) - ( (t(e) %*% e)/ (2*sigma2) )
  return(-logl)                        # return negative log-likelihood because optim() minimizes
}

results_mle_calc <- optim(fn=lin_likelihood, y=data_2$y, X=cbind(1, data_2$x), par=c(0,1,1), hessian=TRUE, method="BFGS")
print(results_mle_calc$par)

coef(lm_model)

# check with glm function
glm_model <- glm(y~x, data=data_2, family = "gaussian")
print(glm_model)

#final check
coef(lm_model)
print(results_mle_calc$par)
coef(glm_model)

#printing
lm_coef <- coef(lm_model)
glm_coef <- coef(glm_model)
mle_coef <- results_mle_calc$par[1:2]


stargazer(lm_coef, glm_coef, mle_coef,
          title = "Comparison of Intercept and Slope Estimates",
          column.labels = c("LM", "GLM", "MLE (method=BFGS)"),
          single.row = TRUE)
```

While the original stargazer output would have been okay, I decided at the last minute that the table could be better organised and thus used the results from *#printing* to manually create my own table:

Table 2: Comparison of Intercept and Slope Estimates Across Models

	lm()	*glm*()	MLE (BFGS)
Intercept	0.1391874	0.1391874	0.139170
Slope	2.7266985	2.7266985	2.726695

In conclusion, the outputs show that GLMs (following a logic of MLE) work equally well as the *lm*() function since both approaches produced the same results.