Problem Set 1

Lisanne van Vucht

Due: February 11, 2026

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 23:59 on Wednesday February 11, 2026. No late assignments will be accepted.

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:

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

1 Answers 1

1.1 Explanation of terms

The Kolmogorov-Smirnov (K-S) test is a "non-parametric test employed to check whether the probability distributions of a sample and a control distribution, or two samples are equal. (Geeks for Geeks, 2025)" It is suitable as a goodness of fit test for contininious distributions (ibid).

In this PS, we want to compare our empirical (created) data set with a normal distibution. This means:

H0: the data follows a normal distribution.
Ha: the data does not follow a normal distribution.

1.2 Two options to calculate the CDF

I start by removing objects, installing and loading libraries, and setting my working directory.

```
rm(list=ls())

detachAllPackages <- function() {
        basic.packages <- c("package:stats", "package:graphics", "package:grDevices", "package:utils", "package:datasets", "package:methods", "package:base")
        package.list <- search()[ifelse(unlist(gregexpr("package:", search()))==1, TRUE, FALSE)]
        package.list <- setdiff(package.list, basic.packages)
        if (length(package.list)>0)  for (package in package.list) detach(package,  character.only=TRUE)
}
detachAllPackages()


pkgTest <- function(pkg){
        new.pkg <- pkg[!(pkg %in% installed.packages()[,  "Package"])]
        if (length(new.pkg))
        install.packages(new.pkg,  dependencies = TRUE)
        sapply(pkg,  require,  character.only = TRUE)
}

setwd("/Users/lisannevanvucht/Documents/TCD/Hillary Term/Quants II/Tutorials/WK II")
```

Next, I generate the data following the instructions:

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

When looking at the structure of the data, I see that the sample size is n = 1000. Therefore, moving on, with this data I create a function to manually compute a KS-test. I will compute a two sided test because we do not have an expectation regarding a direction.

First, I calculate the test-statistic (D) by calclulating the maximum absolute difference between the pbserved and the expected cumaltive distibution functions(CDFs). In the code, I create the observed from the *Cauchy* data. I then subtract the normal CDF (via pnorm) and take the largest absolute gap. This individual value, D, is the strongest deviation of the data from a normal distribution.

Second, I calculate the P value using the second formula as provided on page 1. I've inserted this manually into R. First, I set *sum_term* to 0. I run a loop from k=1 to 1000 to match the sample size (instead of infinity as in the formula). In the loop I start with the calculation behind the sum, as this comes first. The line `term <- exp(exponent)` computes one single term of the series, and `sum_term <- sum_term + term` adds that term to the total. After the loop finishes, *sum_term* contains the sum of all 1000 observations. I then multiply this sum by the constant in front of the series to obtain the final p-value. This p-value tells us how extreme the observed D would be if the data were normally distributed. A small p-value indicates that such a deviation is unlikely under normality.

To validate whether my answers are correct, I also use the R-function *ks.test* to check the D and P value I obtained. The D value from the hand made function is 0.*1354227* which almost matches the R-function's value of 0.13573. However, the p-value as provided by my manual input of the page-1 formula into R returns *5.652523e-29*, which does not match the R-function's value of *2.2e-16*. I tried various things, among other things: I replaced D, with the aforemnetioned value into the formula. I changed the pi value to 3.141593 because I wondered it might be due to malfunctioning of the pi function. I added and deleted various brackets, but nothing made a difference for the outcome to match the function's outcome.

Regardless, both of the p-value's assume that there is a very small probability we would see a deviation this large if the data was normally distributed. This would lead to a strong rejection of the null hypothesis that the created *Cauchy* data is distributed normally and acceptence of the Ha.

```
ks_byhand <- function(data) {
        ECDF <- ecdf(data)
        empiricalCDF <- ECDF(data)

        D <- max(abs(empiricalCDF - pnorm(data)))

        sum_term <- 0.0
        for (k in 1:1000) {
                exponent <- - (((2*k - 1)^2) * (pi^2)) / ((8 * D^2))
                term <- exp(exponent)
                sum_term <- sum_term + term
        }

        p_value <- (sqrt(2*pi) / D) * sum_term

        return(list(
        empiricalCDF = empiricalCDF,
        D = D,
        p_value = p_value
        ))
}

ks.test(data, "pnorm")
```

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.

```

#check data structure
#data
```

1.3 Answer 2

I start by creating the example dataset. First, following the instructions, I set the seed with *set.seed(123)* so I get the same random numbers. Then I continue with `runif(200, 1, 10)`, meaning I generate 200 random values between 1 and 10. After that I continue following the instructios `y` with `ex_data$y <- 0 + 2.75 * ex_data$x + rnorm(200, 0, 1.5)`. This means the true model has a slope of 2,75 if the intercept is 0.

Next I create another hand made function called `ols_manual`. In this I tell the function what the outcome is (`outcome`) the y values, what the input values are our x (`input`), and what my guessed coefficients are (`parameter`). Inside the loop to creat e it, I first compute `n <- ncol(input)` to know how many columns I have in the input matrix. Then I take the first `n` values from `parameter` as my coefficients using `beta <- parameter[1:n]`. I do this because those are the coefficients that match the columns of my input matrix. After that I calculate predicted values with `y_hat <- input %*% beta`, which is the matrix version of "intercept + slope $\times x$". Finally, I compute the sum of squared residuals with `sum((outcome - y_hat)^2)`. This gives me the total squared errors for my current best gues for the coefficients.

Then I use `optim()` to find the coefficients that make this sum of squared residuals as small as possible. I set `fn = ols_manual` so `optim()` uses my ols_manual function. I set `outcome = ex_data$y` and `input = cbind(1, ex_data$x)`. The `cbind(1, ex_data$x)` part creates the input matrix with a column of ones for the intercept and a column with `x` for the slope. I choose starting values with `par = c(1, 1)`, meaning I start with an intercept guess of 1 and a slope guess of 1. Next I set `method = "BFGS"` so the optimizer function uses the BFGS algorithm. Accordingly, Broyden, Fletcher, Goldfarb and Shanno (BFGS) provide useful quasi-Newton updating methods (Gill, 2001).

After running `optim()`, I print the estimated intercept and slope with `round(results_ols$par, 2)`. In my output I get `0.14 2.73`, meaning my estimated intercept is 0.14 and my estimated slope is 2.73.

Finally, running the built-in OLS regression function with `lm(y ~x, data = ex_data)` and print the coefficients using `round(coef(lm(y~x, data=ex_data)), 2)`. This returns `(Intercept) = 0.14` and `x = 2.73`, which matches my `optim()` result, so I do now get the same coeffi-

cients. This means that when $x = 0$, the predicted value of y is 0.14. And on average a one unit increase in x leads to a 2.73 increase in y.

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


ols_manual <- function(outcome, input, parameter) {

        n <- ncol(input)
        beta <- parameter[1:n]

        y_hat <- input %*% beta

        sum((outcome - y_hat)^2)
}

results_ols <- optim(
fn      = ols_manual,
outcome = ex_data$y,
input   = cbind(1, ex_data$x),  # column of 1s = intercept
par     = c(1, 1),              # starting guess
method  = "BFGS",
)

round(results_ols$par, 2)

round(coef(lm(y~x, data=ex_data)), 2)
```

References

CRAN. (n.d.). `get_loglikelihood` documentation. `https://search.r-project.org/CRAN/refmans/insight/html/get_loglikelihood.html`

GeeksforGeeks. (2025). Kolmogorov-Smirnov Test in R Programming. `https://www.geeksforgeeks.org/r-language/kolmogorov-smirnov-test-in-r-programming/`

Gill, J. (2001). *Generalized Linear Models: A Unified Approach*. SAGE Publications.

Posit Community Forum. (2020). Discussion on `set.seed()`. `https://forum.posit.co/t/set-seed-function/89599/4`

Reddit AskStatistics. (2021). Can someone explain log likelihood in an intuitive way? `https://www.reddit.com/r/AskStatistics/comments/l14whl/can_someone_explain_log_likelihood_in_an/`

R-lang.com. (n.d.). Pi in R. `https://r-lang.com/pi-in-r/`

Schlegel, A. (n.d.). Newton-Raphson Method in R. `https://rpubs.com/aaronsc32/newton-raphson-method`