Problem Set 1

Applied Stats II: Ellen Beattie

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).

This function aims to implement a test for Kolmogorov's goodness-of-fit measure with a normal reference distribution for large samples by creating a test statistic D_n and computing the p value, $Pr(D_n < d)$ that returns a measure of surprise of the test statistic under the null i.e. if the data really did originate from a normal distribution how likely is it to see this maximum difference between the theoretical and observed CDFs?

For large samples, to compute the p-value approximation methods are valuable to reduce computation time as explored in Marsaglia, Tsang, and Wang (2003). They use x an asymptotic variable after scaling by size $(\sqrt{n})$ such that $x = \sqrt{n}D_n$. This scaled version means that the limiting distribution is instead $Pr(\sqrt{n}D_n \leq x) \to L(x)$ where x can range from zero to infinity.

The function therefore uses the the limiting form of the distribution Kolmogrov's D_n explored in the paper:

$$\lim_{n\to\infty} Pr\big(\sqrt{n}\, D_n \leq x\big) = 1 - 2\sum_{i=1}^{\infty}(-1)^{i-1}e^{-2i^2x^2}$$

As the p-value is an upper-tail probability $p = 1 - Pr(\sqrt{n}D_n)$, the p-value can be calculated as $P(D > x) = p(x) = 2\sum_{i=1}^{\infty}(-1)^{i-1}e^{-2i^2x^2}$. This formula is translated into a R loop, that iterates over the indexes i until the exponential terms become negligible, summing over the terms to get the probability of observing a deviation this large (from the theoretical cdf) if the null hypothesis was true. This converges rapidly, as the exponential section $e^{(-2i^2)(x^2)}$ shrinks very quickly until the next term is tiny (with the conditions for convergence is the next iteration is smaller than the designed tolerance 'tol' that can be set depending on the specificity).
In addition to this as Marsaglia et al. note that if the scaled value is very large ($d\sqrt{n}$ exceeds about 1.94) it can be beneficial to an extra 'shortcut' condition that returns a probability with sufficient accuracy but protects against excessively slow matrix computations. It instead returns $K(n,d) \approx 1 - 2e^{-(2.000071 + .331/\sqrt{n} + 1.409/n)}$, returning less than .0000005 maximum error. Therefore the very large x, p-value upper-tail probability is $p = 1 - [1 - 2 * summedterms]$ which simplifies to just $2 * summedterms$. These two approximations are included in the

function (depending on the value of x as to which one is used) to determine a p-value for large n Kolmogrov's test statistic.

```r
ks_test <- function(input, tol = 1e-12) { #tol = diff that breaks loop i.e precision

 ECDF <- ecdf(input)
 empiricalCDF <- ECDF(input) #create cdf from input to compare to normal
 D <- max(abs(empiricalCDF - pnorm(input))) #test statistic
 n <- length(input) #sample size

 x <- sqrt(n) * D #scaled test statistic for large n

 if (x == 0) { #if x 0 probability, no difference between distribution, p-value is 1
   p <- 1
 } else if (x > 1.94) { #large x tail
   c <- 2.000071 + 0.331 / sqrt(n) + 1.409 / n #shortcut approx in MTW
   p <- 2 * exp(-c * x^2)
 } else {
   p <- 0 #initialise series at 0
   i <- 1 #index starts at 1
   repeat {
     term <- (2 * (-1)^(i - 1) * exp( -2 * i^2 * x^2)) #Limiting form formula from MWT
     p <- p + term #loop over i's  and sum term each time
     if (abs(term) < tol) break #breaking condition if the increment is small enough
     i <- i + 1 #if not loop through next iteration
   }
 }

 p <- max(min(p,1), 0)  #make sure p in correct range (bounded w 0-1)

 return(list("Test Statistic D" = D, "P-Value" = p)) #print results
}
```

This function uses if else to apply different approximation methods depending on x (the scaled statistic), using the previously mentioned p value formula from Marsaglia et al. 2003. Applying the function to the rcauchy data returns a test statistic of 0.135 and a p-value of 2.76e-16. There is statistically significant evidence to reject the null hypothesis that the data distribution is normal, and accept that empirical distribution of the data differs from a normal distribution.

```r
set.seed(123)
data <- rcauchy(1000, location = 0, scale = 1)
ks_test(data) #implement function on data
```

```
$`Test Statistic D`
[1] 0.1347281
```

```
$`P-Value`
[1] 2.760067e-16
```

Using the built in R function on the same data we can see slight differences in D and P-value likely due to slightly different methods of approximation. The conclusions drawn from the test remain the same however.

```r
ks.test(data, "pnorm") #built in version, testing data against normal
```

```
	Asymptotic one-sample Kolmogorov-Smirnov test

data:  data
D = 0.13573, p-value < 2.2e-16
alternative hypothesis: two-sided
```

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.

```r
set.seed (123)
data2 <- data.frame(x = runif(200, 1, 10))
data2$y <- 0 + 2.75*data2$x + rnorm(200, 0, 1.5)
```

The Newton-Raphson algorithm method to estimate an OLS regression, uses maximum likelihood estimation to estimates for parameters, by using the likelihood function to determine the value of a parameter that gives the greatest probability of observing the data (rather than minimising least squares). The Newton-Raphson algorithm uses iterative weighted least squares to determine the most likely beta values and constant variance given the observed data.
We assume that the points follow a normal probability distribution (as can be explored looking at density graph of y).

```r
library(ggplot2)
ggplot(data2, aes(x = y)) + geom_density()
ggplot(data2, aes(x = x, y = y)) + geom_point()
```

We write down the log likelihood function based on the normal distribution probability density function, take the derivatives with respect to the parameters (betas and sigma squared), set to zero and solve to find find the parameter values (betas and sigma2)that maximise this probability for the inputted data. The optim function works by iterating through different parameter values and their respective maximum likelihood score until it finds the combination with the maxmimum likelihood score and reports the corresponding parameter values (actually does the min but log likelihood is reverse so equivalent to max) . The log transformation of likelihood is used as it allows the summation of probability of likelihoods rather than the more computationally expensive product of probabilities.

Creating the linear likelihood function and using it to run MLE and find the parameter estimates can be seen below. Note: the starting parameter values are '1,1,1' (but this won't effect the outcome). The Hessian Matrix created also can be used to generate the standard errors for the estimates by finding the matrix inverse (without negative as the reverse log likelihood is used and negatives cancel out) then taking the square root of the covariance matrix diagonal to extract the corresponding standard errors for the estimates.

```r
##  From Scratch - Create linear log likelihood function ##
linear_lik <- function(y, X, theta) { #output, input, theta = vector of
    unknown parms
  n      <- nrow(X) #size of sample
  k      <- ncol(X) # number of parameters to estimate
  beta   <- theta[1:k] # vector of coefficients
  sigma2 <- theta[k+1]^2 # variance
  e      <- y - X%*%beta #errors with mean of Xbeta (as normal distribution)
  logl   <- -.5*n*log(2*pi)-.5*n*log(sigma2) - ( (t(e) %*% e)/ (2*sigma2) )
  return(-logl) #as optim function finds min reverse log likelihood
}

# Run MLE using linear likelihood and data
linear_MLE <- optim(fn=linear_lik,  #using scratch function
                    y =data2$y,  #outcome var
                    X =cbind(1, data2$x),  #input var plus 1's for intercept
                    par=c(1,1,1),  #parameter starting points
                    hessian=TRUE, #find standard errors
                    method="BFGS")

linear_MLE$par # see the parameters for intercept, x and sigma2

#Find Standard Errors
H <- linear_MLE$hessian #extract the hessian matrix
cov_matrix <- solve(H) #compute inverse matrix H^-1 (hessian for neg. log
    likelihood so just take inverse)
se <-sqrt(diag(cov_matrix)) #taking the square root of the covariance matrix
    for ses
se

par <- c(0.13983, 2.72655,-1.43907) #parameters from model
se <- c(0.25141, 0.04136, 0.07191) #se's
names(par) <- c("Intercept", "x", "Sigma2")
comb <- paste0(par, "\n(", se, ")") #combined in format usual regression
comb #see format of 'estimate (se)'

linear_MLE_output <- data.frame(Parameter = names(par), #make table
      Estimate = comb, #put in estimates as combines above
      stringsAsFactors = FALSE)
xtable::xtable(linear_MLE_output)  #create latex compatible table
```

Parameter	Estimate
Intercept	0.13983 (0.25141)
x	2.72655 (0.04136)
Sigma2	-1.43907 (0.07191)

These parameter estimates are the values that best explain the data according to the likelihood. Given that it solves linear regression using the identity link the interpretation of estimates are the same as OLS.
A one unit increase in x is associated with, on average, a 2.726 increase in y, overwhelmingly statistically significant (2.72655/0.04136 = 65). It is also very near the specified 2.75 slope when creating the data (slightly impacted because of the noise). The intercept is not statistically significant from zero (.12983/.25141 = 0.55 < 1.96) which is likely given the inputted data was zero.

The same data is ran for a linear OLS regression using lm function. We find the same estimates values for the parameters using OLS. This is as expected as OLS is a special case of MLE, they are both solving the same optimisation problem and under normal errors maximising the likelihood is equivalent to minimising the sum of squares.

```r
## OLS LM
LM <- lm(data2$y~data2$x) #fit linear regression model using OLS
stargazer(LM,
          type = "latex",
          title = "OLS Linear Regession lm")
```

Table 1: OLS Linear Regession lm

	Dependent variable:
	y
x	2.727***
	(0.042)
Constant	0.139
	(0.253)
Observations	200
R^2	0.956
Adjusted R^2	0.956
Residual Std. Error	1.447 (df = 198)
F Statistic	4,298.687*** (df = 1; 198)
Note:	*p<0.1; **p<0.05; ***p<0.01

This result is the same additionally using the built in Gaussian GLM function but the log likelihood and AIC scores are displayed.

```r
# GLS Gaussian
GLM_gaussian <- glm(data2$y~data2$x, family = "gaussian") #estimate using MLE
    with iteratatley reweighted least squares
stargazer(GLM_gaussian ,
          type = "latex",
          title = "GLM gaussian")
```

Table 2: GLM gaussian

	Dependent variable:
	y
x	2.727***
	(0.042)
Constant	0.139
	(0.253)
Observations	200
Log Likelihood	−357.653
Akaike Inf. Crit.	719.306
Note:	*p<0.1; **p<0.05; ***p<0.01