Lab 5 - Computational Statistics (732A89)

Helena Llorens Lluís (hllor282), Yi Yang (yiyan338)

QUESTION 1 Bootstrap for regression

First, we use `lm()` to fit a cubic regression model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

,where x represnts the *concentration* of the fertilizer(%) and y represents the *yield*(mg).

```
##
## Call:
## lm(formula = Yield ~ poly(Fertilizer, 3, raw = TRUE), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -58.893 -17.142  -3.893  17.716  58.107
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      199.284      5.442  36.616   <2e-16 ***
## poly(Fertilizer, 3, raw = TRUE)1   62.634     72.165   0.868   0.3881
## poly(Fertilizer, 3, raw = TRUE)2 -298.766    165.952  -1.800   0.0757 .
## poly(Fertilizer, 3, raw = TRUE)3  158.664     91.587   1.732   0.0872 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.47 on 77 degrees of freedom
## Multiple R-squared:  0.646,  Adjusted R-squared:  0.6322
## F-statistic: 46.84 on 3 and 77 DF,  p-value: < 2.2e-16
```

To reduce model complexity, we remove the cubic term and fit a *quardic model*

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

We then estimate the coefficients with their 95% confidence intervals. The regression curve plot and confidence intervals for the coefficients are shown below:

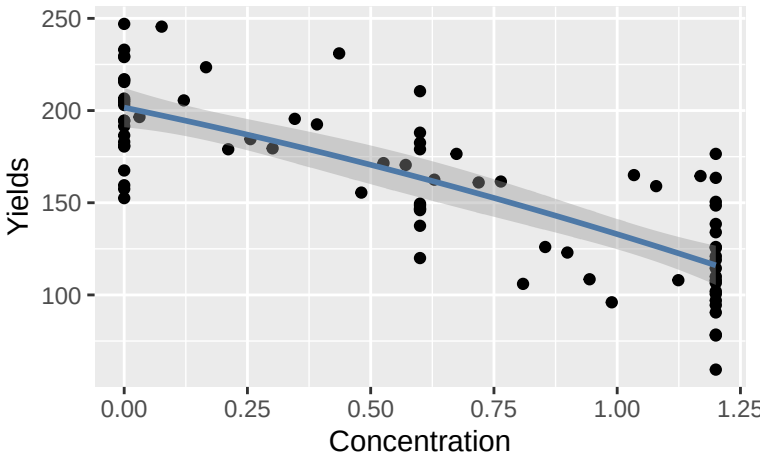

Figure 1: Yields vs Concentration

Table 1: 95% Confidence Interval of parameters

	2.5 %	97.5 %
(Intercept)	190.97432	212.242998
poly(Fertilizer, 2, raw = TRUE)1	-103.16614	-7.791748
poly(Fertilizer, 2, raw = TRUE)2	-51.57084	25.228592

We now use the **bootstrap method** with 10,000 replicates to derive a 95% confidence interval for β_1 using percentile method.The histogram and the 95% confidence interval are shown below:

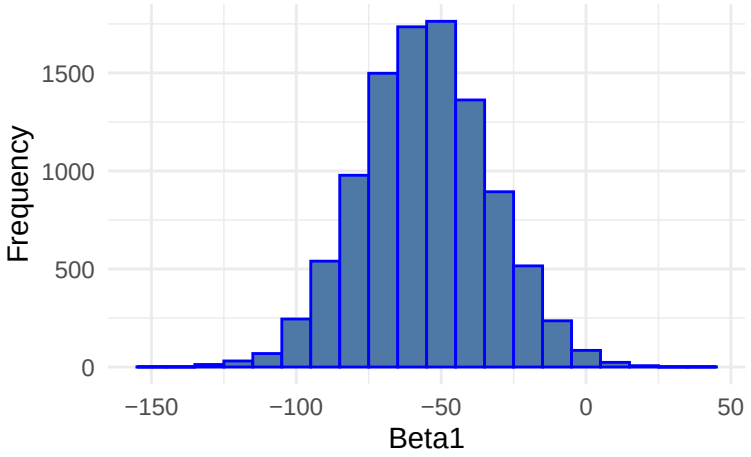

Figure 2: Bootstrap Coefficient Distribution

```
## 95% CI for beta 1 : -98.47781 -11.3802
```

Now we use the package `boot` to compute a 95% confidence interval for β_1 using both percentile method and BCa(bias-corrected and accelerated) method.The table below summarizes the 95% confidence intervals obtained using different methods:

Table 2: 95% Confidence Interval of different methods

	Lower Bound	Upper Bound	Interval Width
lm	-103.16614	-7.791748	95.3744
bootstrap	-98.47781	-11.380203	87.0976
boot_percentile	-99.46221	-12.067414	87.3948
boot_BCa	-98.38221	-10.665462	87.7167

Discussion of Confidence Interval Differences

From the results, we observe the following:

1.**The confidence interval(CI) using `lm()`is wider than the bootstrap CIs.** This is because `lm()` assumes that data follows a normal distribution.If the data sample size is small or the data is not normally distributed this assumption may not hold, leading to a less accurate CI.

2.**The bootstrap method produces narrower confidence interval without distributional assumption.** This allows it to better adapt to the actual data distribution.

3.**The CIs obtained using the manual bootstrap method and the `boot` package with the `percentile` method are similar.** However, the CI from the `boot` package with the `BCa` method is slightly different. This difference may be due to the BCa method correcting for bias, suggesting that the data may have some skewness.

QUESTION 2: Simulation of power curves

To investigate the power of the Sign test for detecting deviations from the null hypothesis $H_0 : \mu = 0$ against the alternative $H_1 : \mu > 0$, we simulate random samples from the Gumbel distribution using the Inverse Transformation Method.

Generating Gumbel Random Variables

The cumulative distribution function (CDF) of the Gumbel distribution with scale parameter set to 1 and location parameter $\mu + c$, where $c = log(log(2))$ is given by

$$F(x) = e^{-e^{-(x-\mu-c)}}, \quad where\ c = log(log(2))$$

To generate random variables from this distribution, we use the Inverse Transformation Method. Specifically, if $U \sim Uniform(0, 1)$, then setting $F(x) = U$ and solving for X gives the inverse CDF:

$$F^{-1}(u) = \mu - c - log(-log(u))$$

This equation allows us to generate Gumbel random variables by:

1. Drawing U from a $Uniform$ distribution on the interval $(0, 1)$.
2. Applying the transformation:
$$X = \mu - c - log(-log(U))$$

This is done by the function `generate_gumbel`, which takes the parameters `mu`, `c` and `n` (number of independent observations to generate). It returns n observations from the Gumbel distribution.

Hypothesis testing

We simulate $n = 13$ independent observations from the Gumbel distribution and perform the Sign test to evaluate the following hypotheses:

$$H_0 : \mu = 0 \; vs \; H_1 : \mu > 0$$

The Sign test is appropriate here because it tests the median of the distribution. Under the null hypothesis, the median is 0. If the observations are consistently greater than 0, we would expect to reject H_0 more frequently, reflecting greater power.

This test is performed by our function `sign_test` , with input parameters `X` and `alpha`, and gives as whether the null hypothesis was rejected.

Objective of the simulation

The goal of this simulation study is to estimate the power of the Sign test, defined as the probability of rejecting H_0 when the alternative hypothesis is true.

We calculate the power for a range of μ values from 0 to 2. Since the median of the Gumbel distribution is μ, as μ increases above 0, we expect the probability of rejecting H_0 to increase. This leads to a power curve that illustrates how sensitive the Sign test is to different values of μ.

We have chosen the grid for μ with a step of 0.1, since a step size of 0.1 provides a fine enough resolution to capture the trend in the power curve without excessive computational cost.

On the other hand, we have performed 1000 repetitions for every μ because it balances precision and computational cost effectively. Increasing the repetitions further would only marginally improve precision but require significantly more processing time. Moreover, with 1000 repetitions the standard error of the estimated power is approximately:

$$SE = \sqrt{\frac{p(1-p)}{n}} = \sqrt{\frac{0.5(1-0.5)}{1000}} = 0.0158$$

which is a reasonable precision for power estimation.

On the following plot, we observe how the the power of the Sign test increases as we increase μ.

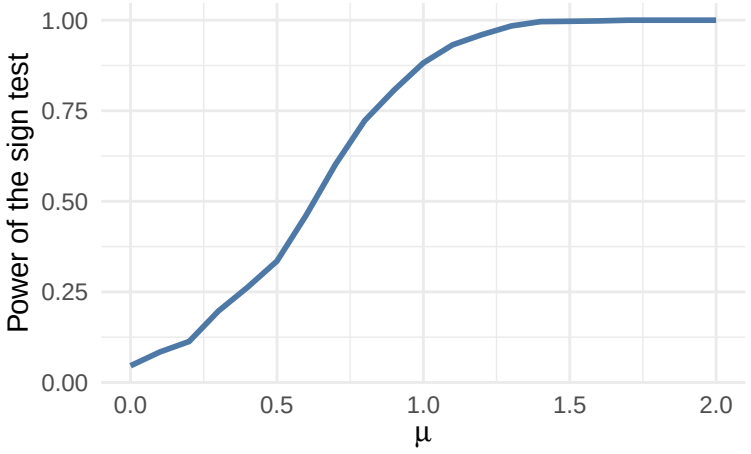

Figure 3: Power Curve of the Sign Test

Conclusion

The power curve of the Sign test, as shown in Figure 1, demonstrates how the test's ability to reject the null hypothesis ($H_0 : \mu = 0$) increases as the true value of μ becomes more positive. For small values of μ the power is low, indicating a low probability of correctly rejecting the null hypothesis when the true median is only slightly greater than 0. But as μ increases, the power rapidly rises, reaching approximately 0.85 when μ is around 1.

For values of μ greater that 1.5, the power approaches 1 indicating that the Sign test is almost certain to reject the null hypothesis when the true median is much larger than 0.

Overall, this power curve illustrates that the Sign test is most effective for detecting moderate to large positive deviations in the median but has limited power for detecting small effects.

APPENDIX

Question 1

```
library(ggplot2)
library(boot)
data <- read.csv("kresseertrag.dat",header=FALSE,sep = "")
colnames(data) <- c("Number","Fertilizer","Yield")
data <- as.data.frame(data)

#a
modelA <- lm(Yield~poly(Fertilizer,3,raw = TRUE),data = data)
summary(modelA)

confint(modelA,level = 0.95)


#b
#remove a term
modelB <- lm(Yield~poly(Fertilizer,2,raw = TRUE),data = data)
summary(modelB)

#coefficients with 95% confidence interval
confint(modelB,level = 0.95)

#plot
#plot(data$Fertilizer,data$Yield,main="Yields vs Concentration",
 #    xlab = "Concentration",ylab = "Yields")
#lines(data$Fertilizer,fitted(modelB),col="blue")

ggplot(data,aes(x=Fertilizer,y=Yield))+
  geom_point()+
  labs(title = "Yields vs Concentration",x="Concentration",y="Yields")+
  geom_line(aes(y=fitted(modelB)),color="#4E79A7")
  #geom_line(aes(y=fitted(modelA)),color="red")

#c bootstrap for beta
bo <- 10000                          #bootstrasp replicates
bs <- c()
set.seed(12345)
```

```r
#save the results
for (i in 1:bo) {
  #sampling using indcies
  indices <- sample(1:nrow(data),size=nrow(data),replace = TRUE)
  bootstrapData <- data[indices,]
  model_bootstrap <- lm(Yield~poly(Fertilizer,2,raw = TRUE),data = bootstrapData)
  bs <- c(bs,coef(model_bootstrap)[2])
}
hist(bs)
bss <- sort(bs)
ci95 <- c(bss[round(bo*0.25)],bss[round(bo*0.975)])
ci95


#d bootstrap using boot package
beta1 <- function(data,i){
  model <- lm(Yield~poly(Fertilizer,2,raw = TRUE),subset=i,data = data)
  coef(model)[2]
}

cb <- boot(data,beta1,R=10000)
perc <- boot.ci(cb,type = "perc")
bca <- boot.ci(cb,type = "bca")
perc

 result <- c(CI[2,],95.3744,ci95,87.0976,perc$percent[4],perc$percent[5],87.3948,bca$bca[4],bca$bca[5],8
 result_mt <- matrix(result,byrow = TRUE,ncol=3)
 rownames(result_mt) <- c("lm","bootstrap","boot_percentile","boot_BCa")
 colnames(result_mt) <- c("Lower Bound","Upper Bound","Interval Width")
 kable(result_mt,caption = "95% Confidence Interval of different methods")
```

Question 2

```r
library(ggplot2)

# Inverse CDF function
f_inv <- function(u, mu, c){
  mu + c - log(-log(u))
}

# Generate n observations from a Gumbel distribution with mu = mu
generate_gumbel <- function(mu, n, c){
  # generate random uniform
  U <- runif(n)
  # inverse CDF function for the Uniform random distribution
  X <- f_inv(U, mu, c)
  return(X)
}

# Perform sign test over sample X (Gumbel distribution generated) and returns if H0: mu = 0 is rejected
sign_test <- function(X, alpha){
  # count positives
```

```r
  positives <- sum(X > 0)
  # perform sign test
  test <- binom.test(positives, length(X), p = 0.5, alternative = "greater")
  # returns TRUE if the test is rejected
  return(test$p.value < alpha)
}

# Parameter initialization
n <- 13
mu <- 0
c <- log(log(2))
nreps <- 1000
alpha <- 0.05
mu_vals <- seq(from = 0, to = 2, by = 0.1)
power <- numeric(length(mu_vals))

# Perform nreps tests for each value of mu
for(i in 1:length(mu_vals)){
  count <- 0
  for(j in 1:nreps){
    # generate sample of size n from a gumbel distribution
    X <- generate_gumbel(mu_vals[i], n, c)
    # perform sign test
    test <- sign_test(X, alpha)
    # update count (adds 1 if the test is rejected)
    count <- count + test
  }
  # calculate the power for each mu
  power[i] <- count / nreps
}

# plot of the power for each mu
plot_data <- data.frame(mu = mu_vals, power = power)
ggplot(plot_data, aes(x = mu, y = power)) +
  geom_line(col = "#4E79A7", linewidth = 1) +
  theme_minimal() +
  labs(x = expression(mu), y = "Power of the sign test")
```