Computational Statistics Lab 5

Authors: Christian Kammerer, Jakob Lindner

Question 1:

a. Read in the dataset and fit a cubic regression model $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$, where x = concentration and y = yield, using the R-function lm.

```
set.seed(42)
data <- read.table("kresseertrag.dat", header=FALSE)

model <- lm(V3~V2+I(V2^2)+I(V2^3), data=data)
summary(model)

##
## Call:
## lm(formula = V3 ~ V2 + I(V2^2) + I(V2^3), 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 ***
## V2            62.634     72.165   0.868   0.3881
## I(V2^2)     -298.766    165.952  -1.800   0.0757 .
## I(V2^3)      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
```

b. You might improve the model in a. by removing or adding (a) model term(s). For the chosen model, estimate the coefficients in the model together with their 95%-confidence intervals using the R-function lm. Create a plot for yield vs. concentration and add the estimated regression curve to the plot.

The summary of the fitted model shows, that the term $\beta_1 x$ has a very low significance. To simplify the model we remove the term which results in the following model: $y = \beta_0 + \beta_2 x^2 + \beta_3 x^3 + \epsilon$

Using the confint function for lm we get the confidence interval for the parameters.

```
model <- lm(V3~I(V2^2)+I(V2^3), data=data)
summary(model)
```

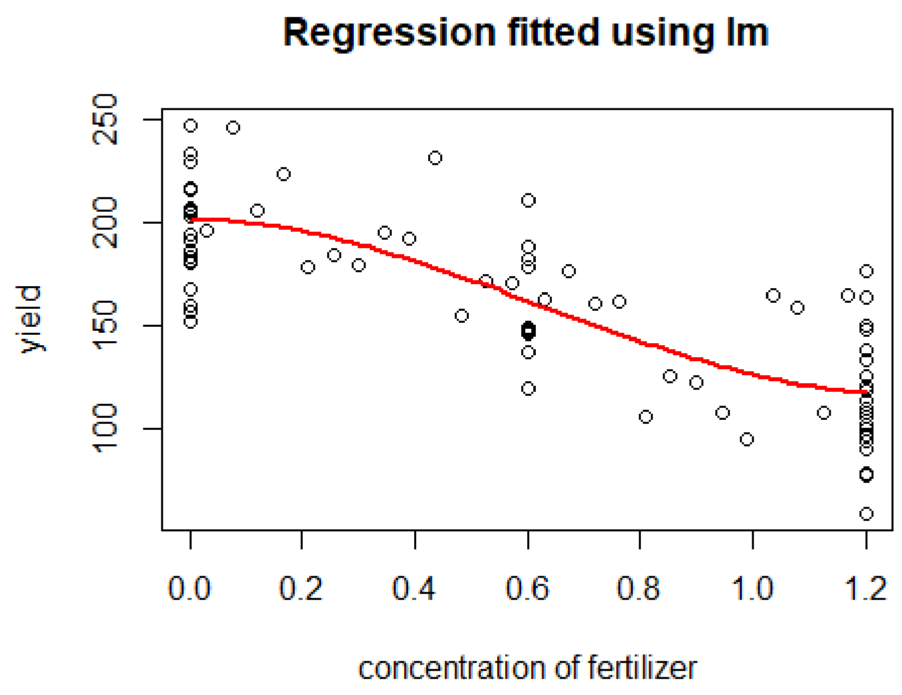
```
##
## Call:
## lm(formula = V3 ~ I(V2^2) + I(V2^3), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -58.187 -16.358  -3.187  16.313  58.813
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  201.315      4.906  41.035  < 2e-16 ***
## I(V2^2)     -158.351     36.912  -4.290 5.08e-05 ***
## I(V2^3)       83.564     29.969   2.788  0.00665 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.43 on 78 degrees of freedom
## Multiple R-squared:  0.6425, Adjusted R-squared:  0.6334
## F-statistic: 70.11 on 2 and 78 DF,  p-value: < 2.2e-16
```

```
model$coefficients
```

```
## (Intercept)     I(V2^2)     I(V2^3)
##   201.31467  -158.35104    83.56384
```

```
confint.lm(model, level=0.95)
```

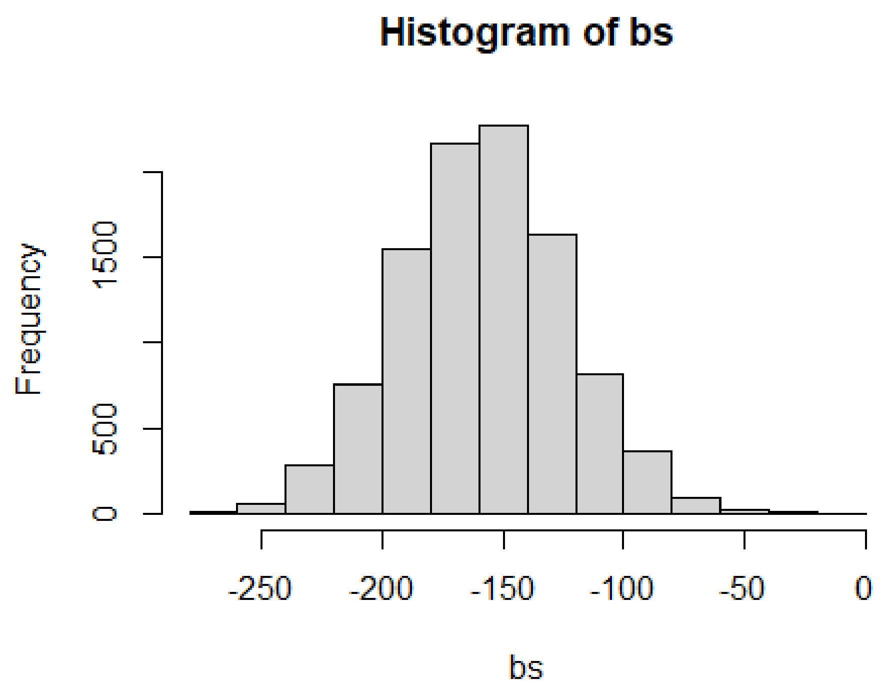
```
##                  2.5 %    97.5 %
## (Intercept)  191.54781 211.08153
## I(V2^2)     -231.83687 -84.86522
## I(V2^3)       23.89916 143.22852
```



c. Derive a 95%-bootstrap confidence interval for one of the model parameters βi based on the percentile method. Do not use a bootstrap package for this calculation; program the bootstrap on your own. Use at least 10000 bootstrap replicates. Plot a histogram with the bootstrap distribution for the parameter.

For this task we adjusted the code from the lecture to train a model on the sampled data in each iteration. The values for β_2 are saved and plotted in the histogram.

```
bo <- 10000
bs <- numeric(length(bo))
for (l in 1:bo){
  x  <- data[sample(nrow(data), replace=TRUE), ]
  m <- lm(V3~I(V2^2)+I(V2^3), data=x)
  bs <- c(bs, m$coefficients[2]) # getting coefficient for beta2
}
```



```
## 95%-confidence interval using percentile method:  -225.1169 -90.62593
```

d. Derive now 95%-confidence intervals for the chosen parameter using the package boot with percentile and BCa method.

The “boot”-package offers the functionality to get confidence intervals using the percentile and the BCa method.

```
fert <- function(data, i){
  m <- lm(V3~I(V2^2)+I(V2^3), subset=i,data=data)
  m$coefficients[2]
}

cb <- boot(data, fert, R=10000)

ci_percentile <- boot.ci(cb, type = "perc")
ci_bca <- boot.ci(cb, type = "bca")
ci_percentile

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = cb, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   (-226.8,  -90.0 )
## Calculations and Intervals on Original Scale
```

```
ci_bca
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = cb, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   (-226.3,  -89.1 )
## Calculations and Intervals on Original Scale
```

e. Compare the confidence intervals from b., c., and d. and comment on it. Do you judge that the intervals are similar or do you see relevant differences? What is your overall conclusion from these confidence interval results about that model parameter?

The confidence intervals retrieved from bootstrapping in c. and d. are very similar around (-226, -90). With (-231.83687 -84.86522) the CI calculated using confint.lm() is a bit lower.

Confint assumes normality according to the documentation, while bootstrapping does not require to make any assumptions on the distribution. But the confidence interval for the parameter retrieved with confint is still close to the result from bootstrapping and not a relevent difference. Therefore, the parameter is almost normally distributed. The shape of the plotted histogram above confirms this statement.

Question 2

a. Generate Gumbel random variables using the Inverse Transformation Method

In the task statement we are instructed to assume a β value of 1, therefore effectively eliminating and thus simplifying our calculations.

In the code block below you can see two functions implemented. `inverse_transformation` takes a value u as input and applies the *inverse of the cumulative density function* of a gumbel distribution with a location parameter μ.

The `sample_gumbel` function samples n random variables from the distribution $U \sim Uniform(0,1)$ (the range of values a *CDF* puts out) and then applies the `inverse_transformation` on them. This is equal to n random variables sampled from a gumbel distribution with the aforementioned location parameter μ.

```
n <- 13

inverse_transformation <- function(u, mu){
  c <- log(log(2))
```

```r
  x <- mu + c -log(-log(u))
  return(x)
}

sample_gumbel <- function(n, mu){
  uniform_rvs <- runif(n, 0, 1)
  gumbel_rvs <- sapply(uniform_rvs, inverse_transformation, mu=mu)
  return(gumbel_rvs)
}

gumbel_rvs <- sample_gumbel(n, 0)
```

b. Simulate the power of the Sign test

Instead of relying on the implementation in the BDSApackage, we decided to implement our own sign test.

This consists of a function called hypothesis_test, which performs a *hypothesis test* given a sample of random gumbel variables, and tests whether the *location parameter* / μ of the distribution these variables are sampled from is different from 0.

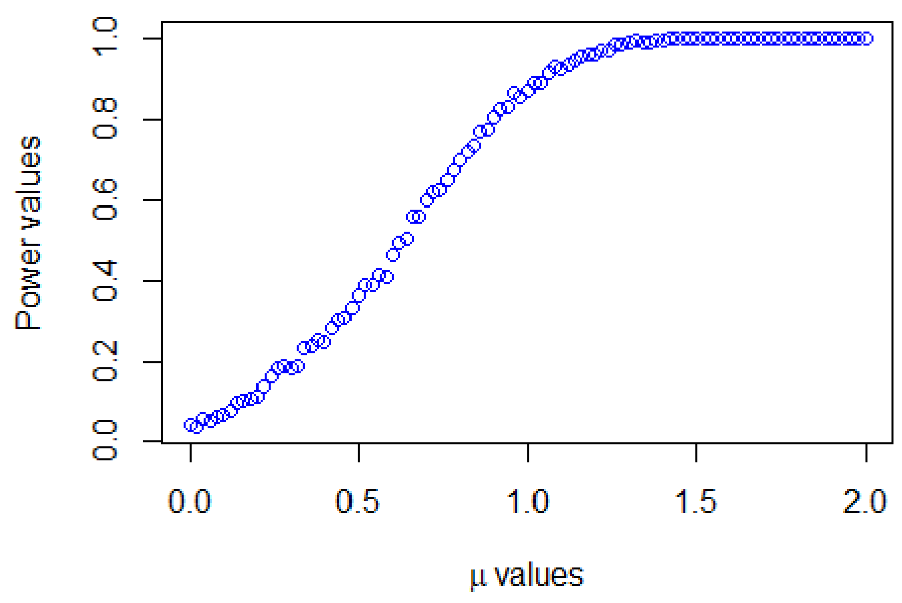
We then extract the *p-values* from these tests and return them. The sign_test function produces 1000 random samples of size n from a gumbel distribution with *location parameter* μ. It then computes the previously mentioned hypothesis test and affiliated p-values. Given a significance level α, it then computes the fraction of hypothesis tests, which are below the significance level. This fraction is the *power* of the *sign test* for a given value for μ. We then repeat this procedure for 100 different μ values in the interval $[0,2]$ and plot the associated power values.

We chose to repeat the sign test a 1000 times, as we rely on the *Law of Large Numbers* to ensure that our estimate for the power of the test. In statistics the minimum power for a test to be viable is typically set at $\hat{p} = 0.8$. At such a power level, the *standard error* (SE) of our estimate would be $SE = \frac{\hat{p}(1-\hat{p})}{n} = \frac{0.8 \cdot 0.2}{1000} \approx 0.0126$.

```r
hypothesis_test <- function(gumbel_rvs){
  x <- sum(ifelse(gumbel_rvs > 0, 1, 0))
  result <- binom.test(x, length(gumbel_rvs), p = 0.5, alternative =
"greater")
  return(result$p.value)
}

sign_test <- function(n, mu, alpha){
  rvs <- lapply(1:1000, function(i) sample_gumbel(n = n, mu = mu))
  p_values <- sapply(rvs, hypothesis_test)
  power <- mean(ifelse(p_values <= alpha, 1, 0))
  return(power)
}
```

Sign power values for gimbel distributions with varying



Appendix

```
# Question 1

set.seed(42)
data <- read.table("kresseertrag.dat", header=FALSE)

model <- lm(V3~V2+I(V2^2)+I(V2^3), data=data)
summary(model)


model <- lm(V3~I(V2^2)+I(V2^3), data=data)
summary(model)

model$coefficients
confint.lm(model, level=0.95)

plot(data$V2, data$V3, main="Regression fitted using lm", ylab="yield",
xlab="concentration of fertilizer")
x <- seq(0,1.2,0.01)
df_x <- data.frame(V2=x)
y <- predict(model, newdata=df_x)
lines(x,y, col="red", lwd=2)



# adjusted code from lecture slides
```

```
bo <- 10000
bs <- numeric(length(bo))
for (l in 1:bo){
  x  <- data[sample(nrow(data), replace=TRUE), ]
  m <- lm(V3~I(V2^2)+I(V2^3), data=x)
  bs <- c(bs, m$coefficients[2]) # getting coefficient for beta2
}


hist(bs)
bss <- sort(bs)
ci95 <- c(bss[round(bo*0.025)], bss[round(bo*0.975)])
cat("95%-confidence interval using percentile method: ", ci95, "\n")


library(boot)

fert <- function(data, i){
  m <- lm(V3~I(V2^2)+I(V2^3), subset=i,data=data)
  m$coefficients[2]
}

cb <- boot(data, fert, R=10000)

ci_percentile <- boot.ci(cb, type = "perc")
ci_bca <- boot.ci(cb, type = "bca")
ci_percentile
ci_bca




# Question 2

library(latex2exp)
library(ggplot2)

#' Function for performing an inverse gumbel transformation
#'
#' @param u Random variable sampled from uniform distribution U~(0,1). Value
we
#' are applying the inverse gumbel cdf on.
#' @param mu Mean of the gumbel distribution also known as location parameter
#' @returns gumble-distributed random variable

inverse_transformation <- function(u, mu){
  c <- log(log(2))
  x <- mu + c -log(-log(u))
  return(x)
}

#' Function for sampling random variables from a gumbel distribution
#' @param n number of sampled random variables
#' @param mu location parameter (median) of gumbel distribution
#' @returns vector of n random variables from gumbel distribution with mu as
median
sample_gumbel <- function(n, mu){
  uniform_rvs <- runif(n, 0, 1)
  gumbel_rvs <- sapply(uniform_rvs, inverse_transformation, mu=mu)
  return(gumbel_rvs)
}

#' Function for performing hypothesis test given a list of random variables
#' (Tests if the median of the distribution is not 0)
#' @param gumbel_rvs vector of random variables
#' @returns p-value of test
hypothesis_test <- function(gumbel_rvs){
  x <- sum(ifelse(gumbel_rvs > 0, 1, 0))
  result <- binom.test(x, length(gumbel_rvs), p = 0.5, alternative =
"greater")
  return(result$p.value)
}

#'Function for performing sign test given.
#'@param n number of samples to feature
#'@param mu location parameter of gimbel distribution for which we test
#'@param alpha significance level we use in the hypothesis test
#'@returns power value of sign_test
sign_test <- function(n, mu, alpha){
  rvs <- lapply(1:1000, function(i) sample_gumbel(n = n, mu = mu))
  p_values <- sapply(rvs, hypothesis_test)
  power <- mean(ifelse(p_values <= alpha, 1, 0))
  return(power)
}

n <- 13
alpha <- 0.05
mus <- seq(0, 2, by = 0.02)
result_list <- lapply(mus, function(mu) sign_test(n = n, mu = mu, alpha =
alpha))
plot(x = mus, y = result_list, type = "b", xlab = TeX("$\\mu$ values"), ylab
= "Power values", col = "blue",
     main = TeX("Sign power values for gimbel distributions with varying
$\\mu$"))
```