Project 2
Computational Numerical Statistics
DM, FCT-UNL
2022-2023
Group 6 : Afonso Ribeiro 59895, Francisco Freitas 60313, Miguel Santos 64474 and Pedro Anjos 66519

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

---

1 Theory

jackknife-after-bootstrap

Jackknife-after-bootstrap is a method used to find the error estimate of a bootstrap estimate. It can be applied to any bootstrap statistic, not only standard error. After bootstrapping, we apply the jackknife to each of the newly generated samples.

Here is the algorithm for the calculating the variance statistic, after drawing the boostrap samples:

- For $i = 1, 2, ..., n$, leave out the data in index i. call the result $\widehat{SE}_{B(i)}$
- define
$$\widehat{var}_{jack}(\widehat{se}_B) = (n-1)\frac{\sum_1^n(\widehat{se}_{B(i)} - \widehat{se}_{B(.)})^2}{n},$$
where $\widehat{se}_{B(.)} = \sum_{i=1}^n \widehat{se}_{B(i)}/n$ (the mean of the bootstrap standard error).

Bias-Corrected-accelerated bootstrap confidence interval

The Bias-Corrected-accelerated bootstrap confidence interval (abbreviated often to BCa) is an improved version of the percentile method to obtain the confidence interval. Altough oftentimes it gives better results it can still be can give erratic results on small sample sizes. It corrects for bias and skewness in the distribution of bootstrap estimates.

Starting off we will need the bias correction parameter z_0 which is directly obtained from the proportion of bootstrap samples less than the original estimate $\hat{\theta}$:

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\#\{\hat{\theta}^* < \hat{\theta}\}}{B}\right)$$

Where ϕ^{-1} is the inverse function of the standard normal cumulative function distribution function and $\hat{\theta}^*$ is the bootstrap samples.

After that we will the need the acceleration $\hat{a}$. There are various ways to calculate the acceleration but we will use the jackknife method for simplicity. Let $\hat{\theta} = s(x)$ be the statistic we want to calculate. let $x_{(-i)}$ be the original sample with the value with index i removed, let $\hat{\theta}_{(-i)} = s(x_{(-i)})$ and $\hat{\theta}_{(.)} = \sum_{i=1}^n(\hat{\theta}_{(-i)})/n$. we can now calculate the acceleration with the following expression;

$$\hat{a} = \frac{\sum_{i=1}^n(\hat{\theta}_{(.)} - \hat{\theta}_{(-i)})^3}{6\{\sum_{i=1}^n(\hat{\theta}_{(.)} - \hat{\theta}_{(-i)})^2\}^{3/2}}$$

finally, having both $\hat{a}$ and $\hat{z}_0$ we can now calculate the adjusted intervals endpoint a_1 and a_2 with the following expressions:

$$a_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z^{(\alpha)}}{1 - \hat{a}(\hat{z}_0 + z^{(\alpha)})}\right)$$

$$a_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z^{(1-\alpha)}}{1 - \hat{a}(\hat{z}_0 + z^{(1-\alpha)})}\right)$$

where Φ the standard normal cumulative distribution function and $z^{(i)}$ the 100αth percentile point of a standard normal distribution.

1.1 Method of Moments

The method of moments is a statistical technique that uses the sample moments of a data set to estimate the parameters of a population distribution. The sample moments are defined as the sample mean, sample variance, and other statistics calculated from the data.
For example, suppose that the problem is to estimate k unknown parameters $\theta_1, \theta_2, \ldots, \theta_k$ characterizing the distribution $f_X(x;\theta)$ of the random variable X. The first k moments of the true distribution (the "population moments") can be expressed as functions of the θs:

$$\begin{aligned}
\mu_1 &\equiv E[X] = g_1(\theta_1, \ldots, \theta_k) \\
\mu_2 &\equiv E[X^2] = g_2(\theta_1, \ldots, \theta_k) \\
&\vdots \\
\mu_k &\equiv E[X^k] = g_k(\theta_1, \ldots, \theta_k)
\end{aligned}$$

Suppose a sample of size n is drawn, resulting in the values $x_1, \ldots, x_n$. For $j = 1, \ldots, k$, let

$$\hat{\mu_j} = \frac{1}{n}\sum_{i=1}^{n} x_i^j$$

be the j-th sample moment, an estimate of μ_j.
The method of moments estimator for $\theta_1, \theta_2, \ldots, \theta_k$ denoted by $\widehat{\theta}_1, \widehat{\theta}_2, \ldots, \widehat{\theta}_k$ **is defined as the solution** (if there is one) to the system of equations

$$\begin{cases}
\hat{\mu_1} &= g_1(\widehat{\theta}_1, \widehat{\theta}_2, \ldots, \widehat{\theta}_k) \\
\hat{\mu_2} &= g_2(\widehat{\theta}_2, \widehat{\theta}_2, \ldots, \widehat{\theta}_k) \\
\vdots & \\
\hat{\mu_k} &= g_k(\widehat{\theta}_1, \widehat{\theta}_2, \ldots, \widehat{\theta}_k)
\end{cases}$$

For instance, let's take the Uniform Distribution on the interval $[a, b]$, $U(a, b)$. If $X \sim U(a, b)$ then we have

$$\mu_1 = E[X] = \frac{1}{b-a}\int_{[a,b]}(xdx) = \frac{1}{2}(a+b)$$

$$\mu_2 = E[X^2] = \frac{1}{b-a}\int_{[a,b]}(x^2dx) = \frac{1}{3}(a^2 + ab + b^2)$$

Solving the system, we obtain

$$\hat{a} = \mu_1 - \sqrt{3(\mu_2 - \mu_1^2)}$$
$$\hat{b} = \mu_1 + \sqrt{3(\mu_2 - \mu_1^2)}$$

Given a set of samples x_i we can use the sample moments $\widehat{\mu}_1$ and $\widehat{\mu}_2$ in order to estimate a and b.

Method of the Secant

The method of the secant (or secant method) is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite-difference approximation of Newton's method.

Although Newton's method is fast, it is required to know the derivative of f in order to use it. In most realistic applications (real-life problems), this can be an impediment for the usage of this method, as the functional form of the derivative is, in many times, not known. A natural way to get around this problem would be to estimate the derivative using

$$f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}, \text{ for } \epsilon \ll 1.$$

The secant method uses the previous iteration to do something similar. It approximates the derivative using the previous approximation. As a result it converges a little slower than Newton's method to the solution:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

The iterates x_n of the secant method converge to a root of f is, if the initial values x_0 and x_1 are sufficiently close to the root. The order of convergence is φ (where $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio). In this case, the secant method's convergence is superlinear (grows faster than any linear one), but not quadratic, as Newton's method.

2 Problem 1

When $T = \overline{X}$, show that:

2.1 $T = T_{jack}$

If $T = \overline{X}$ then $t = \overline{X}$ and

$$t_1^* = \frac{1}{n-1}(x_2 + \ldots + x_n) = \frac{1}{n-1}\sum_{i=1, i\neq 1}^{n}(x_i)$$
$$t_2^* = \frac{1}{n-1}(x_1 + x_3 + \ldots + x_n) = \frac{1}{n-1}\sum_{i=1, i\neq 2}^{n}(x_i)$$
$$\ldots$$
$$t_n^* = \frac{1}{n-1}(x_1 + \ldots + x_{n-1}) = \frac{1}{n-1}\sum_{i=1, i\neq n}^{n}(x_i)$$

Since $T_{jack} = \frac{1}{n}\sum_{i=1}^{n}(t_i^*) = \overline{t^*}$, therefore

$$\begin{aligned} T_{jack} &= \frac{1}{n}\sum_{i=1}^{n}(t_i^*) = \frac{1}{n}\sum_{i=1}^{n}(\frac{1}{n-1}\sum_{j=1,j\neq i}^{n}(x_j)) \\ &= \frac{1}{n}\frac{1}{n-1}\sum_{i=1}^{n}(\sum_{j=1,j\neq i}^{n}(x_j)) \end{aligned}$$

Considering that
$\sum_{i=1}^{n}(c) = nc$ and $\sum_{i=1}^{n}(\sum_{j=1,j\neq i}^{n}(x_j)) = ((x_2+...+x_n)+(x_1+x_3+...+x_n)+\ldots+(x_1+...+x_{n-1}))$
thus
$\sum_{i=1}^{n}(\sum_{j=1,j\neq i}^{n}(x_j)) = (n-1)(x_1+\ldots+x_n)$

$$\begin{aligned} T_{jack} &= \frac{1}{n}\frac{1}{n-1}\sum_{i=1}^{n}(\sum_{j=1,j\neq i}^{n}(x_j)) \\ &= \frac{1}{n}\frac{1}{n-1}\sum_{i=1}^{n}(n-1)(x_1+\ldots+x_n) \end{aligned}$$

$$\begin{aligned} &= \frac{1}{n}(\frac{1}{\cancel{(n-1)}}\cancel{(n-1)}\sum_{i=1}^{n}(x_1+\ldots+x_n) \\ &= \frac{1}{n}(x_1+...+x_n) \\ &= \overline{x} \end{aligned}$$

Since $x_1,\ldots,x_n$ is a realization of $X_1,...,X_n$ we can conclude that $\mathbf{T} = \mathbf{T_{jack}}$

2.2 $V(T_{jack}) = \frac{n-1}{n}\sum_{i=1}^{n}(T_i^* - T_{jack})^2$ simplifies to $\frac{S^2}{n} = V(\overline{X})$

$$\begin{aligned} V(T_{jack}) &= \frac{n-1}{n}\sum_{i=1}^{n}(T_i^* - T_{jack})^2 \\ &= \frac{n-1}{n}((\frac{1}{n-1}(x_2+\ldots+x_n)-\overline{X})^2+\ldots+(\frac{1}{n-1}(x_1+\ldots+x_{n-1})-\overline{X})^2) \\ &= \frac{n-1}{n}((\frac{1}{n-1}((x_2+\ldots+x_n)-(n-1)\overline{X}))^2+\ldots+(\frac{1}{n-1}((x_1+\ldots+x_{n-1})-(n-1)\overline{X}))^2) \\ &= \frac{\cancel{(n-1)}}{n}\frac{1}{(n-1)^{\cancel{2}}}(((x_2+\ldots+x_n)-(n-1)\overline{X})^2+\ldots+((x_1+\ldots+x_{n-1})-(n-1)\overline{X})^2) \\ &= \frac{1}{n(n-1)}(((x_2+\ldots+x_n)-(n\overline{X}-\overline{X}))^2+\ldots+((x_1+\ldots+x_{n-1})-(n\overline{X}-\overline{X}))^2) \\ &= \frac{1}{n(n-1)}(((x_2+\ldots+x_n)-n\overline{X}+\overline{X})^2+\ldots+((x_1+\ldots+x_{n-1})-n\overline{X}+\overline{X})^2) \end{aligned}$$

$$= \frac{1}{n(n-1)}(((x_2 + \ldots + x_n) - \cancel{n}(\frac{1}{\cancel{n}}\sum_{i=1}^{n}(x_i)) + \overline{X})^2 + \ldots + ((x_1 + \ldots + x_{n-1}) - \cancel{n}(\frac{1}{\cancel{n}}\sum_{i=1}^{n}(x_i)) + \overline{X})^2)$$

$$= \frac{1}{n(n-1)}(((x_2 + \ldots + x_n) - (x_1 + \ldots + x_n) + \overline{X})^2 + \ldots + ((x_1 + \ldots + x_{n-1}) - (x_1 + \ldots + x_n) + \overline{X})^2)$$

$$= \frac{1}{n(n-1)}((-x_1 + \overline{X})^2 + \ldots + (-x_n + \overline{X})^2) = \frac{1}{n(n-1)}((-(x_1 - \overline{X}))^2 + \ldots + (-(x_n - \overline{X}))^2)$$

$$= \frac{1}{n(n-1)}((-1)^2(x_1 - \overline{X})^2 + \ldots + (-1)^2(x_n - \overline{X})^2) = \frac{1}{n(n-1)}((x_1 - \overline{X})^2 + \ldots + (x_n - \overline{X})^2)$$

$$= \frac{1}{n(n-1)}\sum_{i=1}^{n}(x_i - \overline{X})^2$$

Considering that $S^2 = (\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{X})^2}{n-1}})^2 = \frac{\sum_{i=1}^{n}(x_i-\overline{X})^2}{n-1}$**, thus**

$$V(T_{jack}) = \frac{1}{n(n-1)}\sum_{i=1}^{n}(x_i - \overline{X})^2$$
$$= \frac{S^2}{n} = V(\overline{X})$$

2.3 Consider the observed sample referring to the survival times of some electrical component pertaining to a car assembly factory.

1552	627	884	2183	1354	1354	1014	2420	71	3725
2195	2586	1577	1766	1325	1299	159	1825	965	695

```
survivalTimes = c(1552, 627, 884, 2183, 1354, 1354, 1014, 2420,  71, 3725,
                  2195, 2586, 1577, 1766, 1325, 1299, 159, 1825, 965, 695)
```

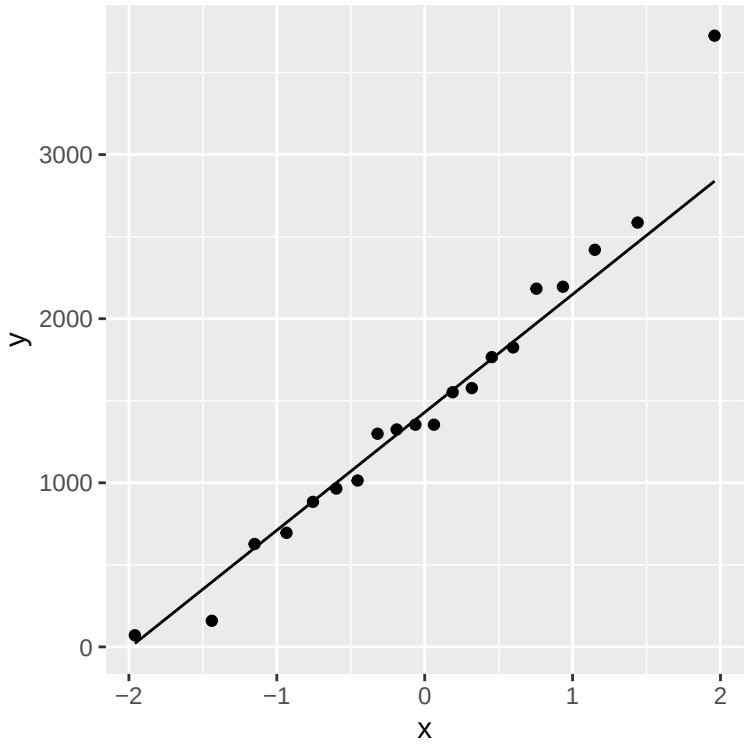
2.3.1 (a) Let μ refer to the mean survival time of that component. Use the non-parametric bootstrap ($B = 10000$ samples) to test the hypotheses

$$H0 : \mu \leq 1020 \quad \textbf{vs} \quad H1 : \mu > 1020,$$

at the usual nominal significance levels. Would it be possible to perform this test using an exact test? If so, do it and compare the results.

```
B = 100000
mu0 = 1020
sd0 = sd(survivalTimes)
n = length(survivalTimes)
```

```
library(ggplot2)
df <- data.frame(survivalTimes)
p1 <- ggplot(df, aes(sample=survivalTimes)) +
    stat_qq( )+
    stat_qq_line()
    xlim(-2,2)

## <ScaleContinuousPosition>
##  Range:
##  Limits:   -2 --    2

plot(p1)
```




```
shapiro.test(survivalTimes)$p.value

## [1] 0.6202625
```

Analyzing the graphic and knowing that shapiro test p-value is greater than 0.05 we can affirm that the survival time estimates follow a normal Distribution. Does mean we can safely perform an exact test:

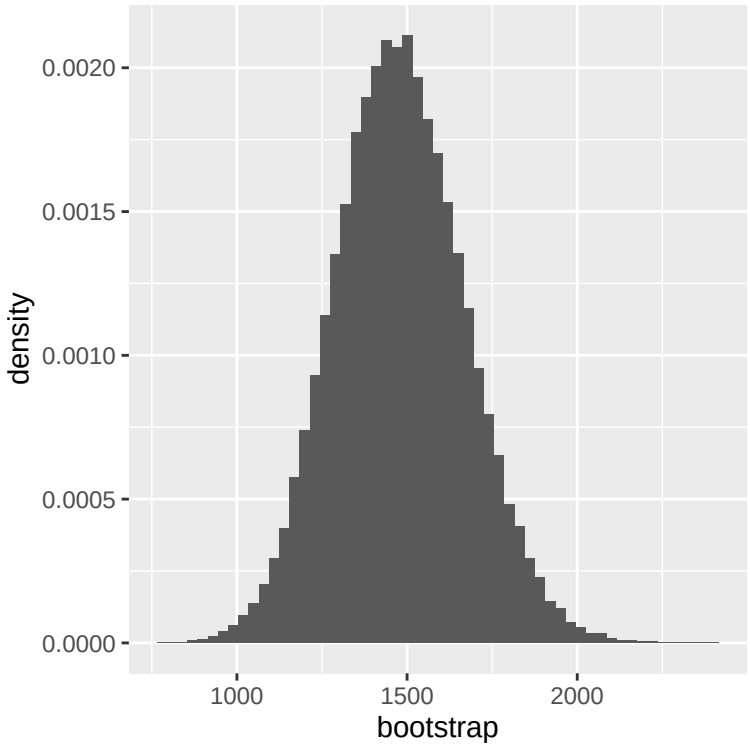
```
p.value.exact = 1-pnorm((mean(survivalTimes)-mu0)/(sd(survivalTimes)/sqrt(n))); p.value.exac

## [1] 0.009150125
```

```r
set.seed(777)
t.obs = (mean(survivalTimes)-mu0)/(sd0/sqrt(n))
t.star = numeric(B)
z=survivalTimes-mean(survivalTimes)+mu0
for(i in 1:B){
    z.star    = sample(z,n,replace=T)
    sd.z.star = sd(z.star)
    t.star[i] = (mean(z.star)-mu0)/(sd.z.star/sqrt(n))
}

## decision on H0 based on the p.value
p.value <- sum(t.star>t.obs)/B; p.value

## [1] 0.00784
```

The exact p.value is greater than the p.value of the boostrap yet there are no significant changes. Both the exact p.value and the bootstrap p.value are less than 0.05 therefore we reject the null hypothesis.

2.3.2 (b) Compute the 90% bootstrap pivotal and percentile confidence intervals for μ. Plot the histogram of the B bootstrap estimates of μ. Which CI do you think is more adequate?

```r
set.seed(777)
alpha = 0.1
t.star = numeric(B)
t= mean(survivalTimes)
for(i in 1:B){
    x.boot = sample(survivalTimes, length(survivalTimes), replace=T)
    t.star[i] = mean(x.boot)
}

### pivotal CI (review)
delta.star = t.star - t
d = quantile(delta.star, c(alpha/2,1-alpha/2))
ci.boot = t - c(d[2],d[1])
names(ci.boot) <- c("5%", "95%")
ci.boot

##      5%     95%
## 1159.10 1781.75

library(ggplot2)
p1 <- ggplot(data.frame(bootstrap = t.star), aes(x = bootstrap)) +
      geom_histogram(aes(y = after_stat(density)), binwidth= 30)
plot(p1)
```




```
## percentile 90% CI
d = quantile(t.star, c(alpha/2,1-alpha/2)); d

##       5%     95%
## 1175.85 1798.50
```

2.3.3 (c) Compute the 90% BCa bootstrap CI for μ.

```
set.seed(777)
u <- c(alpha/2, 1-alpha/2) #desired quantiles
z <- qnorm(mean(t.star < t))
z.u <- qnorm(u)

## calculate accelaration using an estimator
t.star.jack = numeric(length(survivalTimes))
for(i in 1:length(survivalTimes)){
    t.star.jack[i] = mean(survivalTimes[-i])
}
mean.t.star.jack <- mean(t.star.jack)
a.hat <- sum((mean.t.star.jack - t.star.jack)^3)/(6*(sum((mean.t.star.jack - t.star.jack)^2)

#Adjusted quantiles
u_adjusted <- pnorm(z + (z+z.u)/(1-a.hat*(z+z.u)))

#Accelerated Bootstrap CI
quantile(t.star, u_adjusted)
```

```
## 6.170757% 96.05817%
##   1193.45   1822.70
```

2.3.4 (d) Use the boot.ci() function from the R boot package to compute all the above confidence intervals.

```
set.seed(777)
library(boot)
boot.T <- function(data,indices){
    return(mean(data[indices,]))
}
boot.mean <- boot(data=as.data.frame(survivalTimes),statistic = boot.T,R=B)
boot.ci(boot.mean,type=c("basic","norm","perc", "bca"), conf=0.9)

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 100000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.mean, conf = 0.9, type = c("basic", "norm",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Basic
## 90%   (1167, 1789 )   (1161, 1782 )
##
## Level     Percentile            BCa
## 90%   (1176, 1797 )   (1194, 1822 )
## Calculations and Intervals on Original Scale
```

2.4 This car assembly factory needs that all these components are replaced after 1100 hours of service. Let

$$\mathbf{T = \text{number of survival hours of a component.}}$$

One is interested in estimating the proportion of components that live more than 1100 hours, i.e., one wishes to estimate $p = P(T > 1100)$. It is known that X = number of components that live more than 1100 hours in n inspected components, where $p = P(T > 1100)$ is the probability of a success, has distribution $\sim Bin(n, p)$

2.4.1 (e) Show that $\mathcal{P} = \frac{X}{n}$ is an unbiased and consistent estimator of p. Estimate p and SE(P).

$$E(\mathcal{P}) = E\left(\frac{X}{n}\right) = \frac{E(X)}{n}$$

X has an Binomial distribution therefore its expected value is np. Proof:

$$\sum_{k=1}^{n} k\binom{n}{k}p^K(1-p)^{n-k} =$$
$$\sum_{k=1}^{n} k\frac{n!}{k!(n-k)!}p^k(1-p)^{n-k} =$$
$$\sum_{k=1}^{n} \frac{n!}{(k-1)!(n-k)!}p^k(1-p)^{n-k} =$$
$$np\sum_{k=1}^{n} \frac{(n-1)!}{(k-1)!(n-k)!}p^{k-1}(1-p)^{n-k} =$$
$$np\sum_{k=1}^{n} \binom{n-1}{k-1}p^{k-1}(1-p)^{n-k}$$

through the binomial theorem we get:

$$np(p+(1-p))^{n-1} = np*1 = np$$

Knowing this:

$$\frac{E(X)}{n} = \frac{np}{n} = p$$

since $E(\mathcal{P}) = p$ we can affirm that $\mathcal{P}$ is an unbiased estimator.

$$\lim_{n\to\infty} V(\mathcal{P}) = \lim_{n\to\infty} V\left(\frac{X}{n}\right) = \lim_{n\to\infty} \frac{V(X)}{n^2} = 0$$

$\mathcal{P}$ is a consistent estimator since it's variance aproaches 0 as the sample size increases.

```
P.mean = mean(survivalTimes > 1100); P.mean

## [1] 0.65
```

2.4.2 (f) Describe and discuss in detail the non-parametric bootstrap and jackknife techniques. Use both approaches (B = 10000 samples in the case of the bootstrap) to estimate the variance, standard error and bias of $\mathcal{P}$. Compare the results. Check whether there is need to correct the original estimate of p for bias and if such report the corrected estimate of p.

non-parametric bootstrapping

Bootstraping is any resampling method that that uses random sampling with replacement for a given sample. It is used to assess the accuracy of statistical estimates and tests and estimate statistics of the population.

The non-parametric does not make any assumptions about the distribution of the population. This makes it on average more robust to distributional assumptions unlike the parametric bootstrap. With this robustness comes a necessity for a larger sample size than what would be needed for a parametric bootstrap. If the sample size is too small the estimator it might not approximate the real value very well.

Here is algorithm for the non-parametric bootstrap:

step 1 - choose the the amount of bootstrap samples to compute

step 2 - for each bootstrap sample

step 1 - sample the population with replacement

step 2 - calculate the statistic you want to obtain

step 3 - calculate the mean of the the obtain bootstrap samples

Bootstrapping is really simple and easy to understand. It can be applied to complex estimators to derive estimates for the standard error and confidence intervals. Bootstrapping also avoids the cost of gathering new sample data.

Bootstraping is heavily dependent on the estimator. Using it without care can lead to inconsistent results. When using bootstrap, assumptions about the about the sample are made and it does not guarantee that values are representative of the general population. Bootstrapping can be time consuming and it's not easily automatable using statistical computer packages.

jackknife

The jackknife is another resampling technique.It predates the bootstrap. It was created with the goal of estimating the bias and standard error estimators. The jackknife allows us to obtain samples similar to the original sample by leaving one sample value out at a time, unlike the bootstrap method.

Here is the jackknife algorithm:

- Let $X_1, ..., X_n$ be a random sample from some population $X \sim F(\theta)$ with both F and θ unknown. Let $T = g(X_1, ..., X_n)$ be an estimator of θ.
- Let $x_1, ..., x_n$ be a realization of the random sample above and $t = g(x_1, ..., x_n)$ an estimate of θ.
- Let

  $$x_2, x_3, ..., x_n, \quad x_1, x_3, ..., x_n, \quad ... \ , \quad x_1, ..., x_{n1}$$

  be the n jackknife samples (of size $(n-1)$) and

  $$t_1^* = g(x_2, x_3, ..., x_n), \quad ... \ , \quad t_n^* = g(x_1, ..., x_{n1})$$

  be the n jackknife estimates of θ
- let $t_{jack} = \frac{1}{n} \sum_{i=1}^{n} t_i^* = \bar{t^*}$ (jacknife estimation of θ)

It is normally less computational intensive than the bootstrap method. like the bootstrap, it does not make any assumptions about the distribution of the sample. It is easier than the bootstrap to apply to complex sampling schemes than the bootstrap.

Jackknife only works well for functions with small changes in the statistic therefore it works well with, for example, linear statistics. May that not be the case, it might fail to estimate the variance successfully. Compared to the bootstrap, jackknife is only a crude approximation and therefore, if a more accurate value is desired, bootstrap confidence intervals are a better choice.

```r
### bootstrap
set.seed(777)
B= 10000
t.star.boot=numeric(B)
for(i in 1:B){
    survivalTimes.sample = sample(survivalTimes,length(survivalTimes),replace =T)
    t.star.boot[i] = mean(survivalTimes.sample > 1100)
}

t.star.boot.mean =mean(t.star.boot); t.star.boot.mean

## [1] 0.65045

t.star.boot.bias = t.star.boot.mean - P.mean; t.star.boot.bias

## [1] 0.00045

t.star.boot.var = var(t.star.boot); t.star.boot.var

## [1] 0.01134443

t.star.boot.sd = sqrt(t.star.boot.var); t.star.boot.sd

## [1] 0.1065102

### jackknife
set.seed(777)
t.star.jack=numeric(length(survivalTimes))
for(i in 1:length(survivalTimes)){
    t.star.jack[i] = mean(survivalTimes[-i] > 1100)
}

t.star.jack.mean = mean(t.star.jack); t.star.jack.mean

## [1] 0.65

t.star.jack.bias = (t.star.jack.mean - P.mean) * (n-1); t.star.jack.bias

## [1] 0

t.star.jack.var = mean((t.star.jack - t.star.jack.mean)^2) * (n-1); t.star.jack.var

## [1] 0.01197368

t.star.jack.sd = sqrt(t.star.jack.var); t.star.jack.sd;

## [1] 0.1094243
```

2.4.3 (g) The jackknife-after-bootstrap technique provides a way of measuring the uncertainty associated with the bootstrap estimate $\widehat{SE}(T)$ (T some estimator of interest). Use this technique in order to estimate the standard error of the bootstrap estimate of $SE(P)$ obtained in (d).

```
set.seed(777)
B= 10000
t.star.boot=numeric(B)
t.star.jack=numeric(length(survivalTimes))
for(i in 1:B){
    z.star = sample(survivalTimes,length(survivalTimes),replace =T)
    t.star.boot[i] = mean(z.star > 1100)
}
for(i in 1:B){
    t.star.jack[i] = sd(t.star.boot[-i])
}

t.star.jack.var = mean((t.star.jack - mean(t.star.jack))^2) * (n-1); t.star.jack.var

## [1] 1.029635e-09

t.star.jack.sd = sqrt(t.star.jack.var); t.star.jack.sd;

## [1] 3.208793e-05
```

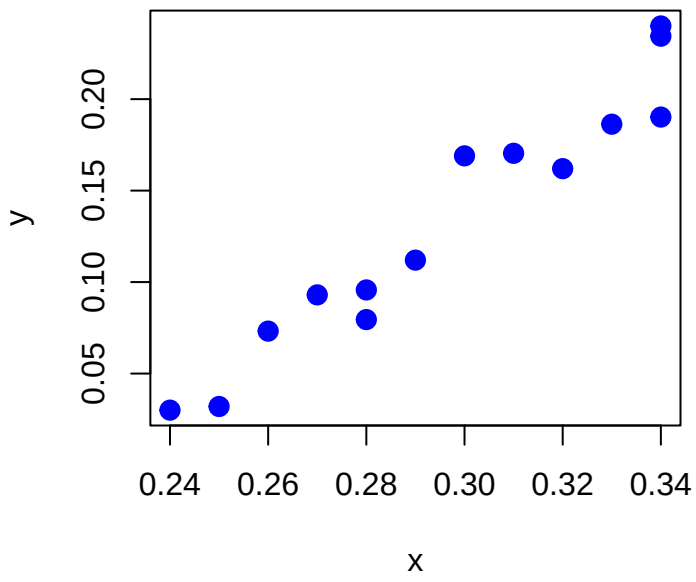
2.5 1.3 Consider the following data

x	0.3400	0.2800	0.3100	0.2800	0.300	0.270	0.320	0.250	0.34	0.3400	0.290
0.2600	0.24	0.3300									
y	0.2344	0.0795	0.1704	0.0957	0.169	0.093	0.162	0.032	0.24	0.1902	0.112
0.0732	0.03	0.1863									

```
x <- c(0.3400, 0.2800, 0.3100, 0.2800, 0.300, 0.270, 0.320, 0.250, 0.34, 0.3400, 0.290, 0.26
y <- c(0.2344, 0.0795, 0.1704, 0.0957, 0.169, 0.093, 0.162, 0.032, 0.24, 0.1902, 0.112, 0.07
```

2.5.1 (a) Graphically inspect that there is a linear trend in the data. Comment on the linear trend. Fit a linear regression model to your data in R presenting and commenting in detail all the summary results referring to the fitted model (returned by the R function summary()). In addition plot the data *versus* the fitted line; and use the R built-in function confint() and report a 90% CI for the slope parameter.

```
x <- c(0.3400, 0.2800, 0.3100, 0.2800, 0.300, 0.270, 0.320, 0.250, 0.34, 0.3400, 0.290, 0.26
y <- c(0.2344, 0.0795, 0.1704, 0.0957, 0.169, 0.093, 0.162, 0.032, 0.24, 0.1902, 0.112, 0.07

plot(x,y,col='blue', pch=20, cex=2)
```




```
data <- data.frame(x,y)
#Fit a linear regression model to your data
model <- lm(y~x, data)
# presenting the summary results referring to the fitted model (returned by the R function
summary(model)

##
## Call:
## lm(formula = y ~ x, data = data)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0275498 -0.0119528 -0.0000411  0.0107442  0.0286795
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.44040    0.04315  -10.21 2.87e-07 ***
## x            1.93573    0.14466   13.38 1.42e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01802 on 12 degrees of freedom
## Multiple R-squared:  0.9372,Adjusted R-squared:  0.932
## F-statistic: 179.1 on 1 and 12 DF,  p-value: 1.424e-08
```

From the plot above, it's possible to conclude that there is somewhat strong relationship between x and y (i.e., when x increases y also increases).

A linear regression is a regression in which the hypothesis class corresponds to the model $y = \theta_1 x_1 + \theta_2 x_2 + ... + \theta_{n+1} + \epsilon$, where each x_n is one dimension of the input space. In our case, the input space has only one dimension and we have a set of (x, y) points, thus our model will have the format $y = \theta_1 x + \theta_2 + \epsilon$ (these thetas are also known as betas)

Call

This item shows the formula used to fit the data. The lm function needs a formula (Y X) and a data source. In our case, the formula was (y x) and a data frame was created (with the two variables) to be our data source

Residuals

This item shows the residuals, that is response minus fitted values (the difference between the predicted values and the actual values, i.e., the smaller the residuals the better). This section divides into five summary points: Min, 1Q, Median, 3Q, and Max, which, when analyzed, should have a symmetrical distribution (sum and mean equal zero). Considering the summary obtained, this condition is verified because the mean is very close to zero (-4.11 × 10-5) and the sum of the errors is also very close to zero

Coefficients

For each variable and the intercept, the theta and other attributes like the standard error, a t-test value and significance are calculated Has seen before, our model is in the format $y = \theta_1 x + \theta_2 + \epsilon$, and by inspecting the summary, we can conclude that $\theta_1 = 1.93573$, $\theta_2 = -0.44040$

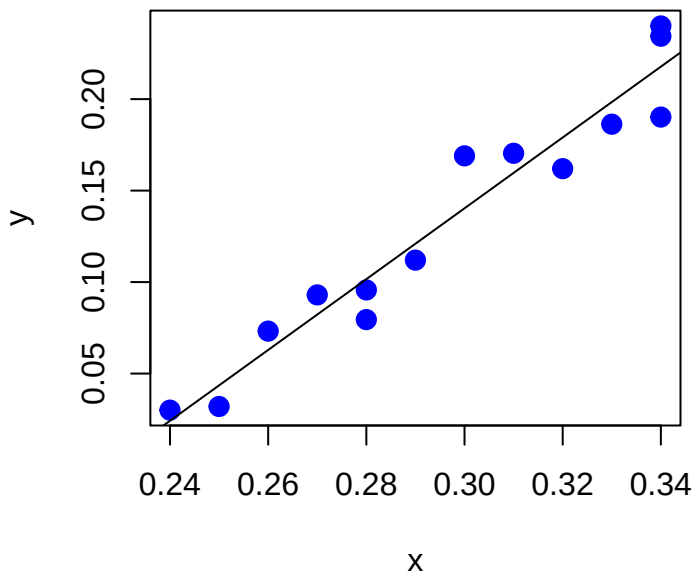
Residual standard error

Every linear model is assumed to contain an error term ϵ, the Residual standard error is the average amount that the response will deviate from the true regression line. With this, we can conclude that $\epsilon = 0.01802$ and that our model follows the expression (for the given data set) $y = 1.93573x - 0.44040 + 0.01802$

R-squared

This statistic is a measure of how much of the variability in the data is explained by the model (how well the model is fitting the actual data). A value of 1 indicates that the model perfectly explains all of the variability in the data, while a value of 0 indicates that the model does not explain any of the variability in the data. In our case was obtained an Adjusted R-squared = 0.932 this means that 93 percent of the variance found in the x variable (dist) can be explained by the y variable.

```
# plot the data versus the fitted line
plot(x,y,col='blue', pch=20, cex=2)
abline(model)
```




```
# use the R built-in function confint() and report a 90% CI for the slope parameter.
confint(model, level = 0.90)

##                    5 %       95 %
## (Intercept) -0.517309 -0.3634904
## x            1.677902  2.1935656

slope_parameter_CI <- confint(model, level = 0.90)[2, ] ; slope_parameter_CI

##      5 %     95 %
## 1.677902 2.193566
```

2.5.2 (b) Carefully check for the linear model's underlying assumptions - use both visual inspection and adequate statistical tests (at the 5% level) to check the assumptions. Does the fitted model validate all the underlying assumptions?

- Linearity: The relationship between X and the mean of Y is linear.
- Homoscedasticity: The variance of residual is the same for any value of X.
- Independence: Observations are independent of each other.
- Normality: For any fixed value of X, Y is normally distributed.

Considering last question and

```r
cor(x,y)

## [1] 0.9680854
```

the linearity assumption seems to be a valid one. To validate Homoscedasticity we applied the **Breusch Pagan Test** (H_0: the variance is constant) with the help of the built in function of the library *lmtest*

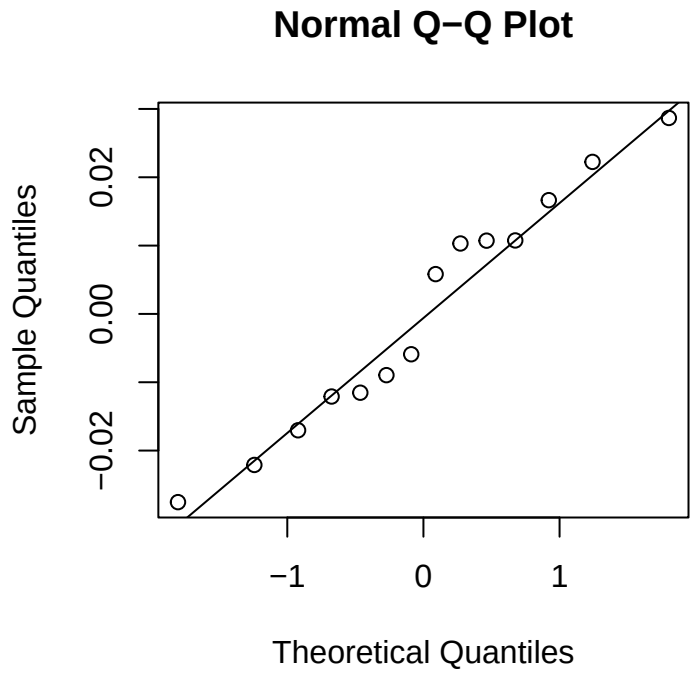
```r
bp_test <- lmtest::bptest(model); bp_test$p.value

##         BP
## 0.06224839
```

Since the p-value obtained (0.06224839) is greater than our alpha (0.05) H_0 it's not rejected and Homoscedasticity is validated One way to determine if the Independence assumption is met is to perform a **Durbin-Watson test**, which is used to detect the presence of autocorrelation in the residuals of a regression (H_0: There is no correlation among the residuals). With the help of the built in function of the library "car", we get

```r
DW_Test <- car::durbinWatsonTest(model); DW_Test$p

## [1] 0.344
```

Since the p-value obtained (0.262) is greater than our alpha (0.05), H_0 it's not rejected and Independence is validated. To verify Normality it's possible to use statistical tests (Shapiro-Wilk test) or by visualization like QQplots (both possibilities applied to the model residuals). A QQplot shows how two distributions quantiles line up, with the theoretical distribution (in our case, the normal distribution). The more lined up, the more adjusted to the theoretical distribution.

```r
qqnorm(model$residuals)
qqline(model$residuals)
```



By analyzing the QQplot we can conclude the adjustment to the Normal Distribution. With **Shapiro-Wilk test** (H_0: the population is normally distributed)

```
shapiro.test(model$residuals)

##
##  Shapiro-Wilk normality test
##
## data:  model$residuals
## W = 0.95742, p-value = 0.6804
```

Since the p-value obtained (0.6804) is greater than our alpha (0.05), H_0 it's not rejected and Normality is validated.

2.5.3 (c) Use the bootstrap of the pairs (with $B = 10000$) to: estimate the bias and standard error of the slope parameter estimator; check if there's need to correct the original estimate and if so report the corrected estimate; construct a pivotal 90% CI for the slope parameter; compare it with the CI obtained in (a).

```
B=10000
set.seed(123)
n = nrow(data)

theta.npb <- matrix(0,B,2)
for(i in 1:B){
  idx      = sample(1:n,n,replace=T)
```

```
  x.npb        = x[idx]
  y.npb        = y[idx]
  model.npb    = lm(y.npb~x.npb)
  theta.npb[i,] = model.npb$coefficients
}

# reporting the bootstrap mean and standard deviation
# for theta0 and theta1; we see there's no need to correct the original estimates
theta   <- model$coefficients
result <- matrix(c(theta[2], mean(theta.npb[,2]), sd(theta.npb[,2]),mean(theta.npb[,2])-thet
colnames(result) <- c("original slope parameter","bootstrap slope parameter","boot.se","boot
rownames(result) <- c("theta1 / slope parameter")
out.table <- cbind(result,abs(result[,4])/result[,3])
colnames(out.table)[5] <- "|bias|/SE"
out.table

##                          original slope parameter bootstrap slope parameter
## theta1 / slope parameter                 1.935734                  1.938906
##                            boot.se boot.bias  |bias|/SE
## theta1 / slope parameter 0.1399878 0.0031721 0.02265983
```

since $|bias|/SE \leq 0.25$ it is not necessary to correct/adjust the original estimate for bias

```
# pivotal bootstrap 95% CI
deltastar = theta.npb[,2] - theta[2]
alpha = 0.1
a = quantile(deltastar,c(alpha/2, 1 - alpha/2))
theta1.IC = theta[2] - c(a[2], a[1])
names(theta1.IC) <- c(paste0(alpha/2 * 100, "%"),paste0((1 - alpha/2) * 100, "%"))
theta1.IC

##       5%      95%
## 1.705220 2.162514
```

Comparing the two Confidence Intervals we can conclude that the pivotal CI is slightly smaller than the one created with the built-in function confint()

3 Problem 2 - Optimization

Let $X \sim \text{Pareto}(1, \alpha)$, which has p.d.f.

$$f(x;\alpha) = \frac{\alpha}{x^{\alpha+1}}, \quad \alpha > 0, \quad x \geq 1.$$

Let

1.977866	1.836622	1.097168	1.232889	1.229526	2.438342	1.551389	1.300618	1.068584
1.183466	2.179033	1.535904	1.323500	1.458713	1.013755	3.602314	1.087067	1.014013
1.613929	2.792161	1.197081	1.021430	1.111531	1.131036	1.064926		

be an observed sample from X.

3.1 2.1 Derive the likelihood, log-likelihood and score functions (simplify the expressions as much as possible). Derive both the maximum likelihood estimator (MLE) and method of moments estimator (MME) of α and use them to estimate α. Why are ML estimators so attractive? Briefly describe the properties of MLEs.

```
os = c(1.977866, 1.836622, 1.097168, 1.232889, 1.229526,
       2.438342, 1.551389, 1.300618, 1.068584, .183466,
       2.179033, 1.535904, 1.323500, 1.458713, 1.013755,
       3.602314, 1.087067, 1.014013, 1.613929, 2.792161,
       1.197081, 1.021430, 1.111531, 1.131036, 1.064926)

    n = length(os)

    #MLE
    MLE = function(x) {n/sum(log(x))}

    MLE.v = MLE(os)

    MLE.v
```

```
## [1] 3.561552
```

```
    #MME
    MME = function(x){mean(x)/(mean(x)-1)}

    MME.v = MME(os)

    MME.v
```

```
## [1] 3.072476
```

The principle of maximum likelihood provides a unified approach to estimating parameters of the distribution given sample data. Although ML estimators $\hat{\theta}_n$ are not in general unbiased, they possess a number of desirable asymptotic properties:

- consistency: $\hat{\theta}_n \overset{n\to\infty}{\to} \theta$
- normality: $\hat{\theta}_n \sim \mathcal{N}(\theta, \Sigma)$, where Σ^{-1} is the Fisher information matrix.
- efficiency: Var $\left(\hat{\theta}_n\right)$ approaches Cramer-Rao lower bound.

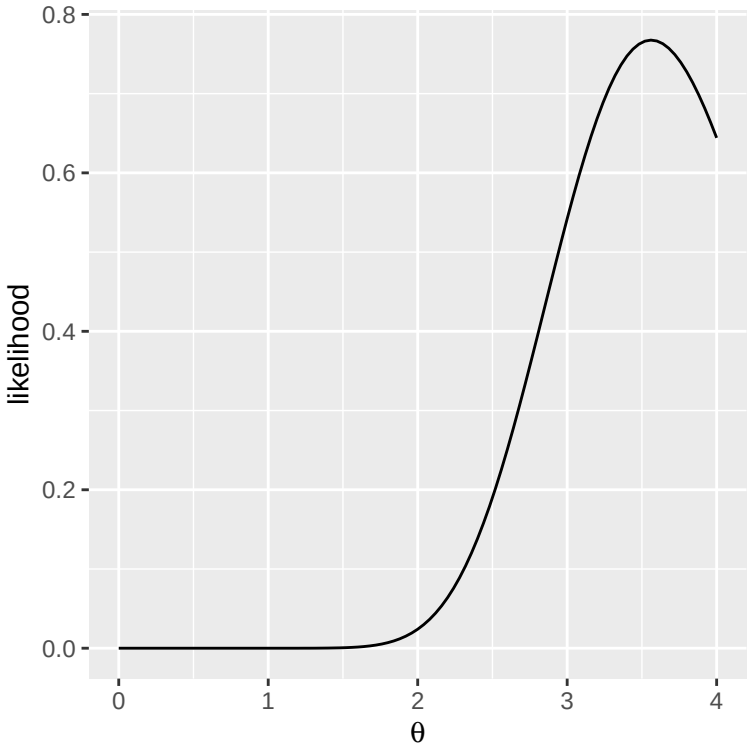
3.2 2.2 Noting that the Pareto distribution belongs to the exponential family, derive the Fisher information $I_n(\alpha)$. Use the Fisher information to estimate the variance of the MLE.
Assume herein that it was not possible to derive the MLE of α and as such, it was not possible to compute the ML estimate of α.

```r
    In = n/(MLE.v^2)
    MLE.var = 1/In

    MLE.var
```

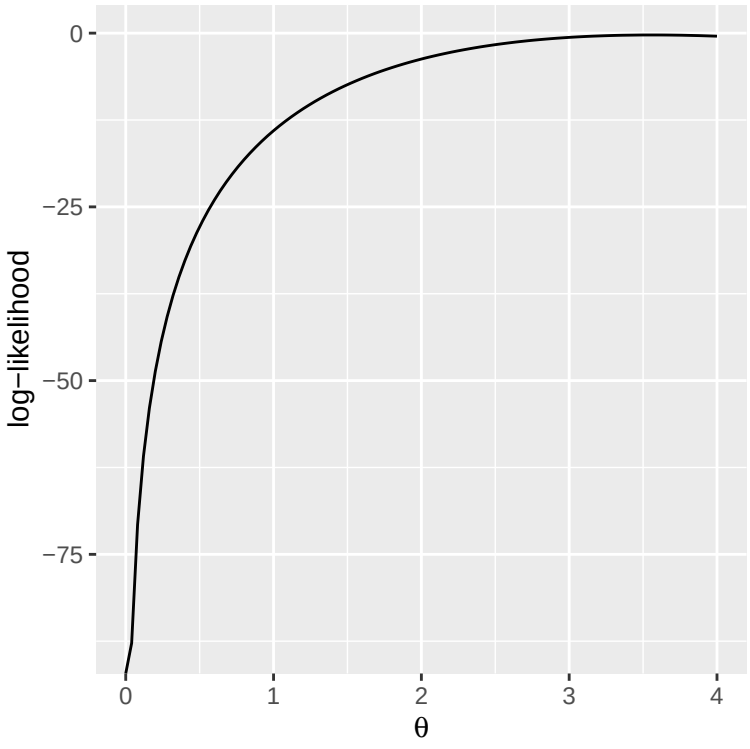
```
## [1] 0.5073862
```

3.3 2.3 Display graphically (side-by-side) the likelihood, log-likelihood and score functions in order to locate the ML estimate of α. Indicate an interval that contains the ML estimate.

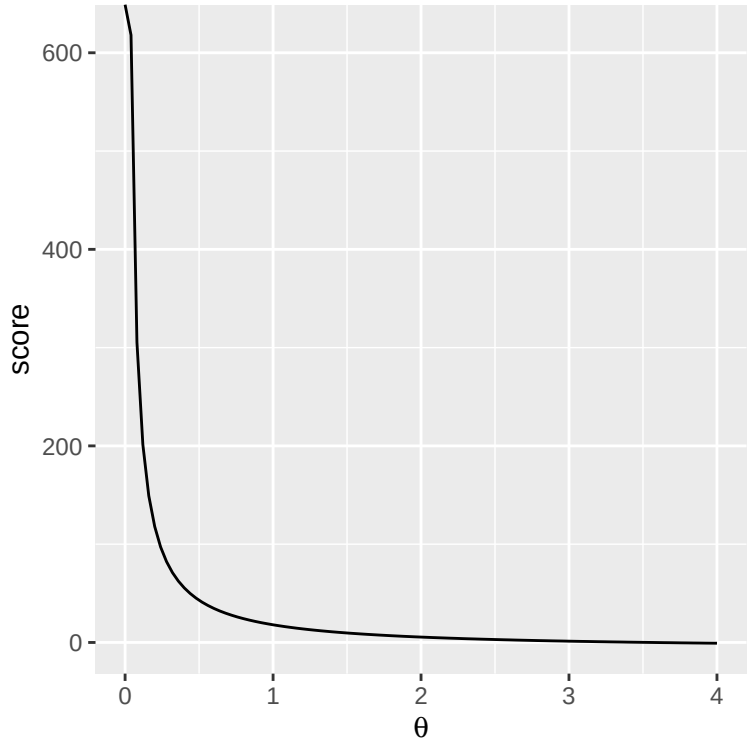
```r
   #Likelihood function
    likh = function(theta){
      l = vector()
      for(t in theta)
        l = c(l, (t^n)*prod(1/(os^(t+1))))
      return(l)
    }

    #Log-Likelihood function
    loglikh = function(theta){
      ll = vector()
      for(t in theta)
        ll = c(ll, n*log(t)-((t + 1) * sum(log(os))))
      return(ll)
    }

    #Score function
    scr = function(theta) {
      s = vector()
      for(t in theta)
        s = c(s, (n/t) - sum(log(os)))
      return(s)
    }

    #Graphical Display
    library(ggplot2)
    base <-
      ggplot() +
      xlim(0, 4)

    base + xlab(expression(theta)) + ylab("likelihood") + geom_function(fun = likh)
```




```
base + xlab(expression(theta)) + ylab("log-likelihood") + geom_function(fun = loglikh)
```




```
base + xlab(expression(theta)) + ylab("score") + geom_function(fun = scr)
```




```
    #If we use interval estimation (2,4):

    #Likelihood function maximum
    optimize(likh,c(2,4),maximum=T)$maximum
```

```
## [1] 3.561555
```

```
    #Log-Likelihood function maximum
    optimize(loglikh,c(2,4),maximum=T)$maximum
```

```
## [1] 3.561547
```

```
    #Score function root
    uniroot(scr,c(2,4))$root
```

```
## [1] 3.561556
```

3.4 2.4 Use the R function maxLik() from library maxLik to approximate the ML estimate of α. Feed maxLik() with the initial estimate of α given by the method of moments.

```
    library(maxLik)
```

```
## Loading required package: miscTools
##
## Please cite the 'maxLik' package as:
```

```
## Henningsen, Arne and Toomet, Ott (2011). maxLik: A package for maximum likelihood
estimation in R. Computational Statistics 26(3), 443-458. DOI 10.1007/s00180-010-0217-1.
##
## If you have questions, suggestions, or comments regarding the 'maxLik' package,
please use a forum or 'tracker' at maxLik's R-Forge site:
## https://r-forge.r-project.org/projects/maxlik/
```

```
    maxLik(loglikh,start=2.81418)
```

```
    # Maximum Likelihood estimation
    # Newton-Raphson maximisation, 1 iterations
    # Return code 1: gradient close to zero (gradtol)
    # Log-Likelihood: -8.016816 (1 free parameter(s))
    # Estimate(s): 2.814179

    #The ML estimate of alpha by R function maxLik() was 2.814179.
```

3.5 2.5 Implement the methods of bissection, Newton-Raphson and secant in R and use them to estimate α. Report all the iterations computed by the different methods plus the corresponding errors. Justify your choice of the initial estimates for each method and discuss the results.

```
bisection <- function(a,b,eps){
    # [a,b]: interval where s verifies Bolzano's theorem
    # eps : is the stopping rule
    alpha.it = vector(); alpha.it[1] = (a+b)/2
      k = 1; diff = 1
      diffs= vector()
      while(diff>eps){
        #If multiplication is negative, reduce b
        if(scr(alpha.it[k])*scr(a)<0){
          b = alpha.it[k]
          alpha.it[k+1] = (a+b)/2
        }
        #If multiplication is positive, raise a
        else{if(scr(alpha.it[k])*scr(a)>0){
          a = alpha.it[k]
          alpha.it[k+1] = (a+b)/2
          #If multiplication result is 0, we apply the stopping rule
        }else{alpha.it[k+1]=alpha.it[k]}
        }
        #Update the iteration difference for the stopping rule
```

```
      diff = abs(alpha.it[k+1]-alpha.it[k])
      diffs[k]=diff
      k = k+1
    }
    result = as.matrix(alpha.it)
    colnames(result)<-"iterations"
    rownames(result)<-1:length(alpha.it)
    retList<-list("res"=result,"error"=diffs)
    retList
  }
  #Bisection result for each iteration
  l1=bisection(2,4,1e-06)
  m1=t(l1$res)
  m1
```

```
##            1   2    3     4      5       6        7        8        9       10
## iterations 3 3.5 3.75 3.625 3.5625 3.53125 3.546875 3.554688 3.558594 3.560547
##                  11       12       13       14       15       16       17
## iterations 3.561523 3.562012 3.561768 3.561646 3.561584 3.561554 3.561539
##                  18       19       20       21
## iterations 3.561546 3.56155 3.561552 3.561553
```

```
  #Error result for each iteration
l1$error
```

```
##  [1] 5.000000e-01 2.500000e-01 1.250000e-01 6.250000e-02 3.125000e-02
##  [6] 1.562500e-02 7.812500e-03 3.906250e-03 1.953125e-03 9.765625e-04
## [11] 4.882812e-04 2.441406e-04 1.220703e-04 6.103516e-05 3.051758e-05
## [16] 1.525879e-05 7.629395e-06 3.814697e-06 1.907349e-06 9.536743e-07
```

For the bisection method, we use the initial estimated interval $[2, 4]$, as it requires a lower bound and a upper bound to start the convergence of the values until we get the closest estimation.

```
  #(Newton-Raphson aux function)
  prime <- function(alpha){
    out = numeric(length(alpha))
    if(length(alpha)==1){out =- n/alpha^2}
    if(length(alpha)!=1){
      for(i in 1:length(alpha)){out[i] = - n/alpha[i]^2}
    }
    return(out)
  }

#Newton-Raphson algorithm to estimate alpha
 NR <- function(x,alpha0,eps){
   # x: observed sample
   # alpha0: first value to iterate over the vector
   # eps: the stopping rule
```

```
  alpha.it = vector()
  alpha.it[1] = alpha0
  k = 1
  diff = 1
  diffs=vector()
  #iterations
  while(diff>eps){
    alpha.it[k+1] = alpha.it[k]-scr(alpha.it[k])/prime(alpha.it[k])
    diff = abs(alpha.it[k+1]-alpha.it[k])
    diffs[k]=diff
    k = k+1
  }
  result = as.matrix(alpha.it)
  colnames(result)<-"iterations"
  rownames(result)<-1:length(alpha.it)
  result
  retList<-list("res"=result,"error"=diffs)
  retList
}
#NR result for each iteration
l2=NR(os,MLE.var,1e-06)
m2=t(l2$res)
m2
```

```
##                    1         2        3        4        5        6        7
## iterations 0.5073862 0.9424892 1.635569 2.520037 3.256979 3.535506 3.561362
##                    8        9
## iterations 3.561552 3.561552
```

```
#Error result for each iteration
 l2$error
```

```
## [1] 4.351029e-01 6.930794e-01 8.844679e-01 7.369422e-01 2.785275e-01
## [6] 2.585580e-02 1.904710e-04 1.018743e-08
```

For the Newton-Raphson method, we use the *MLE.var* initial value as α^0, as it is a reference value for the upper limit, until it converges to the closest estimation.

```
#Secant algorithm to estimate alpha
secant <- function(x, alpha0, alpha1, eps) {
  # x: observed sample
  # alpha0: interval left limit
  # alpha1: interval right limit
  # eps: stopping rule value
  alpha.it = vector()
  alpha.it[1] = alpha0
  32
  alpha.it[2] = alpha1
  k = 2
```

```
  diff = 1
  diffs = vector()
  diffs[1] = diff
  # iterations
  while (diff > eps) {
    alpha.it[k + 1] = alpha.it[k] - scr(alpha.it[k]) *
      ((alpha.it[k] - alpha.it[k-1])/(scr(alpha.it[k]) - scr(alpha.it[k - 1])))
    diff = abs(alpha.it[k + 1] - alpha.it[k])
    diffs[k] = diff
    k = k + 1
  }
  result = as.matrix(alpha.it)
  colnames(result) <- "iterations"
  rownames(result) <- 1:length(alpha.it)
  retList <- list(res = result, error = diffs)
  retList
}
#Secant result for each iteration
l3 = secant(os, 2, 4, 1e-06)
m3 = t(l3$res)
m3
```

```
##            1 2        3        4       5        6        7        8
## iterations 2 4 3.753789 3.537887 3.56283 3.561561 3.561552 3.561552
```

```
#Error result for each iteration
l3$error
```

```
## [1] 1.000000e+00 2.462115e-01 2.159014e-01 2.494271e-02 1.268859e-03
## [6] 8.490594e-06 3.044056e-09
```

For the secant method, we use the initial estimated interval $[2, 4]$, as it requires a lower bound and a upper bound to start the convergence of the values until we get the closest estimation.

3.6 2.6 Show, analytically, that the methods of Newton-Raphson and Fisher scoring coincide in this particular case.

```
#I(alpha)
Ialpha<-function(alpha){
  n/(alpha^2)
}
#Fisher Scoring algorithm
fisherScoring <- function(x,alpha0,eps){
  # x: observed sample
  # alpha0: first value to iterate over the vector
  # eps: the stopping rule
  alpha.it = vector()
  alpha.it[1] = alpha0
  k = 1
```

```
      diff = 1
      diffs=vector()
      diffs[1]=diff
      # iterations
      while(diff>eps){
        alpha.it[k+1] = alpha.it[k]+(1/Ialpha(alpha.it[k]))*scr(alpha.it[k])
        diff = abs(alpha.it[k+1]-alpha.it[k])
        k = k+1
        diffs[k]=diff
      }
      result = as.matrix(alpha.it)
      colnames(result)<-"iterations"
      rownames(result)<-1:length(alpha.it)
      retList<-list("res"=result,"error"=diffs)
      retList
    }
    #Fisher Scoring result for each iteration
    l4=fisherScoring(os,MLE.var,0.000001)
    m4=t(l4$res)
    m4
```

```
##                    1         2        3        4        5        6        7
## iterations 0.5073862 0.9424892 1.635569 2.520037 3.256979 3.535506 3.561362
##                    8        9
## iterations 3.561552 3.561552
```

```
    #Error result for each iteration
    l4$error
```

```
## [1] 1.000000e+00 4.351029e-01 6.930794e-01 8.844679e-01 7.369422e-01
## [6] 2.785275e-01 2.585580e-02 1.904710e-04 1.018743e-08
```

4 References

1. https://en.wikipedia.org/wiki/Methodofmoments(statistics)
2. https://www.statology.org/durbin-watson-test-r/
3. https://www.learnbymarketing.com/tutorials/linear-regression-in-r/
4. https://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html
5. https://rpubs.com/iabrady/residual-analysis
6. McCullagh, P., Nelder, J.A. (1983). Generalized Linear Models. London: Chapman and Hall.
7. Casella, G. and Berger, R. L. (2002). Statistical inference (Vol. 2). Pacific Grove, CA: Duxbury.

8. Givens, G. H. and Hoeting, J. A. (2013). Computational Statistics. John Wiley Sons, Inc.

9. Efron, B. (1980). The Jackknife, the Bootstrap, and Other Resampling Plans. Technical report 63, Division of Biostatistics, Stanford University.

10. Efron, B. and Tibshirani, R.J. (1998). An introduction to the bootstrap. CRC Press LLC.