Exam Project ST522

Björn Eyríkur Helgason

10th of June, 2016

For this project, all functions are contained in the package ST522examn, avialable as follows

```r
if(!require("devtools")){
  install.packages("devtools", repos="http://cran.rstudio.com/");library(devtools)}
```

```
## Loading required package: devtools
```

```r
#install_github("Kozilek/ST522-Faux-Examn-Package")
library(ST522examn)
```

TASK 1

A study compares the effect of daily sports on grading. The parameter of interest is the correlation between the daily time spent for physical activities and the semester average. We start by loading the data.

```r
sportdata <-
  read.table("https://raw.githubusercontent.com/haghish/ST516/master/data/sport.txt")
set.seed(2016)
```

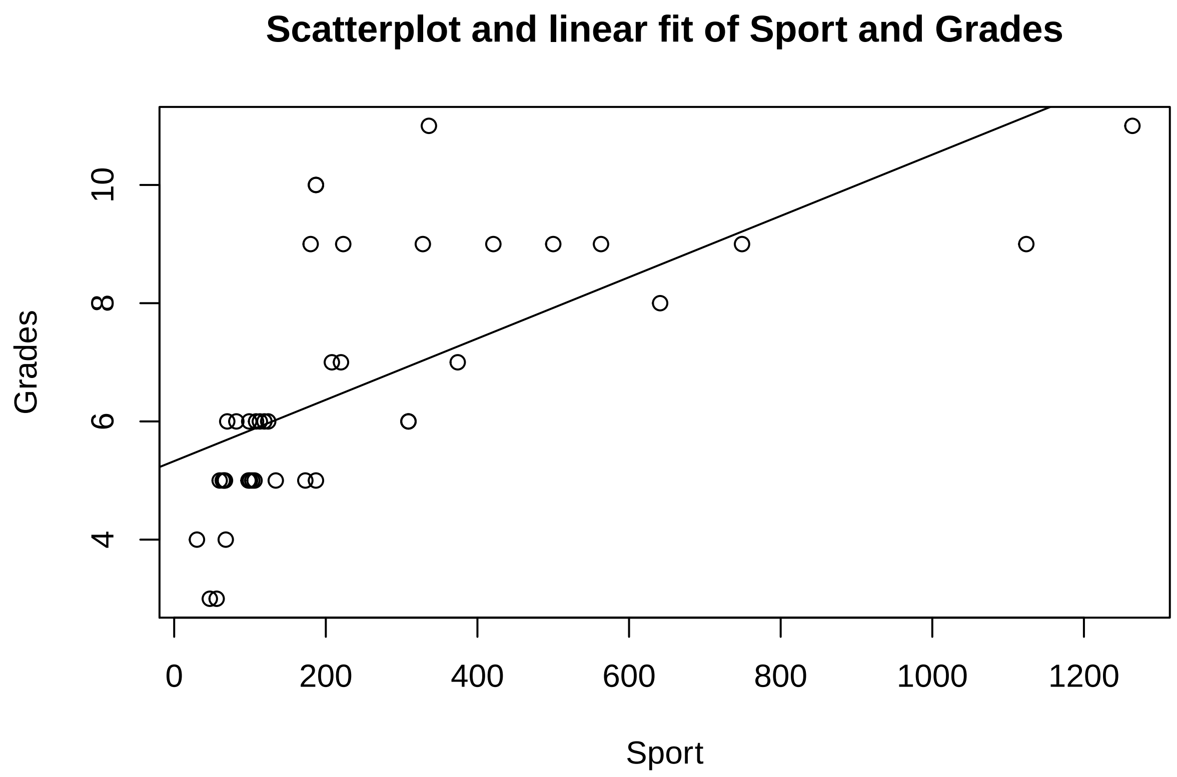
For the purpose of these tasks, the function `bootbivar()` from the package `ST522examn` is the function written for task 3. Applying the function we get

```r
bootbivar(100,sportdata$Sport,sportdata$Grades)
```

```
## Sample correlation = 0.6672733
##
## Estimated correlation = 0.6599804
##
## Estimate standard error = 0.007128649
##
## Estimate 95% confidence interval:
##  0.6460082 0.6739525
##
## Estimated bias = -0.007292879
```

From this we see that there is a correlation coefficient of about `0.66`, which indicates a somewhat strong relation between sport and grading. Observing the scatterplot, with a linear fit,

```r
plot(sportdata$Sport, sportdata$Grades,
     main = "Scatterplot and linear fit of Sport and Grades",
     xlab = "Sport", ylab = "Grades")
abline(lm(sportdata$Grades ~ sportdata$Sport))
```



this does indeed seem plausible. The low standard error and confidence inteval, suggest that the estimated correlation coefficient is reliable, and it seems reasonable to detemine that there is a non-zero relation between sport and grading.

TASK 2

For this task, i use the function `throw` from the package `ST522examn`, to simulate throwing needles. ## Part 1 Using Monte Carlo to evaluate the integral

$$\int_0^{\pi} l \sin\theta d\theta,$$

we simply

```r
rsamp <- runif(1000) # Generate a random sample (size 1000) from a uniform distribtuion
hsamp <- apply(as.array(rsamp)*pi, 1, sin) # Apply the scaled integral inner function
# Note: Two of the pi's cancel out.
mean(hsamp)# Compute the average
```

```
## [1] 0.6289736
```

We compute variance,

```r
var(hsamp)
```

```
## [1] 0.1004942
```

standard devitaion,

```r
sd(hsamp)
```

```
## [1] 0.3170083
```

standard error,

```r
sd(hsamp)/sqrt(1000)
```

```
## [1] 0.01002468
```

and confidence interval.

```r
erMarg <- 1.96 * sd(hsamp)/sqrt(1000)
mean(hsamp) + c(-erMarg, erMarg)
```

```
## [1] 0.6093252 0.6486220
```

Part 2

For the second part of the task, we use `throw` to carry out Buffon's experiment for $N = 10000, l = d = 1$,

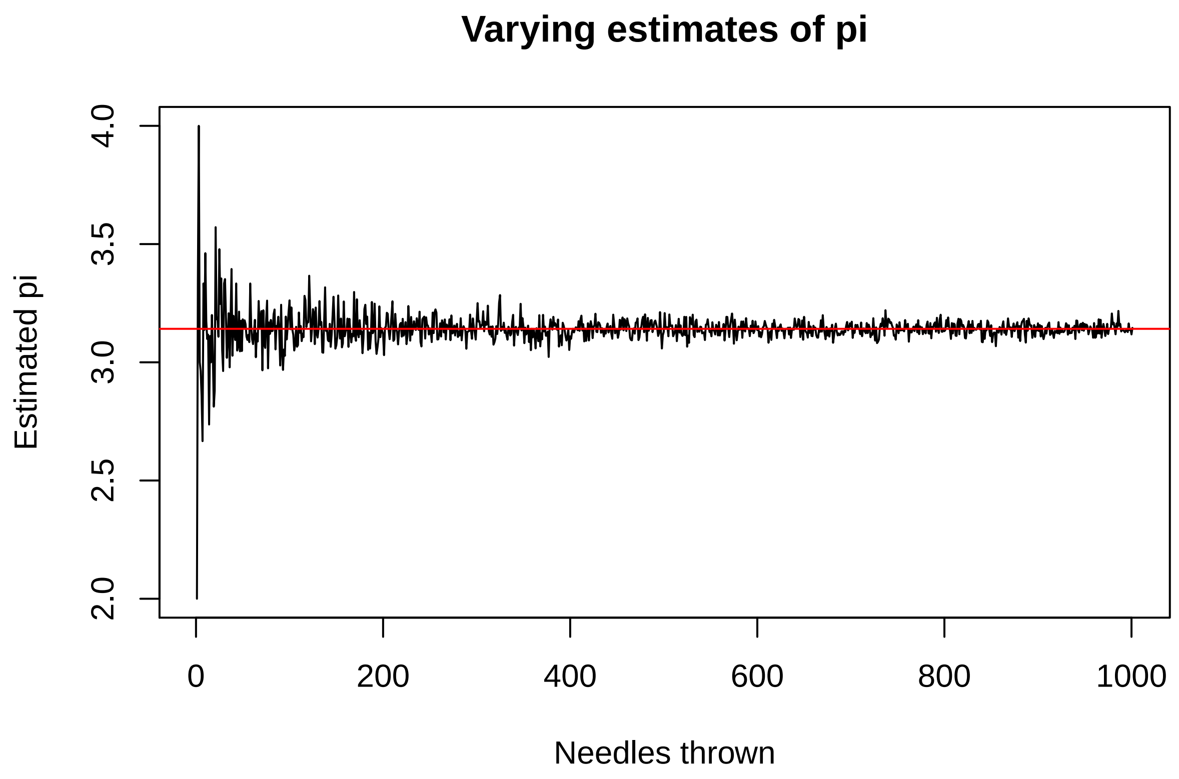
```r
throw(n=10000, PI = TRUE)
```

```
## [1] 3.123048
```

Part 3

We now observe the same experiment, but with N ranging from 1 to 10000.

```r
seql <- c(1,seq(from=10,to=10000, by = 10))
PIests <- apply(as.array(seql), 1, throw, l = 1, d = 1, PI = TRUE)
plot(PIests, type = "l", main = "Varying estimates of pi",
     ylab = "Estimated pi", xlab = "Needles thrown")
abline(a=pi,b=0, col = "red")
```

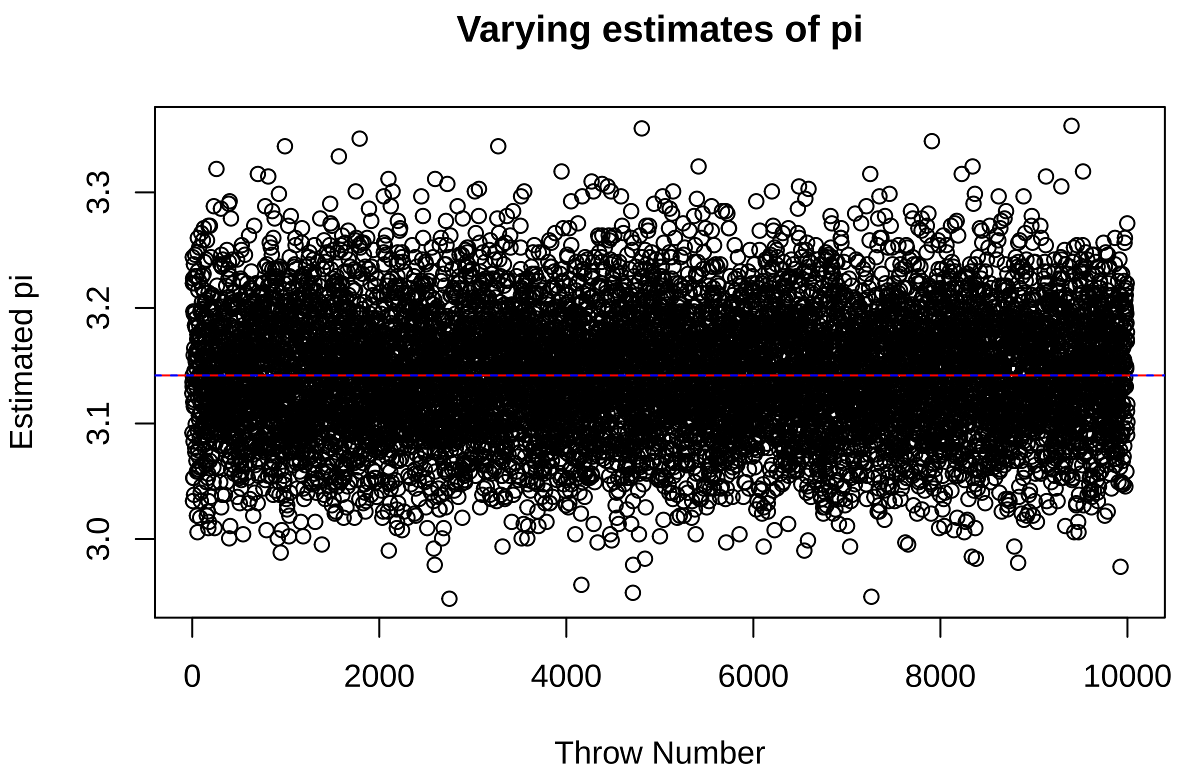


It is clear to see from the plotted estimates, that the estimates approach pi, as N increases, however the precision is typically low. Observing single values, 0, have six digit accuracy.

Part 4

We first recreate Lazzarini's experiment 10000 times,

```r
PIests <- replicate(10000, throw(l = 2.5, d = 3, n = 3048, PI = TRUE))
plot(PIests, type = "p", main = "Varying estimates of pi",
     ylab = "Estimated pi", xlab = "Throw Number")
abline(a=pi,b=0, col = "red")
abline(a=3.141592,b=0, col = "blue", lty = 2)
abline(a=3.141593,b=0, col = "blue", lty = 2)
```



Here, the dashed blue lines indicate the 6 digit precision area. Numeric inspection shows that 0 estimates fell in that area. This leads on to the discussion of Lazzarinis experiments. It is clear that the probability of achieving his results is very low, and as such one should naturally question his "optimal" outcome, however, one cannot deny the possibility of it. Assuming then that Lazzarini simply had a stroke of luck, I would still say that his experiment is flawed, as it was constructed with an intended outcome in mind. That is, both distance between lines and length of needles, combined with the odd choice of N, stronlgy indicate that he had Tsu Ch'ung-chih's result in mind when he constructed his experiment, actively allowing his result to occur, which inherently counteracts the intended randomness of the method. The fact that any other outcome would at its highest have 2 digits accuracy, also shows that the design is in no way a reliable method for estimating pi for multiple digits.

TASK 3

The function for task 1, can be found as `chi.probability` in the package `ST522examn`. For task 2, the function can be found under the name `chi.gof` in the same package.

Part 3

The `chi.gof` function implements syntax processing for defensive programming. The function first requires that x is a numeric vector, with non-zero length, and non-negative entries, of which at least one must be positive. This ensures that the recieved data does not result in zero-divisions, and that the data makes sense, seeing as x should be counting outcomes of a variable, and we cannot have a negative amount of outcomes, nor can say anything about outcomes if we have none. When x is deemed suitable, the function then evaluates p. If p is missing, it is given a uniform distribution, which is safe, as x is of non-zero length. Since p must contain probabilities, we check if all entries are non-negative and sum to one. If so the function is deemed safe to run, with the final check of whether a legal method has been chosen, that is, *“Monte Carlo”* or *“base”*, with their function described in the help file.

Part 4

We first load the data

```
gumdata  <-
  read.table("https://raw.githubusercontent.com/haghish/ST516/master/data/gum.txt")
set.seed(2016)
```

We now assume for our null-hypothesis, that the data is uniformly distributed. Applying `chi.gof`,

```
pval <- chi.gof(gumdata$Number)
print(pval)
```

```
## [1] 0
```

We recieve a miniscule p-value, which strongly indicates that we should reject our null-hypothesis. We can as such say that the cards are unlikely to be uniformly distributed.

Part 5 and 7

To compare functions for the Chi-Square distribution, we use the `method` setting, to implement `pchisq`.

```
npval <- chi.gof(gumdata$Number, method = "base")
cat("R base p-value: ", npval, "\nDifference in values: ", abs(npval - pval))
```

```
## R base p-value:  1.257277e-08
## Difference in values:  1.257277e-08
```

Since the null-hypothesis is so strongly rejected, the difference here is miniscule. Let us instead observe

```
pval <- chi.gof(gumdata$Number, gumdata$expected)
npval <- chi.gof(gumdata$Number, gumdata$expected, method = "base")
cat("R base p-value: ", npval, "\nMonte Carlo p-value", pval,"\nDifference in values: ",
    abs(npval - pval))
```

```
## R base p-value:  0.1399425
## Monte Carlo p-value 0.145
## Difference in values:  0.00505747
```

Here we see a much more noticable difference. This difference stems from the inherent randomness of the Monte Carlo method, as it only provided an estimate of the cdf, where `pchisq` provides a much more accurate result. Since we just evaluated the results for task 7, it seems relevant to discuss here. These p-values are above the standard of 0.05 (5% significance), which indicates that we should not reject the null-hypothesis, that the cards are distributed as described in the data-set. It is as such quite possible that the cards are distributed as so, as there is no strong evidence against it.

Part 6

The p-value itself indicates the probability of the deviation achieved from the expectation occurs, under the null-hypothesis. Simple put, it explains how likely it is that we got the observed amount of each card, instead of the expected. This difference is measured in the T statistic, which also transforms the result into something, which for large values of n is distributed like a Chi Square distribution, with degrees of freedom equal to the number of observations minus one. This fact is what allows us to use the Chi Square distribution for estimating the p-value, and it is why we use the specific formula of the T statistic.

TASK 4

Part 1

For the function see `linreg` in the package `ST522exman`.

Part 2

We now want to generate three variables from the standard normal distribution, with a given correlation. To do this we use the package MASS,

```
if(!require("MASS")){
  install.packages("MASS", repos="http://cran.rstudio.com/");library(MASS)}
```

```
## Loading required package: MASS
```

```
set.seed(2016)
```

which gives us access to `mvrnorm()`, which allows us to generate multiple correlated normally distributed variables, using a covariance matrix. Since the standard deviation happens to be 1 for each variable, the correlation happens to equal the covariance. We can as such generate

```
CorMat <- matrix(c(1.0,0.7,0.3,
                   0.7,1.0,0.0,
                   0.3,0.0,1.0), nrow = 3, ncol = 3)
SNVars <- mvrnorm(n = 100, mu = c(0,0,0), Sigma = CorMat, empirical = TRUE)
cor(SNVars)
```

```
##      [,1]         [,2]         [,3]
## [1,]  1.0 7.00000e-01 3.00000e-01
## [2,]  0.7 1.00000e+00 3.43631e-16
## [3,]  0.3 3.43631e-16 1.00000e+00
```

We now apply `linreg`.

```
linreg(SNVars[,1] ~ SNVars[,2] + SNVars[,3])
```

```
##
## Coefficients:
##                   Estimate Std. Error      t-value     Pr(>|t|)
## (intercept) 4.199419e-17 0.06547212 6.414057e-16 1.000000e+00
## SNVars[, 2] 7.000000e-01 0.06580195 1.063798e+01 5.652712e-18
## SNVars[, 3] 3.000000e-01 0.06580195 4.559135e+00 1.499737e-05
##
## Residuals:
## Residual standard error:  0.6480741  on  97  degrees of freedom
## Multiple R-squared:   0.58 , Adjusted R-squared:   0.5713402
## F-Statistic:  66.97619  on  2  and  97 degrees of freedom,  p-value: 0
```

We do indeed observe that the coefficients are equal to the correlations. This can be explained, as the correlation between two variables explains the strength of a linear relationship between the variables. In this case, the variables are identically distributed, and have the same range, which means that the coefficients, which describe the best linear relationship between the variables, fall in the same range as the correlation. However, this is not always the case. One simple counterexample is that the correlation ranges from -1 to 1, whereas the model coefficient ranges over the entire real line.

Part 3

We start by getting the data

```r
heightdata  <-
  read.table("https://raw.githubusercontent.com/haghish/ST516/master/data/height.txt")
set.seed(2016)
```

We now wish to explore the relation between variables, in order to predict the height of a person. We propose the null-hypothesis, that an individuals height is not predictable by their parents', with the alternate hypothesis that at least one of the parents heights can predict the height of an individual. Applying `linreg` we get

```r
attach(heightdata)
linreg(Height ~ Father + Mother)
```

```
##
## Coefficients:
##               Estimate Std. Error  t-value     Pr(>|t|)
## (intercept) 22.3097055 4.30689678 5.179995 2.742177e-07
## Father       0.3798970 0.04589120 8.278209 4.520223e-16
## Mother       0.2832145 0.04913817 5.763635 1.132338e-08
##
## Residuals:
## Residual standard error:  3.382216  on  895  degrees of freedom
## Multiple R-squared:   0.1088952 , Adjusted R-squared:   0.1069039
## F-Statistic:  54.6856  on  2  and  895 degrees of freedom,  p-value: 0
```

The significant p-value for the F statistic indicates that we should reject the null-hypothesis, as there is strong evidence against it. In fact, from the t-values and their respective p-values, we see that there is strong evidence that both the height of the father's and the mother's heights can be used to predict an individual's height. We see that both these coefficients have small standard deviations, which means that we can make a reasonable assumption, that the coefficents can be used to somewhat accurately predict the height of the individual. For a height measured in inches, an approximate standard deviation of 3.4 is rather small. However, the somewhat low r square values indicate that there is a lot of devitaions unaccounted for. Still, one can conlclude that the parents' heights are usefull for predicting the height of an individual.

TASK 5

The function `dense.estimator` in `ST522examn`, is the function requested in part 1, and `dense.plot` the one from part 2. ## Part 3 We wish to examine the eruptions of olf faithfull. We apply

```
dense.estimator(faithful$eruptions)
```

```
## Bandwith = 0.35
##                   y
## Min.:    0.23634454
## 1st Qu.: 0.15231092
## Median:  0.08403361
## Mean:    0.13655462
## 3rd Qu.: 0.51995798
## Max.:    0.12605042
```

and

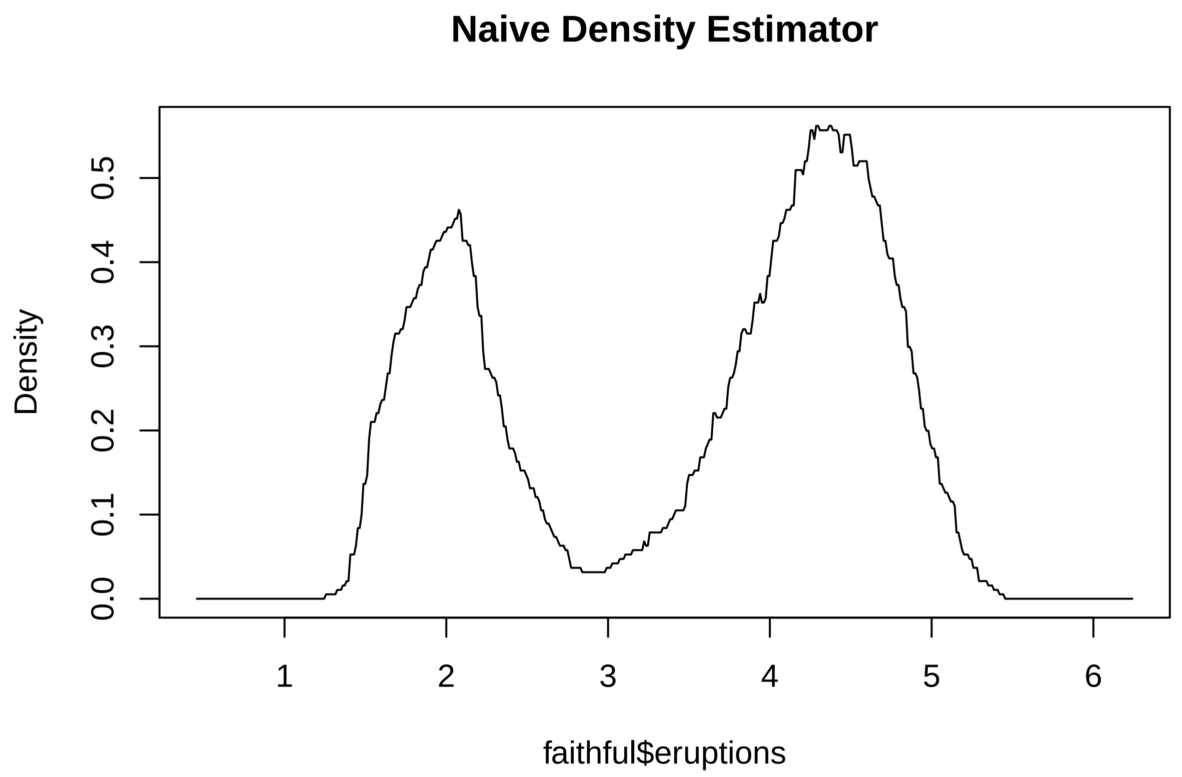
```
dense.estimator(faithful$eruptions, method = "kernel")
```

```
## Bandwith = 0.3719745
##                  y
## Min.:    0.2104631
## 1st Qu.: 0.1839732
## Median:  0.1213168
## Mean:    0.1624644
## 3rd Qu.: 0.4485952
## Max.:    0.1677039
```

We now wish to generate the estimated density plots

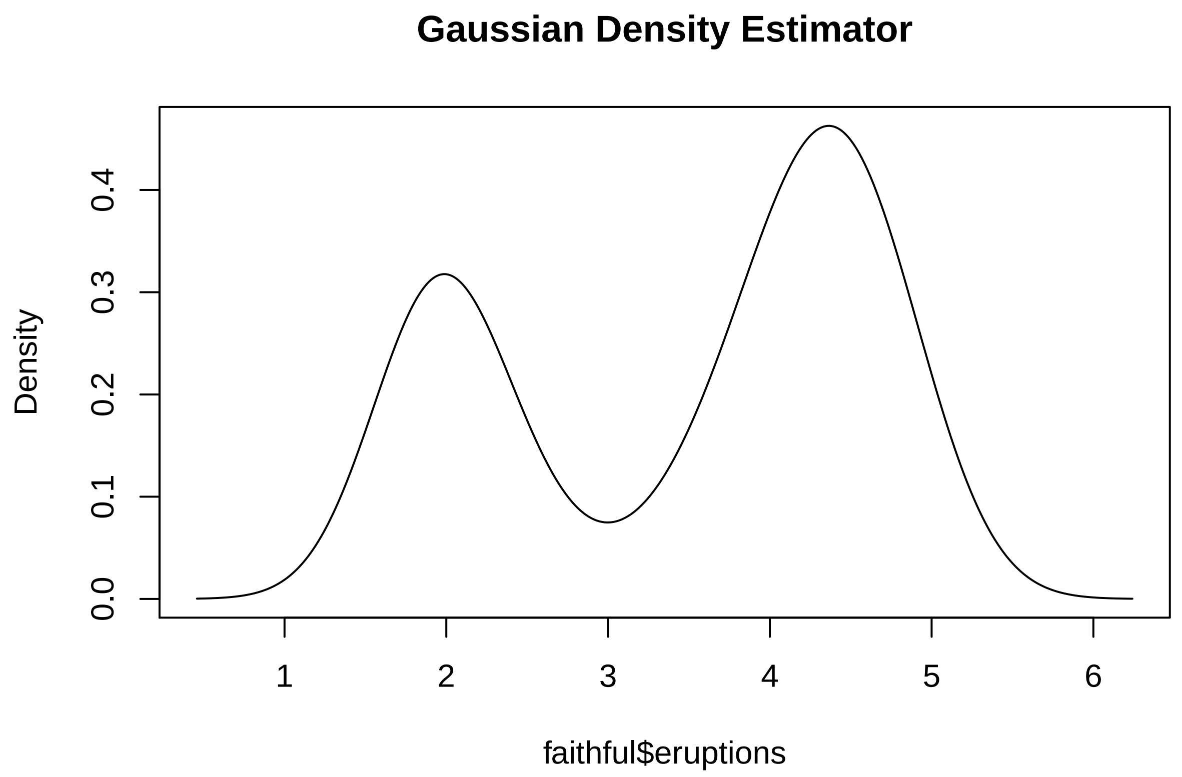
```
dense.plot(faithful$eruptions, main = "Naive Density Estimator")
```

```
## Bandwith = 0.35
```



```
dense.plot(faithful$eruptions, method = "kernel", main = "Gaussian Density Estimator")
```

```
## Bandwith = 0.3719745
```



The two methods utilize different methods for computing the bandwidth of the estimators, namely Sturges' and Silverman's methods. One notable difference between the methods, is that Sturges' was designed to estimate an amount of bins in a histogram, whereas Silverman's was intended for kernel estimation. Both of the methods share the same flaw, that they assume the distribution to be normally distributed. Sturges condsiders a binomial distribution to determine an amount of bins, which works for large sample sizes, but is a weak method for smaller samples. I consider Silverman's rule of thumb more accurate, as it instead is an approximation of which bandwith will minimize the mean square error, assuming that the underlying distribution is normal. The rule has different variations for different kernels, where the gaussian kernel happens to be simplest. The fact that Silverman's method bases itself on an optimal solution, where Sturges' merely approaches one, is why I consider it optimal.

TASK 6

The relevant function can be found as `mcstation` in `ST522examn`. We define the transistion matrix and apply `mcstation`

```
tmat <- matrix(c(.2,.7,0,0,0,.1,
                 .3,0,.7,0,0,0,
                 0,.5,.5,0,0,0,
                 0,0,0,.9,.1,0,
                 0,0,0,.25,.5,.25,
                 .4,0,0,0,.4,.2), nrow = 6)
mcstation(tmat, 3, 10000)
```

```
##                               1      2      3     4      5      6
## Stationary Probability 0.0772 0.0304 0.0128 0.257 0.2322 0.3904
```

TASK 7

To generate a `Beta(2.5,5.5)` distribution, we do as so

```
n <- 1000 # Determine iterations
chain <- numeric(1000) # Initiate the chain
reject <- 0 # We start at no rejections
chain[1] <- runif(1) # Choose a random starting point
for(i in 2:n){ # Repeat for each iteration
  proposed <- runif(1) # Propose a new value
  if(runif(1) < dbeta(proposed,2.5,5.5) / dbeta(chain[i-1], 2.5,5.5)){
    chain[i] <- proposed # Keep proposed if accepted
  } else{
    chain[i] <- chain[i-1] # Keep old value if proposed rejected
    reject <- reject + 1 # Count rejections
  }
}
```

While most steps are simple and explained in the comments, one is not. In the **if** statement, we determine whether or not to reject the proposed value. We do this based on determining the likelyhood of moving from the previous point to the new one, and then use a uniform variable to determine if we do so. We observe a rejection rate of 0.504. To visualize the chain, we observe entries 500-515,

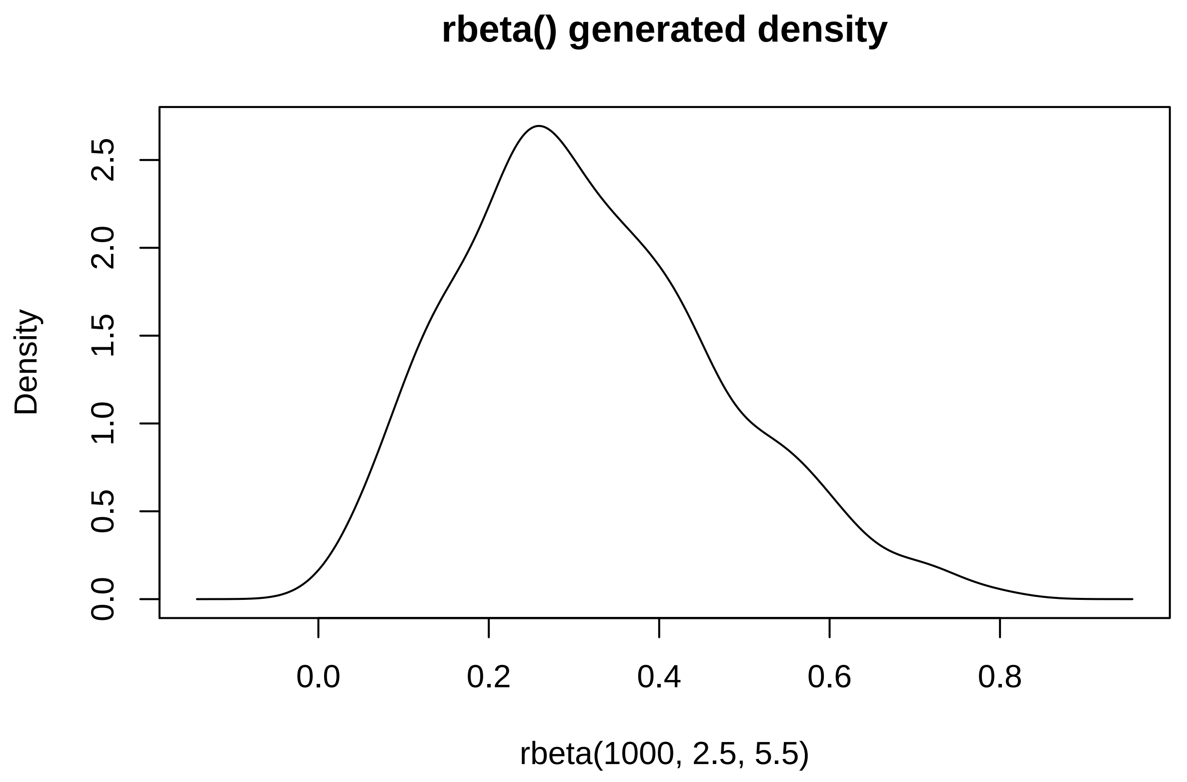
```
print(chain[500:515])
```

```
##  [1] 0.24236811 0.23628058 0.23628058 0.23628058 0.18009721 0.18009721
##  [7] 0.18009721 0.18009721 0.18009721 0.18009721 0.18009721 0.03407199
## [13] 0.10210495 0.51907710 0.51907710 0.09574780
```

We see the effects of the high rejection rate, as the variables often stays the same. Finally, we compare our method to the base beta generator,

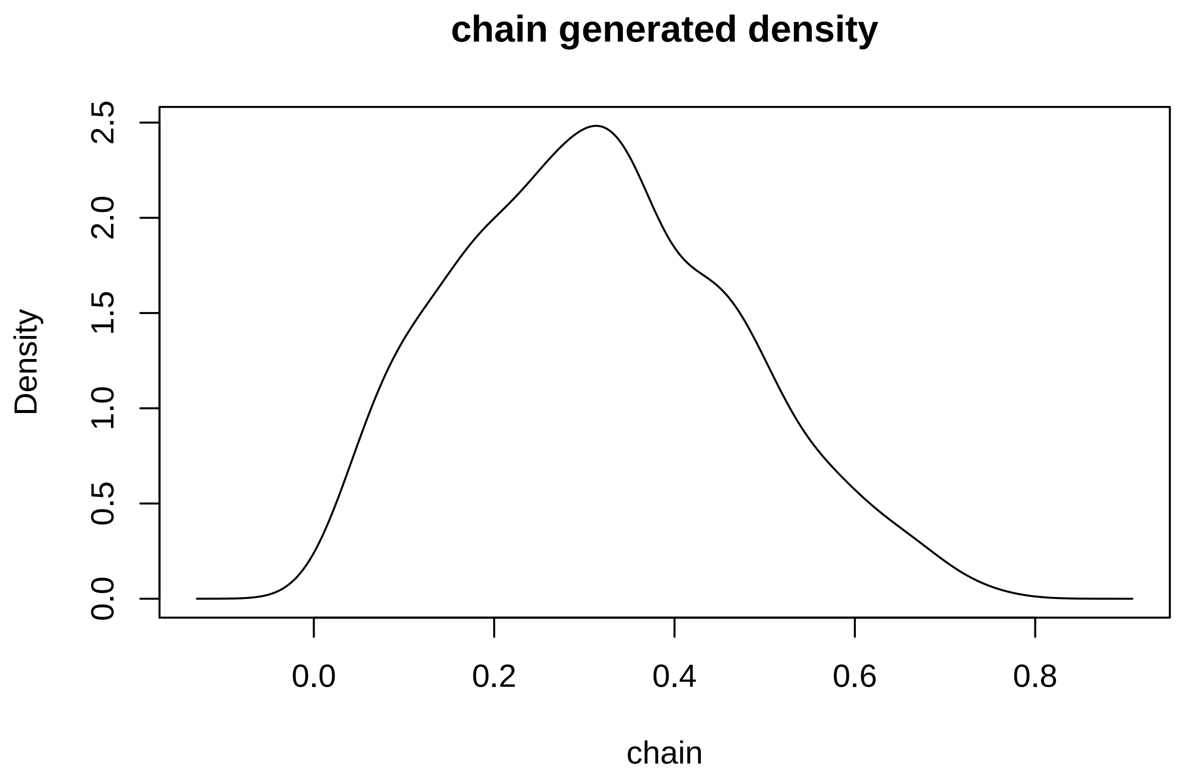
```
dense.plot(rbeta(1000, 2.5,5.5), method = "kernel", main = "rbeta() generated density")
```

```
## Bandwith = 0.03839531
```



```
dense.plot(chain, method = "kernel", main = "chain generated density")
```

```
## Bandwith = 0.03852852
```



We see that these results are indeed quite similar, as one would hope for a successful method. We should also note that one possible source of deviation, aside from the inherent randomness, is the starting point. One would typically discard the first many entries to allow the model to settle in and have a more natural start. Instead I opted to just choose a random starting point, 0.1989637, chosen from the uniform distribution.