ST516: Computational Statistics Exam (Part B)

Katrine Eriksen and Katrine Bach

10 juni 2016

```
#install.packages("devtools")
#install.packages("roxygen2")
#library(devtools)

#install_github("Katrinebch/ExamST516")
#library(ExamST516)
```

Task 4

Matrix regression:

This section is based on (Sheather 2009).

Simple linear regression is a method for fitting a straight line through a set of points such that the sum of squared residuals are as small as possible. For the straight-line regression model

$$E(Y|X = x) = \beta_0 + \beta_1 x$$

where β_0 is the intercept and β_1 is the slope. In the case where Y is predicted from one predictor variable X together with Y and X being linear in the parameters β_0 and β_1, then the multiple regression model looks like the following.

$$E(Y|X_1 = x_1, X_2 = x_2, \dots, X_p = x_p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

Thus

$$Y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_p x_{pi} + e_i$$

for which e_i is the random fluctuation on Y_i such that $E(e_i|X) = 0$.

A way to study $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}, \dots, \hat{\beta}_p$ is to use matrices, so for further notation X is a matrix $(n \times p + 1)$ and Y is $(n \times 1)$ vector. By assuming that the inverse matrix $(X'X)$ exists, then the least squares estimates can be calculated using the

$$\hat{\beta} = (X'X)^{-1} X'Y$$

where the predicted values are given by

$$\hat{Y} = X\hat{\beta}$$

together with the residuals

$$\hat{e} = Y - X\hat{\beta}$$

Which can be used to calculate the residual sum of squares

$$RSS = \sum_{i=1}^{n} \hat{e}_i^2$$

This can be used for estimating the error variance

$$S^2 = \frac{1}{n-p-1} \sum_{i=1}^{n} \hat{e}_i^2$$

which is the unbiased estimate of σ^2. This can be used for generating the standard error

$$se(\beta) = \sqrt{S^2(X'X)^{-1}}$$

If it is assumed that the errors are normally distributed with constant variance then the *T-value* is

$$T_i = \frac{\hat{\beta}_i - \beta_i}{se(\hat{\beta})}$$

where $se(\hat{\beta})$ is the estimated standard deviation of $\hat{\beta}_i$. The degrees of freedom satisfies the following formula

$$df = \text{sample size-Number of mean parameters estimated}$$

An analysis of the variance approach can be used for testing whether there is a linear association between Y and a subset/all of the predictors.
For this the total corrected sum of squares of the Y's is

$$STT = \sum_{i}^{n}(y_i - \bar{y})^2$$

The regression sum of squares

$$SSreg = \sum_{i}^{n}(\hat{y}_i - \bar{y})^2$$

Both *SSreg* and *RSS* can be used to make the F statistics

$$F = \frac{SSreg/p}{RSS/(n-p-1)}$$

The *RSS* together with *SST* can be used for calculating R^2, which is the coefficient of determination of the regression.

$$R^2 = \frac{RSS}{SST}$$

This value is often increased when irrelevant predictor variables is added to the regression equation. To compensate for this increasment the adjusted coefficient of determination is defined.

$$R^2_{adj} = 1 - \frac{RSS/(n-p-1)}{SST/(n-1)}$$

Cholesky Decomposition:

For one of the task we want to generate variables dependent on their correlation. To do this we make use of Cholesky deposition. For using this the correlation matrix C must be created.

$$C = \begin{bmatrix} 1 & 0.3 & 0.7 \\ 0.3 & 1 & 0 \\ 0.7 & 0 & 1 \end{bmatrix}$$

This can be used to generate the matrix U by the Cholesky decomposition.

$$U^T U = C$$

The matrix U can then be used to generate correlated random number R_C from uncorrelated numbers R in the following way.(Sitmo 2009–2015)

$$R_C = RU$$

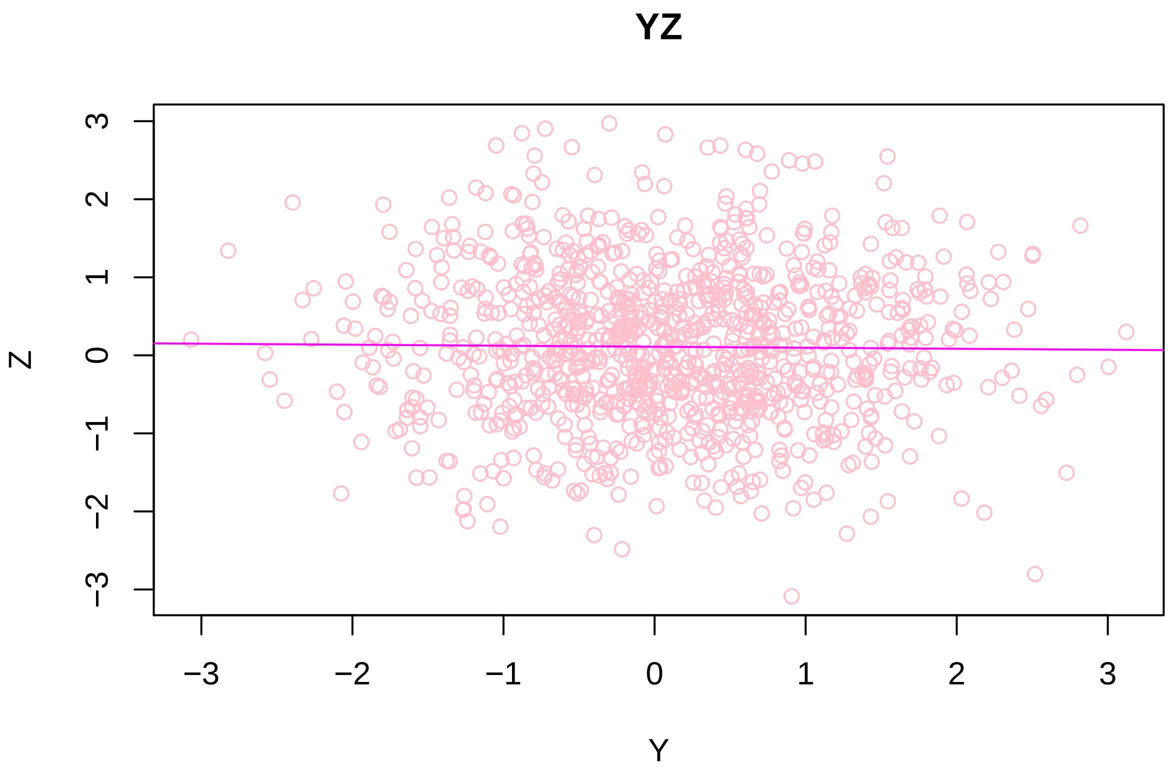
For the first subtask we want to make a function that fits a simple or multi linear regression model.

```r
MatrixRegression = function(formula){

  #Get the matrix so it can be used for evaluations
  call <- match.call()
  matri <- match.call() #Use to make matrix and so the code can be run like the lm function
  matri[[1L]] <- quote(stats::model.frame)
  matri <- eval(matri, parent.frame())

  #Dimension of matrix
  widthmatri <- dim(matri)[2] #Width=p+1
  heightmatri <- dim(matri)[1] #Height = n
  Y <- matri[,1] # Setting Y up
  X <- rep(1,heightmatri) #Start X with olumn of 1's.

  for(i in 2:widthmatri){
    X <- cbind(X,matri[,i]) # Add other variables to X.
  }

  #Calculates the parameters as defined in the theory
  Estimate = as.vector(solve(t(X)%*%X)%*%(t(X)%*%Y))#beta values
  yhat = X%*%Estimate
  ehat = as.vector(Y-X%*%Estimate)
  RSS = sum((ehat)^2)
  SST = sum((Y-mean(Y))^2)
  SSreg = sum((yhat-mean(Y))^2)
  R2=1-(RSS)/(SST)
  df= heightmatri-widthmatri#Generates the degrees of freedom

  Radj = 1- (RSS/(heightmatri-widthmatri))/(SST/(heightmatri-1))
  Ftest = (SSreg/(widthmatri-1))/(RSS/(heightmatri-widthmatri))
  pvalue = 1-pf(Ftest, heightmatri- widthmatri,widthmatri-1)
  S2 = RSS/(heightmatri-widthmatri)
  ResidualStandardError = sd(ehat)
  StandardError = sqrt(diag(S2*solve(t(X)%*%X)))
  Tvalue = Estimate/StandardError
  Pr = 1-pt(Tvalue,df)
  print(data.frame(Estimate, StandardError, Tvalue, Pr))

  #Generates the table
  cat((sprintf("Residual standard error %.4f on %i degrees of freedom \n",ResidualStandardError, df)),
      (sprintf("R-squared:%4f \n", R2 )),
      (sprintf("Adjusted R-squared: %4f\n", Radj )),
      (sprintf("F-statistics %2f with the cooresponding p value %4f\n", Ftest, pvalue )))

}#ends function
```

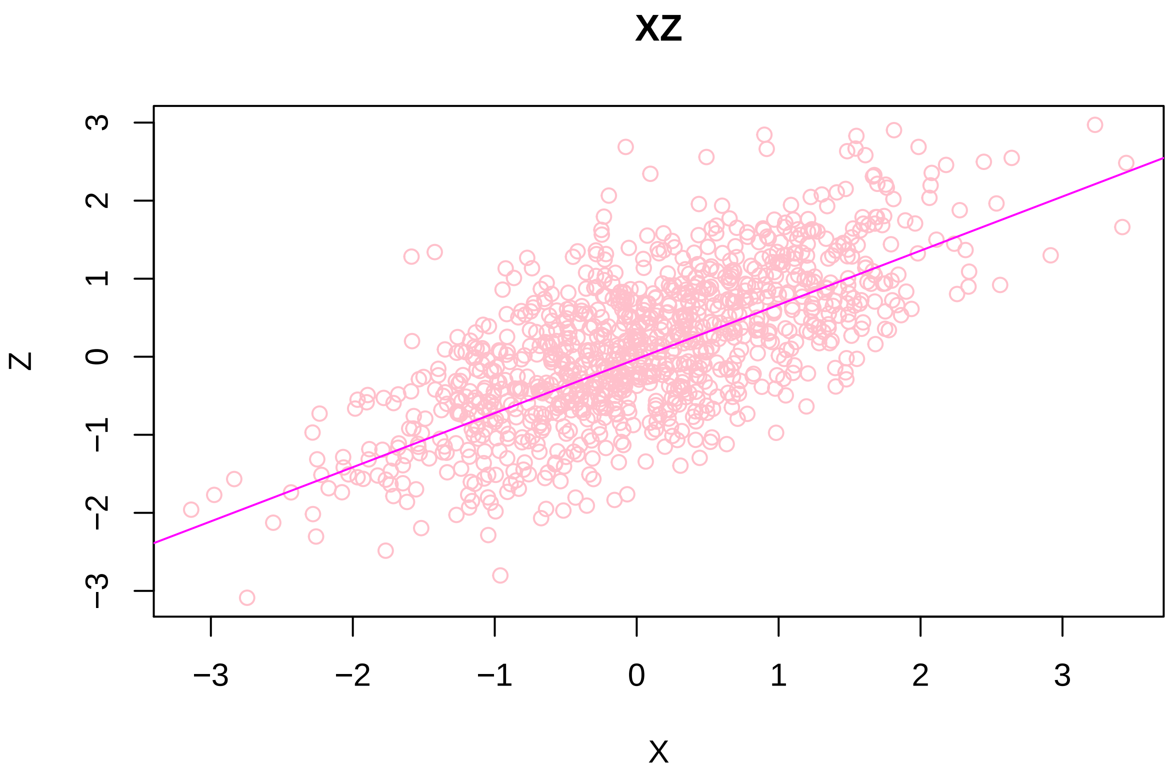
For the next subtask we want to generate 3 variables from a standard normal distribution. For this task we want to generate the variables such that the correlation between the dependent variables and the predictors are 0.7 and 0.3 and the predictors have a correlation of 0.0 with each other. These generated variables should then be visualized to show their correlation together with a regression line.

```r
R <- matrix(cbind(1, .3, .7, .3, 1, 0, .7, 0, 1), nrow = 3)#Correlation matrix
U <- t(chol(R))#Matrix generated using Cholesky
```

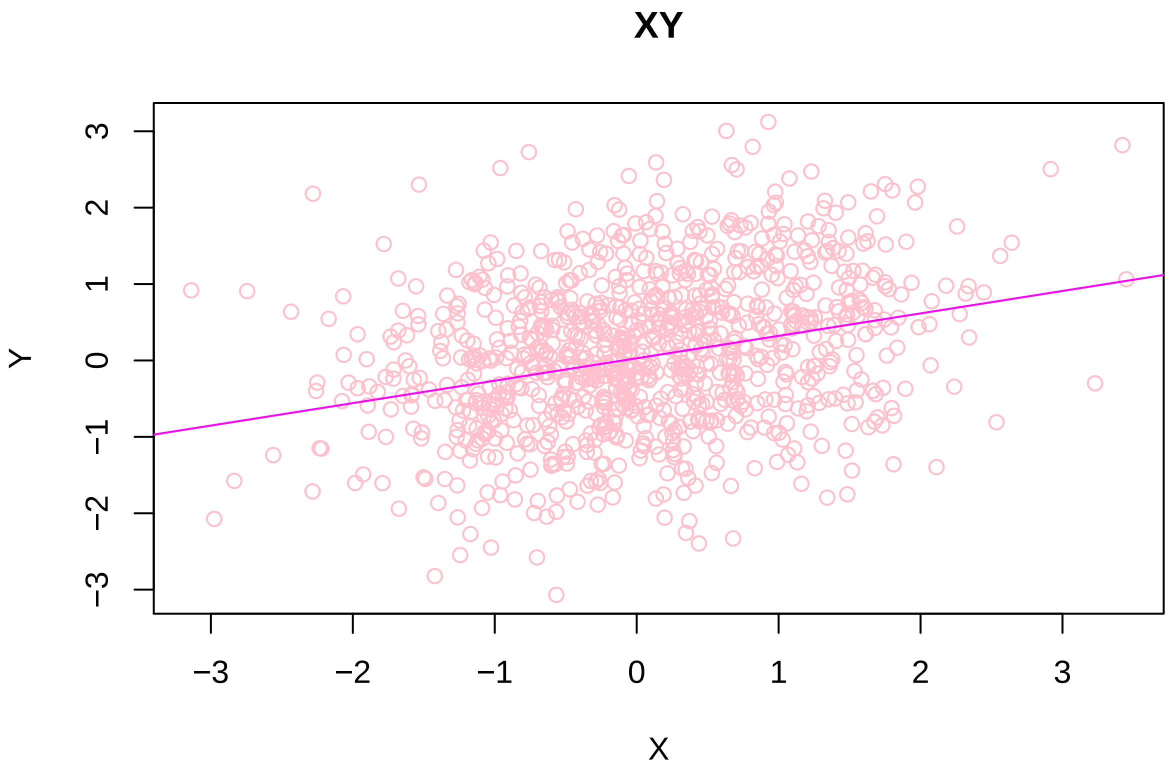
```
nvars <- dim(U)[1]
numobs <- 1000
set.seed(13)#set seed
random.normal <- matrix(rnorm(nvars*numobs,0.1), nrow = nvars, ncol=numobs)#R
X <- U%*%random.normal#Rc
newX <- t(X)

#Makes the plot
#makes the yz-correlation
plot(newX[,2],newX[,3], main = "YZ", xlab ="Y", ylab = "Z", col="pink")
abline(lm(newX[,2]~newX[,3]),col="magenta")
```



```
#makes the xz-correlation
plot(newX[,1],newX[,3], main = "XZ", xlab ="X", ylab = "Z", col="pink")
abline(lm(newX[,1]~newX[,3]),col="magenta")
```



```
#makes the xy-correlation
plot(newX[,1],newX[,2], main = "XY", xlab ="X", ylab = "Y", col="pink")
abline(lm(newX[,1]~newX[,2]), col="magenta")
```



```r
MatrixRegression(newX[,1]~newX[,2]+newX[,3])
```

```
##      Estimate StandardError    Tvalue        Pr
## 1 -0.06152339    0.02110978 -2.914449 0.9981786
## 2  0.30243815    0.02112629 14.315723 0.0000000
## 3  0.69752033    0.02138878 32.611512 0.0000000
## Residual standard error 0.6571 on 997 degrees of freedom
##  R-squared:0.557623
##  Adjusted R-squared: 0.556736
##  F-statistics 628.366697 with the cooresponding p value 0.001590
```

It can be seen that the estimates and the correlation are close being equal. This can be explained by the fact that the best fit is understood to be the fit that minimizes the sum of least squares. It can be shown that the β value that minimizes the problem are

$$\begin{aligned}\beta &= \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2} \\ &= \frac{\sum_{i=1}^{n}x_iy_i - 1/n\sum_{i=1}^{n}x_i\sum_{j=1}^{n}y_j}{\sum_{i=1}^{n}x_i^2 - 1/n(\sum_{i=1}^{n}x_i)^2} \\ &= \frac{\bar{XY}-\bar{X}\bar{Y}}{\bar{X^2}-\bar{X}^2} \\ &= \frac{Cov(X,Y)}{Var(X)}\end{aligned}$$

The definition of the correlation is the following

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$$

For the case where

$$Var(X) = Var(Y)$$

the correlation is

$$Corr(X,Y) = \frac{Cov(X,Y)}{Var(X)}$$

Hence it can be seen that for the case where $Var(X) = Var(Y)$ then the correlation and the estimates is equal. This must apply to our case. However this relation between the estimates and the correlation is not always true, only when the variance of X is equal to the variance of Y. (n.d.)

In the last subtask we want to explore the relation between some of our variables in the text file height. We have chosen first to look at the relation between the height and the gender. We have chosen the significans niveau to be 0.05 with corresponding null hypothesis. This null hypothesis can be rejected if the p value is below the significance level. The rejection of a null hypothesis is considered as a strong conclusion in contrast to the acceptance of a hypothesis which is a weak conclusion because we do not know what the probability is of not rejecting the null hypothesis when it should be rejected(n.d.).

We will now make use of the F-test with the following null-hypothesis.

H_0 : There is no relation between the height and gender.

H_1 : There is a relation between the height and gender.

```r
URL <- "https://raw.githubusercontent.com/haghish/ST516/master/data/height.txt"
data1 <- read.table(URL, header = TRUE)
MatrixRegression(data1$Height~data1$Gender)
```

```
##    Estimate StandardError    Tvalue Pr
## X 64.110162     0.1205759 531.69973  0
##    5.118656     0.1675607  30.54807  0
## Residual standard error 2.5076 on 896 degrees of freedom
##  R-squared:0.510164
##  Adjusted R-squared: 0.509618
##  F-statistics 933.184603 with the cooresponding p value 0.026107
```

Here we can see that with the F-test we get a p-value that is smaller than our chosen significans niveau. We therefore reject the H_0 hypothesis, which implies that there is a relation between the height and gender.

We also want to explore the influence of the variables on the model, which can be done by the use of the T-test. If the Pr value corresponding to the T-value calculated for the variable is lower than the null-hypothesis, then the hypothesis is rejected. We use the same significans niveau as previous.

H_0 : The variables has no influence on the model

H_1 : The variables has an influence on the model

```r
MatrixRegression(data1$Height~data1$SES)
```

```
##      Estimate StandardError      Tvalue        Pr
## X 66.64716118    0.24959050 267.0260364 0.0000000
##    0.02166828    0.04181041   0.5182508 0.3022056
## Residual standard error 3.5824 on 896 degrees of freedom
##  R-squared:0.000300
##  Adjusted R-squared: -0.000816
##  F-statistics 0.268584 with the cooresponding p value 0.946024
```

Based on the Pr value of *SES* the null hypoteses can not be rejected hence the *SES* variable has no influence on the model.

We now want to explore the influence of the variabeles *SES* and *Gender* in the model using the T-test using the null-hypothesis.

H_0 : The variables has no influence on the model

H_1 : The variables has an influence on the model

We will use the same significance level as before.

```r
MatrixRegression(data1$Height~data1$SES+data1$Gender)
```

```
##      Estimate StandardError      Tvalue        Pr
## 1 63.98800167    0.19523320 327.7516415 0.0000000
## 2  0.02329162    0.02927281   0.7956743 0.2132162
## 3  5.11889766    0.16759528  30.5432095 0.0000000
## Residual standard error 2.5067 on 895 degrees of freedom
##  R-squared:0.510511
##  Adjusted R-squared: 0.509417
##  F-statistics 466.717786 with the cooresponding p value 0.002140
```

Based on the Pr value of *SES* the null hypothesis cannot be rejected, hence the *SES* has no influence on the model. On the other hand for the *Gender* variable the Pr value is below the significance level, hence the null hypothesis is rejected. Therefore the *Gender* varible has an influence on the model.

Task 5

Density:

Density estimation is a collection of methods for contructing an estimate af probability density, as a function of an observed sample of data. A histogram is a type of density estimator as well as kernel density estimation.

Kernel Density Estimation

The Kernel density estimation is an appraoch that is rooted in the histogram methodology. The basic idea is to estimate the density function at a point x using neighboring observations. To do this we can use the naive density estimation or the gaussian kernel estimation.

Naive density estimation

This section is based on (Haghish 2016a).

Consider a histogram contructed from a sample $X_1, \dots, X_n$ with bin width h. From this, the density estimate for a point x within the dataset range, can be viewed to be

$$\hat{f}(x) = \frac{1}{2hn} \cdot k$$

for which k is the number of sample point that lies within the interval $(x-h, x+h)$. This estimate can be rewritten into the following form.

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x - X_i}{h}\right)$$

For which the kernel density function $K(t) = 1/2I(|t| < 1)$ is used as a weight function. This function is referred to as the *naive density estimator*. Hence from above it can be seen that the naive density estimator uses each point of estimation x as the center and a bin of width $2h$.

For this density estimator the Struges optimal width can be used

$$h = \frac{R}{1 + log_2 n}$$

where R is the range of the sample, and n is the sample size.

Gaussian kernel density estimation

This section is based on (Haghish 2016a).

We consider the standard normal density function as the weight function

$$K = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

which is called the kernel weight. The kernel weight together with the kernel estimate creates the Gaussian kernel

$$\hat{f}_K(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{x - X_i}{h}\right)$$

This can be used with the Silverman width to make the Gaussian kernel, which is used in this project.

$$h = 0.9\min(sd, IQR/1.34)n^{-1/5}$$

In the first subtask we have to write a function that estimates the density using the naive estimator or the gaussian kernel.

```
densi <- function(x,d=NULL,h=NULL, method="naive"){
  #warnings/stops
  if (!is.numeric(x)){stop("x must be numeric")}
  if (length(x)<1){stop("The length of the x cannot be negative")}
  if (length(x)==0){warning("The input must exist")}

  if (method=="naive"){
  }else if (method=="kernel"){
  }else {stop("The method does not exist for this program")}


  n<-length(x)

  #Generates h
  if(length(h)==0){#Generates h if there is no input value
    if (method=="kernel"){#Generates h according to Silverman.
      h <- 0.9*min(c(IQR(x)/1.34,sd(x)))*n^(-1/5)
    }else{
      h<- (max(x)-min(x))/(1+log2(n))#Generates h according to Struges.
    }
  }else{

    h <- h
  }

#Generates d
  if(length(d)==0){# Generates d to be the minimum, maksimum, mean and all the quantiles.
    minimum<-densi(x,min(x),h,method)
    maksimum<-densi(x,max(x),h, method)

    quantile1<-quantile(x,names=FALSE)[2]
    q1<-densi(x,quantile1,h,method)

    quantile3<-quantile(x,names=FALSE)[4]
    q3<-densi(x,quantile3,h,method)

    medians <- densi(x,median(x),h,method)
    means <- densi(x,mean(x),h,method)
  #Creates the table
    smoke <- matrix(c(min(x),minimum,quantile1,q1,median(x),medians,mean(x),means,quantile3,q3,max(x),ma
    colnames(smoke) <- c("x","y")
    rownames(smoke) <- c("Min","1st Quantile","Median", "Mean", "3dr Quantile", "Max")
    smoke <- as.table(smoke)
    print(h)
    return(smoke)

  }else{#If the input argument is not NULL then the d value is used.
    d=d
  }

  #Implementation of Gaussian kernel according to theory.
  if(method=="kernel"){
    kern<-0
    for (i in 1:n) {
      kern[i] <-(1/h)*dnorm((d-x[i])/h)
    }
    summ<- sum(kern)
    f<-(1/n)*summ
    return(f)
  }else{# Implementation of naive method according to theory.
    nav <- 0
    for(i in 1:n){
      if(abs((d-x[i])/h)<1){
        w <- 1/2
      }else{
        w <- 0
      }
      nav[i] <-(1/h)*w
    }
    summ <- sum(nav)
    f <- (1/n)*summ
    return(f)
  }

}
```

In the second subtask we write a function that calls our function densi, made in the previous subtask, and creates a density plot based on the specified method.

```
densiplot <- function(x,n=500, method="naive", from= min(x)-(sd(x)/3), to=max(x)+(sd(x)/3)){
  #warnings/stops
  if (!is.numeric(x)){stop("x must be numeric")}
  if (length(x)<1){stop("The length of the x cannot be negative")}
  if (length(x)==0){warning("The input must exist")}

  if (!is.numeric(n)){stop("n must be numeric")}
  if (n<1){stop("The number of points to be plottet must be positive")}

  if (method=="naive"){
  }else if (method=="kernel"){
  }else {stop("The method does not exist for this program")}

  if (!is.numeric(from)){stop("The point must be numeric")}
  if (!is.numeric(to)){stop("The point must be numeric")}
  #Plots the density function
  happy<-c()
  for (d in seq(from,to,length.out = n)) {
    happy1<-densi(x,d=d,method=method)
    happy<-c(happy,happy1)
  }
  plot(seq(from,to,length.out = n),happy, type = "l")
}
```

In subtask 3 we use our *R* function *densi* with the *Gaussian kernel* method on the *faithfull*-data without any argument for *d* and *h*.

```
densi(faithful$eruptions, method="kernel")
```

```
## [1] 0.334777
```

```
##                      x         y
## Min          1.6000000 0.2132889
## 1st Quantile 2.1627500 0.3106146
## Median       4.0000000 0.3850462
## Mean         3.4877831 0.1547187
## 3dr Quantile 4.4542500 0.4782279
## Max          5.1000000 0.1577281
```

For a drawn X, these outputs gives the probability of X being equal to d. Now we run it again using the naive density estimation and again without specifying *d* and *h*.

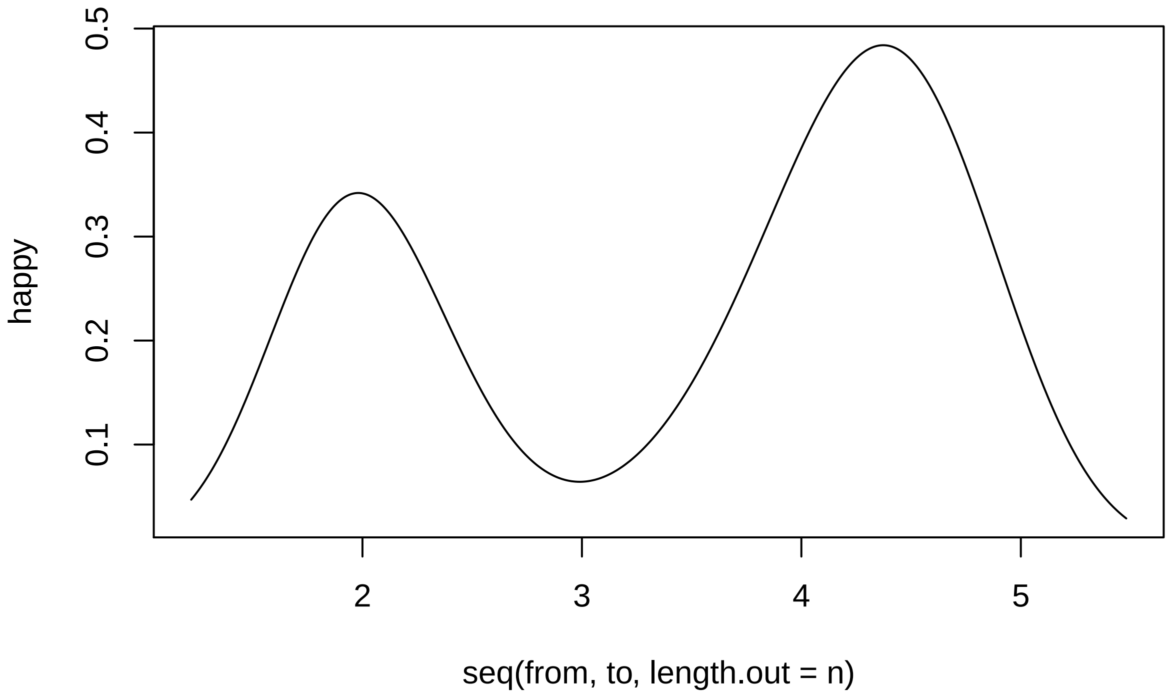
```
densi(faithful$eruptions)
```

```
## [1] 0.385146
```

```
##                      x         y
## Min          1.6000000 0.2434142
## 1st Quantile 2.1627500 0.3913718
## Median       4.0000000 0.3913718
## Mean         3.4877831 0.1336392
## 3dr Quantile 4.4542500 0.5536479
## Max          5.1000000 0.1240935
```

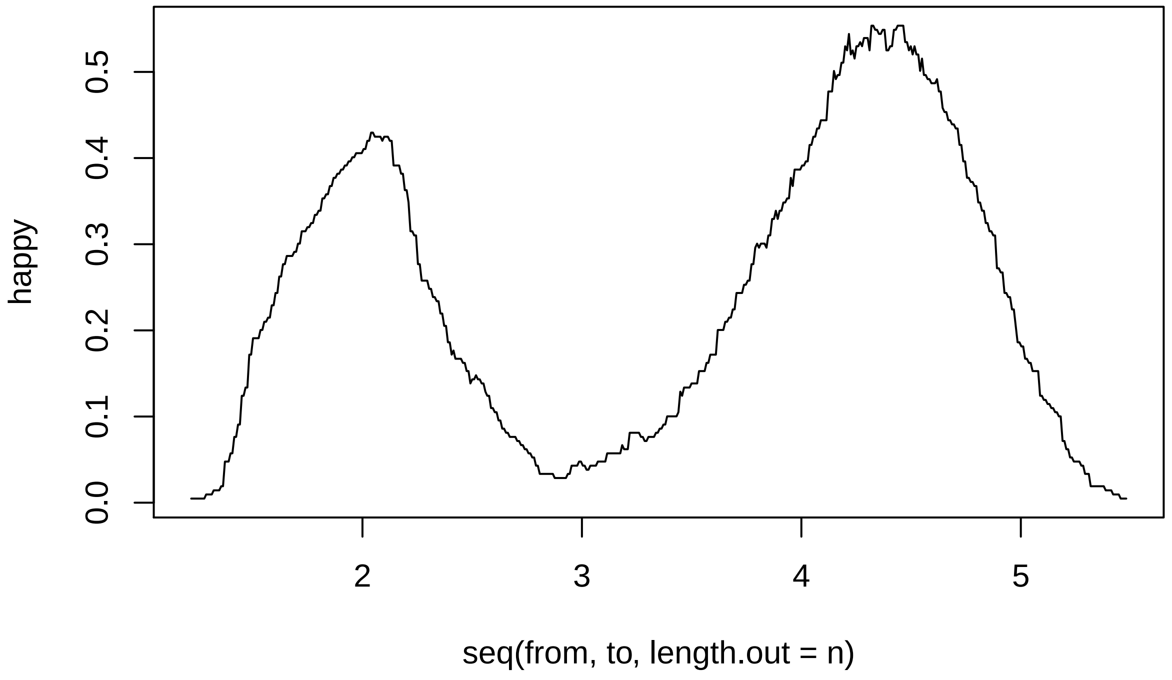
We will now show the plot for the gaussion kernel density estimation.

```
densiplot(faithful$eruptions, method="kernel")
```



Now we run *densiplot* using the naive density estimation.

```r
densiplot(faithful$eruptions)
```



In subtask 3 we have to compare the two formulas we used for calculating the bin width h, and discuss how they influence the density function.

A histogram is a piecewise constant approximation of the density function. While data is often contaminated by noise, the estimator that presents the most details is not necessarily better. The choice of a bin width for a histogram is a choice of smoothing parameter. In case of both under- and oversmoothing important features can be lost. The Sturges is based on the assumption that the sampled population is normally distributed.

From this assumption it is logical to choose a family of discrete distribution that converge in distribution to normal as the number of classes and sample size n tends to infinity. The most obvious candidate using this description is the bionomial distribution with 1/2 as the probability of success (Haghish 2016a). Because of the fact that the Sturges width makes the assumption that the sampled population is normally distributed, this method works the best when this is the case. The Sturges method also tends to oversmooth when the sample size is large, but it also makes bad bin widths when the sample size is less then 30. In the case of a small sample size the result trends to show the data poorly.

In contrast to Struges Silverman creates an appropriate h for a wide range of distributions, while this can be used for function that are not necessarily normal, unimodal or symmetric.(Haghish 2016a)

Since a large bin widths may hide patterns in the data and small bin widths may yield graphs that are difficult to interpret for both methods, it is difficult to see which method is the best. If we look at the bin width, we can see that with the Sturges calculation of h we have a bin width 0.3348 and with the use of the Silverman method we get 0.3851. We can therefore conclude that with the use of the Silverman we get a better bin width than with the Sturges method because the bin width is not as small as with the Sturges bin width. Furthermore it is practically that you can by Silverman use functions that does not necessarily have a normal distributions. Therefore we think that the Silverman method for finding the bin width is a better method than Sturges, but the difference is very small for our example.

Task 6

Markov chain:

This section in based on (Sheldon 2013), (Haghish 2016b) and (Sigman 2007).

Markov Chain is a random process that undergoes transitions from one state to another on a state space. A state space is the set of values which a process can take. Markov Chain is memoryless i.e the probability distribution of the next state depends only on the current state and not on the sequence of events that preceded it. Markov property defines a serial dependence only between adjacent periods.

Definition: The sequence $\{X_t | t \geq 0\}$ is a Markov Chain if for all times $t \geq 0$ and all states $i_0, \ldots, i, j \in \mathcal{S}$

$$P(X_{t+1} = j | X_0 = i_0, X_1 = i_1, ..., X_{t-1} = i_{t-1}, X_t = i) = P(X_{t+1} = j | X_t = i) = P_{ij}$$

P_{ij} denotes the probability that the chain, whenenver in state i, moves next into state j, and is referred to as a *one-step* transition probability. So the transition probability is the probability of moving from one state to another.

Property 1: Markov Chain is irreducible if all states communicate with all other states i.e given that the chain is in state i, there is a positve probability that the chain can enter state j in finite time, for all pairs of states (i, j)

Property 2: The process must be in some state after it leaves states i, the transition probability satisfy

$$\sum_{j=1}^{N} P_{ij} = 1$$

Property 3: For an irreducible Markov Chain, π_j denotes the long-run proportion of time that the process is in state j. The π_j is called stationary probability

$$\sum_{j=1}^{N} \pi_j = 1$$

Property 4: The stationary probability π_j which is *the proportion of time in which the Markov chain has just entered state j*

$$\pi_j = \sum_{i=0}^{\infty} \pi_i P_{ij}$$

To make the Markov Chain the following psedo code is used

1. Choose an inital value, $X_0 = i_0$ and set $n = 1$.
2. Generate Y_{i_0} and set $X_1 = Y_{i_0}$
3. if $n < N$, then set $i = X_n$ and generate Y_i. Set $n = n + 1$ and set $X_n = Y_i$. Otherwise stop
4. Go back to step 3

where Y_i is generated using a uniform random distributed U. If $U \leq P_{i,0}$ then $Y_i = 0$ and if $\sum_{k=0}^{j-1} P_{i,k} < U \leq \sum_{k=0}^{j} P_{i,k}$, then $Y_i = j$.

In the first subtask we want to write a function that simulates the stationary distribution of a discrete Markov Chain. The function *MarkovChain* returns a table that include the stationary probability for each state.

```
MarkovChain <- function(p,k,n){
  #Warning/stops
  if (!is.matrix(p)){stop("The transition probability matrix must be a matrix")}
  if (dim(p)[1]!=dim(p)[2]){stop("The transition probability matrix must be a square matrix")}

  if(!is.numeric(n)){stop("n must be numeric")}
  if(n==0){warning("Must be positive and larger than 0")}
  if(n<0){stop("Must be positive")}

  if(!is.numeric(k)){stop("The inital state must be numeric")}
  if(k>dim(p)[1]){stop("The initial state must exist")}
  if (k<1){stop("The inital state must be positive and larger than zero")}

  #Sets the inital values
  set.seed(12)# Set seed
  x <- 0
  Y<- 0
  i<-k
  x[1]<-i
  t <- 1
  U <- runif(1)

  #The first state
  #Generate inital Y
  for (j in 1:dim(p)[1]) {#Generates a for loop from 1 to the dimension of p
    if (U<= p[i,1]){ #If U is less than or equal to the number in the first column
      Y[i] <- 1
    }else if( sum(p[i,1:j-1])<U && U<= sum(p[i,1:j])) {#Otherwise U must be
      #between the sum of probabilities in the row until j-1 and the sum of
      #probabilities in the row until j
      Y[i]=j
    }
  }
  #Set X2 to be equal to Y
  x[2]<-Y[i]


  #The general
  # Generate Y as above
  while (t<n){
    i<-x[t]
    U<-runif(1)
    for (j in 1:dim(p)[1]) {
      if (U<= p[i,1]){
        Y[i] <- 1
      }else if( sum(p[i,1:j-1])<U && U<= sum(p[i,1:j])) {
        Y[i]=j
      }
    }
    t=t+1
    #Set X at the given time equal to Y
    x[t]<-Y[i]

  }#End while

  return(prop.table(table(as.matrix(x)))) #Returns a table

}#End function
```

At last we want to create the transition probability marix based on the diagram we have presented in figure 1. We initiate the chain at state 3 and run the simulation 10000 times to get the stationary probabilities for all the states.

```
p=matrix(c(.2,.3,0,0,0,.4, .7,0,.5,0,0,0, 0,.7,0,0,0,0, 0,0,.5,.9,.25,0, 0,0,0,.1,.5,.4, .1,0,0,0,.25,.2
print(p)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0.2  0.7  0.0 0.00  0.0 0.10
## [2,]  0.3  0.0  0.7 0.00  0.0 0.00
## [3,]  0.0  0.5  0.0 0.50  0.0 0.00
## [4,]  0.0  0.0  0.0 0.90  0.1 0.00
## [5,]  0.0  0.0  0.0 0.25  0.5 0.25
## [6,]  0.4  0.0  0.0 0.00  0.4 0.20
```

```
MarkovChain(p,3,10000)
```

```
##
##      1      2      3      4      5      6
## 0.0510 0.0544 0.0376 0.6106 0.1797 0.0667
```

In the table we can se the stationary probabilities for each state. If the total time spent once the chain reaches the stationary distribution is considered. This tells us that the smallest amount of time is spent in state 1 and the greatest amount of time is spent in state 4 using seed(12).

Task 7

Metropolis-Hastings sampler:

This section is based on (Haghish 2016b) and (Sheldon 2013).

The Markov Chain Monte Carlo (MCMC) method are a general methodological framework introduced by *Metropolis* and *Hastings*. The MCMC is a combination of Markov Chain and the Monte Carlo method. The MCMC use the Monte Carlo to construct a Markov Chain with a stationary distribution and run the chain for a sufficiently long time until the chain converges to a stationary distribution. So the MCMC method provides a sampler that generates random observations from the target distribution.

If we let $b(j), j = 1, \dots, m$ be a positive number with $B = \sum_{j=1}^{m} b(j)$ which is diffucult to calculate. We want to simulate a random variable with the probability mass function

$$\pi(j) = \frac{b(j)}{B}, \quad j = 1, \dots, m$$

A way to do this is to make use of Markov Chain. This Markov Chain can be used to simulate a sequence of random variables whose distribution converge to $\pi(j)$ by finding a Markov Chain that is easy to simulate and whose limiting probabilities are $\pi(j)$.

The Metropolis-Hastings algorithm contructs a time-reversible Markov Chain with the desired limiting probabilities. This is done by letting Q be the transition probability matrix from the Markov Chain, where $q(i, j)$ represent the probability in the row, i, and column j. With the use of this the Markov Chain can be defined as follows. When $X_n = i$, then a random variable with probability $P(X = j)$ is generated. If $X = j$ then X_{n+1} is set equal to j with probability $\alpha(i, j)$ and is set equal to i with probability $1 - \alpha(i, j)$. Then the Markov Chain with the transition probabilities $P_{i,j}$ is given by

$$P_{i,j} = q(i, j)\alpha(i, j), \quad j \neq i$$

$$P_{i,i} = q(i, i) + \sum_{k \neq i} q(i, k)(1 - \alpha(i, k))$$

With the use of these the Markov Chain will have the stationary probabilities

$$\pi(i)P_{i,j} = \pi(j)P_{j,i} \quad j \neq i$$

which can be rewritten so that it is containing α

$$\pi(i)q(i, j)\alpha(i, j) = \pi(j)q(j, i)\alpha(j, i)$$

It can be checked whether this is satisfied using

$$\alpha(i, j) = \min\left(\frac{\pi(j)q(j, i)}{\pi(i)q(i, j)}, 1\right) = \min\left(\frac{b(j)q(j, i)}{b(i)q(i, j)}, 1\right)$$

If $\alpha(i, j) = \pi(j)q(j, i)/\pi(i)q(i, j)$ then $\alpha(j, i) = 1$ and also the other way around.

The main idea for the MCMC algortihm is to generate a Markov Chain such that the stationary distribution is the target distribution. The algortihm must specify, for a given state X_m, how to generate the next state X_{t+1} and generate a candidate link Y for the chain from the proposal distribution. If the candidate link is accepted, the chain moves to state Y at the time $n + 1$ and X_{n+1}. Otherwise the chain stays in state X_n and time X_{n+1}.

All of this can be used to generate the following psedo code of Metropolis-Hastings

1. Choose the transition probability matrix **Q** and some initial state value **k**.

2. Let $n = 0$ and $X_0 = k$
3. Generate a random variable X which fulfills that $P(X = j) = q(X_n, j)$ and generate a random number **U** using *runif*.
4. If $U < \alpha(i, j) = \frac{b(X)q(X,X_n)}{b(X_n)q(X_n,X)}$ then NS=X is used otherwise $NS = X_n$ is used.
5. n=n+1 and $X_n = NS$
6. Go to 3.

For this task we want to generate a beta distribution using the Metropolis-Hastings algorithm.

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)}x^{\alpha-1}(1-x)\beta - 1$$

We want to use the Metropolis-Hastings algorithm to generate Beta$(2.5, 5.5)$ distribution using the proposal candidates from uniform $(0, 1)$ distribution.

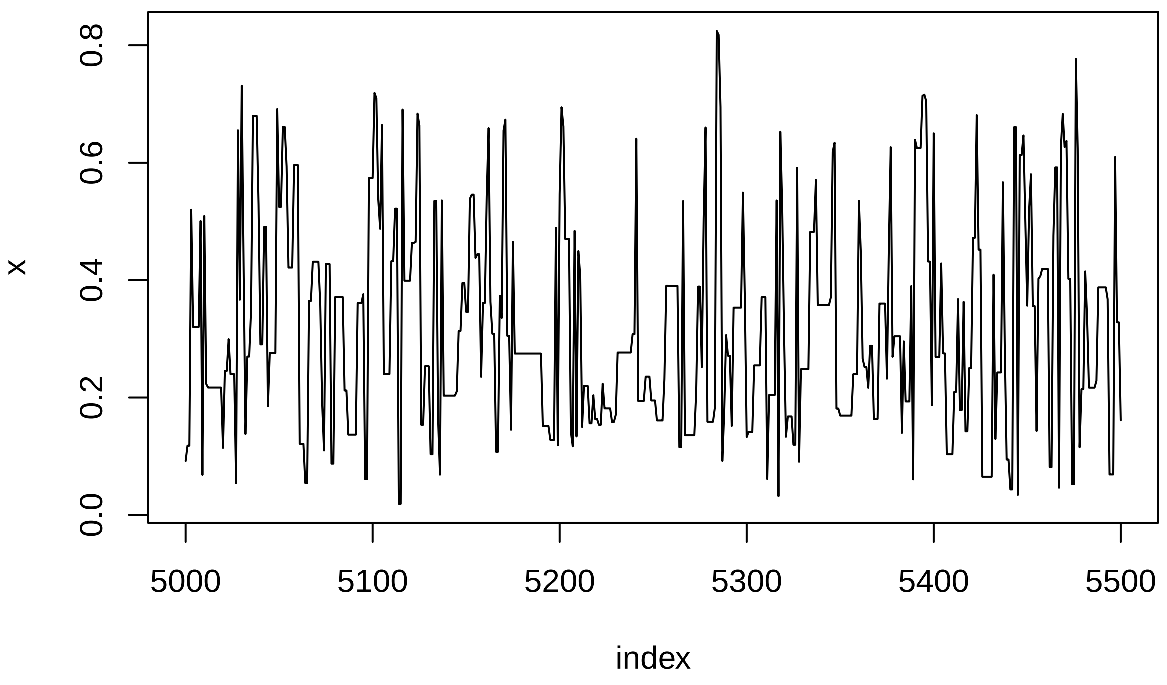
```
set.seed(2016)
n <- 10000#Number of iterations
k<-0#Set the inital value of the number of rejections
u <- runif(n) # Generates n random numbers between 0 and 1
x <- numeric(n) # Set x to be a vector of size n, where each element is equal to zero.
x[1] <- runif(1,0,1)# States that the first element of
#the x vector is a random number between 0 and 1.

for(i in 2:n){# makes a for loop from 2, while the inital value has already been stated.
  xt <- x[i-1]# Sets xt to be the previous value of xi
  y <- runif(1,0,1)# Creates a variable y which is a random number between 0 and 1.

  num <-dbeta(y,2.5,5.5)*dunif(xt)# Sets num to be equal to the density of the betafunction
  #using y as an input variable times the uniform density distribution using
  #the previous x value.
  den <- dbeta(xt,2.5,5.5)*dunif(y)# Sets num to be equal to the density of the
  #betafunction using xt as an input variable times the uniform density distribution
  #using the previous y value.
  alpha <- num/den# Generates the difference between the two distributions.
  if(u[i]<=alpha){# If uniform number is less than or equal to the difference between the
    #two random distributions, then the x value is equal to y, hence y can be used as
    #a proposal candidate for x.
    x[i]<-y
  }else{ # Otherwise the x value is set equal to the previous value
    x[i]<-xt
    k <- k+1#and the rejection number is made biggere.
  }
}
print(k)
```

```
## [1] 5117
```

By running the program it can be seen that the rejection rate is 5117 with $n = 10000$ for seed(2016).

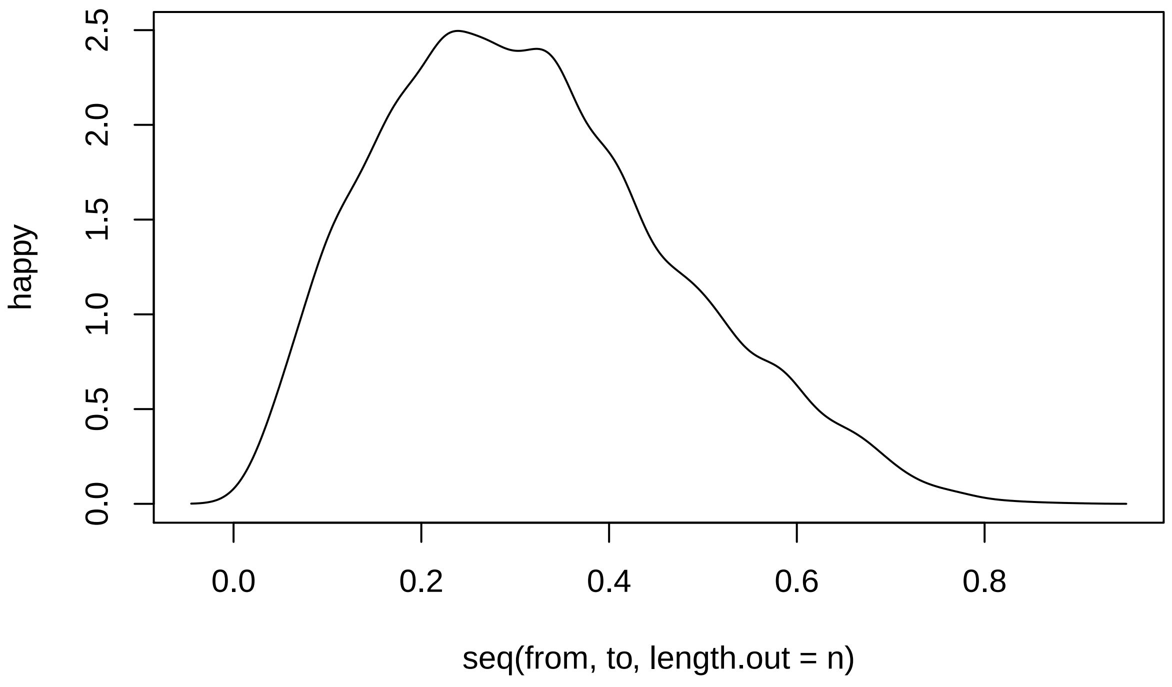
In subtask 7.1.c we want to vizualize a part of the chain generated by Metropolis-Hastings

```r
index <- 5000:5500
y1 <- x[index]
plot(index, y1, type = "l", main="", ylab = "x")
```

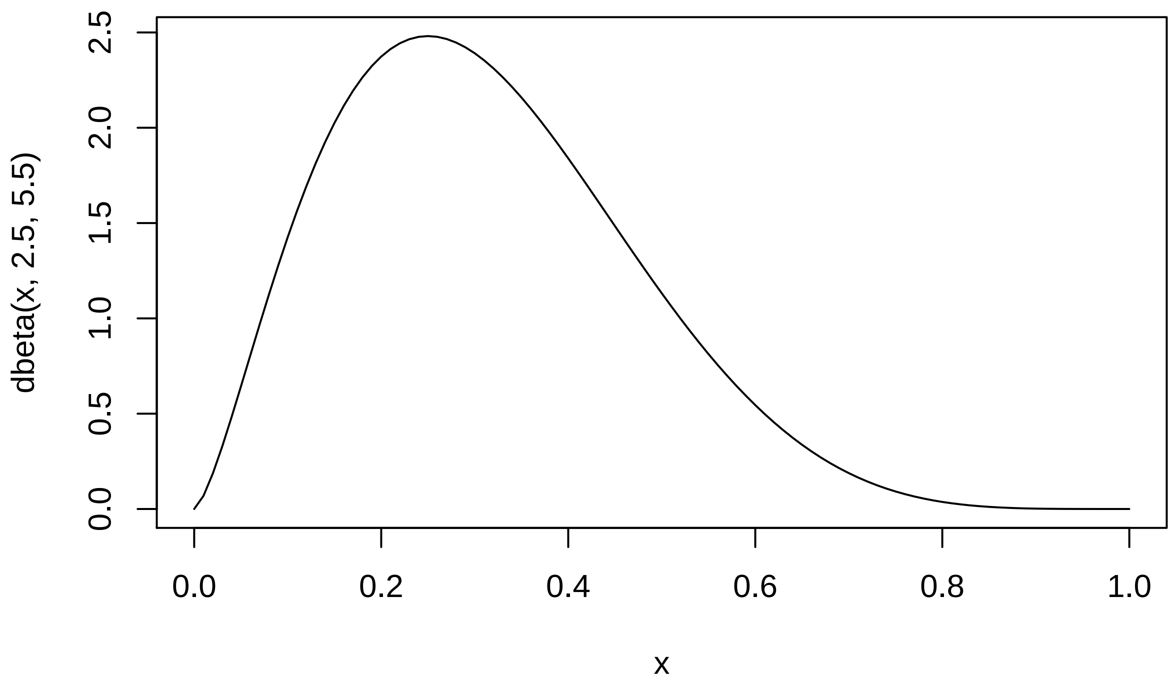


At last we want to generate another beta distribution with the same parameters using rbeta() function. We want to use our densiplot() function from task 5 with the method "kernel".

```r
densiplot(x, method = "kernel")
```



```r
curve(dbeta(x,2.5,5.5))
```



It can be seen that these distributions function are very much alike. The only main difference is the top of *densiplot* has a deviation from the one generated by the build-in function in *R*. So by assumming that the build-in function in *R* provides the correct result, then the use of *densiplot* instead of the build-in function in *R* could result in some misleading results for the maksimum of the plot.

Haghish, E.F. 2016a. "Density."

———. 2016b. "Marcov Chain."

Sheather, Simon J. 2009. *A Modern Approach to Regression with R*. Springer.

Sheldon, Ross. 2013. *Simulation*. Academic Press.

Sigman, Karl. 2007. "Simulating Markov Chains." http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-MC.pdf.

Sitmo. 2009–2015. "Generating Correlated Random Numbers." http://www.sitmo.com/article/generating-correlated-random-numbers/.

n.d. https://en.wikipedia.org/wiki/Simple_linear_regression.

n.d. http://www.uv.es/uriel/4%20Hypothesis%20testing%20in%20the%20multiple%20regression%20model.pdf.