Lab3 - CS - Group6 - Resubmission2

Jaskirat S Marar (jasma356), Filip Berndtsson (filbe354),
Dinuke Jayaweera (dinja628), Raja Uzair Saeed (rajsa233)

13/01/2022

Statement of Contribution

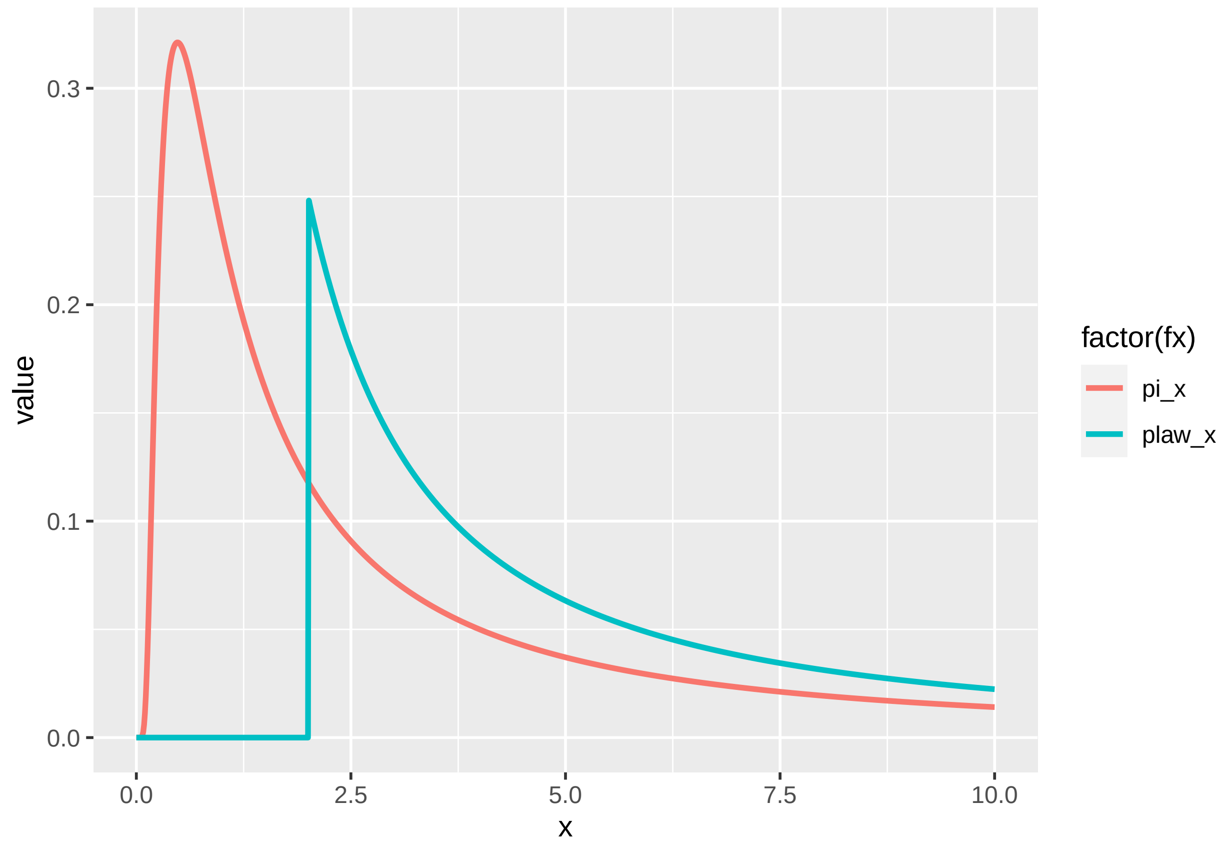
This lab work was divided among group members as follows:

1. Assignment1: Dinuke Jayaweera, Jaskirat Marar, Uzair Saeed
2. Assignment2: Filip Berndtsson

Question1: STABLE DISTRIBUTION

Plotting $f(x)$ and $f_p(x)$ together

```r
# pi(x) with c > 0, support on (0, inf)

target_density <- function(x, c = 1.2) {
  Y <- (c/sqrt(2*pi)) * exp(-(c^2)/(2*x)) * x^(-1.5)
  Y <- ifelse(x>0, Y, 0)
  return(Y)
}

# powerlaw, alpha > 1, Tmin > 0, support on (Tmin, inf)

plaw_distribution <- function(x, tmin = 2, alpha = 1.5) {
  fp_x <- (alpha - 1) / tmin * (x / tmin) ^ -alpha
  fp_x[x <= tmin] = 0
  return(fp_x)
}

x <- seq(0, 10, 0.01)
c <- 1.2

plot_df <- data.frame(x = x, pi_x = target_density(x), plaw_x = plaw_distribution(x))
plot_df1 <- melt(plot_df, id.var = 'x', variable.name = 'fx')
ggplot(plot_df1, aes(x = x, y = value, group = factor(fx), color = factor(fx))) + geom_line(size = 1)
```



We have arbitrarily chosen values of $c = 1.2$, $\alpha = 1.5$ & $T_{min} = 2$ within the definition of the problem.

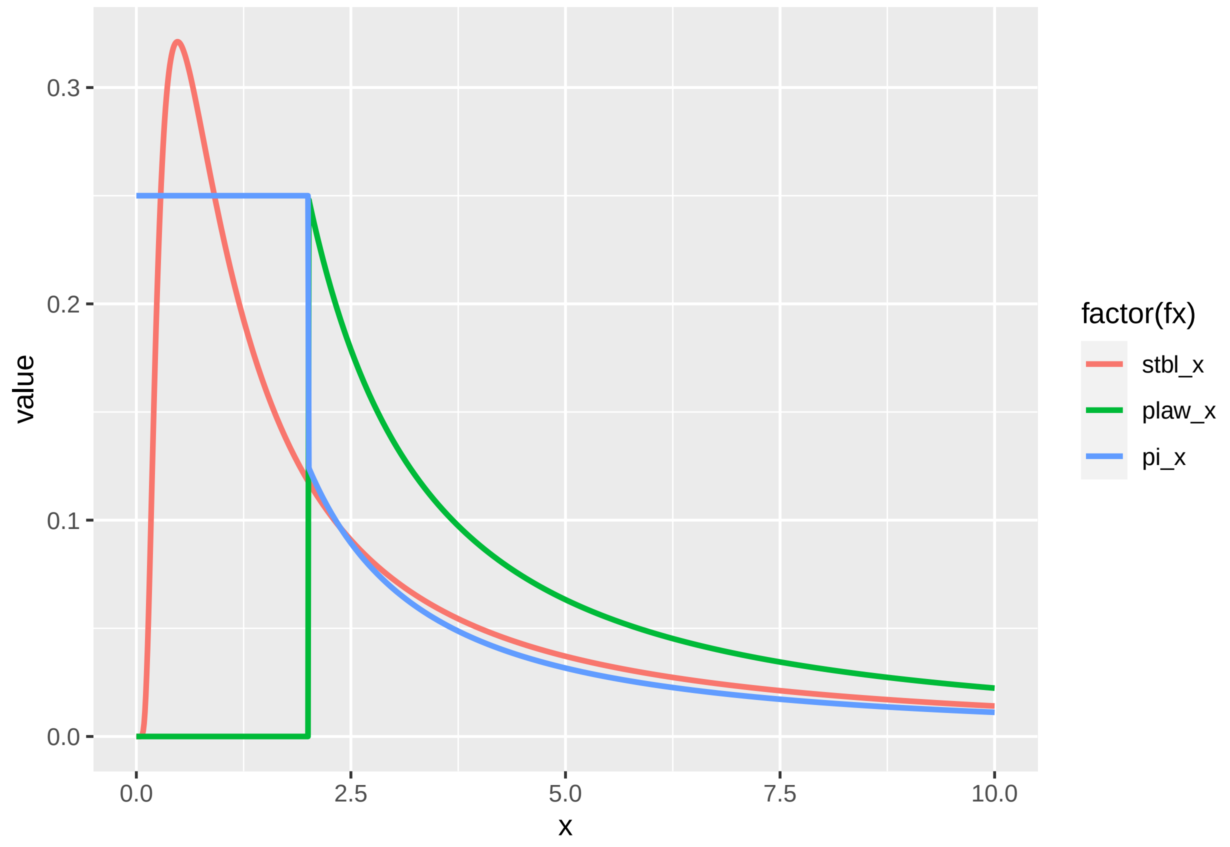
The observed issue with using Power-law distribution alone is that since the Power-law distribution is one sided and exponential, it doesn't fully fit above the stable distribution if even if we set the lower support at the same as that of the stable distribution i.e. $T_{min} = 0$. There is a problem at the upper support also, because the power-law decays much faster than the stable distribution. thus making it necessary to choose a suitable value of multiplication factor 'M' and $\mathrm{T_{min}}$ that shifts the power-law distribution such that the decay can be rectified and also so that the power-law can envelope the target distribution.

But clearly, the problem for the lower support will remain even if we shift the power-law by choosing a better value of $\mathrm{T_{min}}$

We will attempt to rectify this problem by mixing the given power-law density with a uniform density before its lower support. Our resulting density will then be able to cover the data points that lie before $x < T_{min}$

This is done in the following manner where we choose a mixture coefficient = 0.5, to make sure that our mixed distribution function is a pdf. This will be true because we are adding two pdf together wither mixture coefficients adding up to 1, hence ensuring that the resulting majorizing density will also have its density = 1.

```r
# mixing densities

majorizing_density <- function(x, alpha = 1.5, tmin = 2) {
  Y <- rep(0, length(x))
  for (i in 1:length(x)) {
    if (x[i] <= tmin) {
      Y[i] = 0.5 * dunif(1, 0, tmin)
    }  else {
      Y[i] = 0.5 * ((alpha - 1) / tmin * (x[i] / tmin) ^ -alpha)
    }
  }
  return(Y)
}

target_distrib <- target_density(x)
mixed_dist <- majorizing_density(x)

plot_df <- data.frame(x = x, stbl_x = target_distrib, plaw_x = plaw_distribution(x), pi_x = mixed_dist)
plot_df1 <- melt(plot_df, id.var = 'x', variable.name = 'fx')
ggplot(plot_df1, aes(x = x, y = value, group = factor(fx), color = factor(fx), fill = factor(fx))) +
  geom_line(size = 1)
```



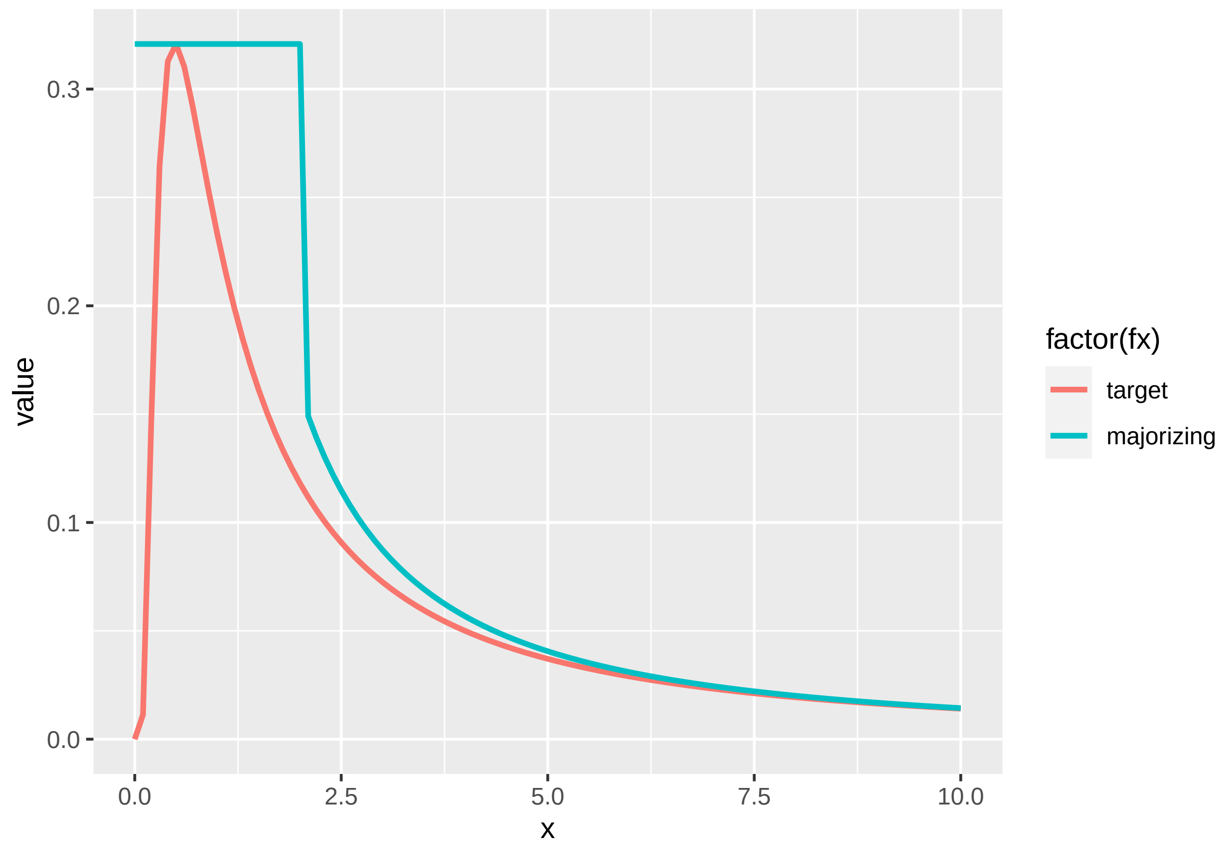
As we can see our resulting density has added a “uniform” part before T$_{\text{min}}$ which envelopes the target distribution, but it has also scaled the power-law part of our distribution down below our target distribution. Now, we move onto finding the majorizing constant that will satisfy the inequality:

$$M * f_p(x) >= f(x)$$

For calculating this we will create a data for a range of x-values and accept the largest value of the multiplier. We obtain the result and use it to shift the majorizing density above the target density. The method is outlines below along with the new plot of the majorizing function:

```
plot_df <- data.frame(x = seq(0,10,by=0.1),
                      target = target_density(x = seq(0,10,by=0.1)),
                      majorizing = majorizing_density(x = seq(0,10,by=0.1)))

plot_df$M <- plot_df$target/plot_df$majorizing
```

```r
M <- max(plot_df$M)

plot_df <- plot_df[,1:3]
plot_df$majorizing <- M*plot_df$majorizing

plot_df <- melt(plot_df, id.var = 'x', variable.name = 'fx')
ggplot(plot_df, aes(x = x, y = value, group = factor(fx), color = factor(fx))) +
  geom_line(size = 1)
```

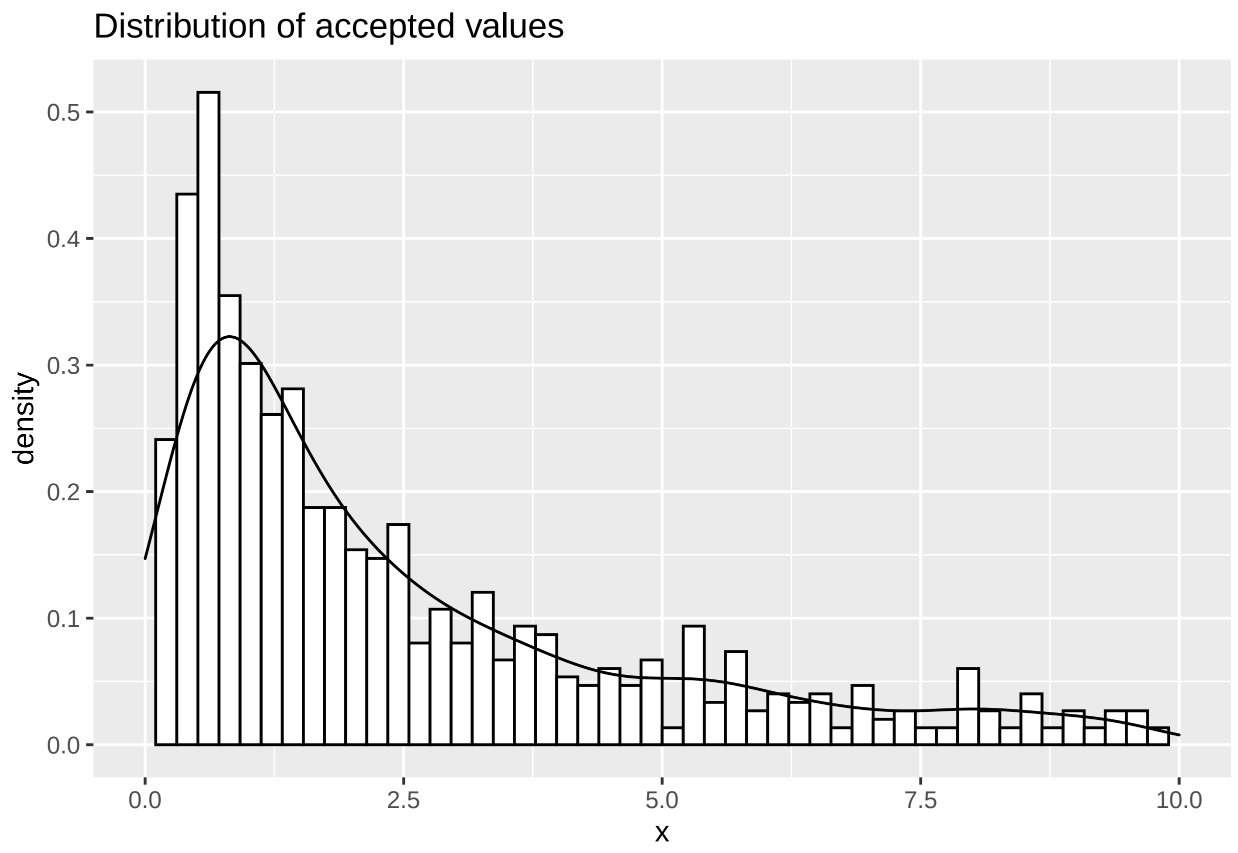


As we can see, our majorizing density is now enveloping the target density. Now we move onto the drawing samples from this distribution using the acceptance rejection method. The approach we used was to initiate a vector accept() where we would storing the accepted samples from the algorithm. The counter helps us loop over our samples from the Uniform distribution and our mixed distribution. For drawing from our mixed distribution we have written a sampling function which has divided the drawing process by initiating a coin toss. For coin toss results ≤ 0.5 we draw from a uniform distribution and in cases of > 0.5 we draw from the power-law distribution using the poweRlaw package.

The accept-reject algorithm is performed with a separate function that takes as input the number of draws needed, the majorizing constant M, and the value of c for the target density (this will be important for performing the last part of the assignment)

```r
# Rejection sampling

sampling_function <- function(alpha = 1.5, tmin = 2) {
  Y <- 0
  coin_toss <- runif(1)
  if (coin_toss <= 0.5) {
    Y = runif(1, 0, tmin)
  }  else {
    Y = rplcon(1, tmin, alpha)
  }
  return(Y)
}

AR <- function(n, c, M = M) {
  count = 0
  accept = c()
  reject = 0
  Y = 0
  U = 0
  i = 1
  c = c

  while (length(accept) < n) {
    U <- runif(1)
    Y <- sampling_function()
    count = count + 1
    if(U <= target_density(x = Y, c = c)/(M*majorizing_density(x = Y))){
      accept[i] = Y
      i = i+1
    } else reject = reject + 1
   }
  return(list(accept = accept, reject = reject, count = count))
}

artest <- AR(n = 1000, c = 1.2, M)

df_accept <- data.frame(x = artest$accept)

ggplot(df_accept, aes(x=x)) +
  geom_histogram(aes(y=..density..), fill="white", color="black", bins=50)+
  xlim(c(0, 10))+ggtitle("Distribution of accepted values") +
  geom_density()
```

```
## Warning: Removed 268 rows containing non-finite values (stat_bin).

## Warning: Removed 268 rows containing non-finite values (stat_density).

## Warning: Removed 2 rows containing missing values (geom_bar).
```



```
print(paste("Rejection rate = ", round(artest$reject/artest$count,4)))
```

```
## [1] "Rejection rate =  0.2337"
```

```
  cat("\n")
```

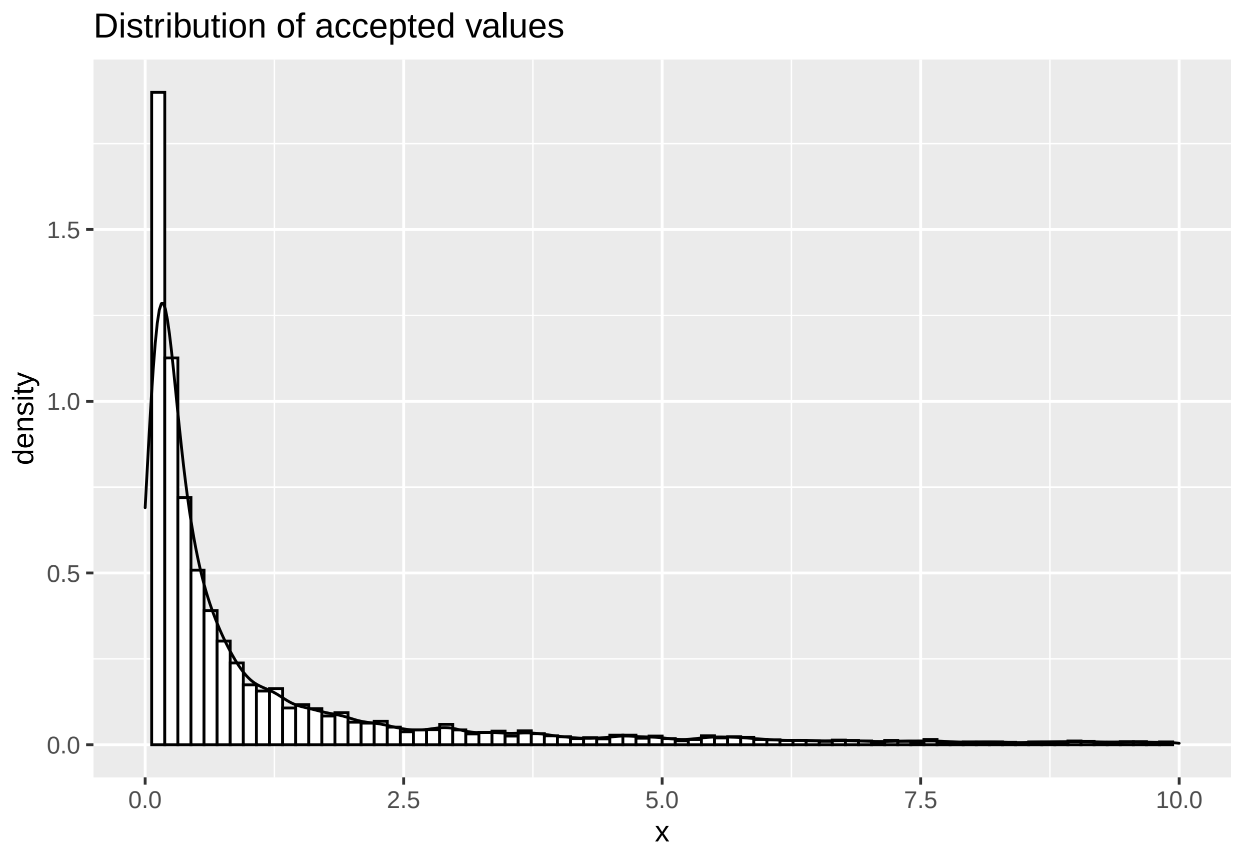
As you can see that our sample generator is able to draw a sample that follows the same distribution as our target function. The rejection rate is $\sim 22\%$

We then ran the sampling for different values of c and graphed them for analysis. As you can see, since we are using a different value of c, we need to optimize the majorizing density each time.

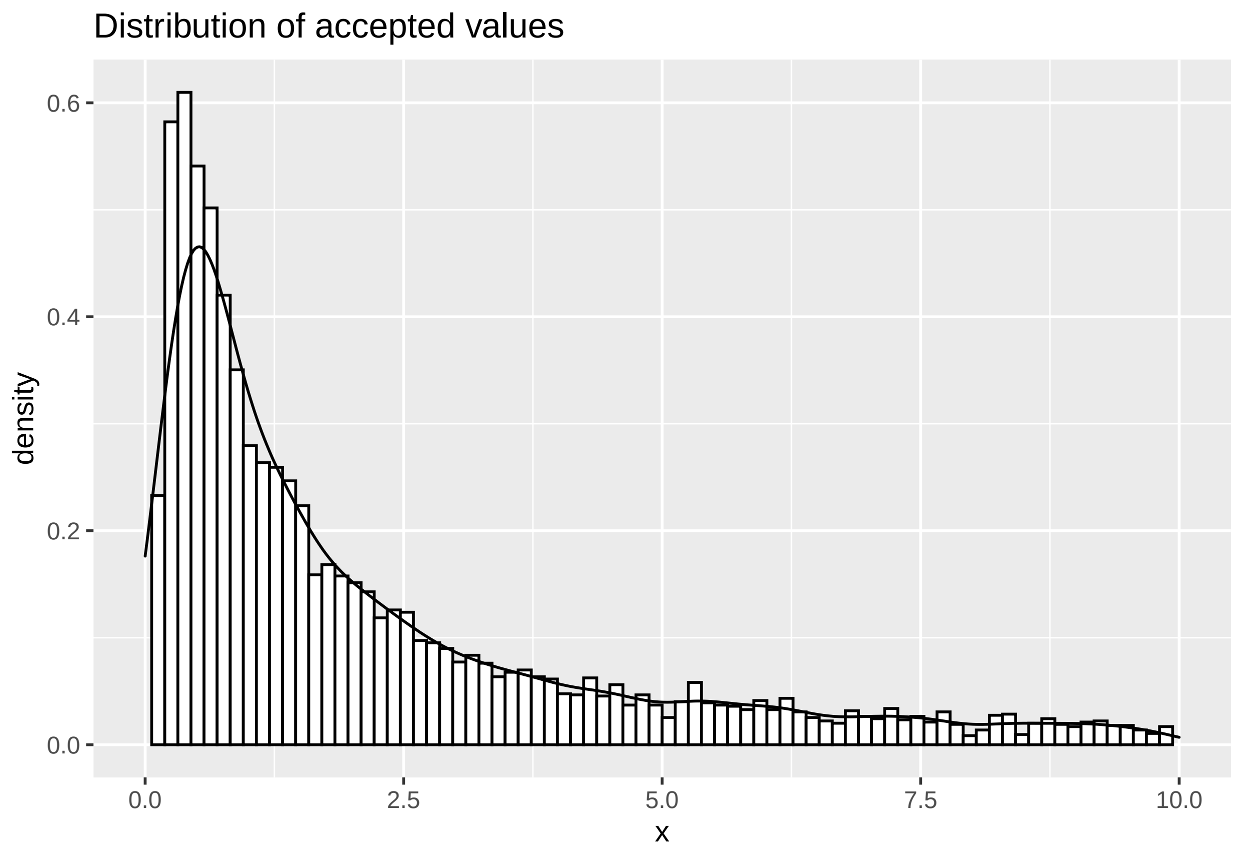
```
## Warning: Removed 1202 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 1202 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



```
## [1] "c = 0.5 ; Rejection rate =  0.8623"
##
## [1] "Mean = 1084.161 ; Variance =  1819928612.5136"

## Warning: Removed 2537 rows containing non-finite values (stat_bin).

## Warning: Removed 2537 rows containing non-finite values (stat_density).

## Warning: Removed 2 rows containing missing values (geom_bar).
```

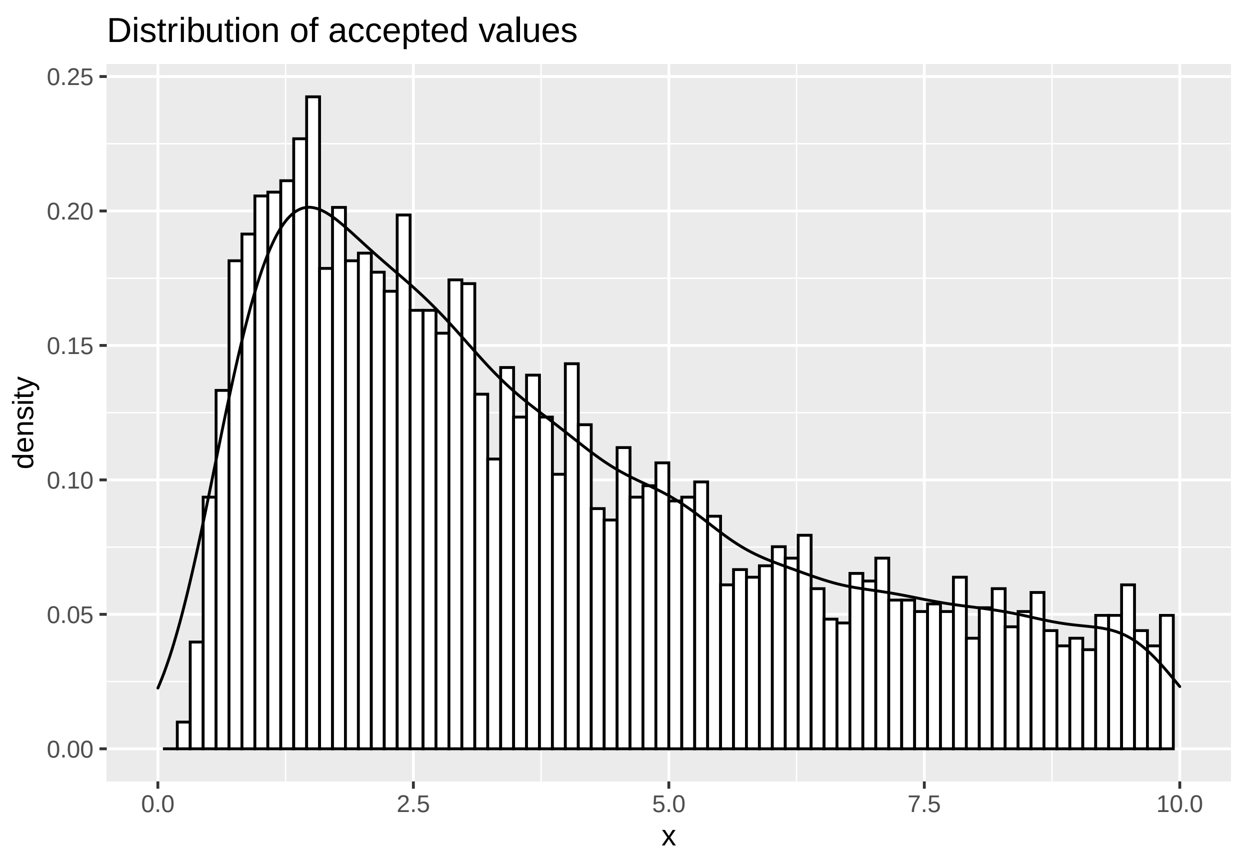


```
## [1] "c = 1 ; Rejection rate =  0.4578"
##
## [1] "Mean = 1732354.0794 ; Variance =  25058823128957256"
```

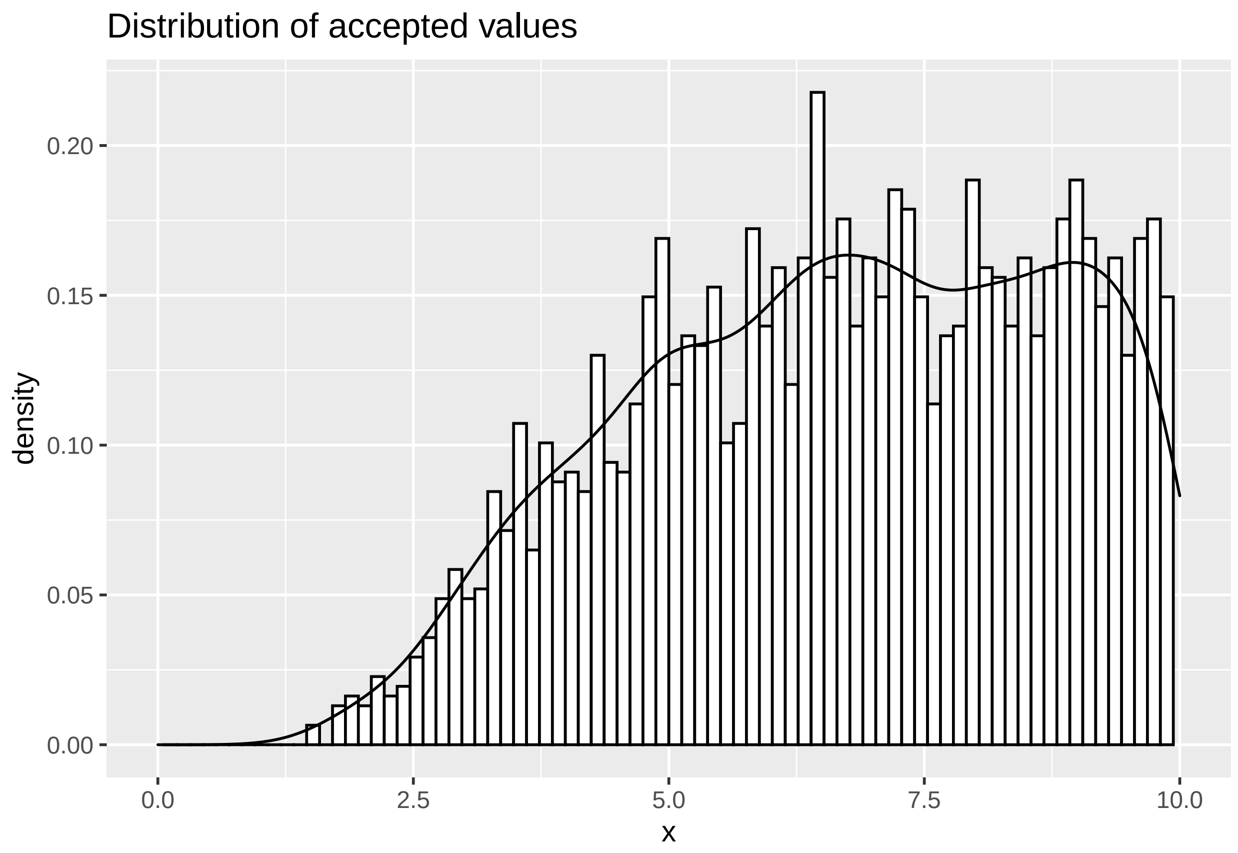
```
## Warning: Removed 4428 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 4428 rows containing non-finite values (stat_density).
```

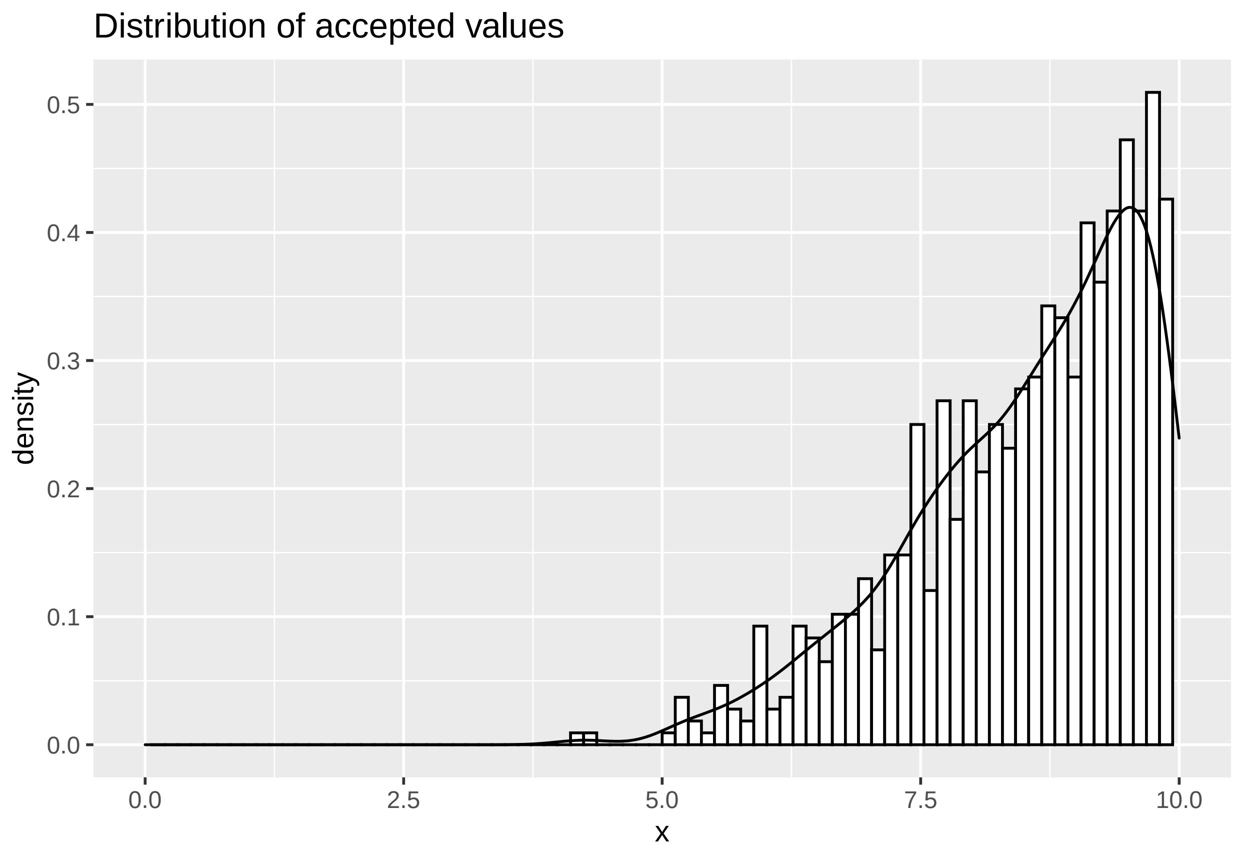
```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



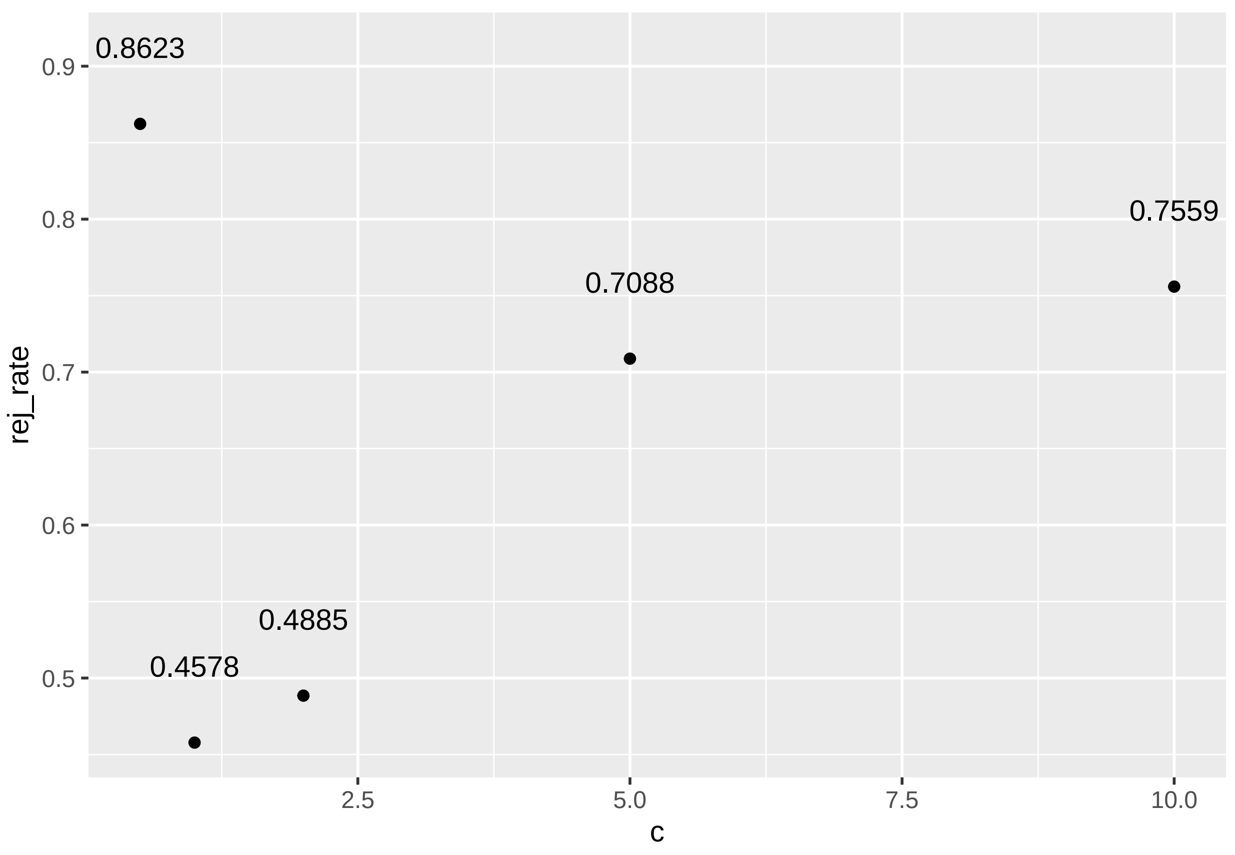
```
## [1] "c = 2 ; Rejection rate =  0.4885"
##
## [1] "Mean = 74427.6828 ; Variance =  28089318438545.6"
## Warning: Removed 7569 rows containing non-finite values (stat_bin).
## Warning: Removed 7569 rows containing non-finite values (stat_density).
## Warning: Removed 2 rows containing missing values (geom_bar).
```

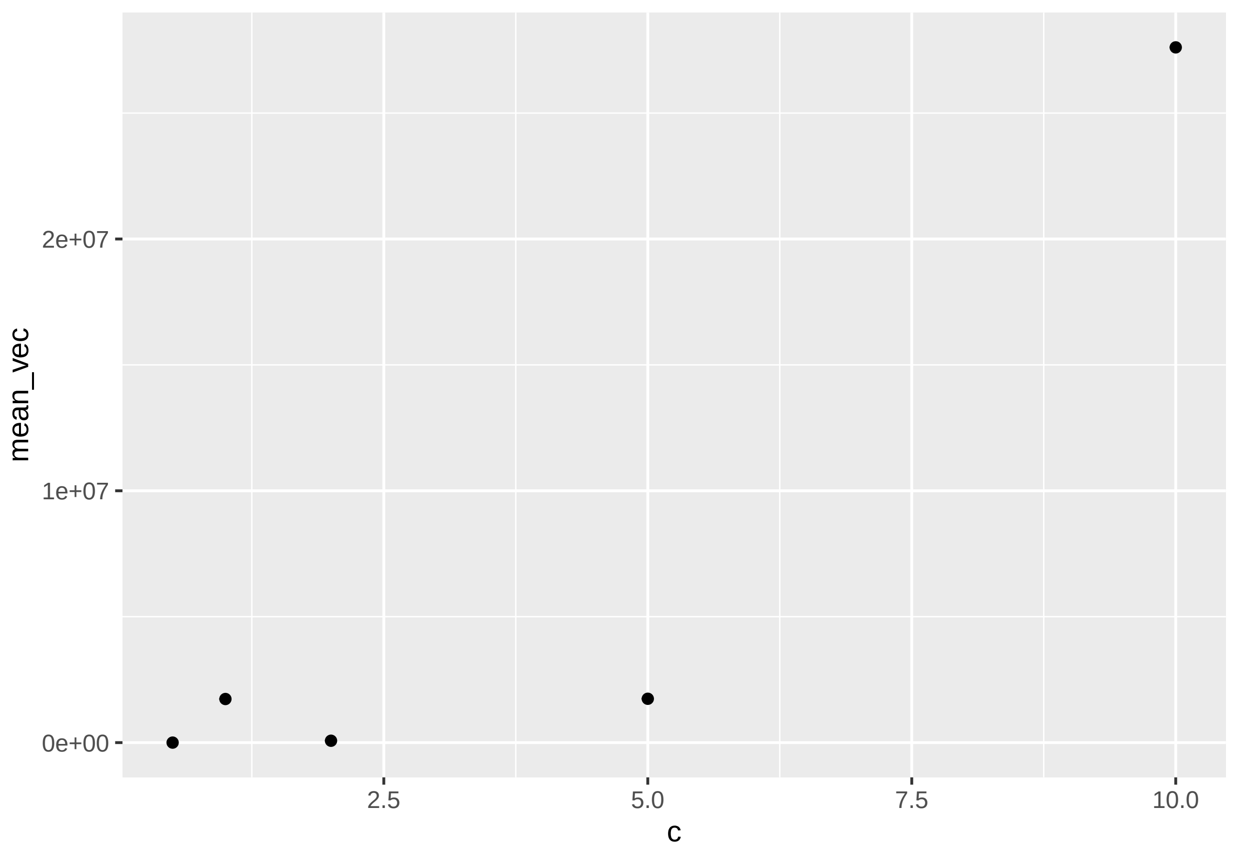


```
## [1] "c = 5 ; Rejection rate =  0.7088"
##
## [1] "Mean = 1741711.2207 ; Variance =  19092963094561788"

## Warning: Removed 9147 rows containing non-finite values (stat_bin).

## Warning: Removed 9147 rows containing non-finite values (stat_density).

## Warning: Removed 2 rows containing missing values (geom_bar).
```

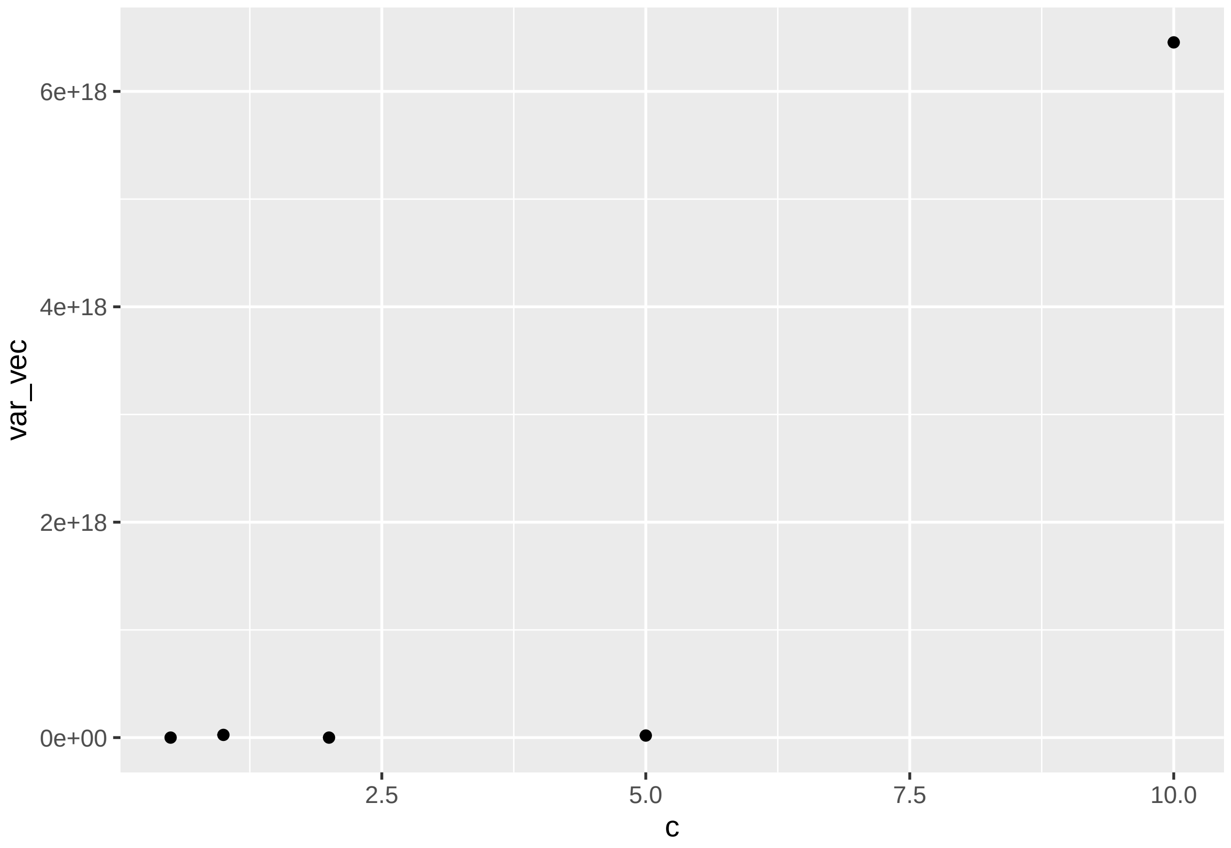


```
## [1] "c = 10 ; Rejection rate =  0.7559"
##
## [1] "Mean = 27608743.2435 ; Variance =  6455195697413608448"
```


0.8623
0.7559
0.7088
0.4885
0.4578
rej_rate
0.9
0.8
0.7
0.6
0.5
2.5
5.0
7.5
10.0
c



mean_vec
2e+07
1e+07
0e+00
2.5
5.0
7.5
10.0
c



We are able to generate outputs for different values of c that match the target density. The rejection rates are on the higher side. We could observe that the rejection rates were lowest in the vicinity of $c \sim 1/$ and much higher as the value of c becomes larger or smaller than 1. The mean and the variance on the other had were stable for smaller values of c, but became much higher and erratic as c increases. This might be because, for the mean and variance for the target distribution may not be a simple function of c (i.e. linear,quadratic etc). Hence it is hard to draw inferences for the relationship between c and the mean & variance of the sample population.

Question2: Laplace Distribution

2.1

We first create the function that we will use for sampling from the double exponentail distrubution $D(0,1)$. The function takes in the arguments μ , b, and n. These have the following default values $\mu = 0$, $b = 1$ and $n = 10000$. The function first use the runif() function to give us 10000 $U(0,1)$ distributed values , we then subtract 0.5. These are then used in our sampling function as seen below and then returned. To get $De(0.1)$ we use $X = \mu - b * sgn(U)ln(1 - 2|U|)$
Here we derive the above formula starting at the CDF of the double exponentail distrubution.

$$F(x) = \begin{cases} \frac{1}{2}e^{\frac{x-\mu}{b}} & \text{if } x \le \mu \\ \frac{1}{2} + \frac{1}{2}[1 - e^{-\frac{x-\mu}{b}}] & \text{if } x > \mu \end{cases}$$

This gives us

$$\frac{1}{2} + sgn(x-\mu)\frac{1}{2}(1 - exp(-\frac{|x-\mu[]}{b}))$$

Suppose $p \in (0,1)$ and let $p = F(x)$.
Note $F(\mu) = \frac{1}{2}$. Suppose that $p \le F(\mu)$. Then $p = \frac{1}{2}e^{\frac{x-\mu}{b}} \to 2p = e^{\frac{x-\mu}{b}} \to ln(2p) = \frac{x-\mu}{b} \to x = \mu + bln(2p)$
Suppose $p > F(\mu)$, then we get $p = \frac{1}{2} + \frac{1}{2}\left[1 - e^{-\frac{x-\mu}{b}}\right]$
$\to 2p - 1 = 1 - e^{-\frac{x-\mu}{b}} \to 2 - 2p = e^{-\frac{x-\mu}{b}} \to ln(2-2p) = -\frac{x-\mu}{b} \to \mu - bln(2-2p) = x$
We have for $0 < p < 1$

$$F^{-1}(x) = \begin{cases} \mu + bln(2p) & \text{if } p \le 0.5 \\ \mu - bln(2-2p) & \text{if } p > 0.5 \end{cases}$$

Now we need to find a general expression.Sign differences can be see above.

$$\begin{cases} 2p & \text{if } p \le 0.5 \\ 2-2p & \text{if } p > 0.5 \end{cases}$$

Here we re-write things until we get it on the desired form

$$2\begin{cases} p & \text{if } p \le 0.5 \\ 1-p & \text{if } p > 0.5 \end{cases} = 2\begin{cases} p + 0.5 - 0.5 & \text{if } p \le 0.5 \\ 1 - p + 0.5 - 0.5 & \text{if } p > 0.5 \end{cases} = 2\begin{cases} (p-0.5) + 0.5 & \text{if } p \le 0.5 \\ 0.5 - (p-0.5) & \text{if } p > 0.5 \end{cases}$$

$$2(0.5) + 2\begin{cases} p - 0.5 & \text{if } p \le 0.5 \\ -(p-0.5) & \text{if } p > 0.5 \end{cases} = 1 - 2\begin{cases} -(p-0.5) & \text{if } p \le 0.5 \\ p - 0.5 & \text{if } p > 0.5 \end{cases} = 1 - 2|p - 0.5|$$

This gives us

$$F^{-1}(p) = \mu + sgn(p-0.5)ln(1 - 2|p-0.5|), 0 < p < 1$$

Let $p \in U(0,1)$
Then

$$x = \mu bsgn(p-0.5)ln(1 - 2|p-0.5|)$$

is $De(\mu, b)$
Define

$$Q = P - 0.5, U(-0.5, 0.5)$$

Then

$$X = \mu b sgn(Q) ln(1 - 2|Q|)$$

is $De(\mu, b)$

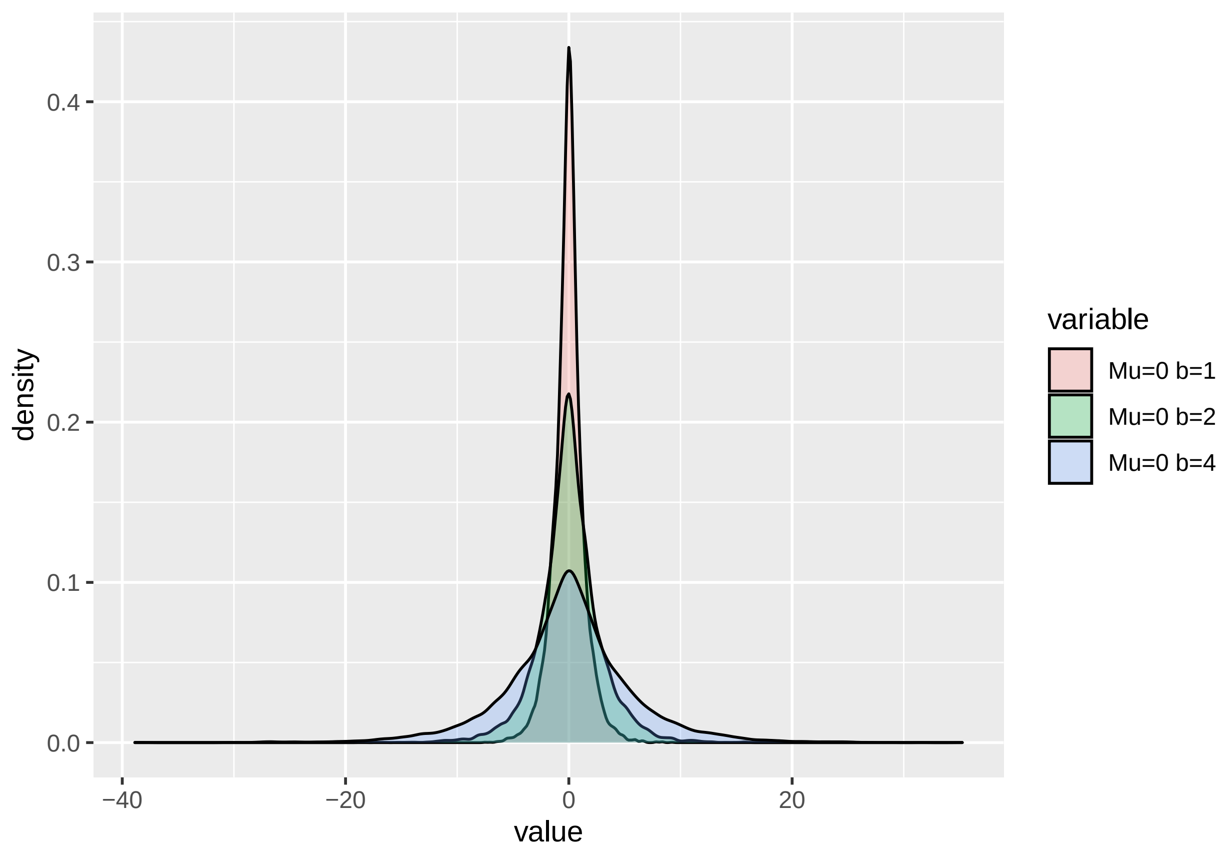
```
library(ggplot2)
library(reshape2)
rlaplacefunc <- function(mu=0, b=1 , n= 10000){
  p <- runif(n,-1/2 , 1/2)
  y <- mu - b*sign(p)*log(1-2*abs(p))

  return(y)
}
dlaplacefunc <- function(x, alpha, beta) 1/(2*beta) * exp(-abs(x - alpha)/beta)
```

We now create the needed data to produce the plots we want to check if our function produces the desired results. The column names are changed as to make plotting easier. All the data is then collected in a single data frame and we use the melt() function to make it easier to plot.

```
test_1 <- as.data.frame(rlaplacefunc())
# Rename column for better plots
colnames(test_1) <- "Mu=0 b=1"

test_2 <- as.data.frame(rlaplacefunc(0,2))
colnames(test_2) <- "Mu=0 b=2"

test_3 <- as.data.frame(rlaplacefunc(0,4))
colnames(test_3) <- "Mu=0 b=4"

plot_df <- cbind(test_1 , test_2 , test_3)
data<- melt(plot_df)
```

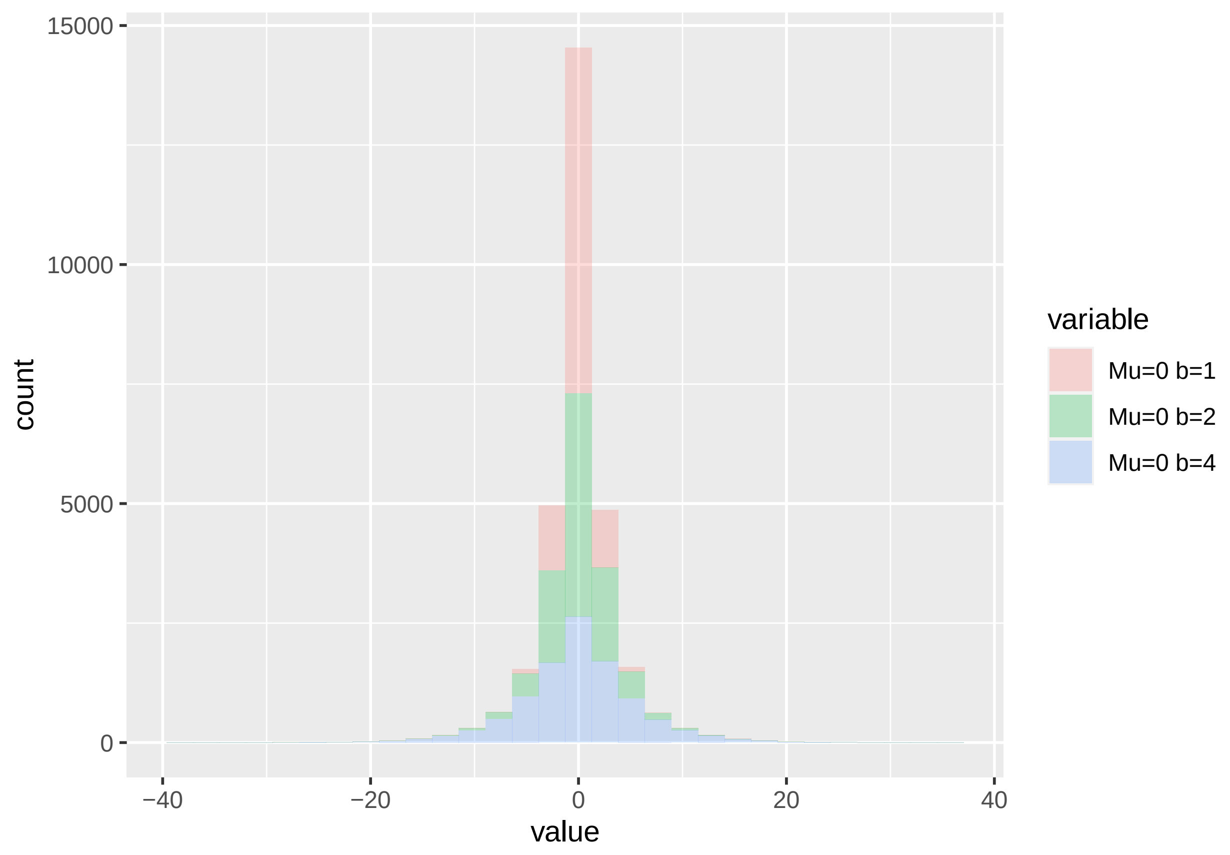
```
## No id variables; using all as measure variables
```

```
p1<- ggplot(data,aes(x=value, fill=variable)) + geom_density(alpha=0.25)
p2<-ggplot(data,aes(x=value, fill=variable)) + geom_histogram(alpha=0.25)
p1
```



```
p2
```

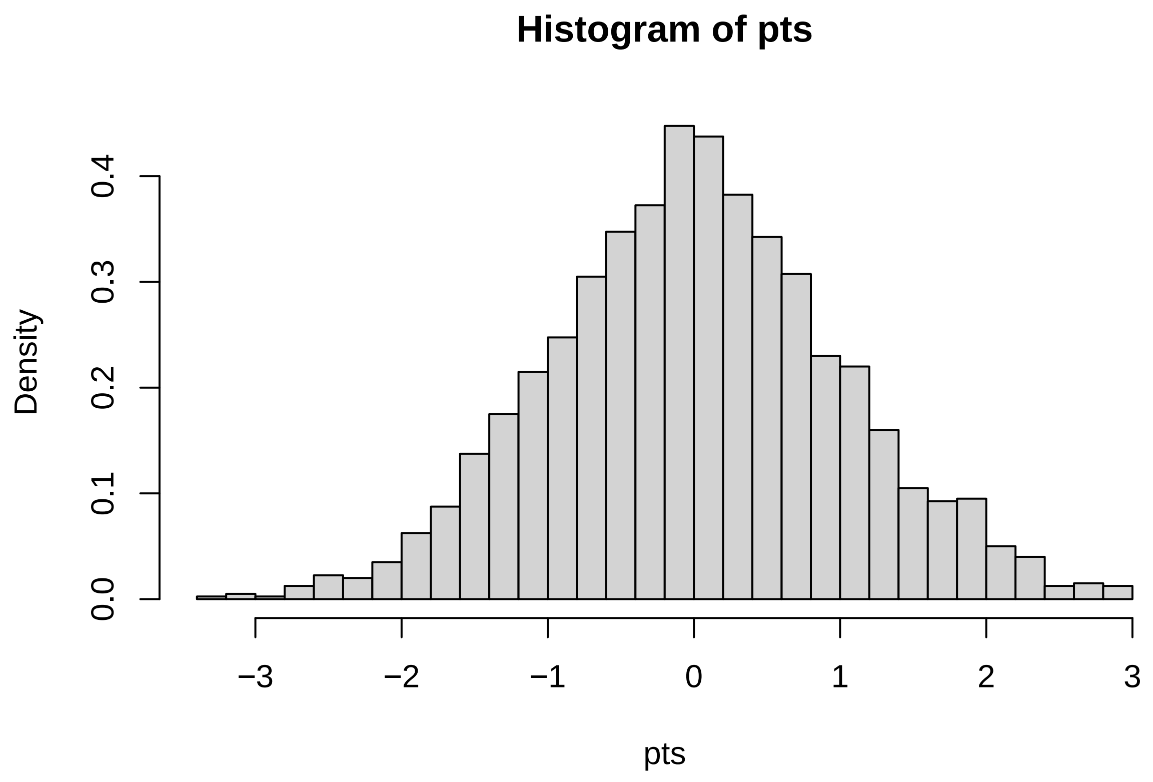
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Above we can see both a density plot and a boxplot for different values for b. These results seems in line with what we should expect.

2.2

```
set.seed(1234)
C = 0
n_draws = 0
n_samples = 2000
rNormal = function(alpha, beta, mu, sigma){

  ratio = function(x) dnorm(x, mu, sigma) / dlaplacefunc(x, alpha, beta)
  if(C == 0) C <<- max(ratio(mu + sigma^2/beta), ratio(mu - sigma^2/beta), ratio(alpha))

  repeat {
    draw = rlaplacefunc(alpha, beta , 1)
    n_draws <<- n_draws + 1
    keep = ratio(draw)/C > runif(1)
    if(keep)
      return(draw)
  }
}

pts = replicate(n_samples, rNormal(alpha=0, beta=1, mu=0, sigma=1))
hist(pts, probability = T , breaks = 40)
```



```
empirical_rejection_rate = 1 - n_samples/n_draws
theoretical_rejection_rate = 1 - 1/C
```

Above is the code we use for the sampling. First we initiate our global variables outside of the function, here we have C , numbers of draws and the size of the sample. Following this we initiate the function used for sampling. This function takes α, β, μ and σ as arguemnts. Where the first two refer to the normal distrubution and the last two refers to the double exponential distribution. Within this function we define another function called ratio, this function calulates the ratios between the two distrubutions. After this we feature a if statement, so if $C == 0$ we assign the value returned from the max() function to C. After this we have the repeat{} part where we first make one draw using our rlaplacefunc from 2.1 , after this we +1 to our variable keeping track of numbers of draws. Following this we have a variable telling us if we should keep draw or not, this variable is TRUE/FALSE and its true when the function ratio with draw as argument divided by C is bigger then a $Unif(0, 1)$ value. The reason this is done is so that we end up with a simulation from the desired distrubtuion.

To now finally receive our plot of the density we use the replicate function where n_samples is how many times the function will be runned , and then we call our function. We then plot this and calculate the Empirical rejection rate and the theoretical rejection rate.

```
print("Empirical rejection rate")
```

```
## [1] "Empirical rejection rate"
```

```
empirical_rejection_rate
```

```
## [1] 0.2357661
```

```
print("Theoretical rejection rate")
```

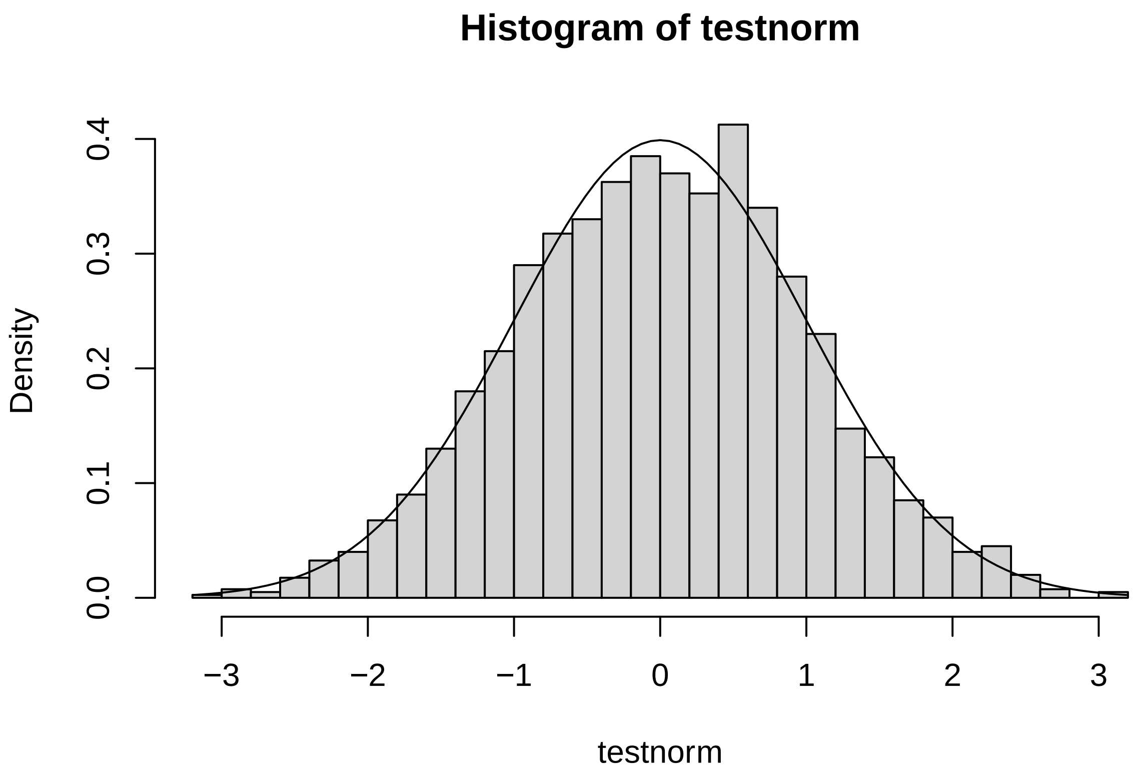
```
## [1] "Theoretical rejection rate"
```

```
theoretical_rejection_rate
```

```
## [1] 0.2398265
```

Here we compute the rejection rate at 23.57 and the theoretical rejection rate should be 23.98 , these values are extremely close and should only get closer as n increases.

```
testnorm <- rnorm(2000 , 0 ,1)
hist(testnorm, probability = T , breaks = 40)
curve(dnorm(x ,0,1), add=T)
```



Lastly we generate 2000 $N(0,1)$ variables and plot these to compared to our previous plot. These look pretty much the same so we conclude that our first plot was indeed correct.