Stochastic Simulation Coursework

Joshua Tegegne

$$f_X^*(x) = \begin{cases} \frac{1}{(x)(5-x)} e^{-\frac{1}{7}(-0.6+log(\frac{x}{5-x}))^2} & 0 < x < 5 \\ 0 & \text{otherwise} \end{cases}$$

In this project we are given a probability density function up to proportionality. We will apply a rejection scheme to simulate from the original distribution X, then investigate how effective our algorithm is and interrogate its accuracy. After we will use Monte Carlo integration to estimate the normalising constant associated with $f_X^*(x)$, discussing the results throughout.

```
CID = 01705625 #finding parameters for f
sprintf("Largest prime factor of CID is %s", max(primeFactors(CID)))
```

```
## [1] "Largest prime factor of CID is 2729"
```

```
print("Corresponding parameters are (a,b,c,d) = (0,5,7,-0.6)")
```

```
## [1] "Corresponding parameters are (a,b,c,d) = (0,5,7,-0.6)"
```

1 : Rejection scheme

We will define the relevant functions and outline the rejection scheme as follows:

$$g_Y^*(x) = \begin{cases} 0.15 & 0 \le x < 3 \\ \frac{11x-21}{80} & 3 < x \le 5 \\ 0 & \text{otherwise} \end{cases} \quad f_X(x) = \frac{f_X^*(x)}{\int_0^5 f_X^*(x)dx} \quad g_Y(x) = \frac{g_Y^*(x)}{\int_0^5 g_Y^*(x)dx} = \frac{g_Y^*(x)}{1.025} \quad M = \sup_{x\in(0,5)} \frac{f_X^*(x)}{g_Y(x)}$$

1) Generate random sample $U = u \sim U(0,1)$ distribution
2) Generate random sample $Y = y \sim g_Y$ distribution
3) If $u \le \frac{f_X^*(y)}{Mg_Y(y)}$, accept and set X = y
4) Else go back to step 1

Note that the rejection sampling method is possible only if the support of g_Y encompasses that of f_X^* and $f_X^*(x) \sim O(g_Y(x))$. This is true as X and Y have supports of $(0,5)$ and $f_X^*(x)$ is bounded. First let us prove that our scheme works, that the accepted samples from Y are have a probability density function f_X

$$P(Y \le x | X \text{ is accepted}) = \frac{P(Y \le x, X \text{ is accepted})}{P(X \text{ is accepted})} = \frac{\int_{-\infty}^{x} \int_0^{\frac{f_X^*(y)}{Mg_Y(y)}} g_Y(y)dudy}{\int_{-\infty}^{\infty} \int_0^{\frac{f_X^*(y)}{Mg_Y(y)}} g_Y(y)dudy} = \frac{\int_{-\infty}^{x} \frac{f_X^*(y)}{Mg_Y(y)} g_Y(y)dy}{\int_{-\infty}^{\infty} \frac{f_X^*(y)}{Mg_Y(y)} g_Y(y)dy} =$$

$$= \frac{\int_{-\infty}^{x} f_X^*(y)dy}{\int_{-\infty}^{\infty} f_X^*(y)dy} = \frac{\frac{f_X^*(y)}{f_X(y)} \int_{-\infty}^{x} f_X(y)dy}{\frac{f_X^*(y)}{f_X(y)} \int_{-\infty}^{\infty} f_X(y)dy} = \frac{F_X(x)}{1} = F_X(x)$$

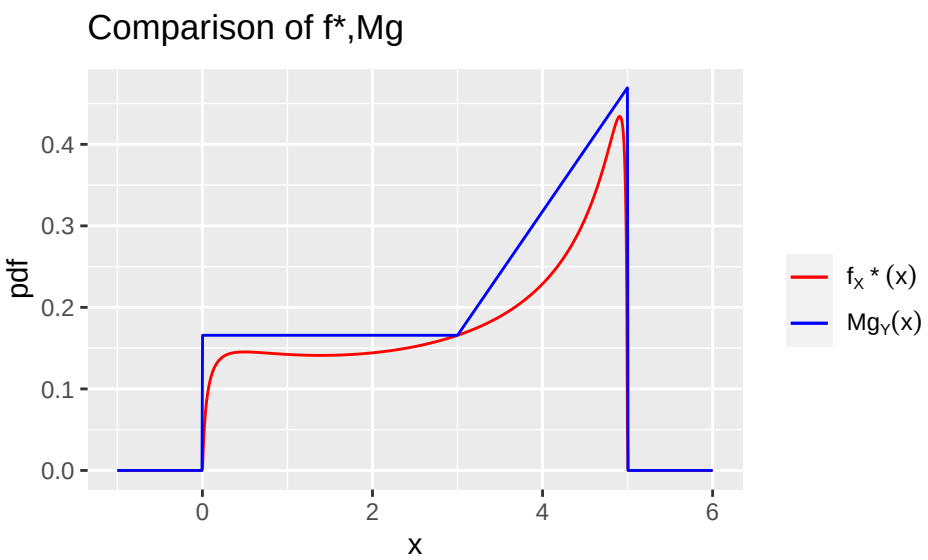
We have shown that the cumulative distribution function of the accepted samples is $F_X(x)$, so for the probability density function must be $f_X(x)$.

To carry out step 2 in our algorithm and generate a random sample from Y, it will be key to use the probability integral transformation. This states that if $F_Y(y)$ is a continuous cumulative distribution function and $U \sim U(0,1)$ this implies that $F_Y(Y) \sim U(0,1)$ or $F_Y^{-1}(U) \sim Y$. We prove this (Angus, 1994) using that continuity of the cumulative distribution function implies that $F_Y(Y) \leq y \Leftrightarrow Y \leq F_Y^{-1}(y)$.

$$P(F_Y(Y) \leq y) = P(Y \leq F_Y^{-1}(y)) = F_Y(F_Y^{-1}(y)) = y \quad \forall y \in [0,1] \implies F_Y(Y) \sim U(0,1)$$

We want to pick an appropriate function g_Y such that it is very easy to sample from, so we pick a piece-wise linear function which is easy to invert. We also want M to be as small as possible so that our acceptance probability is high. We go into more detail on this later, but we can minimize M by ensuring that the re-scaled probability density functions, f_X, g_Y closely mimic each other.

```
a <- 0
b <- 5
c <- 7
d <- -0.6
#define functions
f_star <- function(x) {  ifelse(x> a & x< b, exp((-((d + log((x-a)/(b-x)) )^2)/c))/((x-a)*(b-x)) ,0)}
g_star <-  function(x) { ifelse(x<0, 0, ifelse(x <=3, 0.15, ifelse(x <= 5, (11*x-21)/80, 0))) }
g<- function(x) { ifelse(x<0, 0, ifelse(x <=3, 0.15/1.025, ifelse(x <= 5, (11*x-21)/(1.025*80), 0))) }

x=seq(-1,6,l=1000)

M <- optimise(h <- function(x) {f_star(x)/g(x)}, c(0,5), maximum = TRUE)[[2]]
# the optimise function in R lets us find the maximum value and where the maximum is,
#M is the supremum of f*/g, [[2]] extracts only the supremum

#plot f*, Mg
datf = data.frame(x=x, fx=f_star(x), gx=M*g(x))
label=list(expression(f[X]~"*"~(x)),expression(Mg[Y](x))) #colour coding
ggplot(datf) + geom_line(aes(x,fx,colour="fx"))+
  geom_line(aes(x,gx,colour="fy"))+
  labs(y="pdf", title=expression("Comparison of f*,Mg"))+
  scale_colour_manual("", values=c("fx"="red","fy"="blue"), labels=label)
```

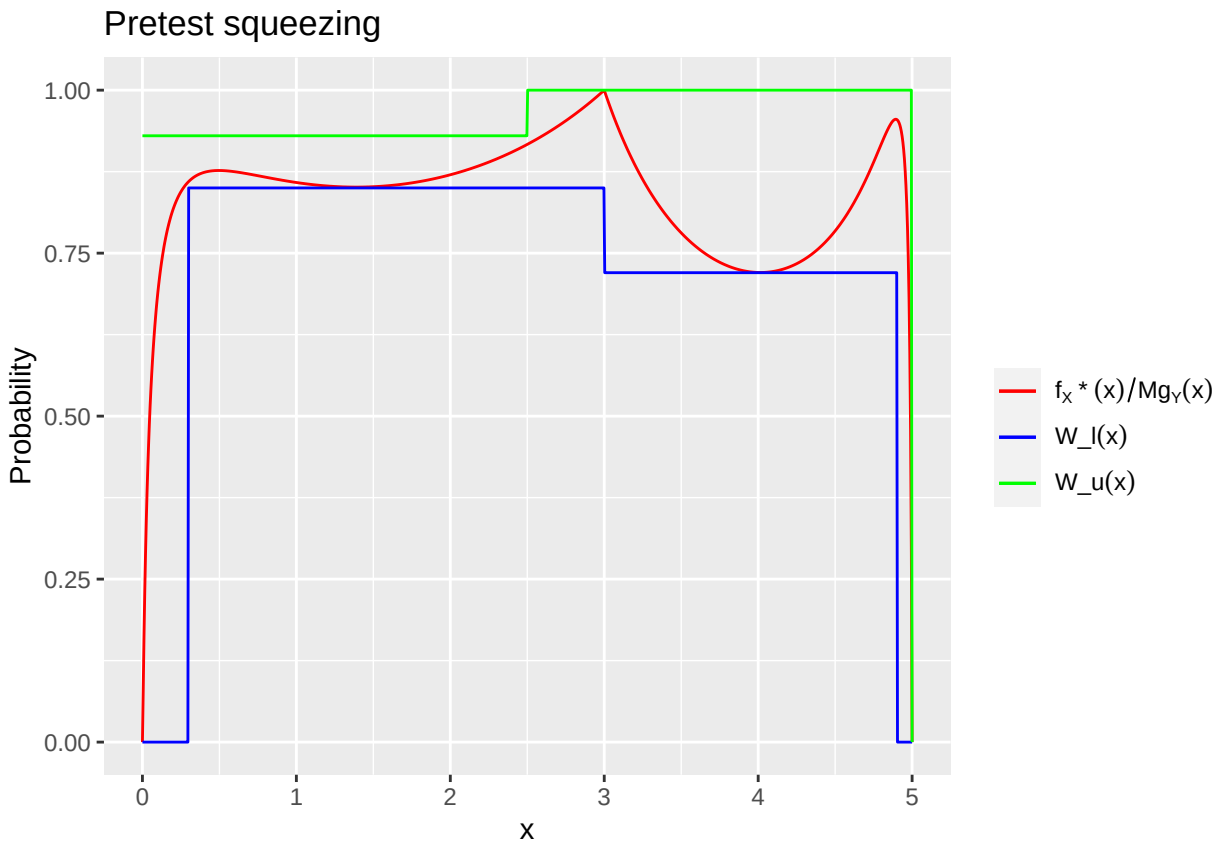


Visually it appears that Mg_y closely overlaps f_X*. Because $\frac{f_X^*(y)}{Mg_Y}$ is almost always close to 1, the acceptance condition $u < \frac{f_X^*(y)}{Mg_Y(y)}$ will be

true for a high proportion of u sampled uniformly in between 0 and 1. Therefore we expect to have a high acceptance rate. However, as y approaches 0 or 5, the ratio $\frac{f_X^*(y)}{Mg_Y(y)}$ approaches 0, but the probability of a sample of y falling within these regions is very low so this shouldn't significantly affect our acceptance rate.

We also aim to use pretest squeezing, because it would be computationally expensive to check if $u \leq \frac{f_X^*(y)}{Mg_X(y)}$ as this function contains exponentiation. We will avoid this as much as possible by finding an lower and upper bound functions $W_L(y), W_U(y)$ for $\frac{f_X^*(y)}{Mg_X(y)}$.

```
x=seq(0,5,l=1000)
func <- function(x){ifelse(x>0 & x<5, f_star(x)/(M*g(x)), 0)}
#define function we want to pretest squeeze with
W_u <- function(x){ifelse(x<0,0,ifelse(x < 2.5, 0.93,ifelse(x < 5, 1,0)))}
W_l <- function(x) {ifelse(x<0.3,0,ifelse(x< 3,0.85,ifelse(x<4.9, 0.72,0)))}
#coords of maximum and minimum of func

dataf = data.frame(x=x, ax=func(x), bx=W_l(x), cx = W_u(x))
label=list(expression(f[X]~"*"~(x)/Mg[Y](x)),expression(W_l(x)),expression(W_u(x)))
ggplot(dataf) + geom_line(aes(x,ax,colour="ax"))+
  geom_line(aes(x,bx,colour="bx"))+ geom_line(aes(x,cx,colour="cx"))+
  labs(y = "Probability", title="Pretest squeezing")+
  scale_colour_manual("", values=c("ax"="red","bx"="blue","cx"="green"), labels = label)
```



$$W_U(x) = \begin{cases} 0.93 & 0 < x \leq 2.5 \\ 1 & 2.5 < x \leq 5 \\ 0 & \text{otherwise} \end{cases} \qquad W_L(x) = \begin{cases} 0.85 & 0.3 < x \leq 3 \\ 0.72 & 3 < x \leq 4.9 \\ 0 & \text{otherwise} \end{cases}$$

These two simple functions bound from above and below, so $W_L(y) \leq \frac{f_X^*(y)}{Mg_X(y)} \leq W_U(y)$. Instead of testing if $u \leq \frac{f_X^*(y)}{Mg_Y(y)}$, we now do the following in these 3 cases: 1) $u \leq W_L(y) \implies u \leq \frac{f_X^*(y)}{Mg_X(y)}$, so accept y\ 2)

$u \geq W_U(y) \implies u \geq \frac{f_X^*(y)}{Mg_X(y)}$, so reject y\ 3) $W_L(y) < u < W_U(y) \implies$ we do not know if we accept/reject, test $u \leq \frac{f_X^*(y)}{Mg_Y(y)}$\

Below we implement the rejection scheme, with pretest squeezing to generate a sample of size n.

$$F_Y^{-1}(u) = \begin{cases} 0 & u < 0 \\ \frac{41u}{6} & 0 \leq u < \frac{18}{41} \\ \frac{21}{11} + \sqrt{\frac{164u}{11} - \frac{648}{121}} & \frac{18}{41} \leq u < 1 \\ 1 & u \geq 1 \end{cases}$$

```
rejection <- function(n){
  vec <- numeric(n) # initialise vector of sample, total iterations and number of accepted Y
  i <- 0 # initialise  number of accepted samples
  count <- 0 # initialise total iterations
  while (i < n) {
    u <- runif(1)  #sample from standard uniform
    u2 <- runif(1)
    if (u2 < 18/41) {y = 41*u2/6} else {y = 21/11 + (164*u2/11 - 648/121)**0.5}
    #use sample from standard normal to find sample from Y
    if (u< f_star(y)/(M*g(y))) { #if acceptance condition is met, move to next element
      vec[i] <- y
      i = i + 1}
    count = count + 1

  }
vec <- c(data.frame(vec), n/count)
vec # vec contains a dataframe of samples, and the acceptance rate
}
```

Acceptance probability

In our algorithm we accept our sample of $Y \sim g_Y$ as a sample of X if $u < \frac{f_X^*(y)}{Mg_Y^*(x)}$. We now want to calculate the acceptance probability, $\mathrm{P}(U < \frac{f_X^*(Y)}{Mg_Y^*(Y)})$ where $U \sim (0,1)$.

$$\text{Acceptance probability} = P(U < \frac{f_X^*(Y)}{Mg_Y(Y)}) = \int_{-\infty}^{\infty} \int_0^{\frac{f_X^*(y)}{Mg_Y(y)}} 1 * g_Y(y) du dy = \int_{-\infty}^{\infty} \frac{f_X^*(y)}{Mg_Y(y)} g_Y(y) dy$$

We have used that the joint probability density function of Y and U must be the product of their individual probability density functions by independence. Now we simplify:

$$= \int_{-\infty}^{\infty} \frac{f_X^*(y)}{Mg_Y(y)} g_Y(y) dy = \frac{1}{M} \int_{-\infty}^{\infty} f_X^*(y) dy = \frac{1}{M} \int_0^5 f_X^*(y) dy$$

```
accep_prob <- integrate(f_star,0,5)[[1]]/(M) # acceptance probability
expected_Z <- 1/accep_prob # expected number of trials to first accept of Y
```

We obtain that the acceptance probability is 0.828 to 3 significant figures - meaning that we accept X approximately 83 percent of the time, despite the fact that we use an extremely simple overlapping function g_Y. Having a high acceptance probability is desirable as it reduces the average computational time of our algorithm.

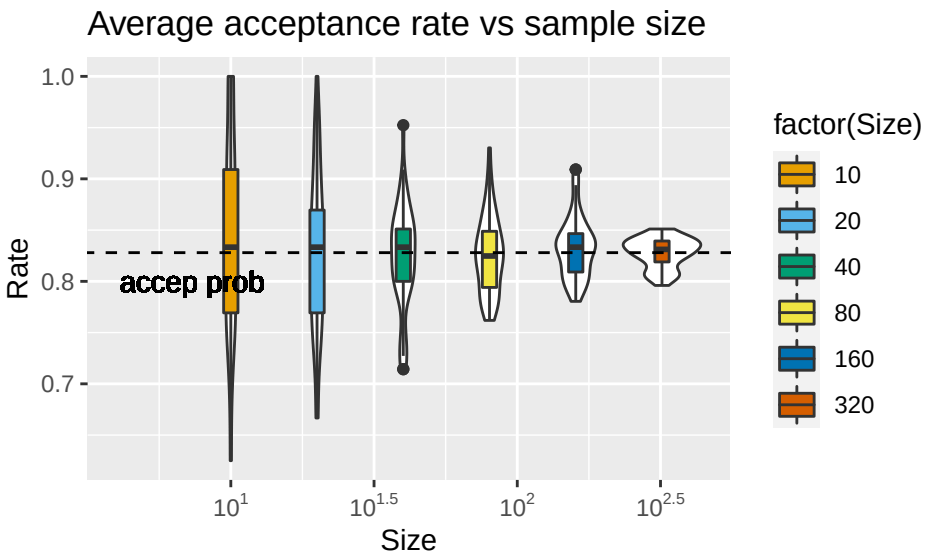
If we denote the acceptance probability as p, we can now determine the sampling distribution. Let Z denote the number of iterations required to obtain a sample size of n, or to accept n samples. In other words Z is the number of independent Bernoulli trials, each with success probability p, before obtaining n successes. This is similar to the negative binomial distribution, and has the following probability density function (Cook, John D. 2009):

$$f_Z(z) = \binom{z-1}{n-1} p^n (1-p)^{z-n} \quad z = n, n+1, n+2 \ldots \quad E(Z) = \frac{n}{p}$$

Hence, the expected number of iterations to require a sample size of n is 1.21n. Now let us look at what this means in practise - we will look at the acceptance rate of our algorithm required to obtain samples of different sizes. We will work with samples sizes of 10,20,40, and so on, and compute the average computational time for 50 samples for each size.

```
sizes <- c(10,20,40,80,160,320)
k = 50
myData = data.frame(matrix(nrow = 0, ncol = 2)) #initialise dataframe
for (i in 1:length(sizes)) {
  for (j in 1:k){
     myData <- rbind( myData, c(rejection(sizes[i])[[2]],sizes[i]) )
  }# each row contains a corresponding acceptance rate and sample size
}
names(myData) <- c("Rate", "Size")
```

```
ggplot(myData, aes(x = Size, y = Rate, group = Size, shape=factor(Size))) +
  geom_violin() + geom_boxplot(aes(fill=factor(Size)), width=0.05) + scale_x_log10(breaks = trans_breaks
              labels = trans_format("log10", math_format(10^.x))) + geom_abline(intercept = accep_prob,
```



Above we can see violin plots and box plots for each size. The advantage of doing both is that it is easier us for us to see the distribution of the data, rather than only the quartiles. A key trend that we observe is that as the sample size increases, there is a smaller variance in the acceptance rate and that the average acceptance rate becomes closer to to the actual acceptance probability. Intuitively this makes sense, but is necessarily true because as a result of the Law of Large Numbers,the acceptance rate converges in probability towards the acceptance probability as the number of iterations n tends to infinity.

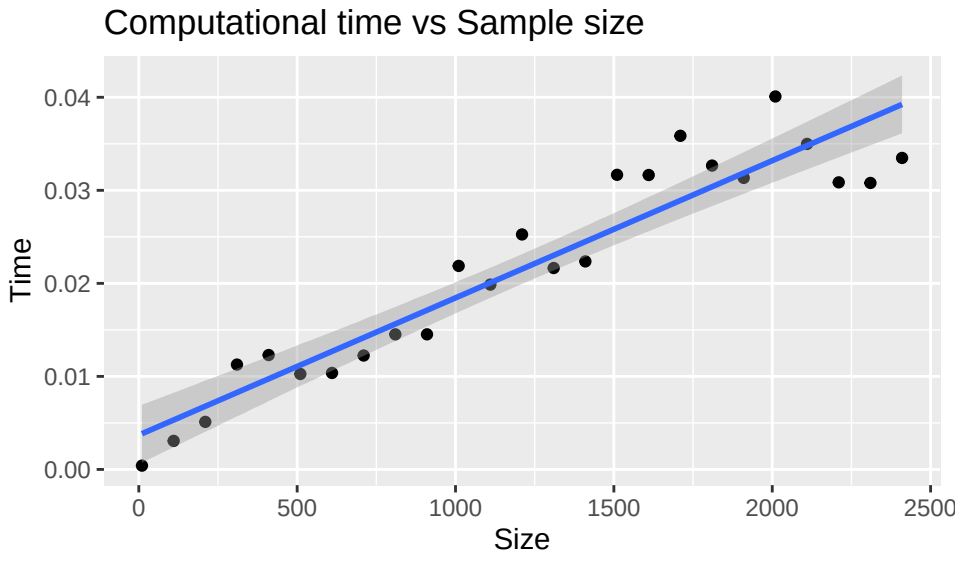
Computational time

In this section we investigate how the computational time of our algorithm varies as the sample size increases. In our algorithm we first have a finite number of operations and a while loop. For sufficiently large n the number of iterations that the while loop takes can be bounded above by a multiple of n, as the acceptance rate tends to the converge probability as n tends to infinity. Inside each iteration of while loop, we also have a finite number of operations as we only perform variable assignments, comparisons and numerical operations, all of which are 1 operation each. Therefore the total number of operations P for our algorithm, to create a sample size of n, is bounded above : $|P(N)| < m * N$ for some $m > 0$ and N sufficiently large, so $P(N) \sim O(N)$.

Below we work out the computational time of our code for different sample sizes, again taking the mean over each size to obtain more accurate values for the computational time.

```
sizes = seq(10,2500,100)
k = 25 #25 repeats for each sample size
time_vals <- data.frame(matrix(nrow=0,ncol=2))
#each row will contain a corresponding sample size and time

for (i in 1:length(sizes)){
  for (j in 1:k){
    p <- sizes[i]
    t0 <- Sys.time()
    x <- rejection(p)
    t1 <- Sys.time() #compute time taken to perform rejection
    time_vals <- rbind(time_vals, c(p,t1-t0))
  }
}
names(time_vals) <- c("Size", "Time")
```

```
time_means <- to_vec(for (i in sizes) mean(time_vals[time_vals["Size"] == i][(k+1):(2*k)]))
#use comprehension to calculate mean time for each sample size
df <- cbind(sizes, time_means)
df <- data.frame(df)
names(df) <-  c("Size", "Time") #convert to dataframe and plot
Rsquared <- round(summary(lm(df[,2] ~ df[,1]))$r.squared, 3)
ggplot(df, aes(x=Size, y=Time)) + geom_point() + geom_smooth(formula = y ~ x, method=lm) +
  labs(title="Computational time vs Sample size")
```



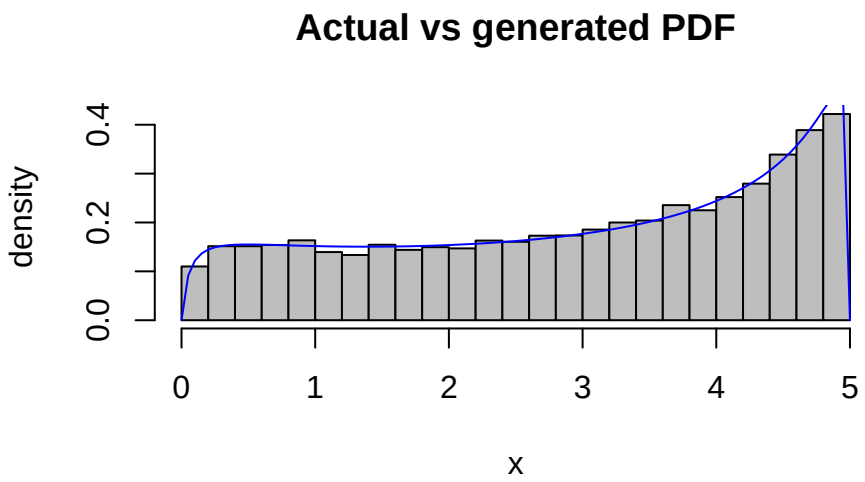
When plotting our data we fit a linear model, with coefficient of determination $R^2 = 0.891$, where R^2 can be interpreted as the fraction of the total variance that is explained by the model. Because R^2 is close to 1, our linear model is very reliable, so it is safe to assume that the computational time is directly proportional to the sample size we seek to generate. This exemplifies that the number of operations is of $O(N)$ and linearly increases. Generally it seems as though the computational time of our code is low, but we could improve on this if we used a different overlapping function g_Y. It is likely that the square root function, that is used when we generating samples with the inverse cumulative distribution, is going to be making the largest contribution to the computational time. Perhaps a function where the inverse cumulative density function is a low degree polynomial, could improve computational time as it would be less expensive to sample from Y. However, it is very difficult to find such a function that would also keep our acceptance probability high.

2: Verification of rejection scheme

In this section we will look at different types of plots and statistical tests to determine if our code can accurately generate sampling that come our desired distribution f_X.

Diagnostic plots

```
n = 10000 #sample size
data <- rejection(n)
data <- data[1][[1]] # extract only samples
hist(data, freq=FALSE, breaks = 30,  xlab = "x", ylab= "density",
     main = "Actual vs generated PDF", col = "grey") #histogram of sample
f <- function(x){f_star(x)/integrate(f_star,0,5)[[1]]}
curve(f, from=0, to=5, col = "blue", add = TRUE)
```

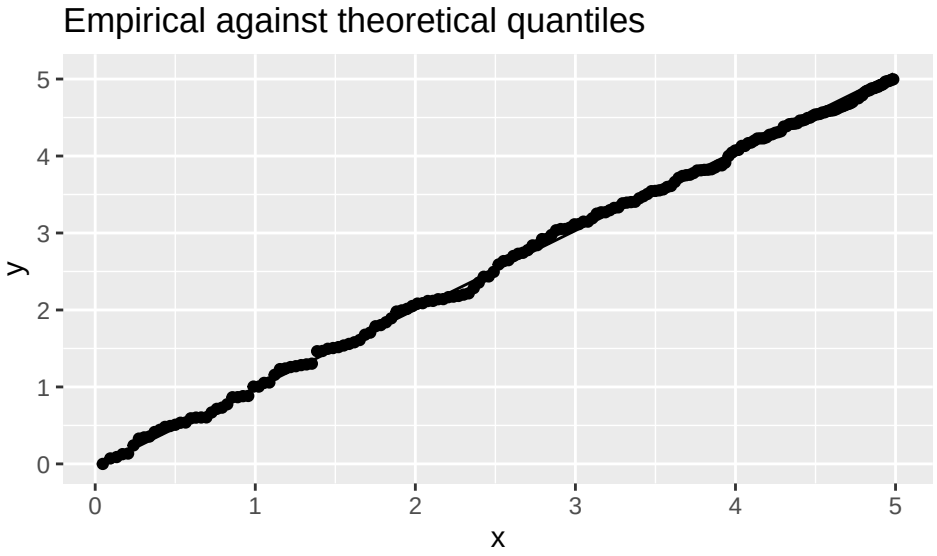



```
# plot (f = f*/(area under f*)) against sample, the actual pdf
```

Above we have a histogram of our samples from X and the probability distribution $f_X(x)$. We use the formula $f_X(x) = \frac{f_X^*(x)}{\int_0^5 f_X^*(x)dx}$ to plot the curve of the probability density function $f_X(x)$. We use the inbuilt R function to estimate the integral of f_X^* with absolute error less than 2×10^{-5}. With a sufficiently large sample size it appears that the data samples came from a distribution that is very similar to $f_X(x)$.

Now we will use a qqplot, a graphical way of comparing the quantiles of f_X and the data samples. As f_X is not one of the standard distributions we have to find the quantiles of f_X ourselves. We work with a sample size of 200 because if we worked with sample sizes of a higher order, the code would take too long to run. In our algorithm we can work with f_X^* to find the quaniles of f_X due to proportionality,

```r
n = 200

x = rejection(n)[1][[1]]
x_plot = data.frame(x=x) # generate sample and save into dataframe
mycdf = function(p,q){
c <- integrate(f_star,0,5)$value
#create function that finds quantiles by calculating where the CDF intersects
#with horizontal lines that hit y-axis at probabilities in [0,1]
integrate(f_star,0,p)$value/c-q }
qfunc <- function(q){
bisect(mycdf, 0, 5, q=q)$root
}
qhn1 <- function(p) unlist(lapply(p, qfunc))
#lapply applies vector to function
#qhn1 converts probabilities to quantile values
ggplot(x_plot, aes(sample=x))+
labs(title="Empirical against theoretical quantiles", xlab = "")+
stat_qq(distribution=qhn1) +
stat_qq_line(distribution=qhn1)
```

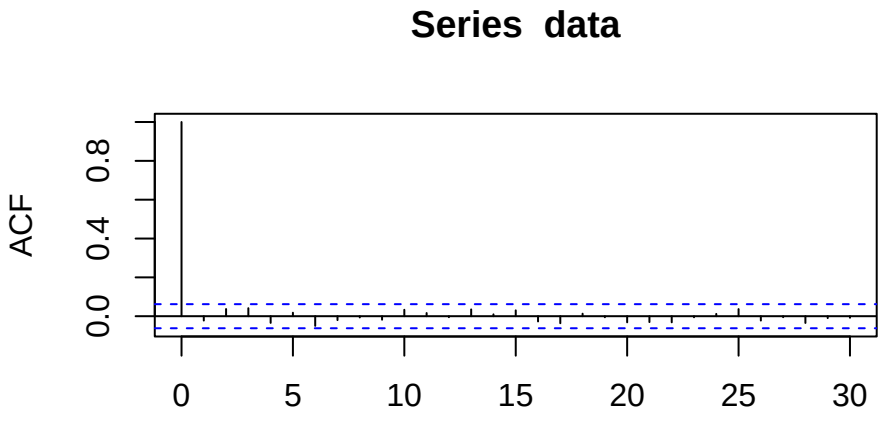


The data points are generally close to the line, so the sample quartiles and expected quartiles are almost equal so it is likely that the samples have distribution f_X. It appears as though there are minor oscillations above and below the line, but this most likely comes from the fact that we are taking a random sample of finite size; as if we increase the size we would yield greater accuracy.

We will now plot the autocorrelation, which is at the correlation between the sample and a translation of the sample by varying times, known as lags. In R we can calculate the sample autocorrelation at lag k, $\hat{p_k}$ if we know $X_1, .., X_n$. We define these estimators for the autocorrelation as follows:

$$\hat{p_k} = \frac{\sum_{i=k+1}^{N}(X_t - \bar{X})((X_{t-k} - \bar{X})}{(X_t - \bar{X})^2}$$

```r
n = 1000 # sample size 1000
data <- rejection(n)
data <- data[1][[1]]
acf(data)
```



Trivially, $p_0 = 1$ thus the sample autocorrelation at lag 0 is always 1 but the sample autocorrelation is very small for non-zero lags. The blue lines represent 95 percent asymptotic confidence intervals of the form $(\frac{-1.96}{\sqrt{N}}, \frac{1.96}{\sqrt{N}})$, where N is the sample size. Note that the 1.96 comes from the fact that we assume the sample autocorrelation is asymptotically normal: $P(|Z| < 1.96) = 0.95$ if $Z \sim N(0, 1)$. All the autocorrelation values fall within this region (or are close to). Therefore, we can say with high confidence that the autocorrelations of our sample are statistically insignificant- meaning that on average there is little to no relation between data points X_t, X_{t+k}. This is a key feature of an independant distribution.

Statistical tests

We now will perform the two sample Kolmogorov–Smirnov test (Zaiontz Charles 2001). We take two samples of equal length, $X_{1,1}, .., X_{1,N}, X_{2,1}, .., X_{2,N}$. We define piecewise cumulative distribution functions $F_1(x), F_2(x)$ and the test statistic D_N as follows:

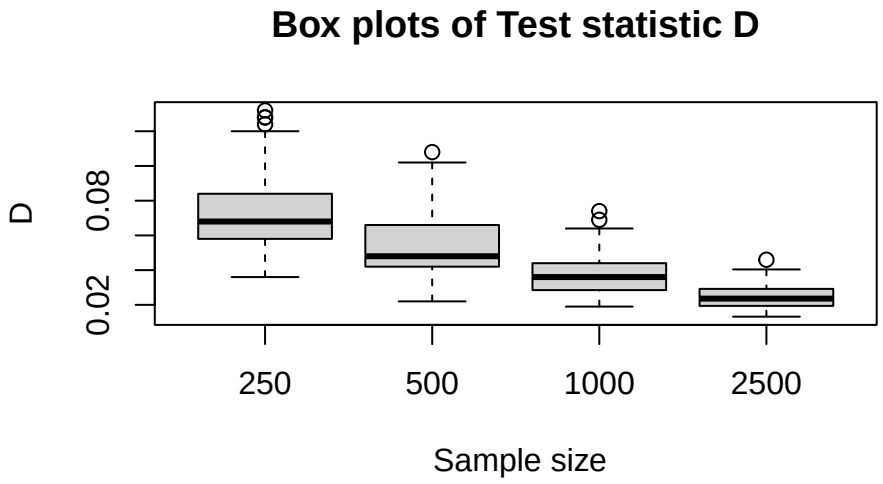
$$F_1(x) = \frac{1}{N}\sum_{i=1}^{n} I(X_{i,1} \leq x) \quad F_2(x) = \frac{1}{N}\sum_{i=1}^{n} I(X_{i,2} \leq x) \quad D_N = \sup_{x>0} |F_1(x) - F_2(x)|$$

If N is infinitely large and the samples came from the same distribution, we would expect our test statistic D_N to be 0. We perform multiple 2 sided tests at significance level 0.02 to see if D_N is statistically significant for varying sample sizes.

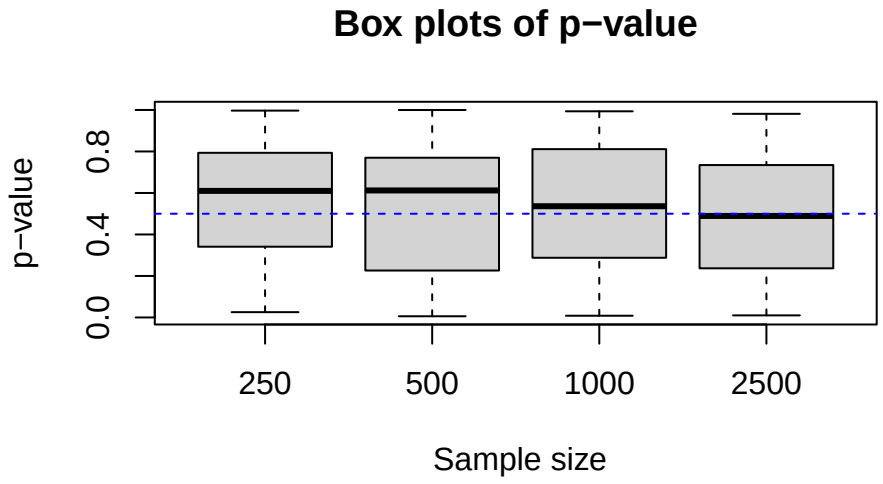
H_0 : Samples come from same distribution $\quad H_1$: Samples do not come from same distribution $\quad \alpha = 0.02$

```
size = c(250,500,1000,2500)
k = 100
d_vals <- matrix(nrow = 4, ncol = k) # each row contains simulations for a fixed sample size
p_vals <- matrix(nrow = 4, ncol = k)

for (i in 1:4){
  for (j in 1:k){ #iterate through every element of matrices
    sample_1 <- rejection(size[i])[1][[1]]
    sample_2 <- rejection(size[i])[1][[1]] # take 2 samples of same size
    suppressWarnings({test <- ks.test(sample_1, sample_2)})
    d_vals[i,j] = test$statistic
    p_vals[i,j] = test$p.value # run ks test and add p-value and test statistic to matrices
  }
}
```

```r
#transpose matrices so that a box plot is done on every row of matrices
boxplot(t(d_vals), main = "Box plots of Test statistic D",
        xlab = " Sample size", ylab = "D", names = size)
```




```r
boxplot(t(p_vals), main = "Box plots of p-value",
        xlab = " Sample size", ylab = "p-value", names = size)
abline(h=0.5, col="blue", lty=2) # horizontal line at p = 0.5
```



When looking at the box plot of our test statistic D, as we increase the sample size, the variance decreases and the mean decreases, getting closer to 0. The mean p-value for a given sample size also moves closer to 0.5, but the variance seems unaffected. We reject H_0 if $p < \frac{0.02}{2} = 0.01$ as our test is two sided. For all 400 p-values (100 for each size) our p-values do not fall in said region, therefore we do not reject H_0. It is safe to assume that the samples come from the same distribution.

Now that we know our samples come from the same distribution, we want to test if this distribution is f_X, using a goodness of fit test (Corder, Gregory W., and Dale I. Foreman 2014). Define as follows:

$$x_i = \frac{i}{10}, i = 0, .., 50 \quad O_i = \sum_{i=1}^{N} I(x_i < X_i \leq x_{i+1}) \quad E_i = \sum_{i=1}^{N} N(F_X(x_{i+1}) - F_X(x)) \quad T = \sum_{i=0}^{N} \frac{(O_i - E_i)^2}{E_i}$$

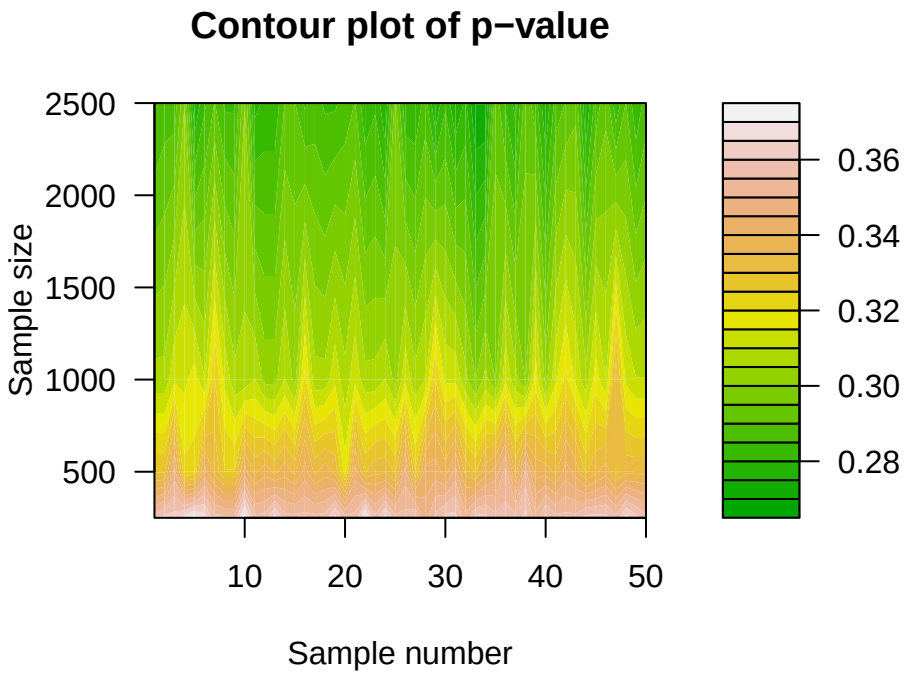
We split up the interval [0,5] into 50 equally sized intervals, O_i and E_i are respectively, the observed and expected number of data points from our sample X_i that lie in each interval. The test statistic T follows a chi-squared distribution with degrees of freedom $N - 1$.

H_0 : Observed and expected frequencies are equal H_1 : Observed and expected frequencies are not equalf_X $\alpha = 0.01$

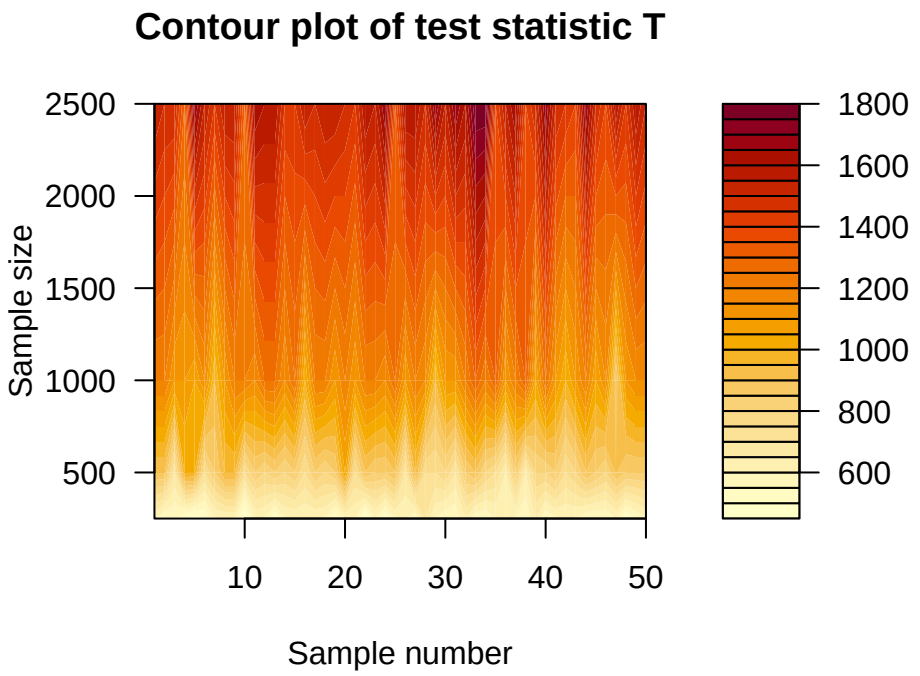
We now conduct the goodness of fit test for sample sizes 250,500,1000 and 2500, repeating 50 times for each size, plotting the test statisic and p-values.

```
f_cdf <- function(p, q) { integrate(f,p,q)[[1]] }
size = c(250,500,1000,2500)
k = 50
n = 100
grid <- seq(0,5,l=51)
#comprehension to calculate O_i,E_i
observed <- to_vec( for (i in 1:50) sum(data>grid[i] & data<=grid[i+1]) )
expected  <- to_vec( for (i in 1:50) n*f_cdf(grid[i],grid[i+1]) )

chi_values <- matrix(nrow = 4, ncol = k) # each row will represent a fixed sample size
p_values <- matrix(nrow = 4, ncol = k)

for (i in 1:4){
  for (j in 1:k){ #iterate through every element of matrices
    sample <- rejection(size[i])[1][[1]]  # take 2 samples of same size
    observed <- to_vec( for (i in 1:50) sum(sample>grid[i] & sample<grid[i+1]) )
    expected  <- to_vec( for (i in 1:50) n*f_cdf(grid[i],grid[i+1]) )
    suppressWarnings({test <- chisq.test(observed, expected)})
    chi_values[i,j] = test$statistic
    p_values[i,j] = test$p.value # run ks test and add p-value and test statistic to matrices
  }
}
 # suppress warnings to prevent error message that chi-squared approximation may be incorrect
```

```
filled.contour(x = 1:50, y = size, t(p_values), color.palette = terrain.colors,
               xlab = "Sample number", ylab= "Sample size", main = "Contour plot of p-value")
```



```
filled.contour(x = 1:50, y = size, t(chi_values), xlab = "Sample number",
               ylab= "Sample size", main = "Contour plot of test statistic T")
```




```
t_mean <- round(rowMeans(chi_values),3) #calculate statistics for each size
p_mean <- round(rowMeans(p_values),3)
t_var <- round(rowVars(chi_values),0)
p_var <- round(rowVars(p_values),6)
p_min <- to_vec(for (i in 1:4) round(min(p_values[1,i]),3))
res <- cbind(size, t_mean, p_mean, t_var, p_var, p_min)
kable(res, col.names =c("Size","Mean(t)", "Mean(p)","Std(t)","Std(p)","Min(p)"))
```

Size	Mean(t)	Mean(p)	Std(t)	Std(p)	Min(p)
250	570	0.360	2653	3.0e-05	0.357
500	831	0.335	7438	5.5e-05	0.362
1000	1145	0.311	8699	4.7e-05	0.362
2500	1543	0.285	12960	4.7e-05	0.367

As expected, the general trend is that as the sample size increases, the sample becomes more representative of f_X and the p-value decreases, meaning the corresponding test statistic is less extreme, and we have more reason to believe the null hypothesis is true. As this is a one sided test, we reject the null hypothesis if $p < \alpha = 0.01$. From the table we can see that all our p-values are greater than 0.35, hence we have insufficient evidence to reject the null hypothesis. This leads us to believe samples actually come from the correct distribution.

3: Monte Carlo procedure

In this section we seek to estimate $\theta = \int_0^5 f^*(x)dx$ using the Monte Carlo integration, so that we can calculate the re-scaled probability density function using that $f(x) = \frac{f(x)}{\int_0^5 f^*(x)dx}$. We will look at 4 unbiased estimators and compare their variances. To calculate the variances it will be necessary to use R's built in integration

function.

1) Crude Monte Carlo

We will use the random variable $U \sim U(0,5)$ as it has a probability density function that is constant on $[0,5]$ with probability $\frac{1}{5}$.

$$\theta = \int_0^5 f^*(x)dx = \int_0^5 \frac{1}{5} 5 f_X^*(x) = E_U(5 f_X^*(U))$$

Set $\hat{\theta_1}$ to be the sample mean as this is unbiased, so $\hat{\theta_1} = \frac{1}{N}\sum_{i=1}^N 5 f_X^*(U_i)$ where U_i are independent samples from U. Note that $E(\hat{\theta_1}) = \frac{1}{N} N E(5 f_X^*(U)) = \theta$. Below we devise a function that generates $\hat{\theta_1}$ with N samples and calculate its variance when $N = 1$, and repeat this for the next estimators.

```
#Take uniform on (0,5), Monte Carlo
monte1 <- function(N){
u <- runif(N)*5
theta1 <- sum(5*f_star(u))/N
theta1
}
var1 <- integrate(k2 <- function(x){5*f_star(x)^2},0,5)[[1]]-integrate(f_star,0,5)[[1]]
```

$$Var(\hat{\theta_1}) = \frac{1}{N} Var(5 f_X^*(U)) = \frac{1}{N}(\int_0^5 25 (f_X^*(u))^2 \frac{1}{5} du - \theta) = \frac{1}{N}(\int_0^5 5 (f_X^*(u))^2 du - \theta) = \frac{0.0817..}{N}$$

2) Importance Sampling

We will attempt to find another estimator by working with a generalization of the arcsine distribution. This is because it is easy to calculate the cumulative density function, to invert it and sample from. We define as follows:

$$\phi(x) = \pi\sqrt{x(5-x)} f_X^*(x) \qquad g_L(x) = \frac{1}{\pi\sqrt{x(5-x)}} \quad \text{for x} \in (0,5)$$

Then represent our integral in terms of these functions:

$$\theta = \int_0^5 f^*(x)dx = \int_0^5 \frac{1}{\pi\sqrt{x(5-x)}} \pi\sqrt{x(5-x)} f^*(x)dx = \int_0^5 g_L(x)\phi(x)dx = E_L(\phi(L))$$

Again we can find an estimator $\hat{\theta_2}$ by taking the sample mean of $\phi(L)$. $\hat{\theta_2} = \overline{\phi(L)} = \frac{1}{N}\sum_{i=1}^N \phi(L_i)$ where $L_1, .., L_n$ are independent samples from L. To generate these samples we first calculate the CDF of L and its inversion:

$$G_L(x) = \int_0^x \frac{1}{\pi\sqrt{u(5-u)}} du = \frac{2}{\pi}\arcsin(\sqrt{\frac{u}{5}})\Big|_0^x = \frac{2}{\pi}\arcsin(\sqrt{\frac{x}{5}}) \implies G_L^{-1}(x) = 5\sin^2(\frac{\pi x}{2})$$

If $U \sim U(0,1)$ is a standard uniform variable, $G_L^{-1}(U) \sim L$. We implement this into the sample code below to calculate $\hat{\theta_2}$ with z samples.

```
phi <- function(x) {pi*((x)*(5-x))**0.5*f_star(x)}
l_inv <- function(x) {5 * sin(pi*x/2)**2}

monte2 <- function(N){
samp <- l_inv(runif(N))
theta2 <- sum(phi(samp))/N
theta2
```

```
}
var2 <- integrate(k2 <- function(x){f_star(x)**2*pi*sqrt(x*(5-x))},0,5)[[1]]-integrate(f_star,0,5)[[1]]
```

$$Var(\hat{\theta_2}) = \frac{1}{N}Var(\phi(L)) = \frac{1}{N}(\int_0^5 \pi\sqrt{x(5-x)}(f_X^*(u))^2du - \theta) = \frac{0.1463..}{N}$$

3) Antithetic variates

The variance of $\hat{\theta_2}$ is greater than $\hat{\theta_1}$, so we will try to apply variance reduction techniques to the crude Monte Carlo estimator $\hat{\theta_1}$. If $U \sim U(0,5)$, then clearly $5-U \sim U(0,5)$ as both these variables have a probability density function that takes value $\frac{1}{5}$ on the interval $[0,5]$. Therefore:

$$\theta = E(\hat{\theta_1}) = E_U(5f_X^*(U)) = E_{5-U}(5f_X^*(5-U)) = E_U(5f_X^*(5-U))$$

We can use this to generate new unbiased estimators,

$$\hat{\theta_1'} = \frac{1}{N}\sum_{i=1}^{N} 5f_X^*(5-U_i) \quad \hat{\theta_3} = \frac{\hat{\theta_1}+\hat{\theta_1'}}{2} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(5f_X^*(U_i)+5f_X^*(5-U_i)) \quad U_i \overset{i.i.d}{\sim} U(0,5)$$

Below we implement this and calculate our estimator $\hat{\theta}$ for a given N, and prove after that it has a much smaller variance than $\hat{\theta_1}$

```
monte3 <- function(N){
u <- runif(N)*5
theta <- 2.5*sum(f_star(u)+f_star(5-u))/N
theta
}

var <- 0.5*var1 +0.5*(integrate(z <- function(x){5*f_star(x)*f_star(5-x)},0,5)[[1]] - integrate(f_star,
```

$$Var(\frac{5f_X^*(U)+5f_X^*(5-U)}{2}) = \frac{1}{4}(Var(5f_X^*(U)) + Var(5f_X^*(5-U)) + 2Cov(5f_X^*(U), 5f_X^*(5-U)))$$

$$= \frac{1}{2}Var(f_X^*(U)) + \frac{1}{2}Cov(5f_X^*(U), 5f_X^*(5-U)) = \frac{1}{2}Var(f_X^*(U)) + \frac{1}{2}(\int_0^5 \frac{25}{5}f_X^*(x)f_X^*(5-x)dx - \theta^2) = 0.00781..$$

$$\implies Var(\hat{\theta_3}) = Var(\sum_{i=1}^{N}\frac{5f_X^*(U_i)+5f_X^*(5-U_i)}{2}) = \frac{1}{N^2}NVar(\frac{5f_X^*(U)+5f_X^*(5-U)}{2}) = \frac{0.00781..}{N}$$

This gives us a significantly smaller variance than the original estimator $\hat{\theta}$, by over a factor of 10 when N is fixed.

4) Hit or Miss

We will use another method of obtaining an estimator for θ. Let $k = \max_{x\in[0,5]} f_X^*(x)$, $U \sim U(0,5)$ and $V \sim U(0,k)$.

$$\hat{\theta_4} = \frac{5k}{N}\sum_{i=1}^{N} I(V_i \le f^*(U_i)) \quad , E(\hat{\theta_4}) = \frac{5k}{N}N\,P(I(V_i \le f^*(U_i)) = 1) = 5k\frac{\text{Area under f*}}{5k} = \int_0^5 f_X^*(x)dx = \theta$$

Our estimator is unbiased like our other 2 estimators, but ideally our estimator will have a variance that is lower given that the sample size is fixed. Below is the calculation for $\hat{\theta_3}$.

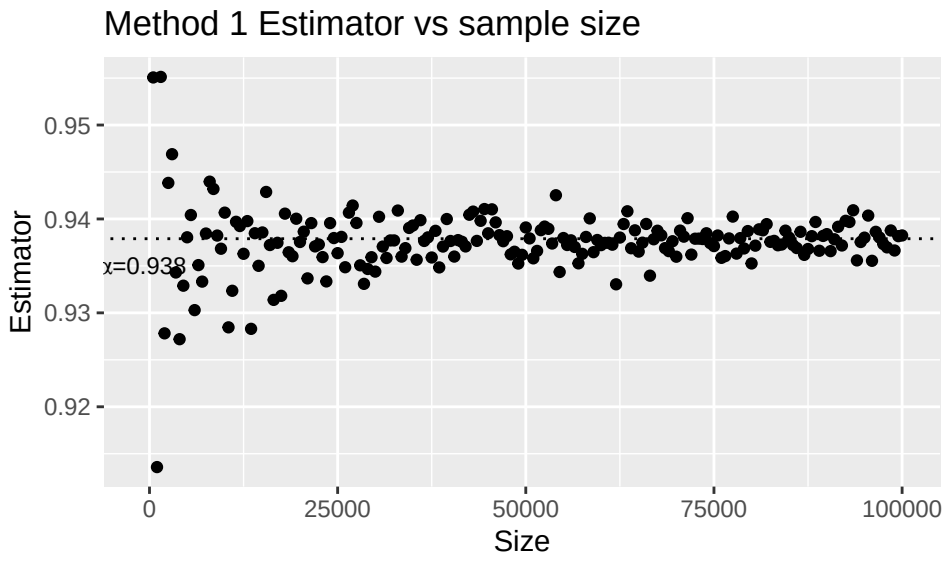
```
fmax<- optimize(f_star, c(0,5), maximum = TRUE)[[1]]
monte4 <- function(N){
k <- 10000
u <- 5 * runif(k)
v <- fmax*runif(k)
theta3 <- fmax*5*sum(v < f_star(u))/k
theta3
}
var2 <- (integrate(f_star,0,5)[[1]])*(5*fmax-integrate(f_star,0,5)[[1]])
```

$$Var(\hat{\theta_4}) = (\frac{5k}{N})^2 NVar(I(V_i \leq f^*(U_i))) = \frac{(5k)^2}{N}\frac{\theta}{5k}(1-\frac{\theta}{5k}) = \frac{\theta(5k-\theta)}{N} = \frac{22.1335..}{n}$$

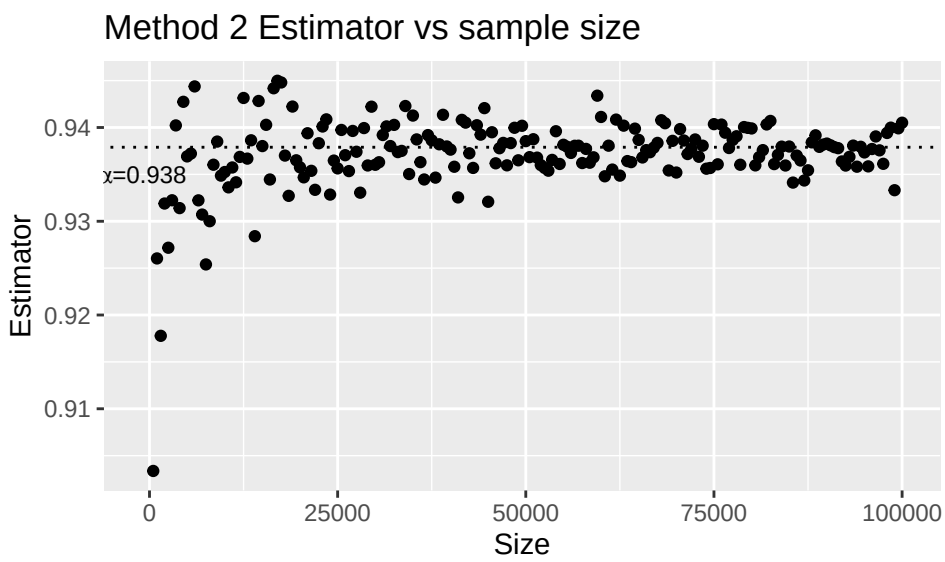
The variance is by far the largest for the Hit and Miss Monte-Carlo method as it is larger by a factor of 10 compared to the variance for $\hat{\theta_1}$ and $\hat{\theta_2}$, so we should avoid from using this method. Let is investigate what values our estimators take as we vary sample size and compare our findings with the estimator variances. We will now plot estimators $\hat{\theta_i}$ for N up to 100,000, and use only the first 3 estimators in a boxplot to be able to make visual interpretations - the variance of $\hat{\theta_4}$ very large so it would be difficult to compare the other 3.

```
sizes = seq(500,100000,500)
estimators1 <- data.frame(matrix(nrow=0,ncol=2))
estimators2 <- data.frame(matrix(nrow=0,ncol=2))
estimators3 <- data.frame(matrix(nrow=0,ncol=2))
estimators4 <- data.frame(matrix(nrow=0,ncol=2))

for (i in 1:length(sizes)){
    p <- sizes[i]
    estimators1 <- rbind(estimators1, c(monte1(p),p))
    estimators2 <- rbind(estimators2, c(monte2(p),p))
    estimators3 <- rbind(estimators3, c(monte3(p),p))
    estimators4 <- rbind(estimators4, c(monte4(p),p))
}
names(estimators1) <- c("Estimator", "Size")
names(estimators2) <- c("Estimator", "Size")
names(estimators3) <- c("Estimator", "Size")
names(estimators4) <- c("Estimator", "Size")
```

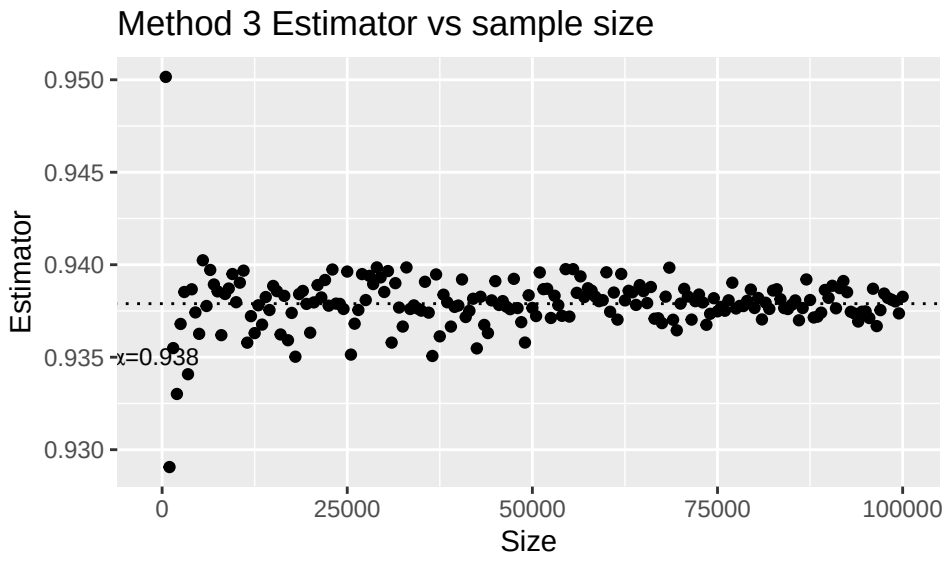
```
ggplot(estimators1, aes(x=Size, y=Estimator)) + geom_point() +  labs(title="Method 1 Estimator vs sampl
```



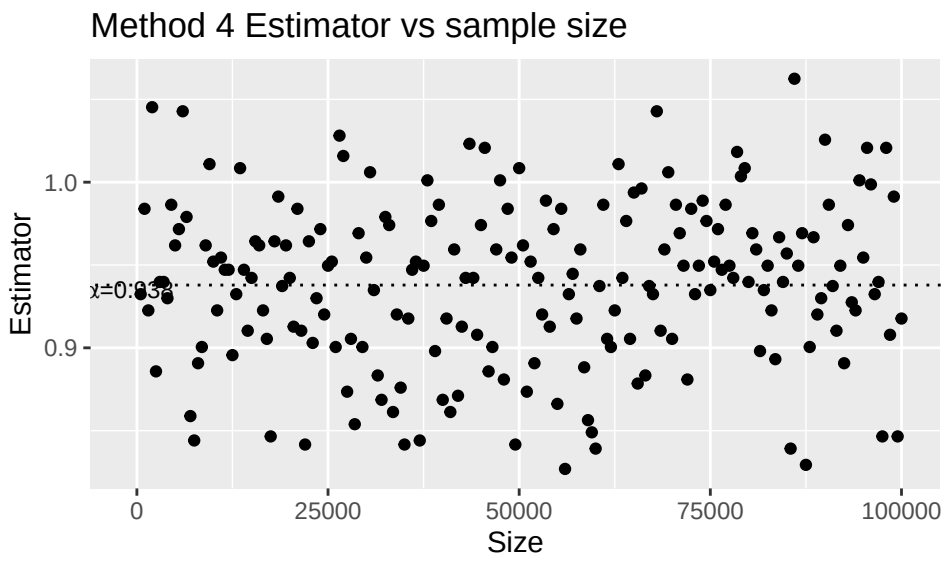

```
ggplot(estimators2, aes(x=Size, y=Estimator)) + geom_point() + labs(title="Method 2 Estimator vs sample
```



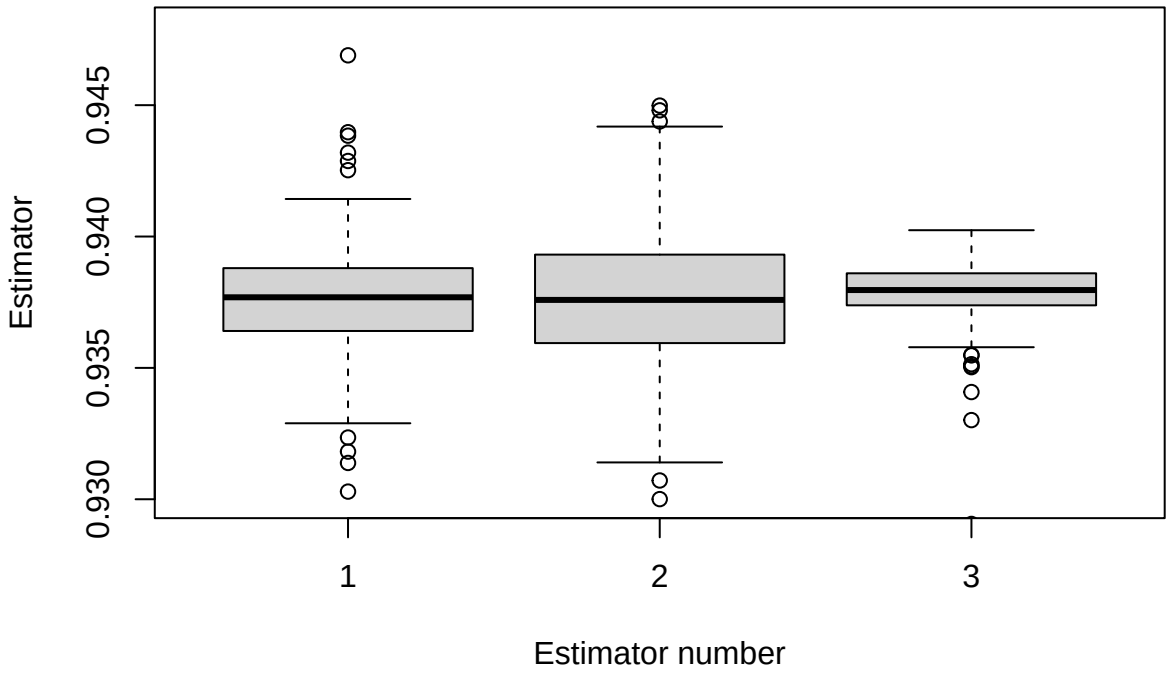

```
ggplot(estimators3, aes(x=Size, y=Estimator)) + geom_point()  +labs(title="Method 3 Estimator vs sample
```




```
ggplot(estimators4, aes(x=Size, y=Estimator)) + geom_point() +  labs(title="Method 4 Estimator vs sample
```




```
boxplot(cbind(estimators1[,1],estimators2[,1],estimators3[,1]), xlab = "Estimator number", ylab= "Estima
```



For $i = 1, .., 4$ the estimators $\hat{\theta}_i$ are unbiased. As $Var(\hat{\theta}_i) = \frac{K}{N}$ for some constant K, $\lim_{N\to\infty} Var(\hat{\theta}_i) = 0$ hence $\hat{\theta}_i$ is consistent. All 4 estimators converges in probability to θ and we can see this in the plots for the first three estimators; as the sample size increases the estimator approaches θ. We do not visually see this trend for $\hat{\theta}_4$ which is scattered in between 0.8 and 1.05, we would need to further increase N. As $\hat{\theta}_3$ has the smallest variance, the estimators for this method approaches θ the fastest.

References

1. Angus, J. E. "The Probability Integral Transform and Related Results." SIAM Review, vol. 36, no. 4, 1994, pp. 652–654., https://doi.org/10.1137/1036146.
2. Cook, John D. 2009. *Notes on the Negative Binomial Distribution.* pg 1 https://www.johndcook.com/ /negative_binomial.pdf [Accessed 18th November 2021]
3. Corder, Gregory W., and Dale I. Foreman 2014. Nonparametric Statistics: a Step-by-Step Approach. Wiley, pg 173.
4. Zaiontz Charles 2001. *Two-Sample Kolmogorov-Smirnov Test.* https://www.real-statistics.com/non-parametric-tests/goodness-of-fit-tests/two-sample-kolmogorov-smirnov-test/ [Accessed 2nd December 2021]