Coursework 1: Stochastic Simulation

Hermine Tranié

11/12/2019

Abstract

In this project, we are simulating random variates in R from a given distribution. We are using the rejection method, with and without squeezing and analyzing the effect of these methods on acceptance probability and running time. We are also analyzing how well our generated data fits the theoritical distribution using various statistics tests including Kolmogorow-Smirnov and Chi-Squared bin test and diagnostic plots. Finally, we are using Monte Carlo methods and others in order to find an estimator of our given distribution normalizing constant, and how to reduce the variance of the said estimator.

Introduction: finding our pdf parameter values

We will first find our parameter values for our density function as per the instructions using my CID = 01400919:

```
library(numbers)
lpm_hmt=max(primeFactors(1400919))
```

We have that the largest prime factor of my CID is 2113 which means, according to the table, that my parameter values are: $a = 0$, $b = 5$, $c = 6$ & $d = 0.60$. This means that my pdf is defined as follows:

$$f_X(x) \propto \begin{cases} \frac{1}{x(5-x)} \exp\{-\frac{1}{6}(0.60 + \log(\frac{x}{5-x}))^2\} & 0 < x < 5; \\ 0, & \text{otherwise.} \end{cases}$$

Let's call this function $f_X^*(x)$.

Find theorical value of normalising constant k

We know that this is the pdf up to proportionality, so let's evaluate its normalizing constant. In order to do so, we can first note the definition of the normal distribution:

$$f_X^{norm}(s) = \frac{1}{\sigma\sqrt{(2\pi)}} e^{-\frac{1}{2}(\frac{s-\mu}{\sigma})^2}, \ -\infty < s < +\infty$$

We have $\int_0^1 f_X^{norm}(s) = 1$ and $X \sim N(\mu, \sigma^2)$ where $N(\cdot)$ indicates the normal distribution.

Now, let's go back to our $f_X^*(x)$ and integrate it on its support to find the value of k:

$$\int_0^5 f_X^*(x)dx = \int_0^5 \frac{1}{x(5-x)} \exp\{-\frac{1}{6}(0.60 + \log(\frac{x}{5-x}))^2\}dx$$

Let's do 2 changes of variable in order to express $f_X^*(x)$ as a normal distribution:

The first one as follows:

$$u = \frac{x}{5-x} \implies \frac{du}{dx} = \frac{5-x+x}{(5-x)^2} = \frac{5}{(5-x)^2} \implies dx = \frac{(5-x)^2}{5}du$$

This change of variables also updates our integral limits: for the bottom limit, $u = \frac{0}{5-0} = 0$, for the top limit, $u = \frac{5}{0} = +\infty$, which yields:

$$\int_0^5 f_X^*(x)dx = \int_0^{+\infty} \frac{1}{x(5-x)} \exp -\frac{1}{6}\left(0.60 + \log\left(\frac{x}{5-x}\right)\right)^2 \frac{(5-x)^2}{5} du = \frac{1}{5}\int_0^{+\infty} \frac{1}{u} \exp\{-\frac{1}{6}(0.60+\log(u))^2\} du$$

The second one as follows:

$$w = \log(u) \implies \frac{dw}{du} = \frac{1}{u} \implies du = udw$$

This change of variables also updates our integral limits: for the bottom limit, $w = \log(0) = -\infty$, for the top limit, $w = \log(+\infty) = +\infty$, which yields:

$$\int_0^5 f_X^*(x)dx = \frac{1}{5}\int_{-\infty}^{+\infty} \frac{1}{u} \exp\{-\frac{1}{6}(0.60+w)^2\} udw = \frac{1}{5}\sqrt{(}6\pi) \int_{-\infty}^{+\infty} \frac{1}{\sqrt{(}2\pi)\sqrt{3}} \exp\{\frac{-(0.60+w)^2}{2.3}\} dw = \frac{1}{5}\sqrt{(}6\pi)$$

as $\int_{-\infty}^{+\infty} f_X^{norm}(w)dw = 1$ by identifying our parameters from the normal distribution defined above: $\mu = -0.6$ and $\sigma^2 = 3 \implies \sigma = \sqrt{3}$.

Hence if we want k such that: $\int_0^5 kf_X^*(x)dx = 1 \implies k = \frac{5}{\sqrt{(}6\pi)}$, we can write that the valid pdf f_X(x) as:

$$f_X(x) = \begin{cases} \frac{5}{\sqrt{(}6\pi)} \frac{1}{x(5-x)} \exp\{-\frac{1}{6}(0.60 + \log(\frac{x}{5-x}))^2\} & 0 < x < 5; \\ 0, & \text{otherwise.} \end{cases}$$

QUESTION 1: Generate using rejection our distribution

Generating from $f_X^*(x)$ using Rejection

The general rejection sampling algorithm to simulate a random variable $X \sim f_X(x)$, requires specification of an envelope distribution $G_Y(y)$, whose range contains the range of X and such that $G_Y(y) = Mg_Y(y) \geq f_X(x)$, $\forall x$, where M is defined to be this in our algorithm (alghough any upper bound would work, the supremum will give the fastest algorithm) $M = \sup_x \frac{f_X(x)}{g_Y(x)}$. Once, we have specified $g_Y(y)$ and M, the general rejection algorithm is given below. [1]

General Rejection Algorithm:

1. Generate $U = u \sim U(0,1)$.
2. Generate $Y = y \sim g_Y(y)$.
3. If $u \leq \frac{f_X(y)}{Mg_Y(y)}$, set $X = y \sim f_X(\cdot)$.
4. Otherwise GOTO 1.

The algorithm also applies if we only know $f_Y^*(x) \propto f_X(x)$ and/or $g_Y^*(y) \propto g_Y(y)$ for the specified ranges of the densities.

If we have,

$$f_X^*(x) \propto \frac{1}{x(5-x)} \exp\left(-\frac{1}{6}\left(0.60 + \log\left(\frac{x}{5-x}\right)\right)^2\right) \quad 0 < x < 5$$

and we have the following g_1 & g_2, that we have picked in order to compare which one has the best accuracy and efficiency, with g_1 being a standord uniform as follows,

$$g_{1_Y}^*(x) = 0.3465,\ 0 \le x \le 5 \implies g_{1_Y} = \frac{1}{5},\ 0 \le x \le 5$$

and g_2 is a more refined fitting curve. Note: I have picked g_2^* by numerical analysis. I had first picked an envelope that had a 73% acceptance probability but I thought that I could do better so I constructed a closer fitting function as follows:

$$g_{2_Y}^*(x) = \begin{cases} 11x & 0 \le x < m_1 \\ 0.36 - \frac{1}{20}x & m_1 \le x < m_2 \\ 5 - x & m_2 \le x \le 5 \\ 0, & \text{otherwise.} \end{cases}$$

Then, we can proceed with the algorithm.

Note that we have found m_1 & m_2 as follows and we have kept them as this for accuracy purposes in our algorithm:

$$11m_1 = 0.36 - \frac{1}{20}m_1 \implies 220m_1 = 7.2 - m_1 \implies 221m_1 = 7.2 \implies m_1 = \frac{7.2}{221}$$
$$0.36 - \frac{1}{20}m_2 = 5 - m_2 \implies -4.64 = -\frac{19}{20}m_2 \implies m_2 = \frac{4.64 * 20}{19}$$

```
#Initialize all major functions and values for this code
#1. Define the distribution we want to simulate from
  fstar <- function(x) 1/(x*(5-x)) * exp(-1/6 * (0.6 + log(x/(5-x)))^2) #Define our function f_star
  k_fstar <- 5/sqrt(6*pi) #Theoritical normalizing constant for fstar
  f <- function(x) k_fstar * fstar(x) #Valid pdf

#2. Define our first function: uniform g1 which we will use for a uniform envelope
  g_1star <- function(y) 0.3465
  k_gstar1 = 1/(5*0.3465) #Compute normalizing constant of g1_star
  g1 <- function(y) g_1star(y) * k_gstar1 #Define g1

#3. Define our second function: g2 which we will use for a second envelope
  #Define the pieces of our function g2 as follows
    g_21 <- function(x) 11*x
    g_22 <- function(x) 0.36 - 1/20 * x
    g_23 <- function(x) 5-x
  #Define the intervals for which g2 is defined
    m1 <- 0.36 * 20 / 221
    m2 <- 4.64 * 20 /19
  #Concetenate g2_star
  g_2star <- function(y){
  ifelse(y <= m1, g_21(y),
         ifelse(y<=m2, g_22(y),g_23(y)))}
  k_gstar2 <- 1/integrate(g_2star,0,5)$value #Compute normalizing constant of g2_star
  g2 <-function(y) g_2star(y) * k_gstar2 #Define g2
```

Compute the M using numerical computation

Using the M we have defined above to be $M_{1,2} = \sup_x \frac{f^*_X(x)}{g^*_{1,2_Y}(x)}$, it yields:

$$M_1 = sup_{0\le x\le 5}\frac{f^*_X(x)}{g^*_{1_Y}(x)} = \sup_{0\le x\le 5}\frac{1}{0.3465x(5-x)}\exp\{-\frac{1}{6}(0.60+\log(\frac{x}{5-x}))^2\} \simeq 1.73$$

$$M_2 = \sup_{0\le x\le 5}\frac{f^*_X(x)}{g^*_{2_Y}(x)} = \sup_{0\le x\le 5}\frac{1}{k_{gstar2}x(5-x)^2}\exp\{-\frac{1}{6}(0.60+\log(\frac{x}{5-x}))^2\} \simeq 1.04.$$

```r
sup1 <- function(x) fstar(x)/g1(x)
sup2 <-function(x) fstar(x)/g2(x)

M_1 <- optimize(sup1,c(0,5), maximum=TRUE)$objective
M_2 <- optimize(sup2,c(0,5), maximum=TRUE)$objective
```

Acceptance schemes

By definition from our lecture notes, we have the following definition for the theoritical acceptance probability of the scheme: $P[acceptX] = \int_{-\infty}^{\infty} h(y)g(y)dy$. Choose $h(y) = \frac{f^*_X(y)}{Mg_Y(y)}$ Hence it yields:

$$P[acceptX] = \int_0^5 \frac{f^*_X(y)}{Mg_Y(y)}g_Y(y)dy = \frac{1}{M}\int_0^5 f^*_X(y)dy$$

The acceptance probability, θ_1 and θ_2, of the algorithm are then:

$$\theta_1 = \frac{1}{M_1}\int_0^5 f^*_X(y)\mathrm{d}y = \frac{1}{M_1}\int_0^5 \frac{1}{y(5-y)}\exp\{-\frac{1}{6}(0.60+\log(\frac{y}{5-y}))^2\}\mathrm{d}y = \frac{\sqrt(6\pi)}{5M_1} = 0.50$$

$$\theta_2 = \frac{1}{M_2}\int_0^5 f^*_X(y)\mathrm{d}y = \frac{1}{M_2}\int_0^5 \frac{1}{y(5-y)}\exp\{-\frac{1}{6}(0.60+\log(\frac{y}{5-y}))^2\}\mathrm{d}y = \frac{\sqrt(6\pi)}{5M_2} = 0.83$$

```r
theta_g1 = 1/M_1 * 1/k_fstar
theta_g2 = 1/M_2 * 1/k_fstar
```

By definition from our lecture notes, the acceptance region as follows:$u \le \frac{f^*_X(y)}{Mg^*_Y(y)}$. For simplification of calculation purposes, we will only be calculating the said condition numerically in our code.

Squeezing for g_2

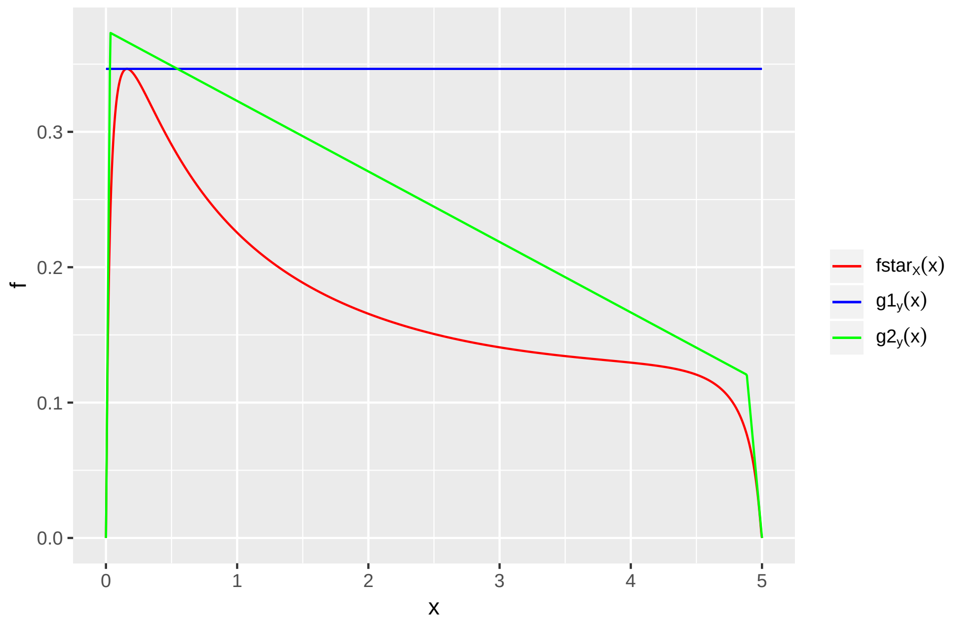
The principle of squeezing is based on the fact that we feed our rejection algorithm an easier condition to satisfy for the generated data to belong to our wanted distribution, before checking the other more computationally demanding conditions. This will save a lot of time as we pick our squeeze close to our distribution so a lot of values will be accepted.
Here I have only picked a bottom squeeze because if the condition has too many logical statements, it ends up being slower than without the squeeze (which defeats the principle of introducing a squeeze).
We want to find 1 function $W_L(x)$ (which doesn't have to be a valid pdf) such that: $W_L(x) \le \frac{f^*_X(x)}{g^*_{2_Y}(x)}$.
We can define it in our code as follows.

```r
wl <- function(x) ifelse(x >= 0.02 & x <= 4.98, 0.739, 0) #Define lower squeeze function
```

Let's now visualize all the important defined functions up to now:

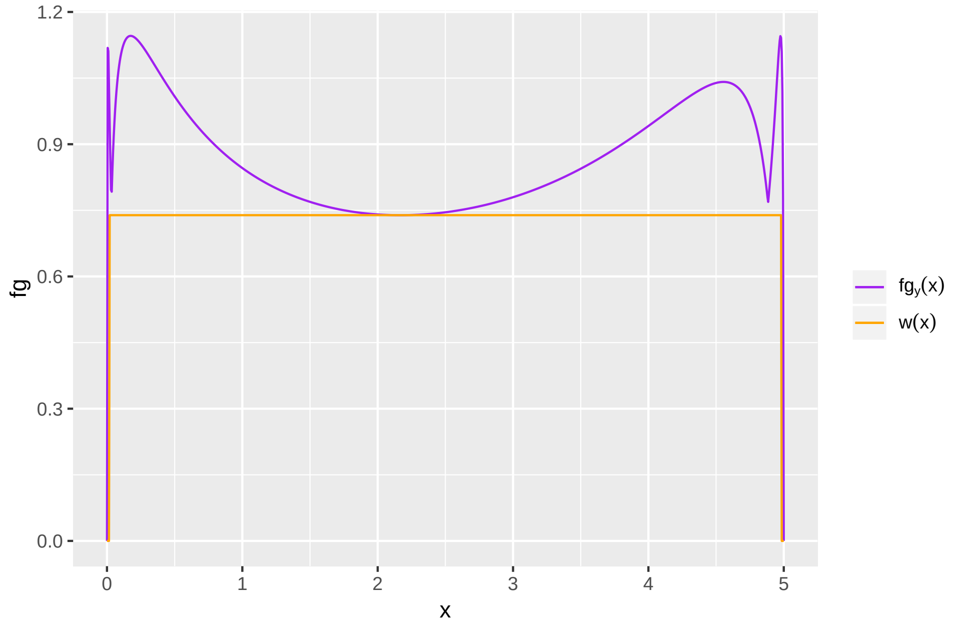
```
#Plot all our functions f_star, g_1star, g_2star
library(ggplot2)
library(plyr)
x <- seq(0+10^-8, 5-10^-8, l=1000) #Define interval for x
x_plot <-data.frame(x=x,f=fstar(x),g1=M_1*g1(x), g2=M_2*g_2star(x)) #Store values
mylabs=list(expression(fstar[X](x)), expression(g1[y](x)), expression(g2[y](x)))
cols <- c("f"="red","g1"="blue", "g2"="green")
p <- ggplot(x_plot)
 p + geom_line(aes(x,f,colour="f"))+
   geom_line(aes(x,g1,colour="g1"))+
   geom_line(aes(x,g2,colour="g2"))+
   scale_colour_manual("", values=cols, labels=mylabs)+
   labs(title= "Fig 1. Distribution and its envelopes")
```

Fig 1. Distribution and its envelopes



Observations for Figure 1: Here we have our distribution up to proportionality (in red) from which we want to generate data. We also have our uniform envelope in dark blue and our more refined envelope in light green.

```
#Plot squeeze wl
library(ggplot2)
library(plyr)
x <- seq(0+10^-8, 5-10^-8, l=1000) #Define interval for x
x_plot <-data.frame(x=x,fg=sup2(x),w=wl(x)) #Store values
mylabs=list(expression(fg[y](x)), expression(w(x)))
cols <- c("fg"="purple", "w"="orange")
p <- ggplot(x_plot)
 p +geom_line(aes(x,fg,colour="fg"))+
   geom_line(aes(x,w,colour="w"))+
   scale_colour_manual("", values=cols, labels=mylabs)+
   labs(title= "Fig 2. Squeezing")
```



Observations for Figure 2: Here we have the plot of fstar/g and its bottom squeeze as we will be implementing a squeeze in our rejection.

Inversion calculations

We will be generating our g_2 using inversion, ie, first we need to make sure that
1. $g_Y^*(y)$ is a pdf (hence find its normalizing constant k_{gstar_2}), ie. $k_{gstar_2} = \frac{1}{\int_0^5 g_Y^*(u)du}$ & $g_Y(y) = k_{gstar_2} g_Y^*(y)$
2. Find its CDF G_{2_Y}, ie. $G_{2_Y}(y) = \int_0^y g_Y(u)du$
3. Invert it with the correct support: $G_{2_Y}(y) = x \implies y = G_{2_Y}^{-1}(x)$

1. Find g_{2_Y}

$$g_{2_Y}(x) = \begin{cases} 11xk_{gstar_2} & 0 \le x < m_1 \\ \left(0.36 - \frac{1}{20}x\right) k_{gstar_2} & m_1 \le x < m_2 \\ (5-x)\, k_{gstar_2} & m_2 \le x \le 5 \\ 0, & \text{otherwise.} \end{cases}$$

2. Integrate g_{2_Y}:

$$G_{2_Y}(y) = \int_0^5 g_{2_Y}(u)du = \begin{cases} \int_0^y 11k_{g_2}udu, \\ \int_0^{m_1} 11k_{g_2}udu + \int_{m_1}^y (0.36k_{g_2} - \frac{1}{20}k_{g_2}u)du, \\ \int_0^{m_1} 11k_{g_2}udu + \int_{m_1}^{m_2} (0.36k_{g_2} - \frac{1}{20}k_{g_2}u)du + \int_{m_2}^y (5k_{g_2} - k_{g_2}u)du, \end{cases}$$

$$= \begin{cases} [\frac{11}{2}k_{g_2}u^2]_0^y, \\ [\frac{11}{2}k_{g_2}u^2]_0^{m_1} + [0.36k_{g_2}u - \frac{1}{40}k_{g_2}u^2]_{m_1}^y, \\ [\frac{11}{2}k_{g_2}u^2]_0^{m_1} + [0.36k_{g_2}u - \frac{1}{40}k_{g_2}u^2]_{m_1}^{m_2} + [(5k_{g_2}u - \frac{1}{2}k_{g_2}u^2)]_{m_1}^y, \end{cases}$$

$$= \begin{cases} \frac{11}{2}k_{g_2}y^2, & 0 \le y < m_1 \\ \frac{11}{2}k_{g_2}m_1^2 + 0.36k_{g_2}y - \frac{1}{40}k_{g_2}y^2 - 0.36k_{g_2}m_1 + \frac{1}{40}k_{g_2}m_1^2, & m_1 \le y < m_2 \\ \frac{11}{2}k_{g_2}m_1^2 + 0.36k_{g_2}m_2 - \frac{1}{40}k_{g_2}m_2^2 - 0.36k_{g_2}m_1 + \frac{1}{40}k_{g_2}m_1^2 + 5k_{g_2}y - \frac{1}{2}k_{g_2}y^2 - 5k_{g_2}m_2 + \frac{1}{2}k_{g_2}m_2^2 & m_2 \le y \le 5 \end{cases}$$

3. Inverse G_2: Solve the following 3 equations:

$$\frac{11}{2}k_{g_2}y^2 = x \implies y = \pm\sqrt(\frac{2x}{11k_{g_2}})$$

We take the positive square root because the values $y \geq 0$ in this interval and the values for x range in $G(0) \leq x \leq G(m_1)$

$$\begin{aligned}
&\frac{11}{2}k_{g_2}m_1^2 + 0.36k_{g_2}y - \frac{1}{40}k_{g_2}y^2 - 0.36k_{g_2}m_1 + \frac{1}{40}k_{g_2}m_1^2 = x\\
\implies & -\frac{1}{40}k_{g_2}y^2 + 0.36k_{g_2}y + (k_{g_2}m_1(\frac{221}{40}m_1 - 0.36) - x) = 0\\
\implies & y = \frac{-0.36k_{g_2} \pm \sqrt((0.36k_{g_2})^2 + 4\frac{1}{40}k_{g_2}(k_{g_2}m_1(\frac{221}{40}m_1 - 0.36) - x))}{2(-\frac{1}{40}k_{g_2})}\\
\implies & y = 20 * 0.36k_{g_2}^2 - 20k_{g_2}\sqrt((0.36k_{g_2})^2 + \frac{1}{10}k_{g_2}(k_{g_2}m_1(\frac{221}{40}m_1 - 0.36) - x))
\end{aligned}$$

We take the negative square root to satisfy our conditions for y and the values for x range in $G(m_1) \leq x < G(m_2)$

$$\frac{11}{2}k_{g_2}m_1^2 + 0.36k_{g_2}m_2 - \frac{1}{40}k_{g_2}m_2^2 - 0.36k_{g_2}m_1 + \frac{1}{40}k_{g_2}m_1^2 + 5k_{g_2}y - \frac{1}{2}k_{g_2}y^2 - 5k_{g_2}m_2 + \frac{1}{2}k_{g_2}m_2^2 = x$$

$$\begin{aligned}
\implies & -\frac{1}{2}k_{g_2}y^2 + 5k_{g_2}y + (m_1k_{g_2}(\frac{221}{40}m_1 - 0.36) + m_2k_{g_2}(\frac{19}{40}m_2 - 4.64) - x) = 0\\
\implies & y = \frac{-5k_{g_2} \pm \sqrt(((5k_{g_2})^2 + 2k_{g_2}(m_1k_{g_2}(\frac{221}{40}m_1 - 0.36) + m_2k_{g_2}(\frac{19}{40}m_2 - 4.64) - x)))}{-2(\frac{1}{2}k_{g_2})}\\
\implies & y = 5k_{g_2}^2 - k_{g_2}sqrt((5k_{g_2})^2 + 2k_{g_2}(m_1k_{g_2}(\frac{221}{40}m_1 - 0.36) + m_2k_{g_2}(\frac{19}{40}m_2 - 4.64) - x))
\end{aligned}$$

We take the negative square root to satisfy our conditions for y and the values for x range in $G(m_2) \leq x \leq G(1)$.

Now let's do the inversion :

$$G_{2_Y}(y) = x \implies y = G_{2_Y}^{-1}(x)$$

$$G_Y^{-1}(x) = \begin{cases} \sqrt(\frac{2x}{11k_{g_2}}), & 0 \leq x < G(m_1)\\ 20 * 0.36 - 20k_{g_2}\sqrt((0.36k_{g_2})^2 + \frac{1}{10}k_{g_2}(k_{g_2}m_1\frac{221}{40}m_1 - 0.36) - x)), & G(m_1) \leq x < G(m_2)\\ 5k_{g_2}^2 - k_{g_2}sqrt((5k_{g_2})^2 + 2k_{g_2}(m_1k_{g_2}(\frac{221}{40}m_1 - 0.36) + m_2k_{g_2}(\frac{19}{40}m_2 - 4.64) - x)), & G(m_2) \leq x \leq 1 \end{cases}$$

```
#Define functions for inversion
  #1. Define inversion scheme functions for g1
    G1 <- function(y) 1/5 * y #Define G1 (calculation by hand)
    inv_G1 <- function(x) 5*x #Define inv_G1
  #2. Define inversion scheme functions for g2
    G2 <- function(y){#Define G2 (calculations by hand)
      ifelse(y<m1, 11/2*y^2*k_gstar2,
        ifelse(y<m2, k_gstar2*(11/2*m1^2+0.36*y-1/40*y^2-0.36*m1+1/40*m1^2),
          k_gstar2*(11/2*m1^2+0.36*m2-1/40*m2^2-0.36*m1+1/40*m1^2 +5*y-1/2*y^2-5*m2+1/2*m2^2) ))}
    #Set the new intervals for inv_G2
    m11=G2(m1)
    m21=G2(m2)
    inv_G2 <- function(x){#Define inv_G2
      ifelse(x<m11, sqrt(2*x/(11*k_gstar2)),
         ifelse(x<m21, (0.36*k_gstar2-sqrt((0.36*k_gstar2)^2 +
          1/10*k_gstar2*(m1*k_gstar2*(221/40*m1-0.36)-x)))/(1/20*k_gstar2),
            (5*k_gstar2-sqrt((5*k_gstar2)^2 +
              2*k_gstar2*(m1*k_gstar2*(221/40*m1-0.36) +
                m2*k_gstar2*(19/40*m2 - 4.64)-x)))/k_gstar2))}
```

In order to simulate from g_Y we use inversion.

Inversion Algorithm for $g_Y(\cdot)$:

1. Simulate $U = u \sim U(0,1)$.
2. Set

$$Y = y = \begin{cases} \sqrt{(}\frac{2u}{11k_{g_2}}), & 0 \leq u < G(m_1) \\ 20 * 0.36k_{g_2}^2 - 20k_{g_2}\sqrt{(}(0.36k_{g_2})^2 + \frac{1}{10}k_{g_2}\left(k_{g_2}m_1\left(\frac{221}{40}m_1 - 0.36\right) - u\right)), & G(m_1) \leq u < G(m_2) \\ 5k_{g_2}^2 - k_{g_2}sqrt((5k_{g_2})^2 + 2k_{g_2}(m_1k_{g_2}\left(\frac{221}{40}m_1 - 0.36\right) + m_2k_{g_2}\left(\frac{19}{40}m_2 - 4.64\right) - u)), & G(m_2) \leq u \leq 1 \end{cases}$$

   $\sim g_{2_Y}(\cdot)$

Rejection Algorithm:

1. Simulate $U = u_1 \sim U(0,1)$.
2. Simulate $Y = y = G_Y^{-1}(u_1) \sim g_Y(\cdot)$.
3. Simulate $U = u_2 \sim U(0,1)$
4. If $u_2 \leq \frac{f_X^*(y)}{Mg_Y(y)}$, then $X = y \sim f_X(\cdot)$.
5. Otherwise GOTO 1.

Here are the following codes:

```
#Rejection scheme without squeezing for g_1
rhn_u <- function(n, text=FALSE){
  #Simulated values in x
  xu <- vector()
  #Estimated acceptance probabilities in p
  pu <- vector()
  len_xu = 0
  multu = theta_g1 #Acceptance probability
  while(len_xu <= n){
    n_to_genu = max((n-len_xu)*multu,10) #Number to generate from
    u1u=runif(n_to_genu)
    yu=sapply(u1u,inv_G1) #Generate g_1 using inversion
    u2u = runif(n_to_genu)
    condu <- (u2u <= fstar(yu)/(M_1*g1(yu))) #Accept X=y if u <= f*(y)/Mg*(y)
    pu <- c(pu, sum(condu)/n_to_genu) #Keep track of estimated acceptance prob
    xu = c(xu,yu[condu]) #Concatenate accepted values
```

```r
    len_xu <- length(xu)
  }
  if (text){
  cat("theoretical acceptance prob=", theta_g1, "\n")
  cat("acceptance prob=", pu, "\n")}
  return(xu=xu[1:n])}
```

```r
#Rejection scheme without squeezing for g_2
rhn <- function(n, text=FALSE){
  #Simulated values in x
  x <- vector()
  #Estimated acceptance probabilities in p
  p <- vector()
  len_x = 0
  mult = theta_g2 #Acceptance probability
  while(len_x <= n){
    n_to_gen = max((n-len_x)*mult,10) #Number to generate from
    u1=runif(n_to_gen)
    y=sapply(u1,inv_G2) #Generate G_2 using inversion
    u2 = runif(n_to_gen)
    cond <- (u2 <= fstar(y)/(M_2*g_2star(y))) #Accept X=y if u <= f*(y)/Mg*(y)
    p<-c(p, sum(cond)/n_to_gen) #Keep track of estimated acceptance proba
    x = c(x,y[cond]) #Concatenate accepted values
    len_x <- length(x)
  }
  if (text){
    cat("theoretical acceptance prob=", theta_g2, "\n")
    cat("acceptance prob=", p, "\n")}
  return(x=x[1:n]) }
```

```r
#Rejection scheme with squezing for g_2
rhn_sq <- function(n,text=FALSE){
  #Simulated values in x
  x_sq <- vector()
  #Estimated acceptance probabilities in p
  p_sq <- vector()
  len_x_sq = 0
  mult_sq = theta_g2 #Acceptance probability
  while(len_x_sq <= n){
    n_to_gen_sq = max((n-len_x_sq)*mult_sq,10) #Number to generate from
    u1_sq = runif(n_to_gen_sq)
    y_sq = sapply(u1_sq,inv_G2) #Generate g_2 using inversion
    u2_sq = runif(n_to_gen_sq)
    cond_sq <- ifelse(u2_sq*M_2 <= wl(y_sq), TRUE, ifelse((g_2star(y_sq) * u2_sq * M_2)
    > fstar(y_sq), FALSE, TRUE)) #Accept if u1 <= wl & u <= f*(y)/Mg*(y) else reject
    p_sq<-c(p_sq, sum(cond_sq)/n_to_gen_sq) #Keep track of estimated acceptance proba
    x_sq = c(x_sq,y_sq[cond_sq]) #Concatenate accepted values
    len_x_sq <- length(x_sq)
  }
  if (text){
    cat("theoretical acceptance prob=", theta_g2, "\n")
    cat("acceptance prob=", p_sq, "\n")}
  return(x_sq=x_sq[1:n]) }
```

Discussion of acceptance rate and computational time

As we have computed by hand before, our theoretical acceptance rate are respectively 50% for the uniform envelope, and 83% for my more refined envelope. We can see after each code chunk, the computational acceptance probability: for the uniform, the acceptance proba is the lowest, for the more refined envelope $M * g_2^*(y)$, clearly, the acceptance probability gets much closer to the theoritical one.

Now let's check the time it takes for each of these set-ups. We expect the squeezing scheme to be the fastest, as explained earlier.

```
n=1000000
#Rejection computational timing
system.time(rhn_u(n,TRUE))
```

```
## theoretical acceptance prob= 0.5011983
## acceptance prob= 0.5021865 0.5023102 0.5017926 0.5038689 0.5026723 0.5027361 0.5020856 0.5005994 0.4

##    user  system elapsed
##   3.102   0.277   3.448
```

```
system.time(rhn(n,TRUE))
```

```
## theoretical acceptance prob= 0.8338595
## acceptance prob= 0.7166975 0.7165444 0.7168637 0.7186437 0.7189393 0.7129393 0.7235055 0.7058564 0.7

##    user  system elapsed
##  10.206   0.227  10.473
```

```
system.time(rhn_sq(n,TRUE))
```

```
## theoretical acceptance prob= 0.8338595
## acceptance prob= 0.7562917 0.7567075 0.7569242 0.7531439 0.756998 0.7538744 0.7482975 0.7484371 0.75

##    user  system elapsed
##  10.191   0.274  10.539
```

We can see that our more tedious envelope takes ~ 5 times longer to generate than the uniform one. However the squeeze takes relatively the same amount of time as the non squeeze. This is because the ifelse statements are computationally heavy and R isn't optimized for these logical statements. In practice, the squeeze should be faster. Also we can see that the numeric acceptance probability is much higher for our second envelope as we have observed theoritically.

QUESTION 2: Diagnostic Plots & Statistics Test

PART 1: Diagnostic Plots

We will only be analysing the diagnostics plots for the g_{2_Y} function as it has the higher acceptance probability and is still quite efficient.

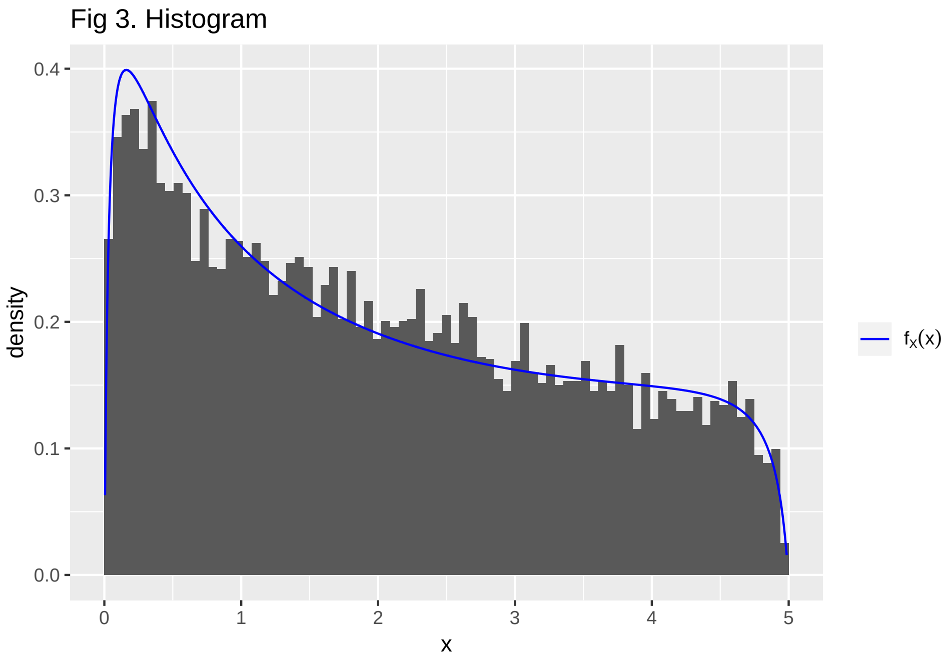
Histogram

First, let's look at the histogram of our generated data (using rejection) and the plot of our distribution to see if they follow the same trend.

```
library(ggplot2)
library(pracma)
library(forecast)
```

```r
n=10000

# Histogram
mylabs=list(expression(f[X](x)))
cols = c("f"="blue")
x=rhn_sq(n) #or rhn_sq
x_plot <- data.frame(x=x,f=f(x))
p <- ggplot(x_plot)
 p+ labs(y="density",title="Histogram and true pdf")+
  geom_histogram(aes(x,y= ..density..),breaks=seq(0, 5, l=80))+
  geom_line(aes(x,f,colour="f"))+
  scale_colour_manual(name="",values=cols, labels=mylabs)+
  labs(title= "Fig 3. Histogram")
```



Observations for Figure 3: We can see that our generated data (the grey bins) follow well the theoritical true pdf (blue line). This means that our rejection with squeezing algorithm is correct. [2]

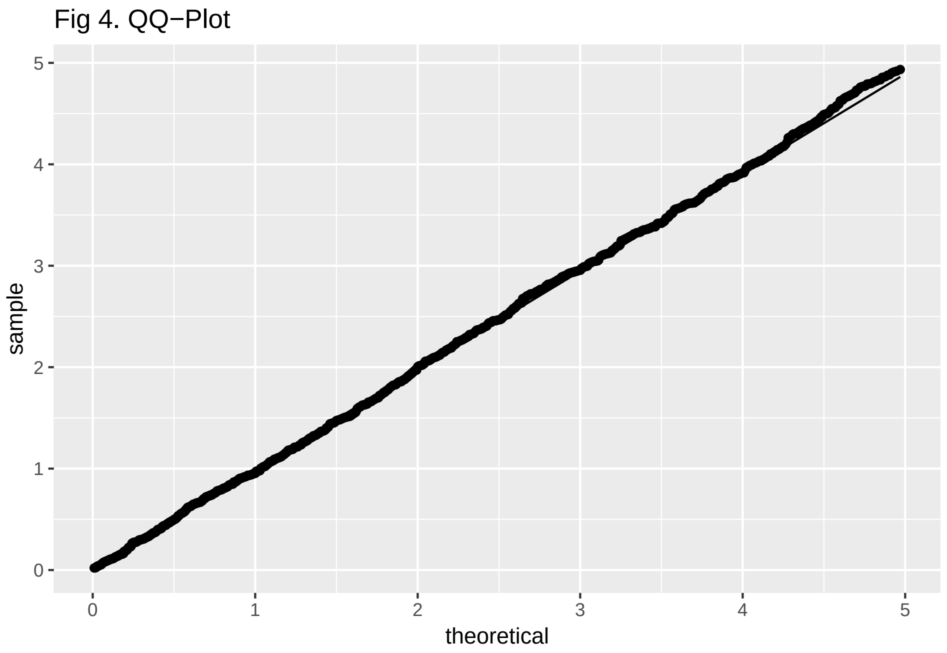
QQ Plot

Now let's do a QQ (quantile-quantile) plot. We plot the quantile of the generated data against the quantile of the hypothetical distribution

```r
#QQ plot
library(ggplot2)
library(pracma)
#Generate some x
n=1000
x=rhn(n)
x_plot = data.frame(x=x)
#Function which uses numerical integration to calculate F(p) - q
mycdf = function(p,q=0){
  integrate(f,0,p)$value-q
}
```

```
#Solving F(x) - q = 0 to calculate the quantiles
qfunc <- function(q){
  #Set lower as 0 and upper as 5, values of CDF
  bisect(mycdf, 0+10^(-6), 5-10^(-6),q=q)$root #Don't include endpoints of (0,5)
}
#Making sure it works on a vector and returns a vector
qhn1 <- function(p) unlist(lapply(p, qfunc))

#Plot
ggplot(x_plot, aes(sample=x))+
  labs(title="Empirical against theoretical quantiles")+
   stat_qq(distribution=qhn1, dparams=list()) +
   stat_qq_line(distribution=qhn1, dparams = list())+
  labs(title= "Fig 4. QQ-Plot")
```

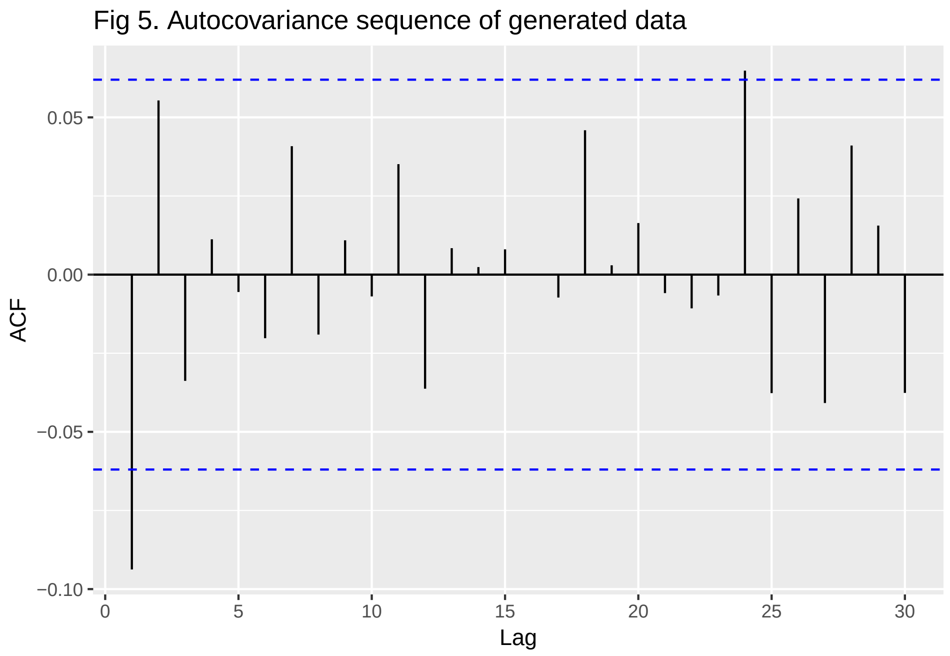


Observations for Figure 4: Here we can see that the points generated from our sample from the squeezed-rejection algorithm follow a straight line from the theoritical quantile. This means that our generated data has the same distribution as our f.[2]

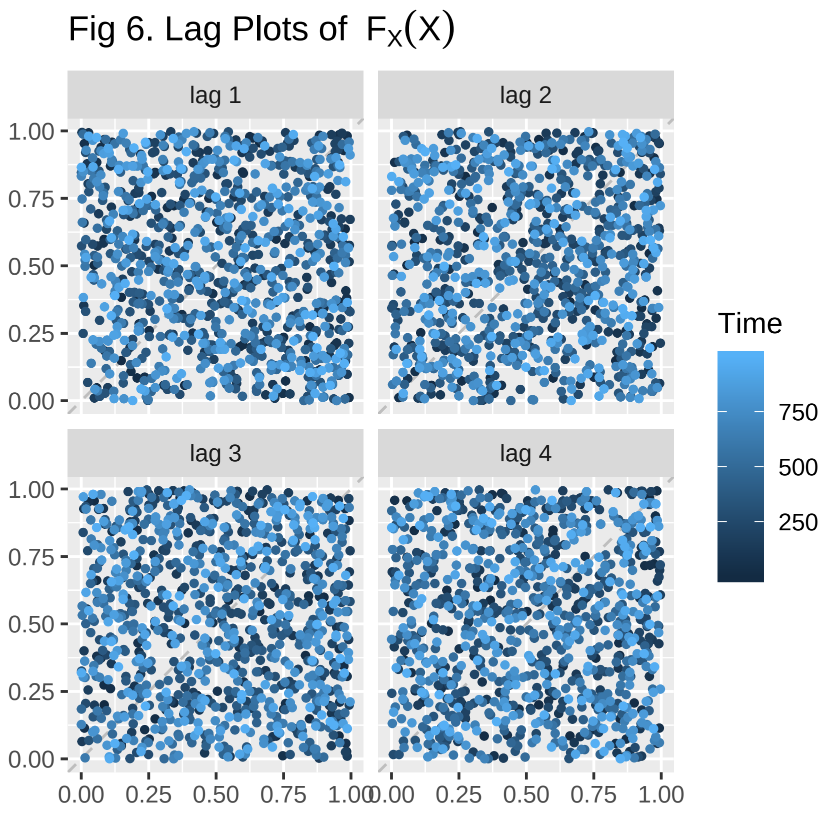
Autocovariance & Lag Plots

The point of doing autocovariance and lag plots is to see whether our data is randomly generated. If there is not pattern in the lag plot or the large majority of the values of the autocovariance sequence stay within the blue lines, then we can conclude that our data is uncorrelated. See as follows:

```
#Autocovariance & Lag Plots
library(forecast)
n=1000
x=rhn(n) #Generate data following our distribution
#Theoritical Cdf
phn <- function(p) unlist(lapply(p, mycdf)) #CDF of f
x_plot <- data.frame(x=x)
ggAcf(x_plot)+
  labs(title="Fig 5. Autocovariance sequence of generated data")
```



```
gglagplot(phn(x_plot$x), lags=4, do.lines=FALSE)+
  labs(title=expression("Fig 6. Lag Plots of " ~ F[X](X)))
```



Observations for Figure 5: We can observe for the autocovariance sequence that all our data appears random and there is almost no outlier, as the majority of our data lies in our blue line range.

Observations for Figure 6: On this lag plot, we can clearly see that the generated points are random as they appear everywhere at different times. [2]

Part 2: Statistical Tests

Here we will be conducting 2 statistics tests on the generated data. Throughout this section we will use our statistics test to test our null hypothesis: H_0 : generated data follows our distribution f^*.

Kolmogorow-Smirnov Test

The Kolmogorov-Smirnov (K-S) test can be used to test the difference between the empirical culumative distribution function (cdf), $F_{X_n}(x)$ and the theoretical cumulative distribution function, $F_X(x)$. The K-S statistics is given by: $D = sup_x \mid F_{X_n}(x) - F_X(x) \mid$. This test statistic is then compared to the critical values of the Kolmogorov distribution to determine the p-value of the test.[1]

KS Simulation Study

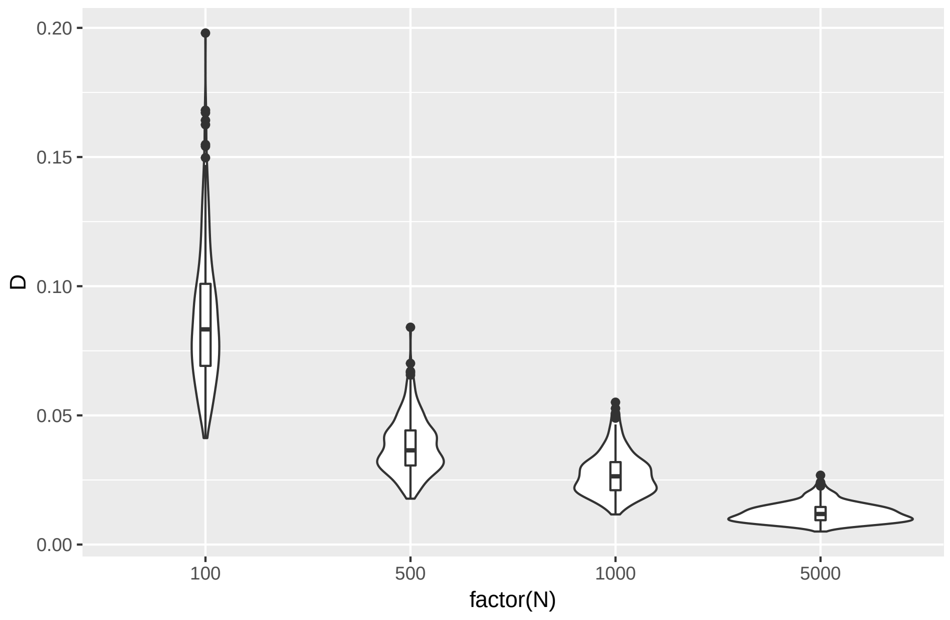
```
#Simulation study of KS test, using data set length stored in n_vals and taking
#m simulations of each length
n_vals = c(100,500,1000,5000) #Number of data points
nn_vals = length(n_vals)
m=300 #Number of simulations
x=rhn(n)
ks.testrhn <- function(x){
  kstestx = ks.test(x,phn) #phn as defined before: CDF of f
  return(c(kstestx$p.value, kstestx$statistic))}
ks.results=data.frame()

for(n in 1:nn_vals){
  n1=n_vals[n]
  x <- matrix(0, nrow=n1, ncol=m)
  # one call to rhn, then split into matrix in order to use the apply function
  x1 = rhn(n1*m)
  for(i in 1:m)  x[,i] <- x1[((i-1)*n1+1):(i*n1)]
  ks.testx= apply(x,2,ks.testrhn)
  ks.results = rbind(ks.results, data.frame(p.value=ks.testx[1,], D = ks.testx[2,],
N=rep(n1,m)))}
```

Plots for KS Simulation Study

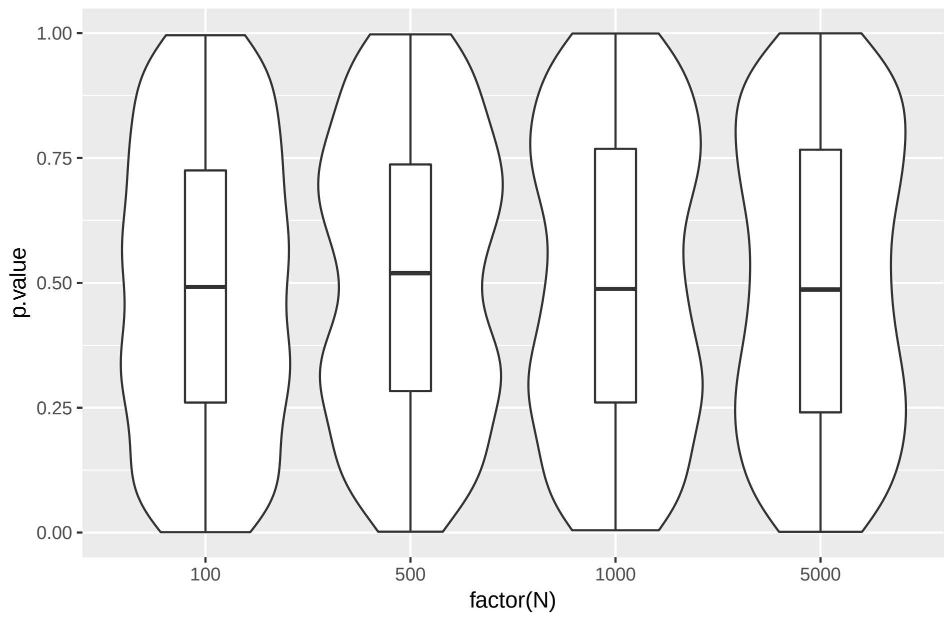
```
#Plot the results of the K-S test
library(ggplot2)
library(knitr)
library(dplyr)
ggplot(ks.results, aes(factor(N), D))+
  geom_violin()+
  geom_boxplot(width=0.05)+
  labs(title="Fig 7. Distributions of the K-S test statistics")
```

Fig 7. Distributions of the K−S test statistics



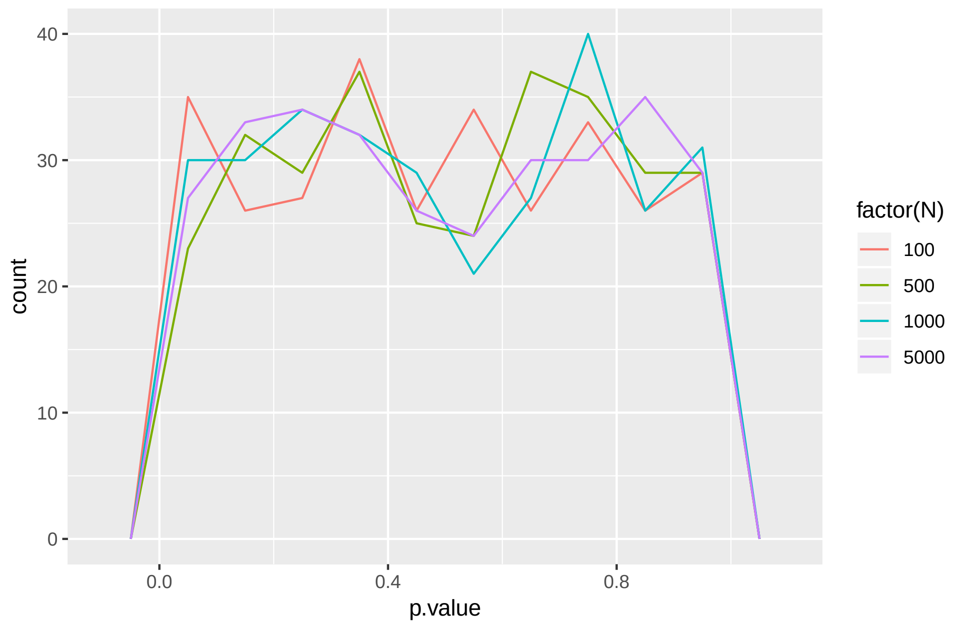
```
ggplot(ks.results, aes(factor(N), p.value))+
  geom_violin()+
  geom_boxplot(width=0.2)+
  labs(title="Fig 8. Distributions of the K-S p-values")
```

Fig 8. Distributions of the K−S p−values



```
ggplot(ks.results, aes(p.value,colour=factor(N)))+
  geom_freqpoly(breaks=seq(0,1,0.1))+
  labs(title="Fig 9. Frequency polygons of p-values")
```

Fig 9. Frequency polygons of p−values



```
ks.table <- ks.results %>% group_by(N) %>% summarise("Mean p-value" = round(mean(p.value),
digits=3), "Std Dev (p)" = round(sqrt(var(p.value)), 3), "Mean D"=round(mean(D),
digits=3), "Std Dev (D)"= round(sqrt(var(D)), 3))
kable(ks.table)
```

N	Mean p-value	Std Dev (p)	Mean D	Std Dev (D)
100	0.489	0.288	0.088	0.027
500	0.510	0.281	0.038	0.011
1000	0.502	0.296	0.027	0.008
5000	0.500	0.294	0.012	0.004

Observations for Figure 7: The value of D is decreasing as we increase the number of data points simulated which makes sense (it means that the difference between the sup of the empirical and theoritical cdf gets smaller) and confirms our data generation validity.

Observations for Figure 8: Our violin plot shows us that our p-values are somewhat uniformly distributed and range on the whole interval.

Observations for Figure 9: Our frequency polygon, coupled with our p-value violin plot helps us understand how the p-value are distributed relative to the number of data points.

Observation for the table: Our table in association with Fig 8. allows us to confirm our null hypothesis that our generated data follows the distribution of f^* as we have that D decreases as we increase the simulation and that our p values are uniformly distributed.[2]

Chi-Squared goodness of fit Test

Now let's look at the Chi-Squared goodness of fit test (if our generated data follows the distribution).
We have a sequence of generated data $X_1, X_2, ..., X_n$ from our rejection algorithm, and we want to split then into Nbins. We have the observed frequencies $O_1, O_2, ..., O_N$ such that: $\sum_{i=1}^{N} O_i = n$.
We have the expected frequencies under our null hypothesis for the distribution of variates by $E_1, E_2, ..., E_N$ with $\sum_{i=1}^{N} E_i = n$. We use the statistics $S = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}$ to measure how different is the observed from

the expected.
We take n to be large so that it yields: $\sum_i \frac{(O_i-E_i)^2}{E_i} \sim \chi^2_{N-1}$. In order to test our hypothesis, we will compare the observed statistic to the chi-squared distribution. [1]

CS Simulation Study

```r
n_vals = c(100, 500, 1000, 5000) #Number of data points
nn_vals = length(n_vals)
m=300 #Number of simulations
x_rhn=rhn(n)
nbin=10 #Number of bins

mycdf1 = function(l,u){integrate(f,l,u)$value} #Find CDF of f

cs.test= function(x_rhn){ #Define function to compute chisq_test
  ob_cs= 5/nbin
  x_cs= vector()
  p_cs= vector()
  s_cs = 0
  for (i in 1:nbin){
    expec= length(x_rhn)* mycdf1(ob_cs*(i-1), ob_cs*i)
    obs= length(x_rhn[x_rhn>(ob_cs*(i-1)) & x_rhn<(ob_cs*i)])
    x_cs= c(x_cs,obs) #Observed data
    p_cs= c(p_cs,expec) #Expected Data
    s_cs= s_cs + (((obs-expec)^2)/expec)
  }
  chisq.test(x_cs,p_cs)
  cbind(chisq.test(x_cs,p_cs)$p.value, s_cs)} #Output p-value and statistic S

cs.results=data.frame() #Define data frame

for(n in 1:nn_vals){
  n1=n_vals[n]
  x <- matrix(0, nrow=n1, ncol=m)
  # one call to rhn, then split into matrix in order to use the apply function
  x1 = rhn(n1*m)
  for(i in 1:m)  x[,i] <- x1[((i-1)*n1+1):(i*n1)]
  cs.testx= apply(x,2,cs.test)
  cs.results = rbind(cs.results, data.frame(p.value_cs=cs.testx[1,], S = cs.testx[2,],
  N_cs=rep(n1,m)))
}
```

Plot for CS Simulation Study

```r
#Plot the table of the Chi-Squared Test
library(knitr)
library(dplyr)
cs.table <- cs.results %>% group_by(N_cs) %>% summarise("Mean p-value" =
round(mean(p.value_cs),digits=3), "Std Dev (p)" =
round(sqrt(var(p.value_cs)), 3), "Mean S"=round(mean(S), digits=3),
"Std Dev (S)"= round(sqrt(var(S)), 3))
```

```
kable(cs.table)
```

N_cs	Mean p-value	Std Dev (p)	Mean S	Std Dev (S)
100	0.259	0.015	9.274	4.302
500	0.239	0.008	8.690	4.239
1000	0.235	0.006	9.052	4.517
5000	0.232	0.003	9.054	4.193

Observation for the table: Our table in association with Fig 11. allows us to confirm our null hypothesis that our generated data follows the distribution of f^* as we have that S decreases as we increase the simulation and that our p values are uniformly distributed.[2]

QUESTION 3: Monte Carlo Integration

Motivation for Monte-Carlo integration: We want to estimate $\theta = \int h(x)dx$, which we can't get analytically. In our case we manage to, but for the purpose of the exercise we will use the following 3 methods in order to get the best estimate for *theta* with the smallest value possible.[1]

1. Crude Monte Carlo
Generate $X_1, X_2, ..., X_n \sim U(0,5)$ with $\theta = \int \phi(x)f(x)dx$ and get our Crude Monte Carlo estimator:

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i)$$

which has variance:

$$var(\hat{\theta}) = \frac{1}{n}\int[\phi(x) - \theta]^2 f(x)dx$$

2. Hit-or-Miss
Generate $U = u_i \sim U(0,5)$ & $V = v_i \sim U(0,c)$, where $c = max(h(x))$ and get our Hit or Miss estimator:

$$\hat{\theta} = 5c\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}_{v_i \le h(u_i)}$$

which has variance:

$$var(\hat{\theta}) = \frac{\theta}{n}(5c - \theta)$$

3. Antithetic Variates
Generate $U = u_i \sim U(0,5)$ & $V = v_i, \sim 5 - U(0,c)$ such that $corr(\phi(u_i), \phi(v_i)) < 0$ (can check numerically!) and get our antithetic sampling estimate:

$$\hat{\theta} = \frac{1}{2n}(\sum_{i=1}^{n}\phi(u_i) + \sum_{i=1}^{n}\phi(v_i))$$

which has variance:

$$var(\hat{\theta}) = \frac{1}{4n^2}var(\sum_{i=1}^{n}\phi(u_i)) + \frac{1}{4n^2}var(\sum_{i=1}^{n}\phi(v_i)) + 2\frac{1}{2n}\frac{1}{2n}cov(\sum_{i=1}^{n}\phi(u_i), \sum_{i=1}^{n}\phi(v_i))$$

$$= \frac{1}{4n^2}(var(\phi(u_i)) + var(\phi(v_i)) + 2cov(\phi(u_i), \phi(v_i)))$$

[3]

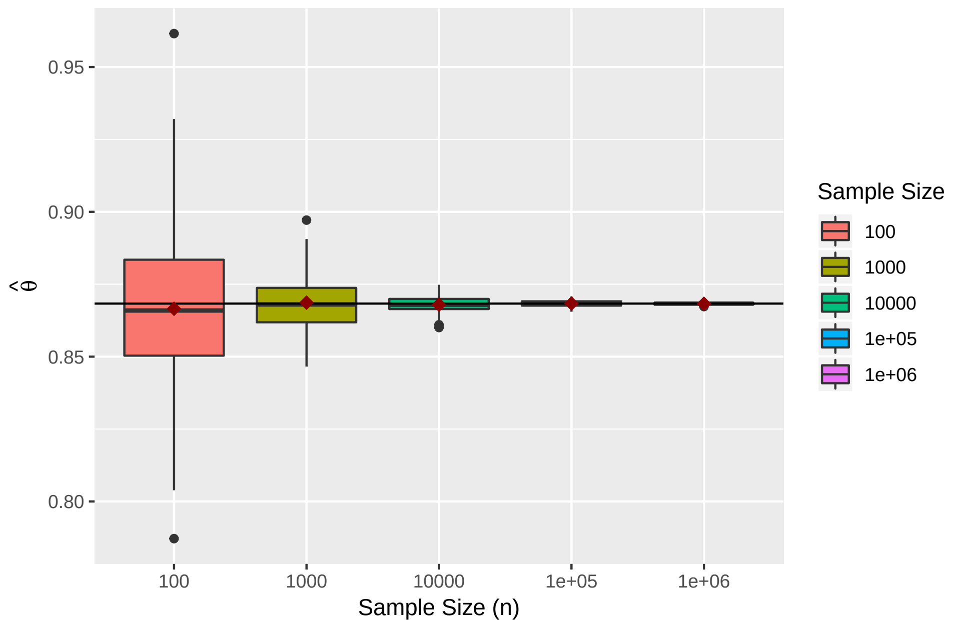
Simulation study for our Crude Monte Carlo estimate of θ

```
library(tidyverse)
n = 1000000
cmc <- function(n){#Crude Monte Carlo estimate
  y<-runif(n,0,5)
  return(t<- mean(5*fstar(y)))}

theta <- 1/k_fstar
nvals <- c(100,1000,10000,100000,1000000)
nnvals <- length(nvals)

m <- 100 #Number of loops
results <- data.frame()
for(j in 1:m){
  for(i in seq(1,nnvals,by=1)){
    results=rbind(results, data.frame(n=toString(nvals[i]),t=cmc(nvals[i]))) #Update cmc
  }
}
ggplot(results, aes(x=n, y=t, fill=factor(n)))+
  geom_boxplot()+
  geom_hline(yintercept=theta)+
  stat_summary(fun.y=mean, colour="darkred", geom="point",
               shape=18, size=3, show_guide = FALSE) +
  labs(title=paste("Fig 10. Crude MC estimation for various sample sizes"))+
  labs(fill="Sample Size")+
  scale_y_continuous(name=expression(hat(theta)))+
  scale_x_discrete('Sample Size (n)')
```

Fig 10. Crude MC estimation for various sample sizes



```
p1_cmc <- filter(results,n==nvals[5])
var_cmc <- var(p1_cmc$t)
theta_cmc <- mean(p1_cmc$t)
cat('Average estimator=', theta_cmc, 'with Variance=', var_cmc)
```

```
## Average estimator= 0.868261 with Variance= 1.118467e-07
```

Observations of Figure 10: This simulation study shows us that as we increase our sample size, our variance decreases for our Crude Monte Carlo estimator. Also note that when the sample size is small (n=100), the mean of estimator isn't exactlu what we are looking for. However as we increase the sample size, the mean gets closer and closer to the true value, on top of overall variance reduction.

Hit-or-Miss

```
# Hit-or-miss Monte Carlo integration code
  c = optimise(fstar,c(0,5),maximum = T)$objective #Compute value of c, max of fstar
  u_hom <- runif(n, 0, 5) #Generate uniforms in rectangle
  v_hom <- runif(n, 0, c) #Generate uniforms in rectangle
  cond_hom <-fstar(u_hom)
  v_hom <- v_hom[v_hom<=cond_hom] #Update v_hom
  theta_hom <- c*5*length(v_hom)/n #Proportion of points in the rectangle
  var_hom <- theta/n*(5*c-theta) #Compute variance
  cat('Average estimator=', theta_hom, "with variance=", var_hom)
```

```
## Average estimator= 0.8687992 with variance= 7.50377e-07
```

This estimate is the worse out of the 3 methods, as we would have expected from the lecture notes.

Antithetic Variates

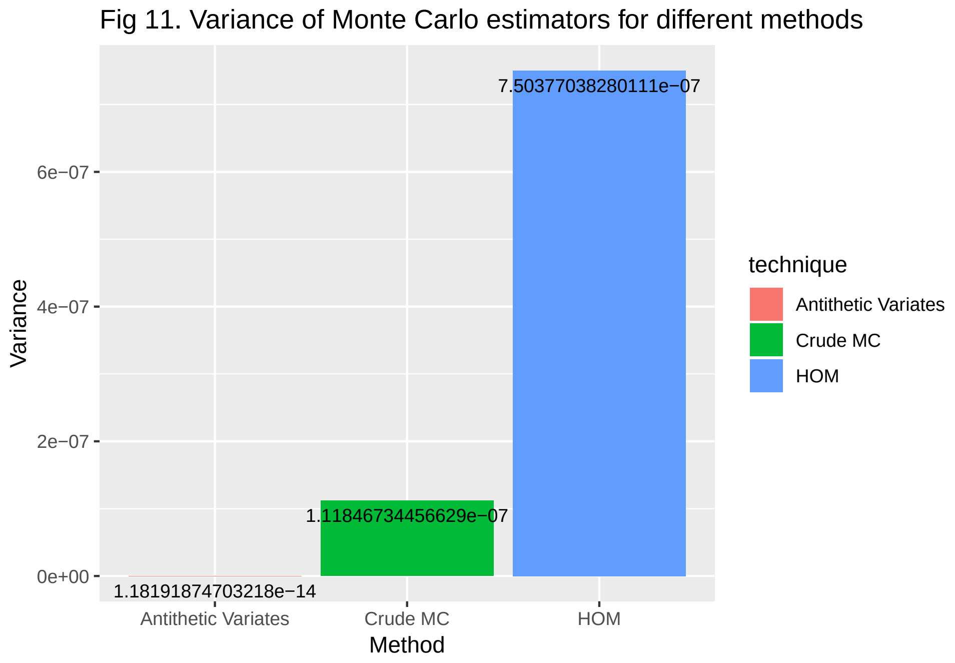
In order to implement a variance reduction, we can use antithetic variates as explained above

```
# Antithetic variates
u1_at <- runif(n,0,5)
u2_at <- 5-u1_at
phi <- function(x) fstar(x)/g2(x)
f1 <- phi(u1_at)
f2 <- phi(u2_at)
corr_at<- cor(f1,f2)
theta_at1 <- mean(f1) #phi(u1)
theta_at2 <- mean(f2) #phi(u2)
theta_at3 <- 1/2*(theta_at1+theta_at2) #Estimator
var_at <- 1/(4*n^2) * (var(f1)+var(f2)+ 2*cov(f1,f2)) #Compute variance
cat('Average estimator=', theta_at3, "with variance=", var_at)
```

```
## Average estimator= 0.8712173 with variance= 1.181919e-14
```

Clearly, the variance has been reduced from the Crude Monte Carlo method.

```
#Variance Reduction Comparison
technique <- c('Crude MC', 'HOM', 'Antithetic Variates')
estimates <- c(theta_cmc,theta_hom,theta_at3)
vars <- c(var_cmc, var_hom, var_at)
df <- data.frame(technique, estimates, vars)
ggplot(data=df,aes(x=technique,y=vars,fill=technique)) +
  geom_bar(stat="identity")+
  geom_text(aes(label=vars), color="black", size=3, vjust=1.6)+
  ggtitle("Fig 11. Variance of Monte Carlo estimators for different methods")+
  labs(y="Variance", x="Method")
```



Observations of Figure 11: Here, we can see that as expected our hit-or-miss technique has the highest variance and that the antithetic variates method gives the lowest variance for our estimator of θ.This was expected as from the lecture notes we have shown that the antithetic variates method allows for a great variance reduction.

Conclusion

In conclusion, I have generated data that follows my given distribution f^*. I have also tested using different methods that this data does indeed match closely the distribution. I have also used Monte Carlo integration methods to compute the normalizing constant of the distribution and variance reduction techniques to get the best possible estimate.

References

[1] E. McCoy. Stochastic simulation, Lectures Notes, Imperial College London, 2020. 2020.
[2] C. Reimann, P. Filzmoser, R. G. Garrett, R. Dutter. Statistical Data Analysis Explained: Applied Environmental Statistics with R. John Wiley & Sons, 2008.
[3] A. Owen. Monte Carlo theory, methods and examples. Chapter 8, Stanford University, 2013.