Stochastic Simulation Coursework

George Hutchings

27/11/2020

Question 1

We aim to generate random variates from the probability density function $f_X(\cdot)$. I will do this by rejection sampling.

My density

Firstly I use my CID to calculate the appropriate parameters for my probability density function.

```
CID <- 1357062
print(max(primeFactors(CID)))
```

```
## [1] 409
```

$$\implies f_X(x) \propto f_X^*(x) \coloneqq \begin{cases} \frac{1}{x(2-x)} e^{-log^2\left(\frac{x}{(2-x)}\right)} & x \in (0,2) \\ 0 & \text{otherwise} \end{cases}$$

Simulating my target density using rejection sampling

The rejection sampling algorithm generates random samples from a density $f_X(x)$, it does this by drawing from another density $g_Y(y)$ and then accepting or rejecting these samples as representative of the target density if they meet certain criteria. For $g_Y(y)$ to be valid we require:

- $f_X(x) > 0 \implies g_Y(x) > 0$
- Existence of a constant c such that for all x s.t $f_x(x) > 0$, $\frac{f_X(x)}{g_Y(x)} \leq c < \infty$

It should be noted that c is not unique, and so to avoid ambiguity we often define M as the infinum of all such c.

General rejection sampling algorithm:

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

It should be noted that this algorithm is completely equivalent (as we showed in lectures) to its application to densities known only up to proportionality, ie: $g_Y^*(y) \propto g_Y(y)$ and $f_X^*(x) \propto f_X(x)$, which gives the natural definition $M^* \coloneqq \sup_x \frac{f_X^*(x)}{g_Y^*(x)}$

Rejection sampling algorithm with unknown normalising constants

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

Upon inspection of the shape of $f_X^*(x)$ (Figure 1) I decided a trapezium shaped envelope would be appropriate. It is a computationally simple function, it can be integrated and then inverted easily (since it is constructed of linear functions) hence can easily be generated from, by inversion as,well as visually appearing to provide a large acceptance probability. I also considered an envelope $h_Z^*(z) \propto \mathcal{N}(1,\, 0.42)$, since this is a valid envelope for $f_X^*(x)$ and provides a marginally better fit (and hence higher acceptance probability), however I decided ultimately not to pursue this envelope since I believed the additional complexities in sampling from a Normal distribution would outweigh the marginal increase in acceptance probability gained in the rejection-acceptance step. Since if I did decide to pursue this method I would first sample a normal distribution, likely by a similar method to Marsaglia's polar method, which in itself, has an acceptance probability of $\frac{\pi}{4}$, and hence the minor improvements of the envelope's acceptance probability would be undone by the acceptance probability of sampling the envelope, (whereas when sampling a trapezium by inversion every value is accepted).

Constructing $g_Y^*(y)$

I shall construct $g_Y^*(y)$ such that it envelopes $f_X^*(y)$ entirely, that is it satisfies the validity conditions we mentioned previously.

Evenness of $f_X^*(x)$

It is useful to note that $f_X^*(x)$ is even about $x = 1$, since for $y \in (-1, 1)$

$$f_X^*(1-y) = \frac{1}{(1-y)(1+y)} e^{-log^2\left(\frac{1-y}{(1-y)(1+y)}\right)} = f_X^*(1+y) \ \square$$

and hence when constructing a valid $g_X^*(x)$ we need only consider $x \in (0, 1]$ since this can then be extended to $x \in (0, 2)$ by making $g_X^*(x)$ even about $x = 1$.

Top edge of$g_Y^*(y)$

First considering the top parallel edge of the trapezium, optimally this would be horizontal and intersect the maximum of $f_X^*(x)$, hence we need to find the maximum of $f_X^*(x)$, this is done by considering:

$$\frac{df_X^*(x)}{dx} = \frac{2\left(x - 2log\left(\frac{x}{2-x}\right) - 1\right)}{x^2(2-x)^2} e^{-log^2\left(\frac{1}{x(2-x)}\right)}$$

This is found easily using the chain rule, quotient rule and substitution (this process has been omitted as it is lengthy and un-insightful). To find the maximum we consider $\frac{df_X^*(x)}{dx} = 0$ for $x \in (0, 2)$ which occurs if and only if $\varphi(x) \coloneqq \left(x - 2log\left(\frac{x}{2-x}\right) - 1\right) = 0$, which $\iff x = 1$. To substantiate this claim, we can clearly see that $x = 1$ is a solution, and is the unique solution in $(0, 2)$, since $\varphi(x)$ is strictly decreasing for all $x \in (0, 2)$ (as $\varphi'(x) = \frac{(x-1)^2+3}{(x-2)x} < 0 \ \ \forall x \in (0, 2)$). Therefore the maximum is $(1, f_X^*(1)) = (1, 1)$ and hence the top part of the trapezium consists of $y = 1>$ It can be verified that it is indeed a maximum by observing figure 1.

Diagonal edges of $g_Y^*(y)$

Now without loss of generality let's consider the left edge of the trapezium. To be optimal, this must begin at $(0,0)$ (ensuring that the support of $g_Y^*(y)$ encompasses that of $f_X^*(x)$, and hence has the form $y = mx$ for some $m \in \mathbb{R}$, and it to only brush past $f_X^*(x)$. This can be found by solving $f_X^*(x) = mx$, $\frac{df_X^*(x)}{dx} = \frac{d(mx)}{dx} = m$. Solving these simultaneous equations we get that $f_X^*(x) = x\frac{df_X^*(x)}{dx}$, which can be simplified (this has been omitted since it is un-insightful) to finding the roots of $e^{3x-4} - \left(\frac{x}{2-x}\right)^4$, to compute this further we require numerical methods and hence I use the nleqslv package. I begin with the initial value 0.7, since we estimate our solution to be near here from Figure 1.

```
# Roots are where the trapezium diagonal 'brushes' fstar
simequ <- function(x) exp(3*x-4) - (x/(2-x))^4
x0 <- nleqslv(0.7, simequ)[[1]] # Finding the root near 0.7
m <- (2*x0 -4*log(x0/(2 - x0)) - 2)/(exp(log(x0/(2 - x0))^2)*(2 - x0)^2*x0^2) # fstar'(x0)
```

By combining the $y = 1$ and $y = mx$ parts of the trapezium, making our function even about $x = 1$ and injective we form our trapezium as this piecewise function:

$$g_Y^*(x) = \begin{cases} mx & x \in \left(0,\ \frac{1}{m}\right) \text{ (left trapezium edge)} \\ 1 & x \in \left[\frac{1}{m},\ 2 - \frac{1}{m}\right] \text{ (top trapezium edge)} \\ -m(x-2) & x \in \left(2 - \frac{1}{m},\ 2\right) \text{ (right trapezium edge)} \\ 0 & \text{Otherwise} \end{cases}$$

Simulating $g_Y(y)$

Finding $G_Y^{-1}(y)$

I aim to simulate $g_Y(y)$ by inversion, to do this I need $G_Y^{-1}(y)$, which we will calculate below, we integrate $g_Y^*(y)$ to get the cumulative density function up to proportionality:

$$G_Y^*(y) = \begin{cases} 0 & y \in (-\infty,\ 0) \\ \frac{my^2}{2} & y \in \left[0,\ \frac{1}{m}\right) \\ y - \frac{1}{2m} & y \in \left[\frac{1}{m},\ 2 - \frac{1}{m}\right] \\ -m\left(\frac{y^2}{2} - 2y\right) 2 - \frac{1}{m} - 2m & y \in \left(2 - \frac{1}{m},\ 2\right] \\ 2 - \frac{1}{m} & y \in (2,\ \infty) \end{cases}$$

It can be noted here that we can easily make this a valid cdf by forcing it to integrate to 1, that is $G_Y(y) \coloneqq \frac{1}{2-\frac{1}{m}} G_Y^*(y) = \frac{m}{2m-1} G_Y^*(y)$

$$\implies G_Y(y) = \begin{cases} 0 & y \in (-\infty,\ 0) \\ \frac{m^2y^2}{4m-2} & y \in \left[0,\ \frac{1}{m}\right) \\ \frac{my}{2m-1} - \frac{1}{4m-2} & y \in \left[\frac{1}{m},\ 2 - \frac{1}{m}\right] \\ \frac{-m^2}{2m-1}\left(\frac{y^2}{2} - 2y\right) + 1 - \frac{2m^2}{2m-1} & y \in \left(2 - \frac{1}{m},\ 2\right] \\ 1 & y \in (2,\ \infty) \end{cases}$$

$$\implies G_Y^{-1}(y) = \begin{cases} \frac{\sqrt{(4m-2)y}}{m} & y \in \left[0,\ \frac{1}{4m-2}\right) \\ y\left(2 - \frac{1}{m}\right) + \frac{1}{2m} & y \in \left[\frac{1}{4m-2},\ 1 - \frac{1}{4m-2}\right] \\ 2 - \sqrt{\frac{(2-4m)(y-1)}{m^2}} & y \in \left(1 - \frac{1}{4m-2},\ 1\right] \end{cases}$$

And hence we can use the method of inversion to generate samples from $g_Y(y)$:

1. Generate $U_i \sim U(0,1)$.
2. Set $Y_i = G_Y^{-1}(U_i)$

Acceptance probability of the algorithm

The acceptance probability, θ, of the algorithm is given by

$$\theta = \int_{-\infty}^{\infty} \frac{f^*(x)}{M^* g_Y^*(x)} g_Y(x) dx = \int_0^2 \frac{f^*(x)}{g_Y^*(x)} \frac{g_Y^*(x)}{2 - \frac{1}{m}} dx = \frac{m}{2m-1} \int_0^2 f^*(x) dx$$

Hence we seek:

$$\int_0^2 \frac{1}{x(2-x)} e^{-log^2\left(\frac{x}{(2-x)}\right)} dx$$

By Letting $x = 2u$ this becomes:

$$\int_0^1 \frac{1}{2u(2-2u)} e^{-log^2\left(\frac{2u}{(2-2u)}\right)} 2du = \frac{1}{2} \int_0^1 \frac{1}{u(1-u)} e^{-\text{logit}(u)^2} du$$

We recognise this as proportional to the logistic normal distribution $P(\mathcal{N}(\mu, \sigma^2)) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\text{logit}(x)-\mu)^2}{2\sigma^2}} \frac{1}{x(1-x)}$ with $\mu = 0$, $\sigma^2 = \frac{1}{2}$, since we know distributions integrate to one, we can conclude:

$$\int_0^2 \frac{1}{x(2-x)} e^{-log^2\left(\frac{x}{(2-x)}\right)} dx = \frac{1}{2}\sigma\sqrt{2\pi} = \frac{\sqrt{\pi}}{2}$$

This gives us the normalizing constant ie $f_X(x) = \frac{2}{\sqrt{\pi}} f_X^*(x)$, and acceptance probability $\theta = \frac{m\sqrt{\pi}}{4m-2} \approx 0.8095$

```
fstar <- function(x) 1/(exp(log(x/(2 - x))^2)*(2 - x)*x)
f <- function(x) fstar(x)*2 / sqrt(pi) # Defining the pdf f
theta  <- (m*sqrt(pi))/(4*m -2) # Theoretical acceptance probability
```

Squeezing

We can also add a squeezing function to our algorithm to reduce the sometimes costly computation of $\frac{f_X^*(x)}{g_Y^*(x)}$. Squeezing is done by formulating two functions, $W_L(x)$ and $W_U(x)$ which are inexpensive and such that $W_L(x) \leq \frac{f_X^*(x)}{M^* g_Y^*(x)} \leq W_U(x)$ and hence we can reject/ accept values based on our squeeze before having to calculate the expensive $\frac{f_X^*(x)}{M^* g_Y^*(x)}$. I have considered a squeeze from below since my envelope is a tight fitting from above, and note that since it is easier to visualise I have considered $W_L^*(y) \coloneqq g_Y^*(y) W_L(x)$. I have naively considered (as in Figure 1):

$$W_L^*(y) = \begin{cases} \frac{4}{3}|x-1| + 1, & x \in \left(\frac{1}{4}, \frac{3}{4}\right) \\ 0 & \text{otherwise} \end{cases}$$

It should be noted that there are more effective squeezes, however for simplicity this function has been chosen, since as we will see later in our case squeezing does not necessarily increase efficiency, due to the caveats of R.

```
# Define gstar (envelope), hstar (a normal distribution) and wl (squeeze)
gstar <- function(x)  ifelse(x <= 1/m, x*m,
  ifelse(x >  1/m & x < 2-1/m, 1,(x-2)*-m))
hstar <-function(x) exp(-0.5 * ((x-1)/0.41)^2)
wlstar <- function(x) ifelse(x >= 0.25 & x <= 1.75, -4*abs(x-1)/3 + 1, 0) # Squeeze function
```

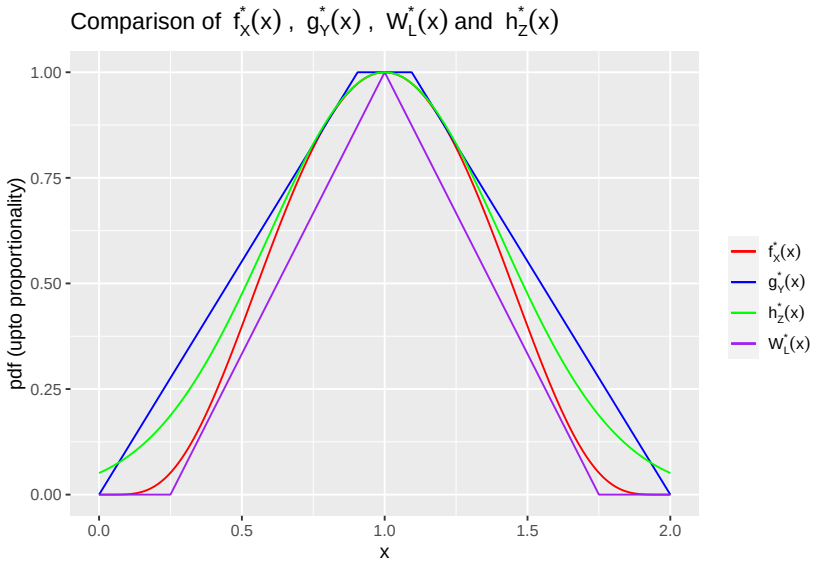

Figure 1: Comparison of un-normalised densities

The Algorithm with squeezing

1. To generate $Y = y \sim g_Y(y)$:
   - Simulate $U_1 = u_1 \sim U(0,1)$
   - Set $y = G_Y^{-1}(u_1) \sim g_Y(\cdot)$
2. Generate $U = u \sim U(0,1)$
3. To reject-accept u:
   - If $u \leq \frac{W_L^*(y)}{g_Y^*(y)}$, set $X = y$ (squeezing step).
   - Otherwise, If $u \leq \frac{f_X^*(y)}{g_Y^*(y)}$, set $X = y$.
4. Otherwise GOTO 1.

This is implemented below:

```
Ginv <- function(x)  ifelse(x < 1/(4*m -2), sqrt((4*m - 2)*x) * 1/m,
                            ifelse(x <= 1 - 1/(4*m -2), (2 - 1/m)*x + 1/(2*m),
                                   2 - sqrt((2-4*m) * (x-1) / (m^2))))
# generating from my distribution
mydistribution <- function(n, verbose=TRUE, squeeze=FALSE)
{
  x <- vector("numeric",0)
  naccept <- 0
  total <- 0
  while(naccept < n){
    count <- max(trunc((n-naccept)*(1/theta)),10)
    u <- runif(count)
    y <- Ginv(runif(count))
    if (squeeze){
      gstaru <- gstar(y) * u
      list <- ifelse(gstaru <= wlstar(y), T,
                     ifelse(gstaru >fstar(y), F, T))
      x <- c(x, y[list])
    } else {
```

```
      x <- c(x,y[(gstar(y) * u) <= fstar(y)])
    }
    total <- total + count
    naccept <- length(x)
  }
  p <- naccept/total
  if (verbose){
    cat("numerical acceptance prob = ", p,"\n")
  }
    list(x = x[1:n], p = p)
  }
```

Function timings

I analyse the timings of generating 100 000 and 1 000 000 over 30 and 20 repetitions respectively.

```
time <-function(n, iterations) # Crude function to analyse the time of the functions
{
  a <- vector("numeric",0) # vector to contain squeezed times
  b <- vector("numeric",0) # non vector to contain squeezed times
  for (i in 1:iterations) {
    set.seed(CID*i)
    a <- c(a, system.time(mydistribution(n, squeeze=TRUE, verbose=FALSE)))
    set.seed(CID*i)
    b <- c(b, system.time(mydistribution(n, squeeze=FALSE, verbose=FALSE)))
  }
  cat("average time for n=",  n, " simulations with squeeze", mean(a), "\n")
  cat("average time for n=",  n, " simulations without squeeze", mean(b), "\n")
}

time(10^5, 30) # Time 10^5 samples 40 times
```

```
## average time for n= 1e+05  simulations with squeeze 0.02591333
## average time for n= 1e+05  simulations without squeeze 0.0179
```

```
time(10^6, 20) # Time 10^6 samples 20 times
```

```
## average time for n= 1e+06  simulations with squeeze 0.25938
## average time for n= 1e+06  simulations without squeeze 0.19508
```

We can see that for $n = 100\,000$ and $n = 1\,000\,000$ the algorithm without the squeeze is faster, this is due to the fact that `ifelse` statements are computationally costly in R, and this additional cost outweighs any computation saved by evaluating the squeeze rather than $f_X(\cdot)$.

I verify below that both with and without the squeeze my algorithm gives the same samples, it can also be noted that the numerical acceptance probability printed below is similar to the theoretical acceptance probability θ, which we would expect.

```
# Ensuring that with and without the squeeze give the same samples and
set.seed(CID)
a <-mydistribution(10^6, squeeze=FALSE, verbose=TRUE)
```

```
## numerical acceptance prob =  0.8089414
```

```
set.seed(CID)
b <-mydistribution(10^6, squeeze=TRUE, verbose=TRUE)
```

```
## numerical acceptance prob =  0.8089414
```

```r
all(a$x == b$x) # Confirm whether the algorithms return identical samples
```

```
## [1] TRUE
```

Question 2

Diagnostic plots

Histogram

A histogram is obtained by splitting our samples into a series of bins (non-overlapping intervals), then plotting these bins with height proportional to the number of samples they contain. This can be scaled such that a pdf can be plotted on the same graph allowing for comparison.

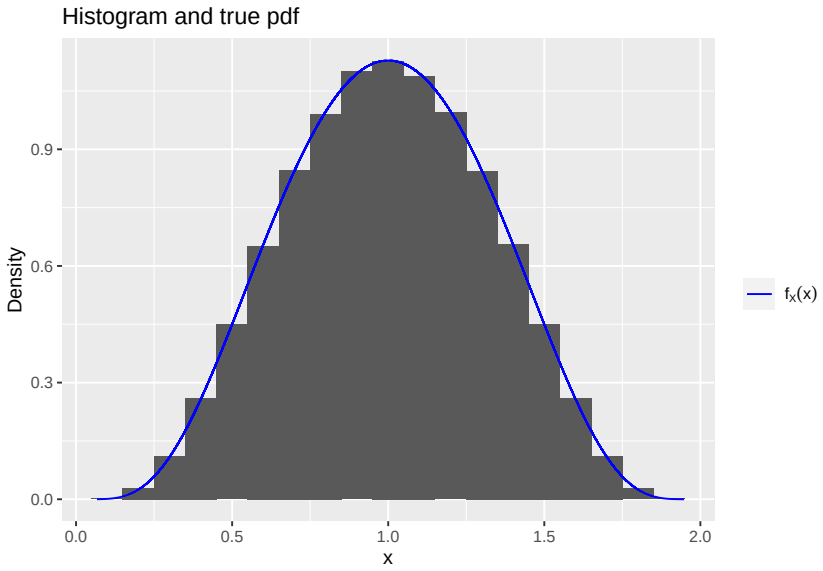

Figure 2: Histogram and true pdf

We can see from Figure 2, that the true pdf, $f_X(x)$ closely follows the histogram generated from our samples, this provides evidence to support the fact that our samples are drawn from $f_X(x)$.

QQ plot

A qq plot plots the quantiles from the sample against the quantiles that we generate from the theoretical cdf. If they form a straight line, that is $y = x$ then the samples can plausibly come from the same distribution as the theoretical cdf of which quantiles they are plotted against. To generate the QQ plot we first need to define our theoretical cdf and the inverse cdf (quantile function), this is done below.

```r
#Theoretical cdf - q
tcdf <- function(upper, q=0){
  sapply(upper, function(x) integrate(f, 0, x)$value ) - q
}

# Inverse cdf
qfunc <- function(q){
  bisect(tcdf, 0+10^-8, 2-10^-8,q=q)$root
}
```

```
# making sure it works on a vector and returns a vector
qf <- function(p) unlist(lapply(p, qfunc))
```

We can see from the below plot (figure 3) that it does indeed form a line very close to $y = x$ hence we can deduce that it is plausible that the samples are from the same distribution as the theoretical cdf represents.

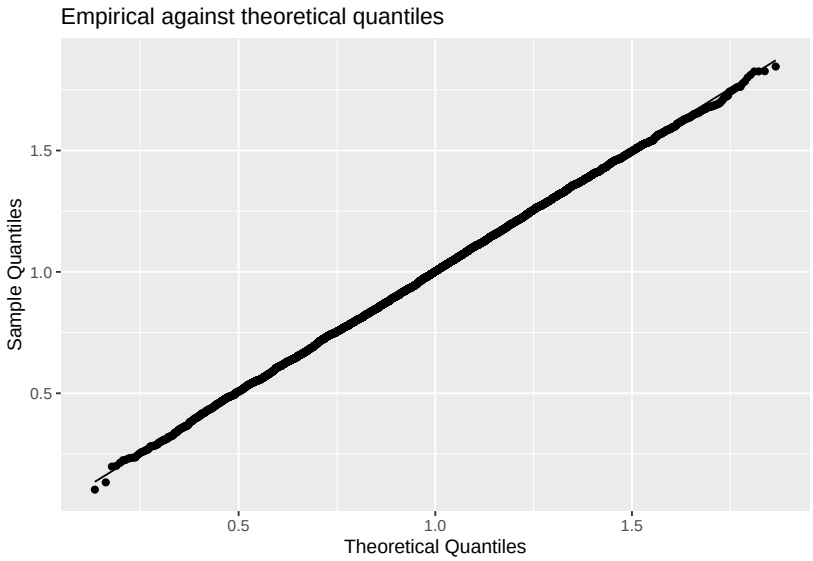

Figure 3: QQ plot of empirical against theoretical

Empirical cdf vs Theoretical Cdf

```
set.seed(CID)
a <-mydistribution(10^3, squeeze=FALSE, verbose=FALSE)
emcdf=ecdf(a$x) # Empirical cdf
# KS plot
mylabs=list("Empirical", "Max difference", "Theoretical")
cols = c("Theoretical"="blue","Empirical"="red","max" = "green")
x_plot = data.frame( x=a$x, theoretical=tcdf(a$x), Empirical=emcdf(a$x))
max <-which.max(abs(x_plot$theoretical-x_plot$Empirical))
p <- ggplot(x_plot)
p+ labs(y="Cumulative frequency",title="Empirical vs Theoretical cdf's of f")+
  geom_line(aes(x,theoretical,colour="Theoretical"))+
  geom_line(aes(x,Empirical,colour="Empirical"))+
  geom_segment(aes(x=x[max], y=theoretical[max], xend=x[max], yend=Empirical[max], colour="max"))+
  xlim(0,2)+
  scale_colour_manual(name="",values=cols, labels=mylabs)
```

In Figure 4, we can see that the empirical and theoretical cumulative density functions almost perfectly align, again supporting the fact that the `mydistribution()` successfully generates samples from $f_X(\cdot)$. This plot shows very similar information to the QQ plot mentioned previously, however it is a different way to visualise it and allows easy visualisation of the K.S statistic, which we will see later.

I will now consider Lag plots and an autocorrelation plot to help determine whether my function produces truly random and uncorrelated samples.

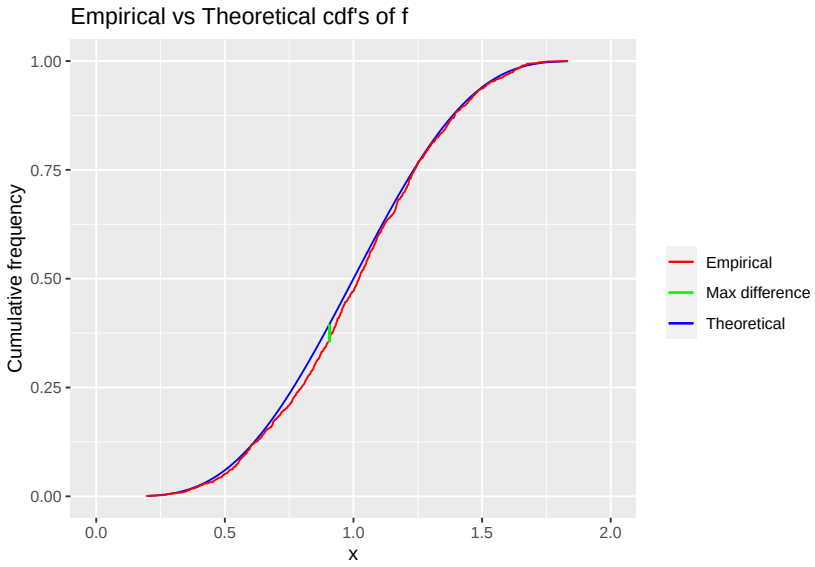

Figure 4: Comparison of the theoretical and empirical cdfs

Lag plot

The samples evaluated at the cdf , $F_X(x_i)$, are compared to those 1,2,3,4 behind in the corresponding lag scatter plots (Figure 5).

We can observe from our lag plots, figure 5, shows to be a random scatter, with no correlation or grouping, which is exactly what we would expect, suggesting that there is no correlation using our algorithm to generate $f_X(\cdot)$.

Autocorrelation plot

Consider an autocorrelation plot, that is where we examine the randomness of the sample and those that lag behind it, if all autocovariances are close to zero, we can conclude that there appears to be no serial correlation in the sample. This is exactly what we observe in figure 6.

```
set.seed(CID)
df <- data.frame(x=a$x)
ggAcf(df)+
  labs(title="Autocovariance sequence of generated Data")
```

Statistical tests

Throughout this section I will consider the following null hypothesis and significance level:

- H_0: the samples are distributed $f_X(\cdot)$
- H_a: the samples are not distributed $f_X(\cdot)$
- Significance level $\alpha = 0.01$

Kolmogorov-Smirnov test

The Kolmogorov-Smirnov test is a test used to determine whether a sample comes from a given distribution. The K.S of statistic of a sample of size n is defined by:

$$D_n = \sup_x |F_n(x) - F(x)|$$

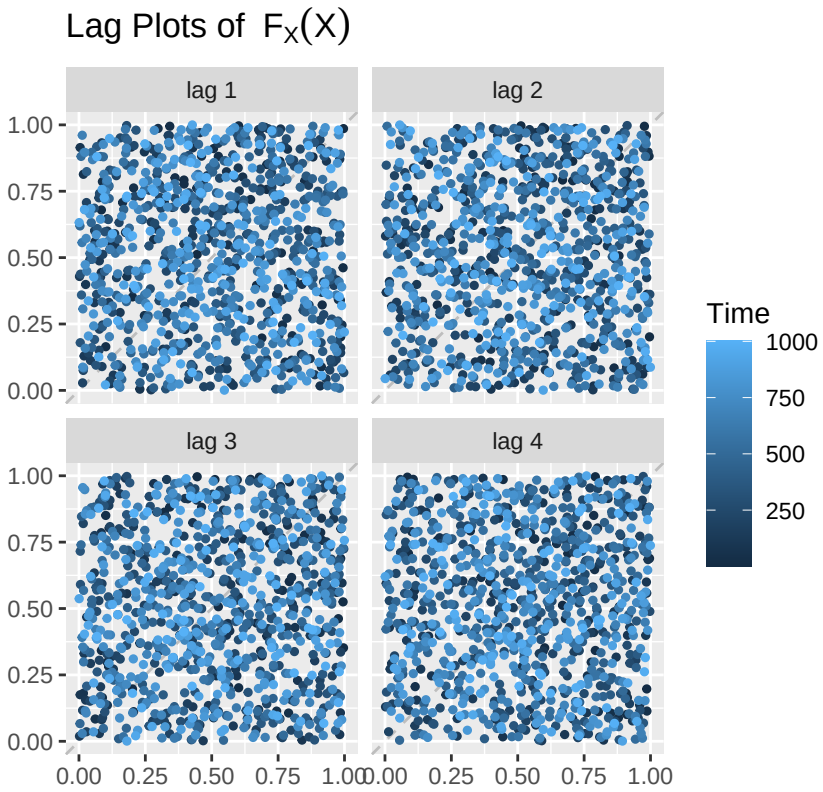

Figure 5: Lag plots of $F_X(x)$

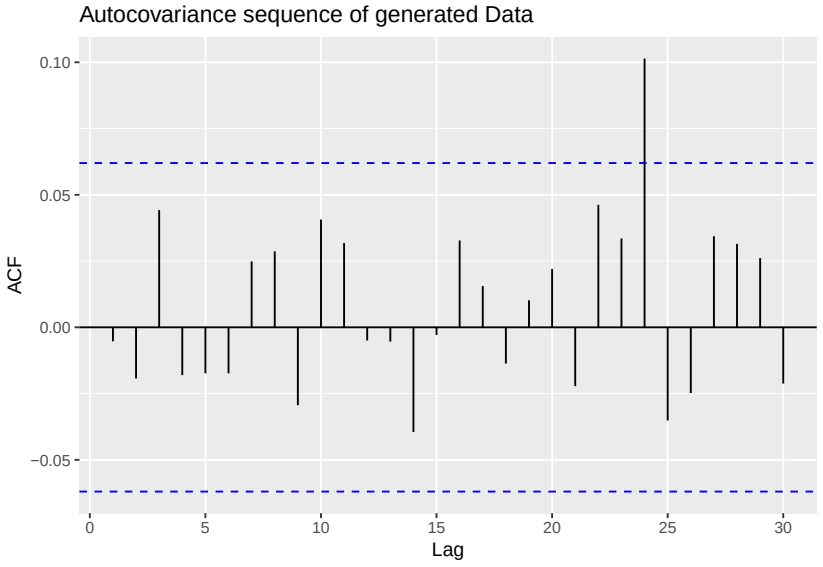

Figure 6: Autocorrelation plot of sequence of generated Data

where $F_n(x)$ represents the empirical cumulative distribution. D_n is then compared to the tails of the Kolomogormov distribution to compute a p-value, an advantage of the K.S test is that the critical values of the test statistic do not depend on the underlying distribution being tested. We can think of the K.S as the maximum vertical distance between the empirical and theoretical cumulative distribution functions, visually this is seen in Figure 4 by the green line, whereby the K.S test is run on the same data below:

```
a <- ks.test(a$x, tcdf)
print(paste0("K.S statistic = ", unname(a$statistic)))
```

```
## [1] "K.S statistic = 0.0391524840237142"
```

```
print(paste0("p-value =",  a$p.value))
```

```
## [1] "p-value =0.0932204531364751"
```

where we do not reject our null hypothesis unless $p < 0.01$, hence we have no reason to reject our null hypothesis from the above test. I will now consider a simulation study of the K.S test for 400 repetitions on various sample sizes.

```
sample_sizes = c(10^2, 4*10^2, 10^3, 4*10^3)
repetitions=400
# Function to perform KS test on a random sample of size 'size'
kstest <- function(size){
  x = mydistribution(size,verbose=FALSE)$x
  kstestx = ks.test(x, tcdf)
  p = kstestx$p.value
  D = unname(kstestx$statistic)
  data.frame(p = p, D = D, N = size)
}
#Function to repeat the kstest a given number of times
repkstest <-function(x){
  df <- rdply(repetitions, kstest(x))
  subset(df, select = -.n)
}
# Merge data frames from various KS tests
df <- data.frame(p=double(),D=double(),N=integer(),stringsAsFactors=FALSE)
set.seed(CID)
for (i in sample_sizes){
  df <- merge(df, repkstest(i), all=TRUE)
}
```

N	Mean p-value	Std Dev (p)	Mean D	Std Dev (D)
100	0.516	0.293	0.085	0.025
400	0.511	0.292	0.043	0.013
1000	0.517	0.293	0.027	0.008
4000	0.506	0.289	0.014	0.004

We would expect our D values to decrease, (since, informally, adding more samples should only decrease the maximum distance between our empirical and theoretical cdfs) which we can see they are from our table and figure 7. We also expect our p-values to be uniformly distributed, which it approximately is (considering the above table and figure 7. Hence we have no reason to reject our null hypothesis.

Chi squared test

The Chi squared test tests whether a sample comes from a given distribution. It works by splitting data into k non overlapping intervals, we then count the number of samples in each interval, let O_j be the observed

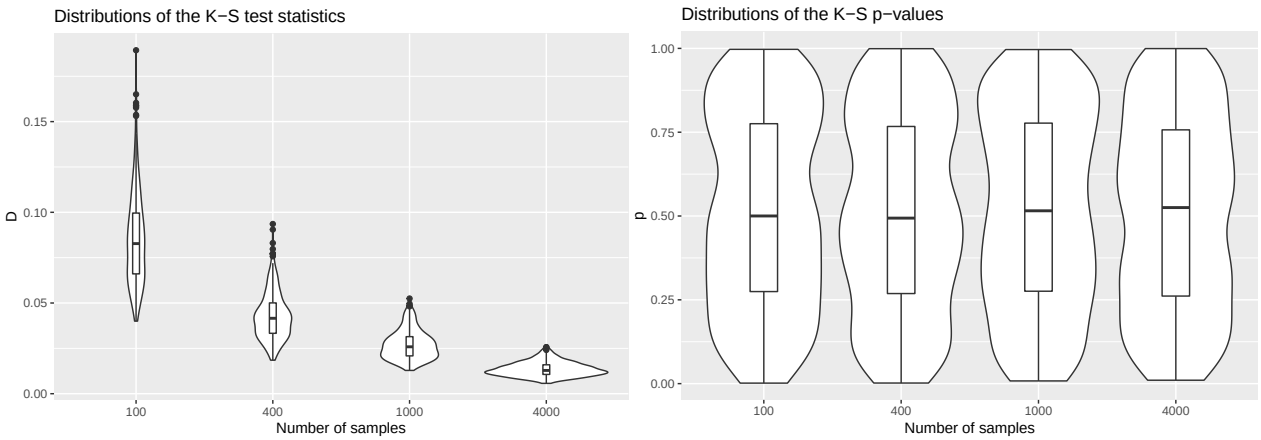

Figure 7: Violin plot of K.S p values and K.S statistics

number of samples in the i^{th} interval, and similarly let E_j be the expected number of samples in the i^{th} interval. Then chi squared statistic is then denoted:

$$X^2 = \sum_{j=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

This can be shown to have a limiting distribution of the chi squared distribution (with degrees of freedom dependent on the number of no.empty bins), under the null hypothesis.The statistic is then compared to a critical value of the chi squared distribution to determine the p-value. We can see from the table below and figure 8 that our p-value appears to be uniformly distributed, hence we have no evidence to reject our null hypothesis.

This test is sensitive to the choice of bins, particularly when the bins are empty or almost empty. I have constructed the first and last bins to be of larger size to avoid this, since it can be noted that my pdf is close to 0 here.

```
bins <- seq(0,2,l=12)
bins <- bins[-c(2,3,10,11)] # Construct bins
p<-numeric(0)
# Calculate the probability of being in each bin
for (i in 1:(length(bins)-1)){
  p <- c(p, integrate(f,bins[i],bins[i+1])$value)
}
sample_sizes = c(500, 10^3, 5*10^3, 10^4)
repetitions = 1000
# Function that performs the chi squared test on a sample
chitest <-function(n){
df <- data.frame(x=mydistribution(n,verbose=FALSE)$x)
df <- transform(df, group=cut(df$x,  breaks=bins,))
count <- count(df, bins = df$group)
count <-cbind(count,p)
chi <- chisq.test(count$n, p = count$p)
data.frame(p = chi$p.value, N = n)
}
# Function to repeat the chi squared test a given number of times
repchitest <-function(x){
  df <- rdply(repetitions, chitest(x))
```

```
  subset(df, select = -.n)
}

# Appling the above functions to for a datafram of all results
df <- data.frame(p=double(),
                 N=integer(),
                 stringsAsFactors=FALSE)
set.seed(CID)
for (i in sample_sizes){
  df <- merge(df, repchitest(i), all=TRUE)
}
```

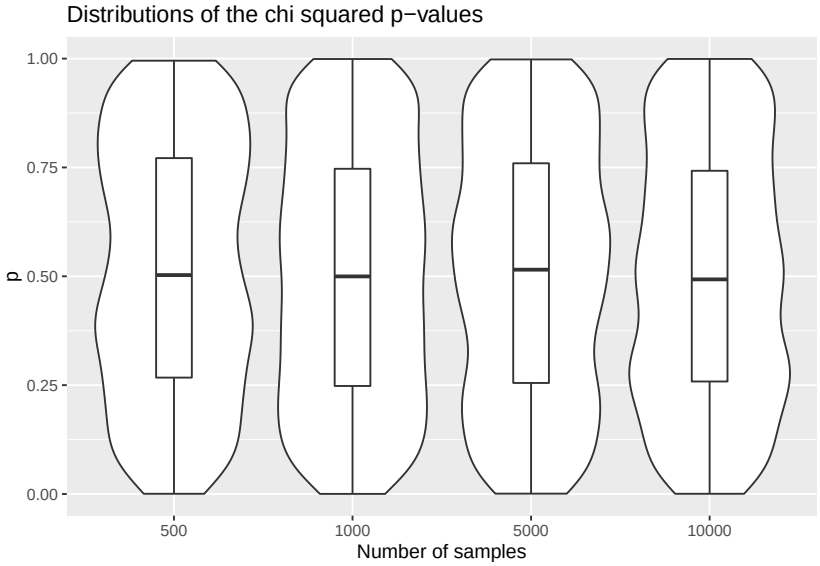

Figure 8: Violin plot of Chi squared p values

N	Mean p-value	Std Dev (p)
500	0.514	0.288
1000	0.501	0.288
5000	0.509	0.290
10000	0.499	0.290

Anderson–Darling Test

The Anderson–Darling test is used to test if a sample has come from a specified distribution. It is similar to the K.S test although places more weight on the tales of the distribution. The test statistic A is defined by:

$$A^2 = -N - \sum_{i=1}^{N} \frac{2i-1}{N} \left(\ln F(Y_i) + \ln(1 - F(Y_{N+1-i}))\right)$$

where $Y_1, Y_2, \ldots, Y_N$ are the samples in increasing order.

A is then compared to the critical values of its distribution which can be calculated, which are then in turn used to calculate the p-values.

I will perform a simulation study with 400 repetitions on various sample sizes.

```
sample_sizes = c(10^2, 4*10^2, 10^3, 4*10^3)
repetitions=400
adtest <-function(n){
ad <- ad.test(mydistribution(n,verbose=FALSE)$x, null = "tcdf",  nullname="theoretical cdf")
data.frame(p = ad$p.value, N = n)
}
repadtest <-function(x){
  df <- rdply(repetitions, adtest(x))
  subset(df, select = -.n)
}
df <- data.frame(p=double(),
                 N=integer(),
                 stringsAsFactors=FALSE)
set.seed(CID)
for (i in sample_sizes){
  df <- merge(df, repadtest(i), all=TRUE)
}
```

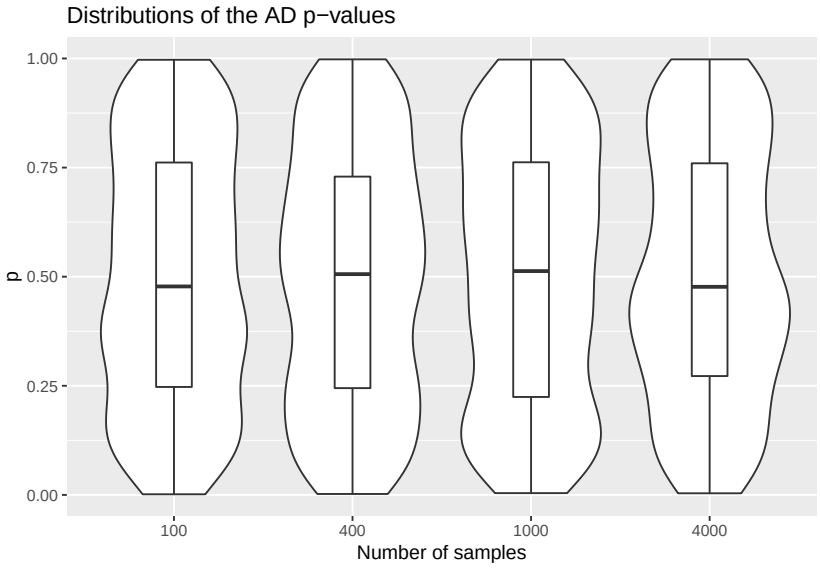

Figure 9: Violin plot of A.D p values

N	Mean p-value	Std Dev (p)
100	0.497	0.290
400	0.495	0.289
1000	0.502	0.292
4000	0.504	0.288

It can again be noted from the table above and figure 9, that p is uniformly distributed, as we would expect, and hence we have no reason to reject our null hypothesis.

Question 3

We aim to integrate $\theta = \int_0^2 f_X^*(x)dx$, we know from question 1, where we analytically integrated this to $\theta = \frac{\sqrt{\pi}}{2}$. Hence the normalising constant of $f_X^*(x)$ is $\frac{1}{\theta}$ since by our calculations we have $\int_0^2 \frac{f_X^*(x)}{\theta} = 1$ I will

now integrate this using:

1. Crude Monte Carlo
2. Hit or Miss Monte Carlo
3. Using Control Variates to reduce variance

Crude Monte Carlo

To evaluate $\theta = \int f^*(x)dx$ consider the decomposition of $f^*(x) = \varphi(x)h(x)$, where $h(x)$ is a valid pdf. The integral then becomes: $\int f^*(x)dx = \int \varphi(x)h(x)dx = E_h[\varphi(x)]$ which we know can be estimated by $\frac{1}{n}\sum_{i=1}^{n}\varphi(X_i)$ where $X_1, X_2, \ldots, X_n \overset{iid}{\sim} h(\cdot)$.

We show from lectures that this is an unbiased estimator for $\int f^*(x)dx$, with variance $\frac{k}{n}$, where $k = \int[\varphi(x) - \theta]^2 h(x)dx$.

For my calculations I define $\varphi(x) = \frac{f_X^*(x)}{g(x)}$ and $h(x) = g(x)$ and hence, I seek $\frac{1}{n}\sum_{i=1}^{n}\frac{f_X^*(X_i)}{g(X_i)}$ where $X_1, X_2, \ldots, X_n \overset{iid}{\sim} g(\cdot)$. And the theoretical variance is calculated using the integrate function in the code below giving $k \approx 0.0921$.

Hit or Miss Monte Carlo

Hit or Miss Monte Carlo calculates $\theta = \int_a^b f^*(x)dx$ by considering the $\tilde{\theta} = c(b-a)\frac{1}{n}\sum_{=1}^{n}\mathbb{I}(v_i \leq h(u_i))$, where $v_i \sim U(0,c)$ and $u_i \sim U(a,b)$, and where c is an upper bound of $f^*(x)$ for $x \in (a,b)$. We calculate in lectures that this is an unbiased estimator for θ by noticing that $\mathbb{I}(V \leq h(U)) \sim$ Bernoulli$[P(V \leq h(U))]$, we also calculate $\frac{k}{n}$ where $k = \theta[c(b-a) - \theta]$ as the variance from lectures.

Applying this to my integral, I note $a = 0, b = 2, c = 1$ where the maximum, c, was calculated in Question 1. I therefore seek $\frac{2}{n}\sum_{=1}^{n}\mathbb{I}(v_i \leq h(u_i))$, where $v_i \sim U(0,1)$ and $u_i \sim U(0,2)$.

And the theoretical variance is calculated below giving $k \approx 0.987$, which we notice is significantly higher than that of Crude Monte Carlo, which is what we would expect.

Control Variates

I seek to reduce the variance of the estimator for $\theta = E_f[\varphi(X)] = E(Z)$, hence I consider the use of control variates. That is I consider the following estimator $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n}[Z_i - (W_i - E[W])]$, where W is a function of X and has known expectation and is correlated with Z. From lectures we know this is an unbiased estimator for θ and has variance $\frac{1}{n}[\text{var}(Z) + \text{var}(W) - 2\text{cov}(Z,W)]$ and hence if $\text{cov}(Z,W)$ is large we have then we will have a small $var(\hat{\theta})$.

Applying this to our problem we see that we can take W to be the polynomial of degree two (which has the benifit of being computationally cheap) which best fits our $Z \coloneqq \frac{f_X^*(x)}{g(x)}$, we can see from figure 10, that this fits $\frac{f_X^*(x)}{g(x)}$ well and visually seems to have strong correlation with it. Calculating the correlation from a large sample we see that the $\text{cov}(Z,W) \approx 0.981$, which is very high leading us to believe there will be a large variance reduction. Which, although we dont calculate the theoretical variance explicitly we see from considering the variance of samples there is a verry large variance reduction.

```
theta = sqrt(pi)/2 # Value we calculated for the integral of fstar prior
g <- function(x) gstar(x)*m / (2*m-1) # Define g
fstar_g <- function(x) fstar(x)/g(x) # Define fstar/g
# Calculate the theoretical variance of crude and hit or miss MC
var_crude <-function(x) (fstar_g(x) - theta)^2 *g(x)
k_crude = integrate(var_crude,0,2)$value
cat("Variance of for MC for n=1 is",k_crude,"\n")
```

```
## Variance of for MC for n=1 is 0.09214605
```

```
x_hm <- theta*(2-theta)
cat("Variance of MC hit or miss for n=1 is",x_hm,"\n")
```

```
## Variance of MC hit or miss for n=1 is 0.9870557
```

```
# Calculate the least squares best fit of a deg 2 poly
x <- seq(0+10^-8,2-10^-8,l=10^7)
y <- fstar_g(x)
fit <- lm(y~poly(x,2,raw=TRUE))
w <- function(x) fit$coefficients[1] + fit$coefficients[2]*x + fit$coefficients[3]*x^2
expect <- function(x) g(x) * w(x) # Integrate to get expectation of w
mn = integrate(expect,0,2)$value # Expected value of w
cat("cor(fstar/g (x),w)=",cor(fstar_g(x),w(x)),"\n")
```

```
## cor(fstar/g (x),w)= 0.9813603
```

```
cat("Theta = ",theta,"\n")
```

```
## Theta =  0.8862269
```

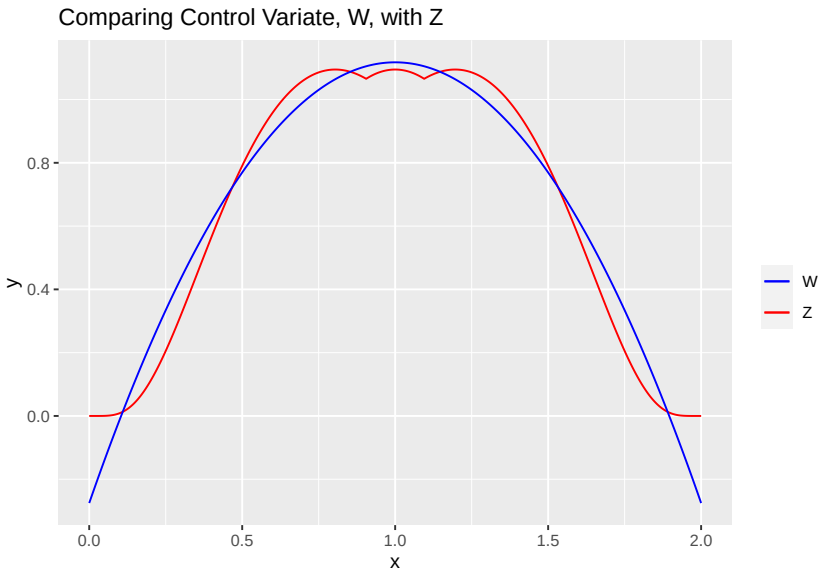

Figure 10: Comparing Control Variate, W, with Z

Firstly I consider analysing the variance and mean of each afore mentioned mentioned methods for one sample of size 1 000 000.

```
n <- 10^6
set.seed(CID)
u <- runif(n)
y <- Ginv(u)

# Crude MC
x <- fstar_g(y)
sigma2 <- var(x)
mu <- mean(x)

# Control variates
x <- x-(w(y) -mn)
```

```r
sigma2 <- c(sigma2,var(x))
mu <- c(mu, mean(x))

# Hit, miss
v <- runif(n, 0, 2)
x <- ifelse(u <= fstar(v), 2, 0) # we calculated the maximum of fstar previously to be 1
sigma2 <- c(sigma2,var(x))
mu <- c(mu, mean(x))

df<- data.frame(
  Method=c("Crude", "Control Variates", "Hit, miss"),
  Variance=sigma2,
  Mean=mu,
  Error=mu-sqrt(pi)/2)
```

Method	Mean (Theta)	Variance
Control Variates	0.8862577	0.0032236
Crude	0.8861835	0.0921875
Hit, miss	0.8855200	0.9868953

We know that from analytically evaluating the integral $\theta \approx 0.886$, and we notice that the mean of all our methods is close to this (as we would expect since they are unbiased), signifying that they all provide a good estimate for the required integral. And also their variance, that is k as we defined prior aligns with the k's we calculated previously (note we did not calculate k for control variates). We also notice that Hit or Miss has significantly higher variance than the Crude method which in turn has much higher variance than when the Control Variates method which is what we would expect.

Below I perform a Simulation study, considering the mean of θ for each of the afore mentioned methods, for various sample sizes with 100 repetitions.

```r
# Putting above into loop to perform simulation study
sample_sizes = c(10^2, 5* 10^2, 10^3)
repetitions = 100
set.seed(CID)

mcint <-function(n){
  u = runif(n)
  y <- Ginv(u)
  x = fstar_g(y)
  x0 <- mean(x)
  x1 <- mean(x-(w(y) -mn))
  v <- runif(n, 0, 2) # we calculated the maximum prior
  x2 <- mean(ifelse(u <= fstar(v), 2, 0))
  data.frame(
    t=c(x0, x1, x2),
    Method=c("Crude","Control Variates","Hit, miss"),
    N=n)
}

repmcint <-function(x){
  df <- rdply(repetitions, mcint(x))
  subset(df, select = -.n)
}
```

```
df <- data.frame(
  t=double(),
  Method=integer(),
  N=integer(),
  stringsAsFactors=FALSE)
set.seed(CID)
for (i in sample_sizes){
  df <- merge(df, repmcint(i), all=TRUE)
}
```

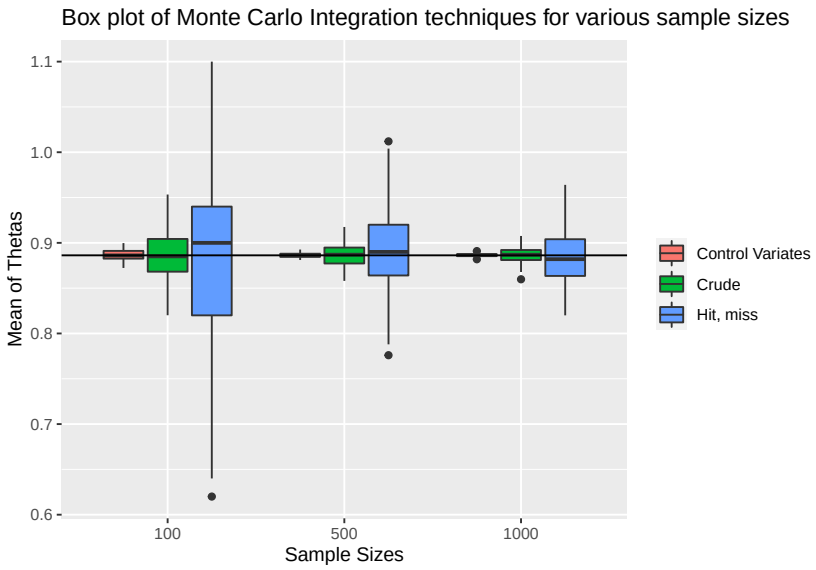

Figure 11: Box plot of Monte Carlo Integration techniques for various sample sizes

Method	Mean of theta	Var of theta
Control Variates	0.88638	0.0000134
Crude	0.88658	0.0003570
Hit, miss	0.88591	0.0042609

We observe from the above table and figure 11 that these methods do indeed seem successful since they all are centered around θ, with there being large decrease in variance between from Hit or Miss MC to Crude MC to our Control Variates method, which is what we would expect.

Bibliography

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, 11/12/2020

McCoy, E, (2020), *Stochastic Simulation*, lecture notes, MATH96054, Imperial College London, delivered Autumn 2020.

McCoy, E, (2020), *FormativeCWex.Rmd*, Exemplar, MATH96054, Imperial College London.

Frederic, P , Lad, F , (2002), *A Technical Note on the Logitnormal Distribution*, https://www.math.canterbury.ac.nz/research/ucdms2003n7.pdf, accessed 11/12/2020