Stochastic Simulation - Assessed Coursework

Tudor Trita Trita - CID: 01199397

13/12/2019

Disclaimer

This is my own work unless stated otherwise. I have relied heavily on the formative CW given by Prof. McCoy, especially in Question 2.

Preliminary Work:

```
suppressMessages(library(numbers,verbose=FALSE,warn.conflicts=FALSE,quietly=TRUE))
suppressMessages(library(tidyverse,verbose=FALSE,warn.conflicts=FALSE,quietly=TRUE))
suppressMessages(library(ggplot2,verbose=FALSE,warn.conflicts =FALSE,quietly=TRUE))
suppressMessages(library(resample,verbose=FALSE,warn.conflicts=FALSE,quietly=TRUE))
suppressMessages(library(GoFKernel,verbose=FALSE,warn.conflicts =FALSE,quietly=TRUE))
suppressMessages(library(forecast,verbose=FALSE,warn.conflicts=FALSE,quietly=TRUE))
suppressMessages(library(pracma,verbose=FALSE,warn.conflicts =FALSE,quietly=TRUE))
knitr::knit_meta(class=NULL, clean = TRUE)
```

```
## list()
```

```
options(warn=-1)
```

The first thing we will do is to find the largest prime factor of my CID Number:

```
CID = 01199397
largest_prime_factor = max(primeFactors(CID))
sprintf("Largest prime factor of %s is %s", CID, largest_prime_factor)
```

```
## [1] "Largest prime factor of 1199397 is 739"
```

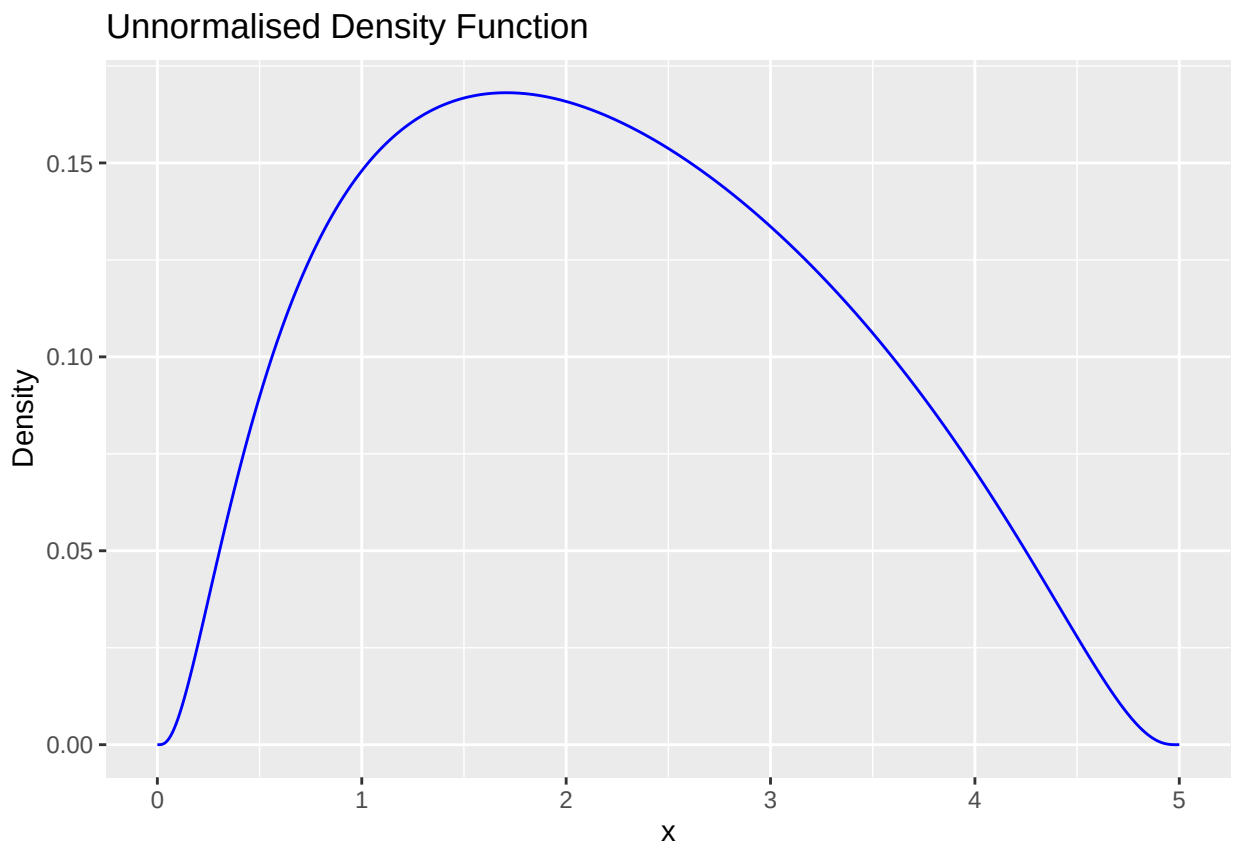
After searching through the Params.pdf file, we find the parameters for the density to be $a = 0, b = 5, c = 2.25, d = 0.30$.

Question 1

We are given $f_X^*(x) = \frac{1}{x(5-x)} \exp\left(-\frac{1}{2.25}\left(0.3 + \log\left(\frac{x}{5-x}\right)\right)^2\right)$. We will go ahead and plot this function for the range $(0, 5)$.

```
fstar <- function(x){
  r = 1/(x*(5-x))*exp(- (1/2.25)*(0.3 + log(x/(5-x)))^2)  # Main computation
  r[is.nan(r)] <- 0  # Setting any nan values at the boundaries to 0
  return(r)
}

x_vals = seq(0, 5, l=1000)
df_fstar = tibble(x=x_vals, fx=fstar(x_vals))
ggplot(df_fstar)+
```

```
  geom_line(aes(x, fx), color="blue")+
  labs(y="Density", title="Unnormalised Density Function")
```

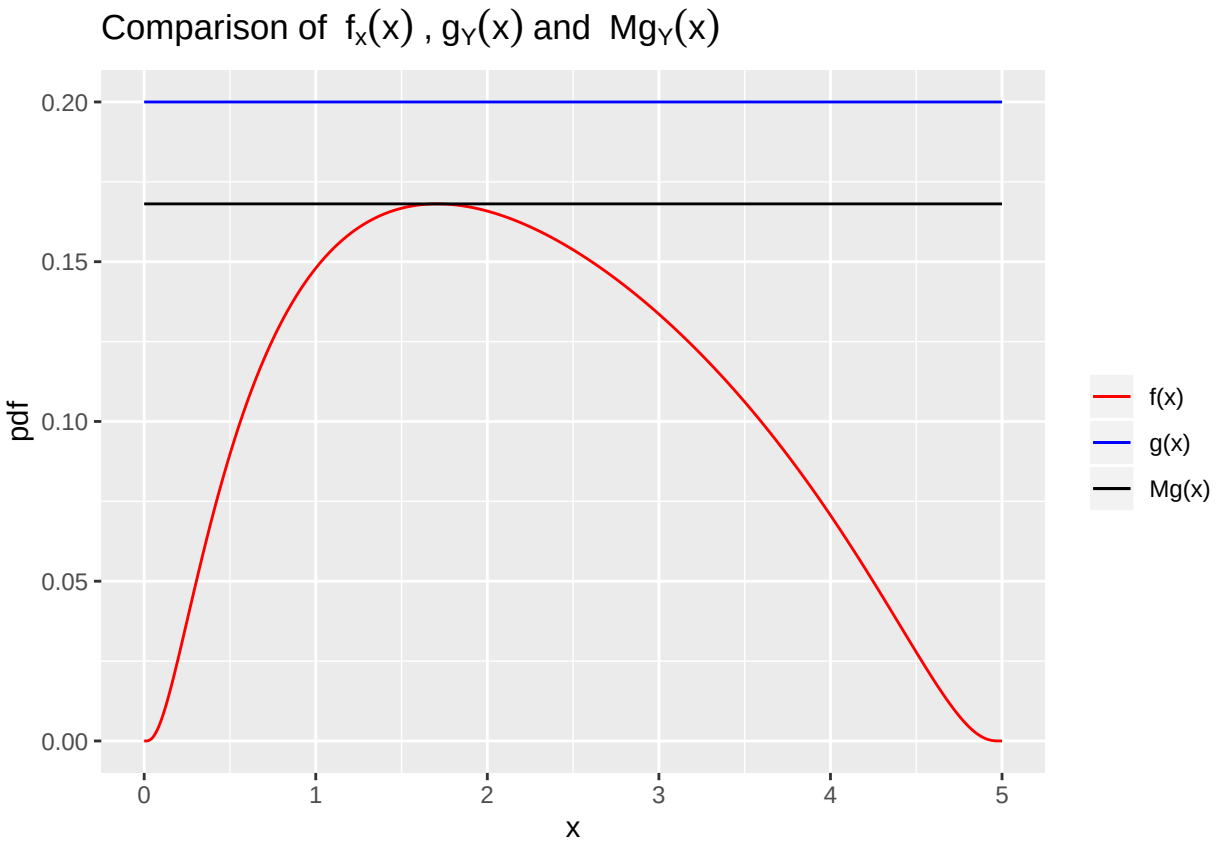


Fitting a Uniform Envelope

To perform rejection sampling, we would like to find a function $g_Y(\cdot)$ which resembles the shape of $f_X^*(\cdot)$.

We begin by fitting a uniform envelope for this example. To fit an uniform density, we can take the function $g_Y(y)_{=}0.2$ between the interval $(0, 5)$. We must then find $M = \sup_x\{f_X^*(x)/g_Y(x)\}$ first. We can do this in R using the *optimize* function.

```
gstar_uniform <- function(y) {rep(0.2, length(y))}  # Forming Uniform gstar
fg1 = function(x) {fstar(x)/0.2}  # Helper function to compute f/g
M1 = optimize(fg1, lower=0, upper=5, maximum=TRUE)$objective  # Computing M for uniform envelope
x_vals = seq(0, 5, l=1000)
df_Mfg1 = tibble(x=x_vals, fx=fstar(x_vals), gx=gstar_uniform(x_vals), Mgx=M1*gstar_uniform(x_vals))  #

# Plot of f(x), g(x) and Mg(x)
p1 <- ggplot(df_Mfg1)
p1 + geom_line(aes(x, fx, colour="f(x)")) +
     geom_line(aes(x, gx, colour="g(x)")) +
     geom_line(aes(x, Mgx, colour="Mg(x)")) +
     labs(y="pdf", title=expression("Comparison of "~f[x](x)~","~g[Y](x)~"and "~Mg[Y](x))) +
     scale_colour_manual("", values=c("f(x)"="red", "g(x)"="blue", "Mg(x)"="black"))
```



As we can see, the shape of the uniform doesn't resemble the shape of the distribution all that well, especially at the tails. However, sampling from an uniform is computationally cheap, especially when using the *runif* function. We will compare our results with a more complicated envelope later in this coursework.

We now proceed to compute the acceptance probability for this envelope. The acceptance probability measures the expected number of values that are accepted at each iteration of the Rejection Sampling Algorithm.

As we don't have the normalised pdf, the Acceptance Probability is given by

$$AP = \frac{\int f_X^* xx}{M}$$

```
integral_fstar = integrate(fstar, 0, 5)[1]$value  # Numerical Integral of fstar in the range
cf = 1/integral_fstar  # Normalising Constant for fstar
acceptance_prob1 = integral_fstar/M1
sprintf("Acceptance Probability for Uniform Envelope = %.1f percent", acceptance_prob1*100)
```

```
## [1] "Acceptance Probability for Uniform Envelope = 63.3 percent"
```

The acceptance probability of approximately 63% is low, and this is because the uniform envelope does not resemble the shape of the density very well.

Rejection Sampling for Uniform Envelope

The rejection sampling technique is a method of generating random variates from a distribution. This method is useful if Inversion from this distribution is not analytically possible, or it may be too expensive to do numerically. The rejection algorithm to the Uniform envelope is as follows:

ALGORITHM:

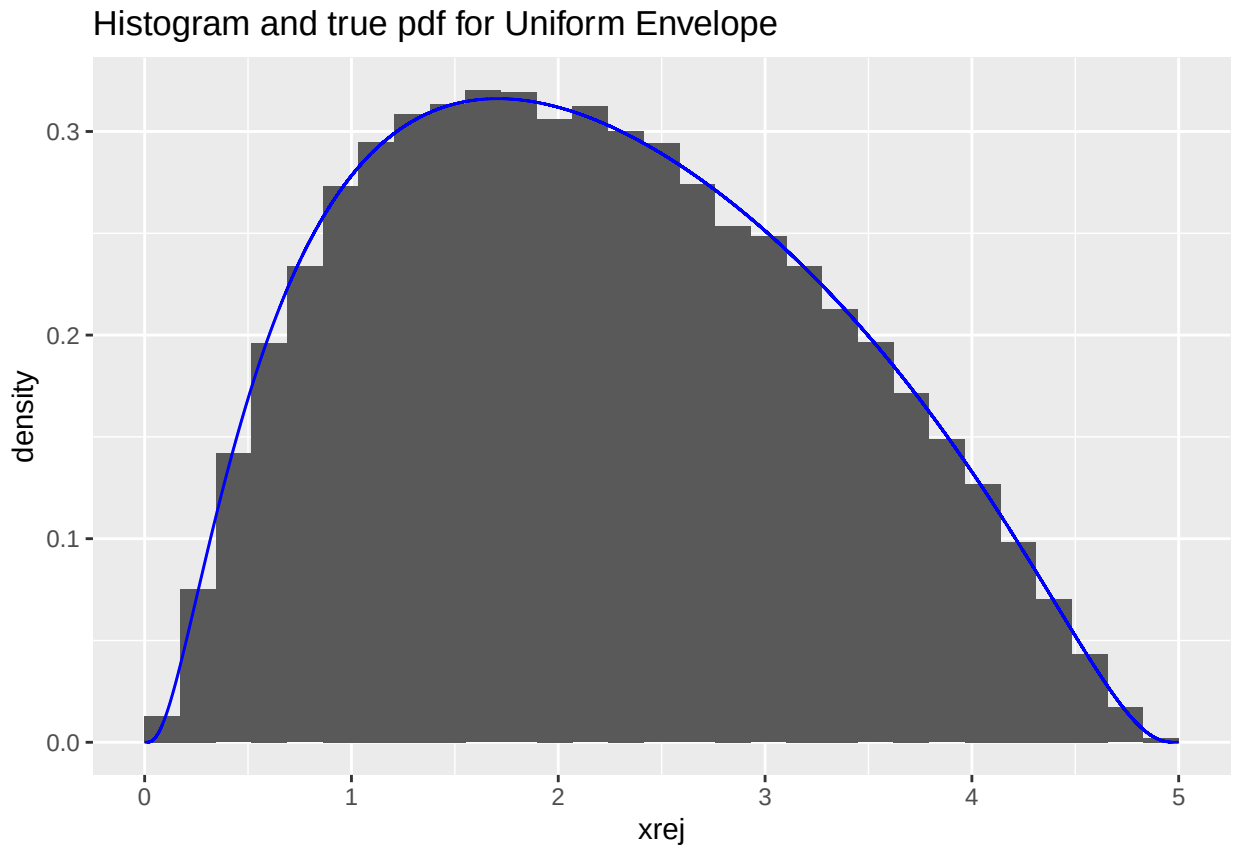
1. Generate $Y = y \sim g(y) = U(0, 5)$
2. Generate $U = u \sim U(0, 1)$
3. If $u \leq f^*(y)/Mg(y)$, set $X = y \sim f_X(\cdot)$
4. Otherwise GOTO Step 1.

The function that implements this algorithm is below:

```
rejection_uniform <- function(n, params){
    M = params[1]
    AP = params[2]
    x <- vector()  # Simulatedv Values in x
    p <- vector()  # Estimated Acceptance Probabilities in P
    len_x = 0

    while(len_x <= n){
      n_to_gen = max((n - len_x)*AP, 10)  # Determine number to generate, not less than 10
      Y = runif(n_to_gen, min=0, max=5)  # Generating from U(0, 5)
      U2 = runif(n_to_gen, min=0, max=1)
      cond <- U2 <= fstar(Y)/(M*0.2)  # Accept X=y if u <= f(y)/Mg(y)
      p <- c(p, sum(cond)/n_to_gen)  # Keeping track of estimated acceptance probability
      x <- c(x, Y[cond])  # Concatenate accepeted values
      len_x <- length(x)
     }
    return(list(x=x[1:n], p=p))
}
```

We now check that this rejection algorithm has worked by fitting a histogram of the generated random variates:

```
n=100000
x_vals = seq(10^-12, 5-10^-12, l=n)
fx1 = fstar(x_vals)

x_rej_unif <- rejection_uniform(n, c(M1, acceptance_prob1))
df_hist1 = tibble(x=x_vals, fx=cf*fx1, xrej=x_rej_unif$x)
phist1 <- ggplot(df_hist1)
phist1 + labs(y="density", title="Histogram and true pdf for Uniform Envelope") +
  geom_histogram(aes(xrej, y= ..density..), breaks=seq(0, 5, l=30))+
  geom_line(aes(x, fx), color="blue")
```



Fitting a Trapezium Envelope:

We now focus on fitting a better shape to fit to the density. After some trial and error and fiddling with parameters, I have found that a trapezium will fit the density well, as the density $g_Y^*(\cdot)$ can be loosely approximated by the function

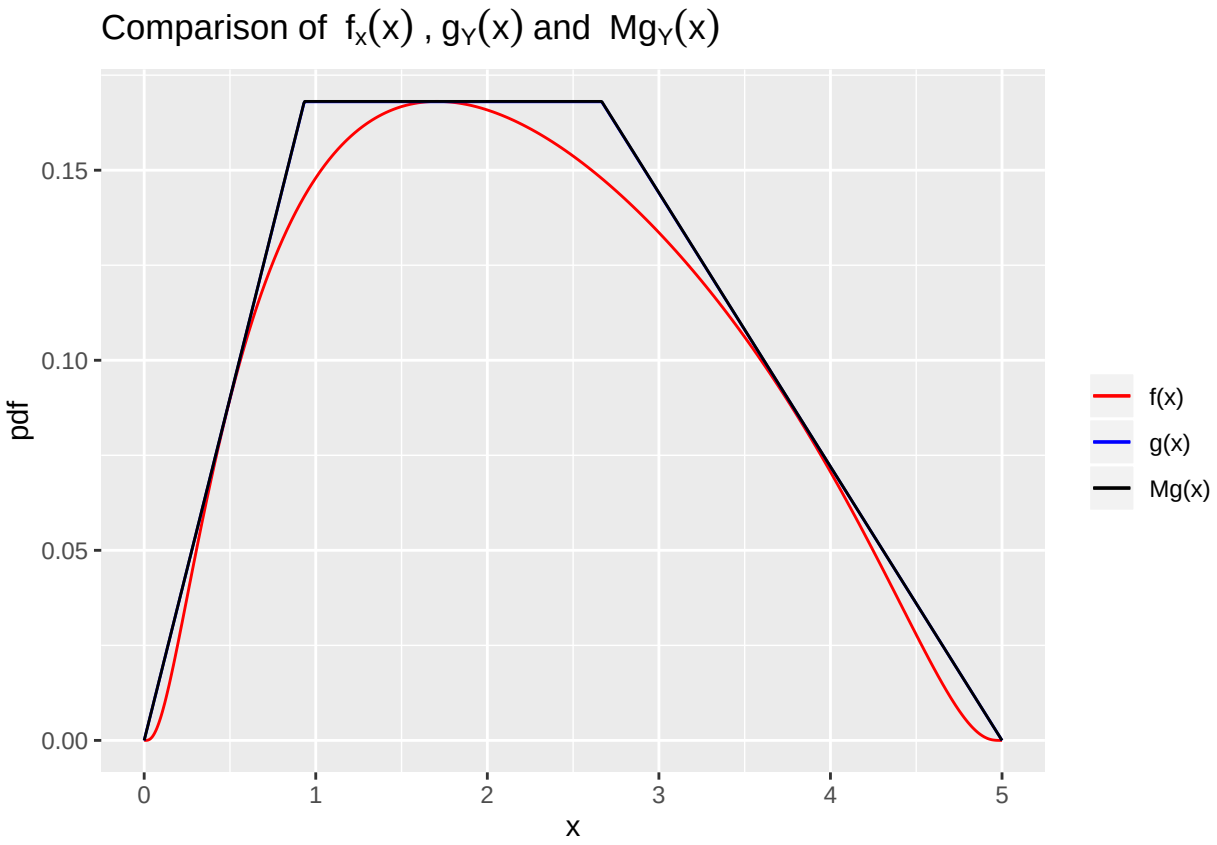
$$g_Y^*(y) = \begin{cases} \frac{9}{50}y, & 0 < y \leq \frac{14}{15}, \\ \frac{21}{125}, & \frac{14}{15} < y \leq \frac{8}{3}, \\ \frac{9}{25} - \frac{9}{125}y, & \frac{8}{3} < y < 5. \end{cases}$$

```
gstar2 <- function(x){
  alpha = 14/15
  beta = 8/3
  cond1 = x <= alpha
  x[cond1] = 0.18*x[cond1]
  cond2 = alpha < x & x <= beta
  x[cond2] = rep(0.168, length(x[cond2]))
  cond3 = x > beta
  x[cond3] = 0.36 -0.072*x[cond3]
  return(x)
}
```

We now plot this function as well as our density to see how it fits.

```
x_vals = seq(0, 5, l=1000)
fg2 = function(x_vals) {fstar(x_vals)/gstar2(x_vals)}
```

```
M2 = optimize(fg2, lower=0, upper=5, maximum=TRUE)$objective
df_Mfg2 = tibble(x=x_vals, fx=fstar(x_vals), gx=gstar2(x_vals), Mgx=M2*gstar2(x_vals))

# Plot of f(x), g(x) and Mg(x)
p1 <- ggplot(df_Mfg2)
p1 + geom_line(aes(x, fx, colour="f(x)")) +
     geom_line(aes(x, gx, colour="g(x)")) +
     geom_line(aes(x, Mgx, colour="Mg(x)")) +
     labs(y="pdf", title=expression("Comparison of "~f[x](x)~","~g[Y](x)~"and "~Mg[Y](x))) +
     scale_colour_manual("", values=c("f(x)"="red", "g(x)"="blue", "Mg(x)"="black"))
```



As we can see, the shape of the trapezium resembles the density very well. The acceptance probability for this envelope is expected to be high as it fits the shape very well.

```
g2temp <- function(x){gstar2(x)}
integral_gstar2 = integrate(g2temp, 0, 5)[1]$value
cg = 1/integral_gstar2
acceptance_prob2 = integral_fstar/(M2*integral_gstar2)
sprintf("Acceptance Probability for Trapezium Envelope = %.1f percent", acceptance_prob2*100)
```

```
## [1] "Acceptance Probability for Trapezium Envelope = 94.0 percent"
```

Rejection Sampling for Trapezium Envelope

To be able perform rejection sampling with the trapezium envelope, we need to sample from the trapezium function CDF. This is luckily relatively cheap to do using the inversion method.

INVERSION ALGORITHM

1. Generate $U_i \sim U(0,1)$
2. Set $X_i := F_X^{-1}(U_i)$

To be able use the inversion algorithm, we need to have access to the CDF and the inverse of the CDF.

The CDF of g is given by (after some tedious algebra) the function:

$$G_Y(y) = c_g G_Y^*(y) = \begin{cases} \frac{9x^2}{100}, & 0 < x \leq \frac{14}{15} \\ \frac{21x}{125} - \frac{49}{625}, & \frac{14}{15} < x \leq \frac{8}{3} \\ \frac{9x}{25} - \frac{9x^2}{250} - \frac{209}{625} & \frac{8}{3} < x < 5 \end{cases}$$

And the associated Inverse CDF is given by (after some even more tedious algebra) the function:

$$G_Y^{-1}(y) == \begin{cases} \sqrt{\frac{100x}{9c_g}}, & 0 < x \leq \frac{9c_g}{100} \cdot \left(\frac{14}{15}\right)^2 \\ \frac{125}{21}\left(\frac{x}{c_g} + \frac{49}{625}\right), & \frac{9c_g}{100} \cdot \left(\frac{14}{15}\right)^2 < x \leq c_g\left(\frac{21}{125} \cdot \frac{8}{3} - \frac{49}{625}\right) \\ 5 - \sqrt{\frac{250}{9}\left(\frac{707}{1250} - \frac{y}{c_g}\right)}, & c_g\left(\frac{21}{125} \cdot \frac{8}{3} - \frac{49}{625}\right) < x < 5 \end{cases}$$

where c_g is the normalising constant for g^* to become a PDF.

```
# CDF Function
CDF_trapezium_unscaled <- function(x){
  alpha = 14/15
  beta = 8/3
  cond1 = x <= alpha
  x[cond1] = 0.09 * x[cond1]^2
  cond2 = alpha < x & x <= beta
  x[cond2] = 0.168*x[cond2] - 0.0784
  cond3 = x > beta
  x[cond3] = 0.36*x[cond3] -0.036 * x[cond3]^2 -0.3344
  return(x)
}

# Inverse CDF
inverse_CDF_trapezium <- function(y, cg){
  alpha = cg * 0.0784  # scaling_fact * 9 * (alpha^2) / 100
  beta = cg * 0.3696  # scaling_fact*(21*beta/125 - 49/625)
  y_scaled = y/cg
  cond1 = y <= alpha
  y_scaled[cond1] = sqrt(y_scaled[cond1]) * 10/3
  cond2 = alpha < y & y <= beta
  y_scaled[cond2] = 125/21 * (y_scaled[cond2] + 0.0784)
  cond3 = beta < y & y < 0.999998  # Careful with Boundary Conditions (errors otherwise)
  y_scaled[cond3] = 5 - sqrt(250/9 * (0.5656 - y_scaled[cond3]))
  cond4 = y >= 0.999998  # Careful with boundary condition (errors otherwise)
  y_scaled[cond4] = 5  # Take it as being at the boundary
  return(y_scaled)
}
```

We now have all the ingredients needed to perform the rejection sampling for the trapezium function. The algorithm is as follows:

ALGORITHM

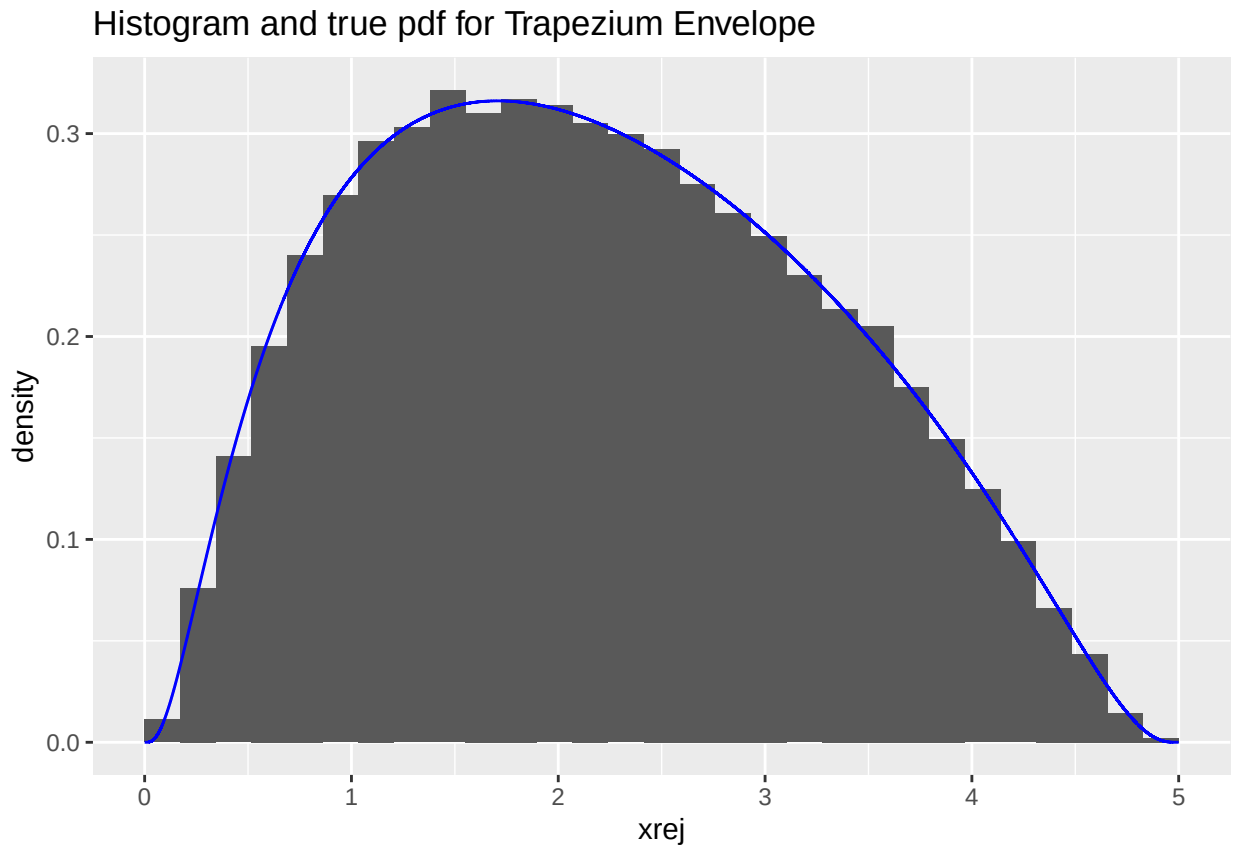
1. Generate $U_1 = u_1 \sim U(0,1)$

2. Form $Y = y = G_y^{-1}(u_1)$
3. Generate $U_2 = u_2 \sim U(0, 1)$
4. If $u_2 \leq f_X^*(y)/Mg_Y^*(y)$, set $X = y \sim f_X(x)$
5. Otherwise GOTO Step 1.

```
rejection_trapezium <- function(n, params){
    M = params[1]
    AP = params[2]
    scaling_factor = params[3]
    x <- vector()  # Simulatedv Values in x
    p <- vector()  # Estimated Acceptance Probabilities in P
    len_x = 0

    while(len_x <= n){
      n_to_gen = max((n - len_x)*AP, 10)  # Determine number to generate, not less than 10
      U1 = runif(n_to_gen, min=0, max=1)
      Y = inverse_CDF_trapezium(U1, scaling_factor)
      U2 = runif(n_to_gen, min=0, max=1)
      cond <- U2 <= fstar(Y)/(M*gstar2(Y))  # Accept X=y if u <= f(y)/Mg(y)
      p <- c(p, sum(cond)/n_to_gen)  # Keeping track of estimated acceptance probability
      x <- c(x, Y[cond])  # Concatenate accepeted values
      len_x <- length(x)
     }
    return(list(x=x[1:n], p=p))
}
```

We will now plot a histogram with the generated random variates as well as the PDF of the density.

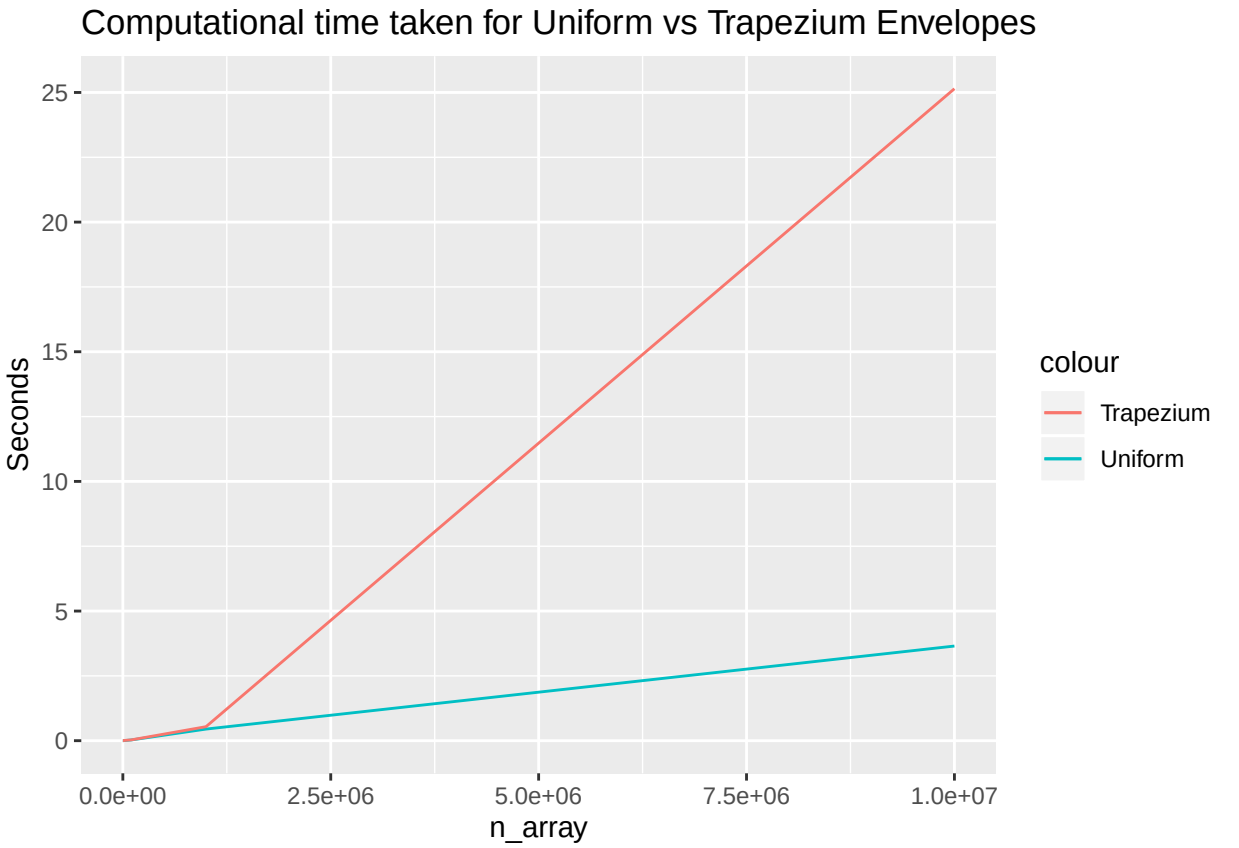
```
# Create Histogram plot for this
n=100000
x_vals = seq(0, 5, l=n)
fx2 = fstar(x_vals)

x_rejection_trapezium <- rejection_trapezium(n, c(M2, acceptance_prob2, cg))
df_hist2 = tibble(x=x_vals, fx=cf*fx2, xrej=x_rejection_trapezium$x)
phist2 <- ggplot(df_hist2)
phist2 + labs(y="density", title="Histogram and true pdf for Trapezium Envelope") +
  geom_histogram(aes(xrej, y= ..density..), breaks=seq(0, 5, l=30))+
  geom_line(aes(x, fx), colour="blue")
```



The reason why we have implemented two rejection methods using different envelopes is to discuss the advantages and disantages of the methods. Clearly, the uniform envelope will be computationally less complex than the trapezium envelope to generate from, but the acceptance probability is much higher for the trapezium envelope. We now want to find out which of the methods is more effective by measuring how long it takes to generate N covariates with both methods, for varying N.

```
n_array = c(100, 1000, 10000, 100000, 1000000, 10000000)
rep_array = seq(1, 5)  # to average
ptm_uniform <- vector()
ptm_trapezium <- vector()

for (n in n_array){
  t1 = proc.time()[[3]]
  for (r in rep_array){
    x_rejection_uniform <- rejection_uniform(n, c(M1, acceptance_prob1))
  }
  ptm_uniform = c(ptm_uniform, (proc.time()[[3]] - t1)/length(rep_array))  # Averaging
  t2 = proc.time()[[3]]
  for (r in rep_array){
    x_rejection_trapezium <- rejection_trapezium(n, c(M2, acceptance_prob2, cg))
  }
  ptm_trapezium = c(ptm_trapezium, (proc.time()[[3]] - t2)/length(rep_array))
}

df_timings1 = tibble(t_unfm=ptm_uniform, t_trzd=ptm_trapezium)
ptimings1 <- ggplot(df_timings1)
ptimings1 + labs(y="Seconds", title="Computational time taken for Uniform vs Trapezium Envelopes") +
```

```
  geom_line(aes(n_array, t_unfm, colour="Uniform"))+
  geom_line(aes(n_array, t_trzd, colour="Trapezium"))
```



From the graph, we can see that the Trapezium envelope does outperform the Uniform method slighlty. The reason why the two methods are similar is due to the complexity-acceptance probability trade-off. For some functions, making your envelope more complex may greatly improve the acceptance probability, but it may not actually increase the number of samples per second generated due to increased computational complexity.

Question 2:

Firstly, we can see that the histograms vs the true pdfs in the Question 1 show a high correspondence between the two, supporting that the samples are generated from the pdf $f_X(x)$.

For this question, we verify the scheme using the following diagnostic plots and tests:

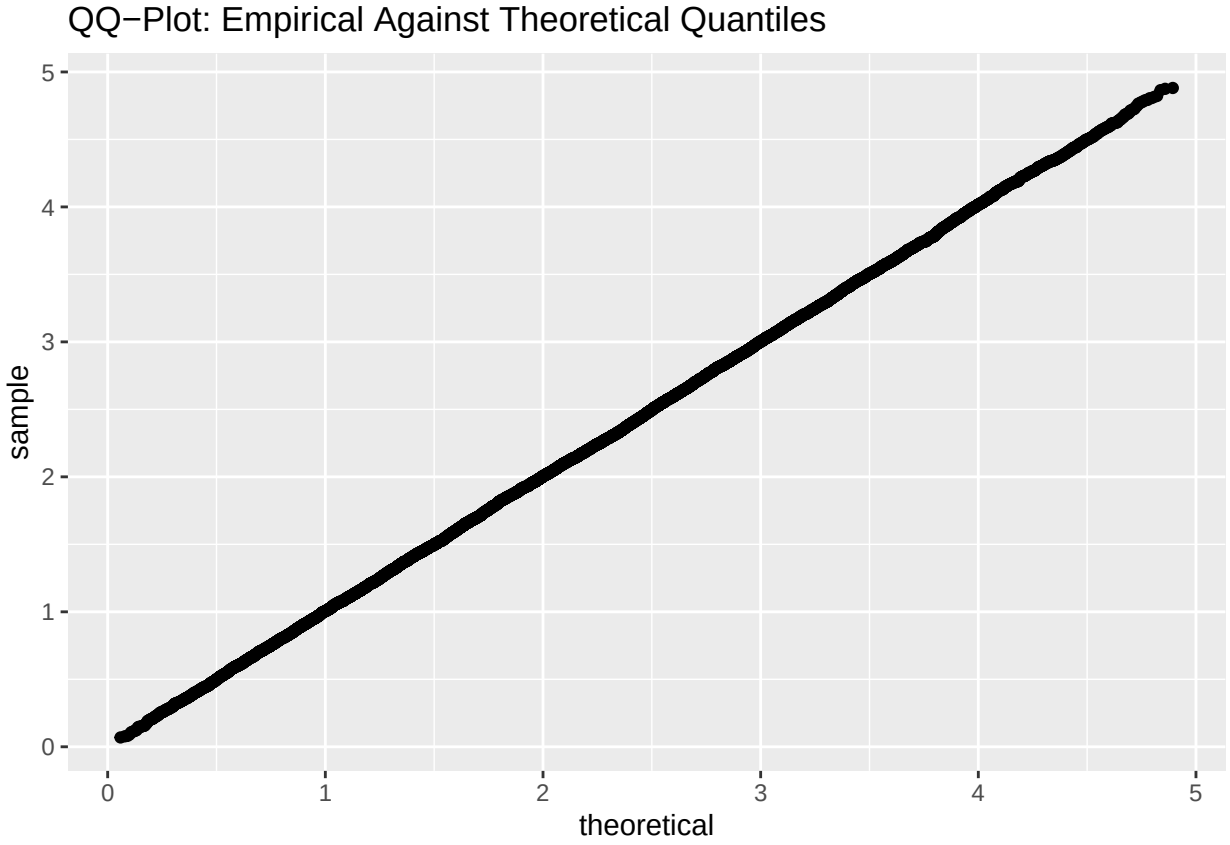
1. QQ Plot
2. Autocovariance Sequence of the Generated Data
3. Lag Plots of $F_X(x)$
4. Kolmogorov-Smirvov Test

QQ PLOT

The QQ Plot is a scatterplot of theoretical against empirical quantiles. It can give a good visual guide of the fit of the distributions. The qqplot is especially sensitive at the tails of the distribution.

```
# Normalised PDF
f <- function(x) cf*fstar(x)
f_cdf_num_single <- function(x) {integrate(f, 0, x)[[1]]}
f_cdf_q <- function(p, q) {f_cdf_num_single(p) - q}
```

```r
quantile_calc <- function(q) {bisect(f_cdf_q, 0, 5, q=q)$root}
qhn <- function(p) {unlist(lapply(p, quantile_calc))}

# Plotting the QQ Plot
n=10000
x_vals = seq(0, 5, l=n)

x_rejection_trapezium <- rejection_trapezium(n, c(M2, acceptance_prob2, cg))

df_qqplot = tibble(x=x_vals, xrej=x_rejection_trapezium$x)
ggplot(df_qqplot, aes(sample=xrej))+
  labs(title="QQ-Plot: Empirical Against Theoretical Quantiles")+
    stat_qq(distribution=qhn)+
    stat_qq_line(distribution=qhn)
```



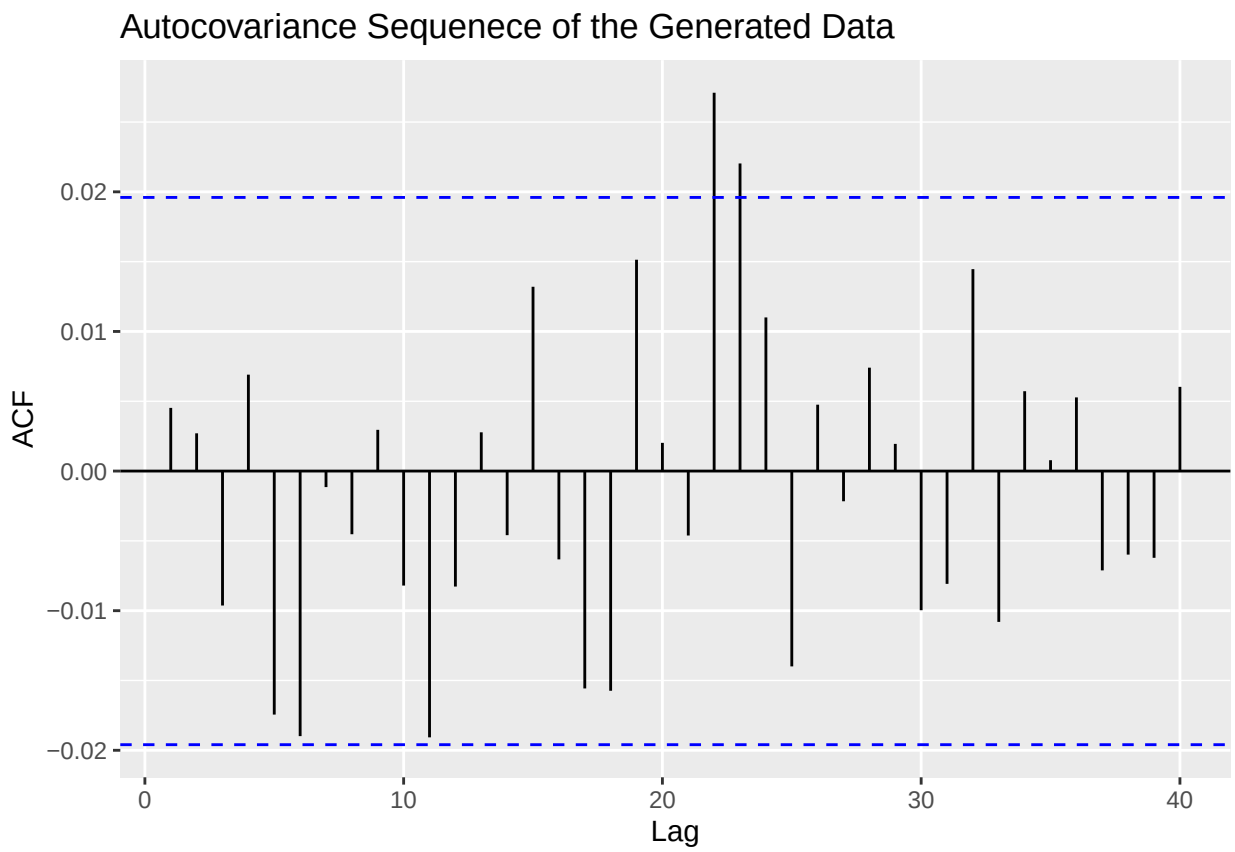
The qqplot shows that the data closely follows the theoretical distribution. This is strong evidence that the rejection sampling technique does indeed sample from $f(x)$.

Autocovariance Sequence

The autocovariance sequence treats our data as a time series and computes the autocovariance sequence for the dataset. For random variates, the sequence should have no obvious pattern and the values should be relatively small.

```r
n = 5000
rej_result <- rejection_trapezium(n, c(M2, acceptance_prob2, cg))
x_rhn <- data.frame(x=x_rejection_trapezium$x)
```

```r
ggAcf(x_rhn)+
  labs(title="Autocovariance Sequenece of the Generated Data")
```

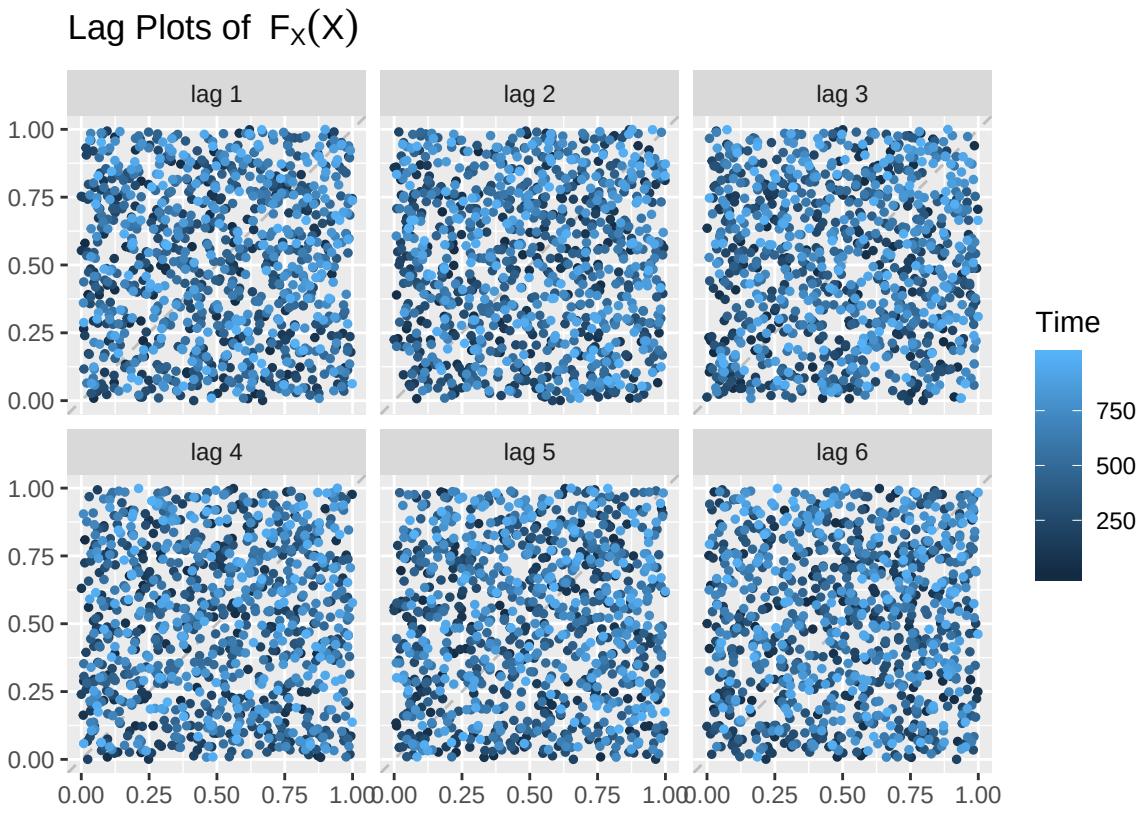


The autocovariance sequence plot shows no significant correlations. This demonstrates that the data generated are distributed independently with pdf $f_X(x)$

LAG PLOT

The lag plots test enables us to check if we have generated truly pseudo-random data. The lag plot of random data should not resemble any patterns, and should be spread throughout the entire plot evenly. If there is a trend in the graphs, this is evidence to believe that there is some signal in the data originating from some unnacounted source.

```r
f_cdf_num <- function(xvec){
  r <- c()
  for (x in xvec){
    r = c(r, integrate(f, 0, x)[[1]])
  }
  return(r)
}

x_rejection_trapezium <- rejection_trapezium(1000, c(M2, acceptance_prob2, cg))
lags_nums = f_cdf_num(x_rejection_trapezium$x)
gglagplot(lags_nums, lags=6, do.lines=FALSE)+
  labs(title=expression("Lag Plots of " ~ F[X](X)))
```



The lag plot shows now no significant patters at any lag, appearing like a random scater, supporting our hypothesis that the data are generated independently from $f_X(\cdot)$.

Statistical Tests

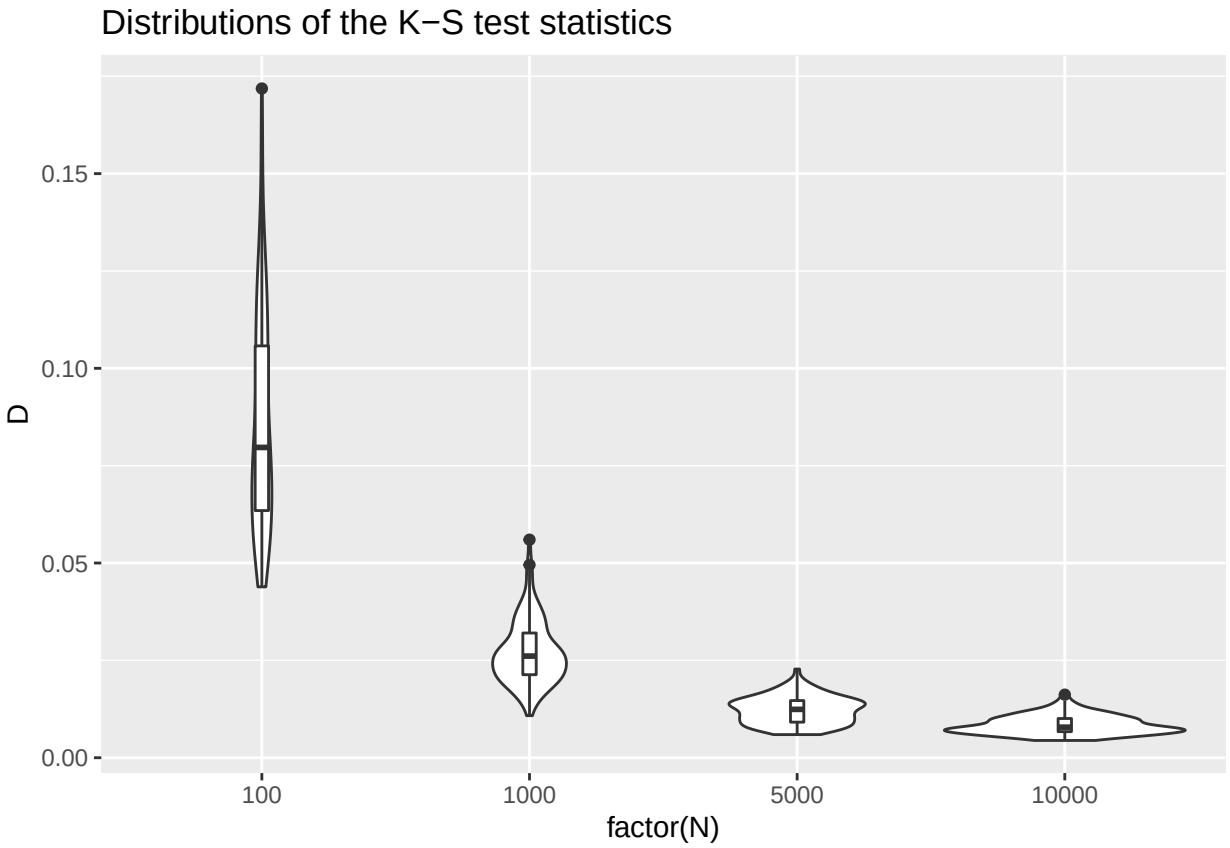
Kolmogorov-Smirvov Test (Note, this section has drawn heavy inspiration from the Formative Coursework Solutions)

The KS Test provides a nonparametric test of goodness of fit. In particular, it measures the error between the empirical CDF and the theoretical CDF

```
# Empirical CDF is the rejection sampling
# Theoretical CDF can be taken as being numerical integration of f

n_vals = c(100, 1000, 5000, 10000)
nn_vals = length(n_vals)
m = 100

ks.testrhn <- function(x){
  kstestx = ks.test(x, f_cdf_num) #, exact=NULL)  # Accounting for ties present as cdf may not be conti
  return(c(kstestx$p.value, kstestx$statistic))
}

ks.results = data.frame()

for (n in 1:nn_vals){
  n1 = n_vals[n]
```

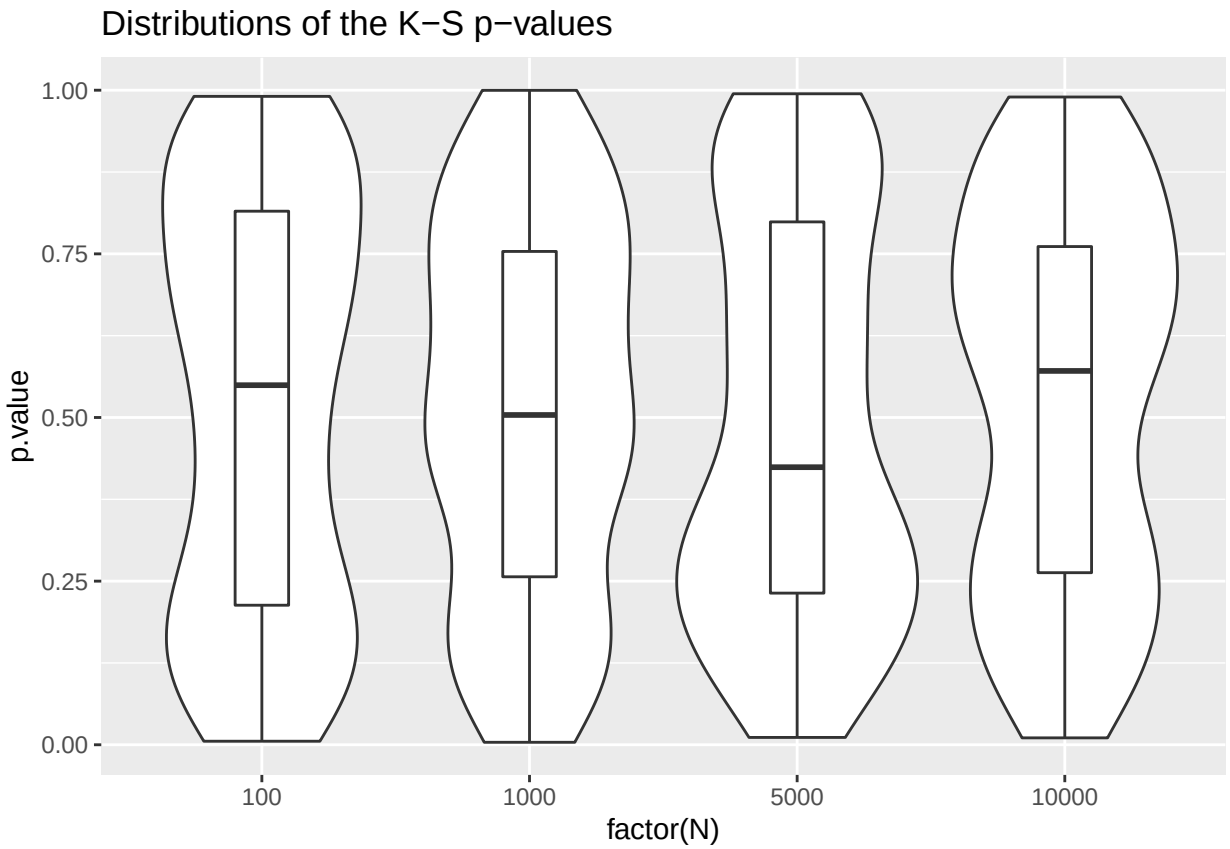
```
  x <- matrix(0, nrow=n1, ncol=m)
  # We do one call to the theoretical cdf, then split into
  # matrix in order to use the apply function
  x1 = rejection_trapezium(n1*m, c(M2, acceptance_prob2, cg))$x
  for(i in 1:m) x[,i] <- x1[((i-1)*n1+1):(i*n1)]
  ks.testx = apply(x, 2, ks.testrhn)
  ks.results = rbind(ks.results, data.frame(p.value=ks.testx[1,], D = ks.testx[2,], N=rep(n1, m)))
}
```

```
ggplot(ks.results, aes(factor(N), D))+
  geom_violin()+
  geom_boxplot(width=0.05)+
  labs(title="Distributions of the K-S test statistics")
```



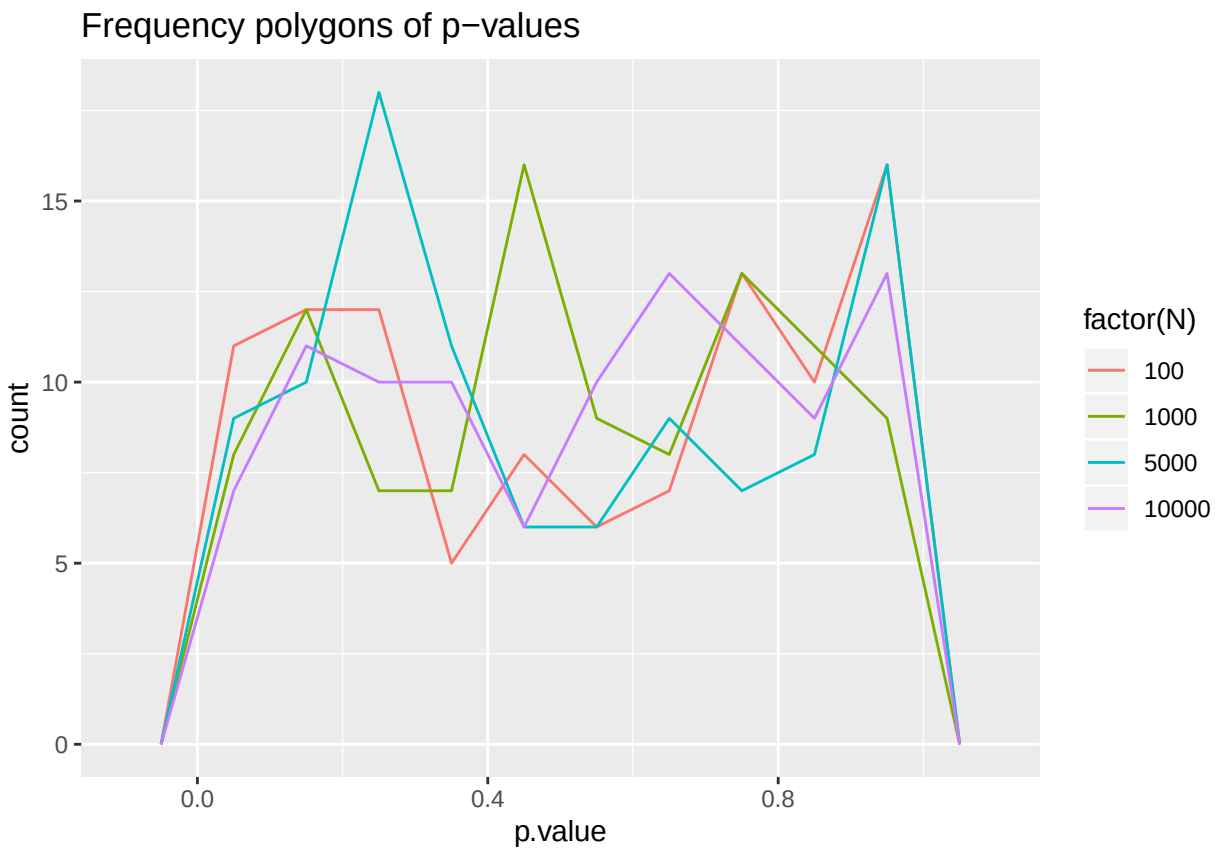
The plot above shows how the distribution of the K-S test statistics become lower as N increases, supporting the null hypothesis.

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



The p-values are roughly uniform, showing no evidence to reject the null hypothesis.

```
ggplot(ks.results, aes(p.value,colour=factor(N)))+
  #geom_histogram(breaks=seq(0,1,by=0.05))+
  geom_freqpoly(breaks=seq(0,1,0.1))+
  labs(title="Frequency polygons of p-values")
```




```
library(knitr)
ks.table <- ks.results %>% group_by(N) %>% summarise("Mean p-value" = round(mean(p.value),digits=3), "S
kable(ks.table)
```

N	Mean p-value	Std Dev (p)	Mean D	Std Dev (D)
100	0.517	0.319	0.086	0.028
1000	0.516	0.286	0.027	0.008
5000	0.496	0.306	0.012	0.004
10000	0.529	0.290	0.008	0.002

From the sample distributions of the p-values, and the summary table, the tests provide no evidence to reject the null hypothesis that the data is composed of random variates from the specified theoretical distribution.

We can see from the distributions of the K-S statistics that the value of D decreases as N (the sample size) increases. The distribution of the p-values is very roughly uniform. All of this supports the decision to not reject the null hypothesis that the data is i.i.d. from the specified theoretical distribution.

Question 3:

Suppose we want to evaluate the value of an integral

$$\theta = \int h(x)dx$$

but we are not able to do so analytically. We can use Monte Carlo (MC) techniques to attempt to evaluate this integral.

The MC techniques we will implement in this questions are:

1. Crude MC
2. Hit-or-Miss MC
3. Variance Reduction Techniques MC

After these implementations, we will do some comparisons of these methods by looking at the mean and variance of the methods.

Crude MC

We can write the integral as

$$\theta = \int \phi(x) f(x) dx$$

where f is a valid pdf and $\phi = \frac{h}{f}$

But we can recognise that the RHS of the integral is an expectation i.e. $\theta = E[\phi(X)]$, and we can use the usual estimator for the expectation, i.e. the sample mean.

The estimator to use is

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

This estimator is unbiased and has variance $var(\hat{\theta}) = k/n$, and we can see that the variance of the estimator drops as the inverse of the sample size.

Implementation below using an Uniform $U(0, 5)$ pdf as our f:

```
MC_crude <- function(N){
  X = runif(N, min=0, max=5)
  Y = fstar(X)
  Ihat =5*sum(Y)/N  # (b-a)*sum(fstar(X))/N
}
```

We now collect some empirical data of this Crude MC:

```
maxfstar = optimize(fstar, lower=0, upper=5, maximum=TRUE)$objective

reps = 1000
N_array = c(100, 1000, 10000, 100000)

MC_crude_mat <- matrix(0, nrow=reps, ncol=length(N_array))

for (i in seq(1, length(N_array))){
  for (j in seq(1, reps)){
    out = MC_crude(N_array[i])
    MC_crude_mat[j, i] = out
  }
}
```

Hit-or-Miss MC

The Hit-or-Miss MC is the most simple and intuitive way of estimating the integral. It turns out that it is also the worst method as it has the highest variance, even though it is unbiased. The intuition is simple, we bound our function by a rectangle, and we sample uniformly within that rectangle. We then count the number

of samples that fall inside the area of the function, divide by the total number of samples and multiply by the area of the rectangle. Mathematically, this is the estimator

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

Implementation below:

```
MC_hit_miss <- function(N, maxf){
  X = runif(N, min=0, max=5)  # Sample x-points
  Y = runif(N, min=0, max=maxf)  # Sample y-points
  cond = Y <= fstar(X)
  number_correct = sum(cond)
  Ihat = maxf*5*number_correct / N  # Using boundaries a=0, b=5, c=max(f), area is (b-a)*c
}
```

```
maxfstar = optimize(fstar, lower=0, upper=5, maximum=TRUE)$objective  # Computing maximum value of fsta
reps = 1000
N_array = c(100, 1000, 10000, 100000)
MC_hit_miss_mat <- matrix(0, nrow=reps, ncol=length(N_array))
for (i in seq(1, length(N_array))){
  for (j in seq(1, reps)){
    out = MC_hit_miss(N_array[i], maxfstar)
    MC_hit_miss_mat[j, i] = out
  }
}
```

Variance Reduction Techniques MC

In this section we will explore the reduction of variance of the Crude MC by using Antithetic Variables. The method relies on taking 'opposite/antithetic' variates, exploiting the fact that the error in the simulated signal may be correlated. We can account for this by taking the 'antithetic' path.

If $\hat{\theta_1}, \hat{\theta_2}$ are unbiased estimators for θ, then

$$E\left[\frac{1}{2}(\hat{\theta_1} + \hat{\theta_2})\right] = \theta$$

which means that the sum of the antithetic variables is still unbiased.

The variance depends on the correlation between the two estimators. If this correlation is negative, then the variance of the sum of the estimators is smaller than the Crude MC variance.

```
MC_crude_antithetic <- function(N){
  X = runif(N, min=0, max=5)
  Ihat =5/N*(sum(fstar(X) + fstar(5-X))/2)
}
```

```
maxfstar = optimize(fstar, lower=0, upper=5, maximum=TRUE)$objective

reps = 1000
N_array = c(100, 1000, 10000, 100000)

MC_crude_antithetic_mat <- matrix(0, nrow=reps, ncol=length(N_array))

for (i in seq(1, length(N_array))){
  for (j in seq(1, reps)){
```

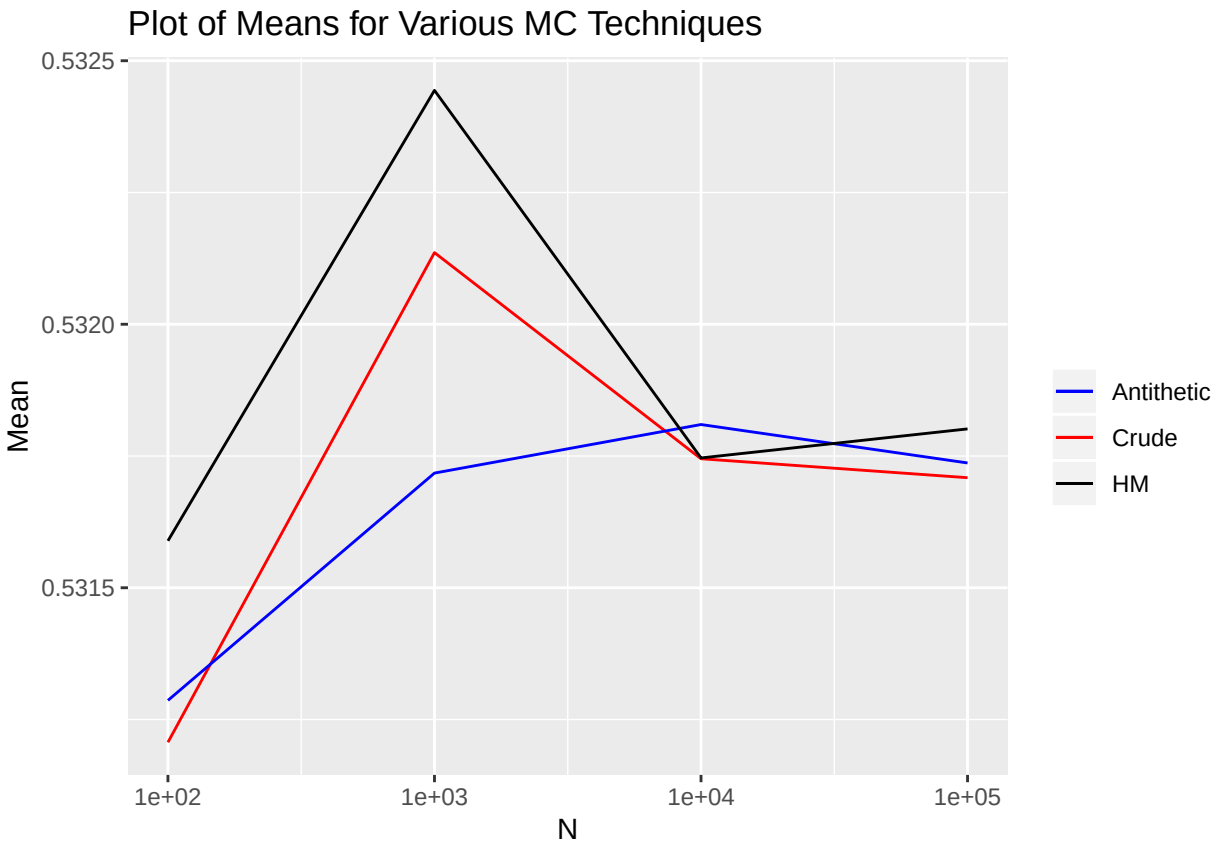
```
    out = MC_crude_antithetic(N_array[i])
    MC_crude_antithetic_mat[j, i] = out
  }
}
```

Comparison of MC Techniques

We now compute the mean and variance of the three MC techniques over 1000 samples for varying N.

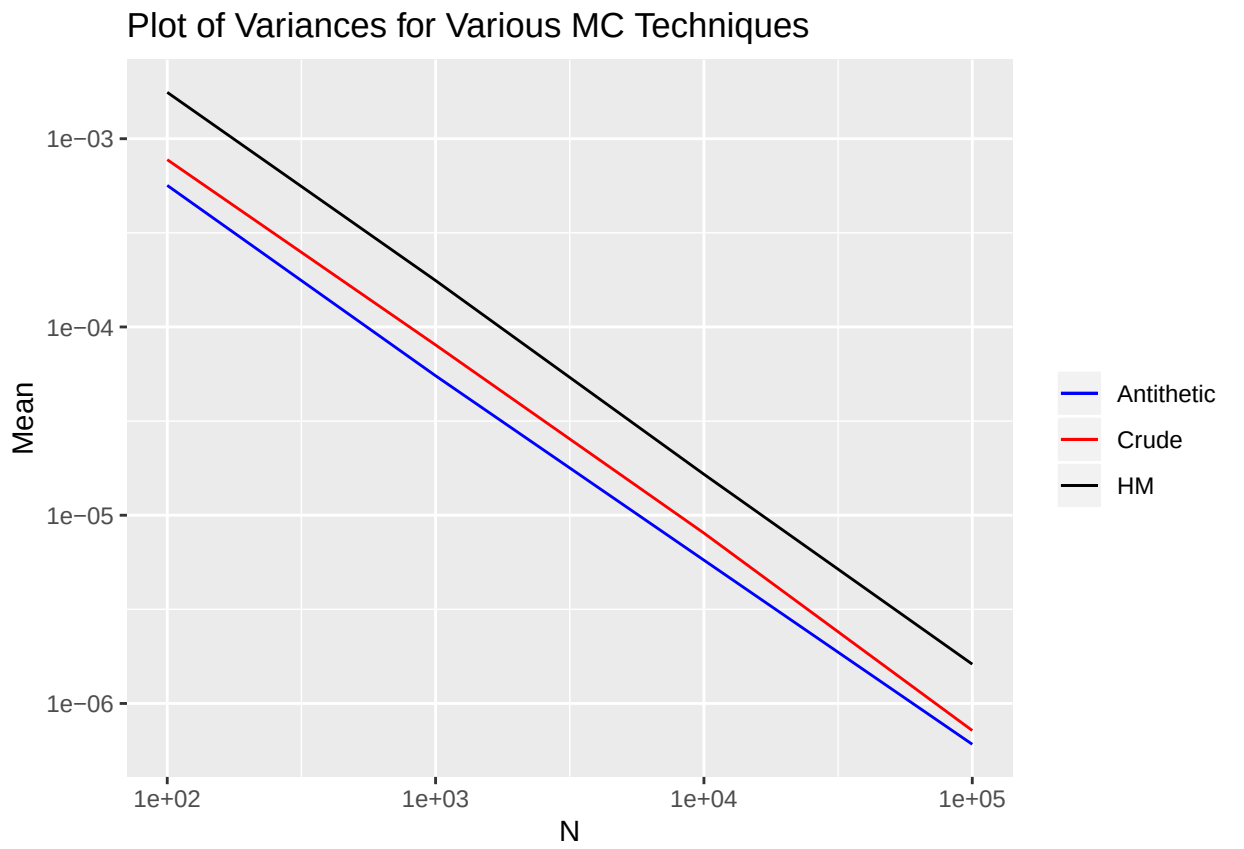
```
mc_crude_means = colMeans(MC_crude_mat)
mc_crude_antithetic_means = colMeans(MC_crude_antithetic_mat)
mc_hit_miss_means = colMeans(MC_hit_miss_mat)

mc_crude_vars = colVars(MC_crude_mat)
mc_crude_antithetic_vars = colVars(MC_crude_antithetic_mat)
mc_hit_miss_vars = colVars(MC_hit_miss_mat)

df_MC_means = tibble(N=N_array, Crude=mc_crude_means, Antithetic=mc_crude_antithetic_means, HM=mc_hit_mi
df_MC_vars = tibble(N=N_array, Crude=mc_crude_vars, Antithetic=mc_crude_antithetic_vars, HM=mc_hit_miss
```

```
# Plot of Means for Techniques
pmeans <- ggplot(df_MC_means)
pmeans + geom_line(aes(N, Crude, colour="Crude")) +
     geom_line(aes(N, Antithetic, colour="Antithetic")) +
     geom_line(aes(N, HM, colour="HM")) +
     labs(y="Mean", title="Plot of Means for Various MC Techniques") +
     scale_colour_manual("", values=c("Crude"="red", "Antithetic"="blue", "HM"="black"))+
     scale_x_log10()
```



This plot shows that for larger N, the three methods agree relatively closely with the 'true' numerically computed value for the integal of 0.531736... This is encouraging, validating our theory that the estimators are unbiased.

```
# Log-Log Plot of Variances for Techniques
pvars <- ggplot(df_MC_vars)
pvars + geom_line(aes(N, Crude, colour="Crude")) +
     geom_line(aes(N, Antithetic, colour="Antithetic")) +
     geom_line(aes(N, HM, colour="HM")) +
     labs(y="Mean", title="Plot of Variances for Various MC Techniques") +
     scale_colour_manual("", values=c("Crude"="red", "Antithetic"="blue", "HM"="black"))+
     scale_x_log10()+
     scale_y_log10()
```



The log-log plot of variances for the three MC techniques clearly shows how the antithetic MC has the lowest variance, hit-and-miss has the highest and Crude is in the middle. Furthermore, we can see that the slope of this line on the log-log plot is roughly -0.5, which agrees with the theory in that the error converges as the square root.

To obtain the Monte Carlo estimate of the normalising constant, all that we need to do is invert the estimate of the integral for large enough N.

```
integral_estimate = MC_crude_antithetic(100000)
cf_estimate = integral_estimate^-1
sprintf("The Monte Carlo Estimate for the Normalising Constant is %.2f", cf_estimate)
```

```
## [1] "The Monte Carlo Estimate for the Normalising Constant is 1.88"
```

THE END