Stochastic Simulations

March 18, 2012

Luke Kanczes

Abstract

The following report begins by generating pseudo-random numbers which we are then able to convert into random variables from a specific distribution. We also explore the technique of Monte Carlo integration, exploring different methods with the objective of variance reduction. A smaller variance allows us to produce an answer to the same degree of accuracy from a smaller simulation. We then explore Monte Carlo Markov Chain techniques, in particular the Metropolis Hastings sampler which has the desired distribution in its equilibrium distribution.

1 Simulating from a specified density

We are asked to consider the following density

$$f(x) \propto \begin{cases} \frac{1}{x(1-x)} \exp\left[-\frac{1}{2}\left(3+\ln\left(\frac{x}{1-x}\right)\right)^2\right] & 0<x<1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and devise and implement a simple, exact and efficient algorithm for simulating from $f(\cdot)$.

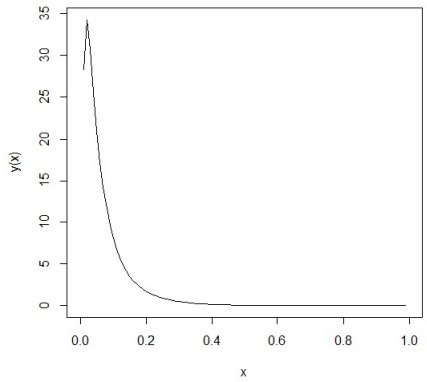

Figure 1: Plot of $f(x)$ up to proportionality as denoted in (1) [Appendix 5.1]

We first note that it is only possible to find the inverse of this function numerically, and therefore we are not able to implement the *Inversion* algorithm. We potentially could implement either the *Rejection* algorithm or the *Ratio of Uniforms* Algorithm to simulate from $f(\cdot)$.

1.1 Ratio of Uniforms

If we consider some function h such that $h(\cdot) \geq 0$ and $\int h < \infty$, we then consider the following region defined in (U,V) space:

$$C_h = \left\{(u,v)\,|0 \leq u \leq \sqrt{h\left(\frac{v}{u}\right)}\right\}$$

If C_h has finite area, and if (U,V) are uniform on C_h, then $X = \frac{V}{U}$ has probability density function $\frac{h}{\int h}$. In other words $f(x) = \frac{h}{\int h}$. If we consider $f(x)$ as defined in (1) we might now wish to consider:

$$K\int_0^1 \frac{1}{x(1-x)} \exp\left[-\frac{1}{2}\left(3+\ln\left(\frac{x}{1-x}\right)\right)^2\right] dx \tag{2}$$

Where K is our constant of proportionality. We can estimate the value of this integral numerically in R using *Monte Carlo Simulations* as in Section 2. As a guide we may use the built-in *integrate* function in R and find its value is $2.506628K$ with absolute error $< 0.00019K$ [Appendix 5.2].

Since we know $\int f(x)dx \approx 2.506628K$ we might want to define K such that

$$K = \left(\int_0^1 \frac{1}{x(1-x)} \exp\left[-\frac{1}{2}\left(3+\ln\left(\frac{x}{1-x}\right)\right)^2\right] dx\right)^{-1} \approx (2.506628)^{-1} \tag{3}$$

and therefore $\int f(x)dx = 1$.

We therefore define our function $h(x)$ as

$$h(x) = \begin{cases} \frac{1}{x(1-x)} \exp\left[-\frac{1}{2}\left(3+\ln\left(\frac{x}{1-x}\right)\right)^2\right] & 0<x<1 \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

And define $f(x)$ as

$$f(x) = \begin{cases} \frac{K}{x(1-x)} \exp\left[-\frac{1}{2}\left(3+\ln\left(\frac{x}{1-x}\right)\right)^2\right] & 0<x<1 \\ 0 & \textit{otherwise} \end{cases} \tag{5}$$

Where K is defined as in (3).

1.1.1 Algorithm

The Ratio of Uniforms algorithm proceeds as follows:

1. Find the bounding rectangle for C_h.
2. Generate $(U_1, U_2) \sim U(0,1)$.
3. Set $U = aU_1$, $V = b + (c-b)U_2$.
4. If $U \leq \sqrt{h\left(\frac{V}{U}\right)}$, set $X = \frac{V}{U}$, otherwise go to step 1.

1.1.2 Find the bounding rectangle

We are required to find a bounding rectangle for C_h. Such a rectangle will always exist provided $h(x)$ and $x^2h(x)$ are bounded in the domain of x. We consider C_h to be bounded by a, b and c, where we define:

$$\begin{aligned} a &= \sup_{x\geq 0} \sqrt{h(x)} \\ b &= \inf_{x\leq 0} x\sqrt{h(x)} \\ c &= \sup_{x\geq 0} x\sqrt{h(x)} \end{aligned}$$

For a we are required to find the supremum of $\sqrt{h(x)}$ for $x \geq 0$. Numerically we find that this is equal to 5.861968 at $x = 0.01864552$. Clearly b takes its infimum at 0 for any value of x such that $x \leq 0$. For c the supremum of $x\sqrt{h(x)}$ for $x \geq 0$ is found numerically to be 0.2865048 when $x = 0.1192146$. [Appendix 5.3]

We therefore have $a = 5.861968$, $b = 0$ and $c = 0.2865048$.

1.1.3 Implement the Algorithm

We can now implement the Ratio of Uniforms algorithm to simulate from $f(\cdot)$. [Appendix 5.4]

The theoretical acceptance value is given as

$$\begin{aligned} \text{Probability of accepting an X} &= \frac{\text{Area}(C_h)}{\text{Area of bounding rectangle}} \\ &= \frac{\frac{1}{2}\int h(x)dx}{a(c-b)} \\ &\approx \frac{\frac{1}{2}\cdot 2.506628}{5.861968 \cdot 0.2865048} \\ &\approx 0.74625 \end{aligned}$$

On implementing the algorithm with $n = 10,000$ we obtain an actual acceptance probability of 0.7760547. Implementing the algorithm again a second time, yields an acceptance probability of 0.7713603, the discrepancy between values due to the nature of a stochastic process. Given the acceptance probability is fairly large we could consider this a relatively efficient algorithm.

1.2 Rejection Algorithm

Here we instead consider some probability density function g such that there exists $\exists M > 0$ with $\frac{f}{g} < M < \infty$.

1.2.1 Algorithm

The Rejection algorithm proceeds as follows:

1. Generate $Y = y \sim g(\cdot)$.
2. Generate $U = u \sim U(0,1)$.
3. If $u \leq \frac{f(y)}{Mg(y)}$ set $X = y$.
4. Otherwise go to step 1.

1.2.2 Probability of acceptance

Here we note first that $M = \sup_x \frac{f(x)}{g(x)}$. The theoretical acceptance probability is given as

$$\begin{aligned} \text{Probability of accepting an X} &= \int_{-\infty}^{\infty} h(y)g(y)dy \\ &= \int_{-\infty}^{\infty} \frac{f(y)}{Mg(y)}g(y)dy \\ &= \frac{1}{M}\int_{-\infty}^{\infty} f(y)dy \\ &= \frac{1}{M} \end{aligned}$$

We therefore wish to find a function $g(x)$ such that M is as close to 1 as possible.

1.2.3 Implementing the algorithm

We shall implement our Rejection algorithm for two different choices of $g(x)$. [Appendix 5.6 & Appendix 5.8]

Uniform Distribution

If we consider the simple case of $g(x)$ equivalent to the uniform distribution we are able to generate from $f(x)$. However, we should note that since $M = 34.36266$ [Appendix 5.5] our acceptance probability is only $\frac{1}{34.36266} = 0.02910136$ and thus the algorithm is particularly inefficient.

Beta distribution

If we instead consider the case when where $g(x)$ takes the form of the Beta$(1.5, 4)$ distribution we are again able to generate from $f(x)$. Here $M = 29.99015$ [Appendix 5.7] and therefore our acceptance probability is $\frac{1}{29.99015} = 0.03334428$ and thus the algorithm is still more inefficient than that obtained using the Ratio of Uniforms, however more efficient than when using $g(x)$ uniform.

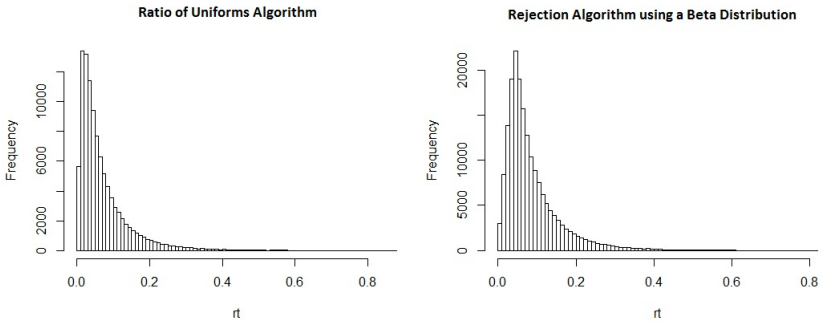

Figure 2: Data simulated from $f(\cdot)$ for $n = 100,000$ using the Ratio of Uniforms algorithm and the Rejection algorithm using a Beta distribution

1.3 Pretesting Squeezing

Both the rejection and the ratio of uniforms methods use membership tests, which are computationally expensive to evaluate. If we consider the Rejection algorithm we may be able to find some functions W_L and W_U which are less computationally expensive to evaluate such that $W_L < \frac{f}{g} < W_U$, $\forall x$. We can do similar with the Ratio of Uniforms algorithm, however in both cases finding such functions is difficult due to the unusual form for the density function.

2 Monte Carlo Integration

We are required to estimate the value of an integral

$$\int_0^1 \frac{1}{x(1-x)} \exp\left[-\frac{1}{2}\left(3 + \ln\left(\frac{x}{1-x}\right)\right)^2\right] dx \tag{6}$$

In Section 1.1 we noted that when using the *integrate* function in R that this value is approximately 2.506628. We shall use several different methods and compare the solutions.

For ϕ a bounded function on (a, b), with $0 \leq \phi \leq c$ we define $\theta = \int_a^b \phi(x)dx = \int_a^b \phi_1(x)f(x)dx$, with $f(x) = \frac{1}{b-a}$, $x\epsilon(a, b)$.

2.1 Direct $\phi - f$ method

Here we estimate θ as

$$\begin{aligned} \hat{\theta} &= \tfrac{1}{n}\textstyle\sum_{i=1}^n \phi_1(X_i) \qquad\qquad X_i \sim U(a,b) \\ &= (b-a)\tfrac{1}{n}\textstyle\sum_{i=1}^n \phi(X_i) \end{aligned}$$

Using θ as defined in (9), for $X_i \sim U(0, 1)$, we estimate $\hat{\theta}$ as

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^n \frac{1}{X_i(1-X_i)} \exp\left[-\frac{1}{2}\left(3 + \ln\left(\frac{X_i}{1-X_i}\right)\right)^2\right] \tag{7}$$

We note $\mathrm{E}\left(\hat{\theta}\right) = \theta$ and that the variance of this value is

$$\begin{aligned}\mathrm{var}\left(\hat{\theta}\right) &= (b-a)^2 \tfrac{1}{n^2} n \mathrm{var}\left(\phi\left(X\right)\right) \\ &= \tfrac{1}{n}\left(\mathrm{E}_f\left(\phi^2(X)\right) - \mathrm{E}_f^2\left(\phi(X)\right)\right) \\ &= \tfrac{1}{n}\left(\int_0^1 \phi^2(x)dx - \theta^2\right)\end{aligned}$$

Using $n = 100,000$ we estimate the value of θ as 2.505269. Running the algorithm again we estimate its value as 2.493166. [Appendix 5.9] The discrepancy in our estimate is to be expected due to the nature of a stochastic process.

2.1.1 Antithetic version

If we suppose that $\hat{\theta}_1$ and $\hat{\theta}_2$ are both unbiased estimators of θ with variances $\mathrm{var}\left(\hat{\theta}_1\right)$ and $\mathrm{var}\left(\hat{\theta}_2\right)$ then $E\left(\frac{1}{2}\left(\hat{\theta}_1 + \hat{\theta}_2\right)\right) = \theta$ and

$$\mathrm{var}\left(\frac{1}{2}\left(\hat{\theta}_1 + \hat{\theta}_2\right)\right) = \frac{1}{4}\mathrm{var}\left(\hat{\theta}_1\right) + \frac{1}{4}\mathrm{var}\left(\hat{\theta}_2\right) + \frac{1}{2}\mathrm{cov}\left(\hat{\theta}_1, \hat{\theta}_2\right)$$

If we suppose $\mathrm{var}\left(\hat{\theta}_1\right) = \mathrm{var}\left(\hat{\theta}_2\right)$ then

$$\begin{aligned}\mathrm{var}\left(\tfrac{1}{2}\left(\hat{\theta}_1 + \hat{\theta}_2\right)\right) &= \tfrac{1}{2}\mathrm{var}\left(\hat{\theta}_1\right) + \tfrac{1}{2}\mathrm{cov}\left(\hat{\theta}_1, \hat{\theta}_2\right) \\ &= \tfrac{1}{2}\mathrm{var}\left(\hat{\theta}_1\right)\left[1 + \tfrac{\mathrm{cov}\left(\hat{\theta}_1, \hat{\theta}_2\right)}{\sqrt{\mathrm{var}\left(\hat{\theta}_1\right)\mathrm{var}\left(\hat{\theta}_2\right)}}\right] \\ &= \tfrac{1}{2}\mathrm{var}\left(\hat{\theta}_1\right)\left[1 + \mathrm{corr}\left(\hat{\theta}_1, \hat{\theta}_2\right)\right]\end{aligned}$$

If $\mathrm{corr}\left(\hat{\theta}_1, \hat{\theta}_2\right)$ is large and negative, $\mathrm{var}\left(\frac{1}{2}\left(\hat{\theta}_1 + \hat{\theta}_2\right)\right) < \mathrm{var}\left(\hat{\theta}_1\right)$. So for $X_i \sim U(0,1)$, we can estimate θ as

$$\hat{\theta}^* = \frac{1}{2n}\sum_{i=1}^{n}\left(\phi(X_i) + \phi(1 - X_i)\right) \tag{8}$$

And we have $\mathrm{var}\left(\hat{\theta}^*\right) = \frac{1}{2}\mathrm{var}\left(\hat{\theta}_1\right)\left[1 + \mathrm{corr}\left(\hat{\theta}_1, \hat{\theta}_2\right)\right]$. Using $n = 100,000$ we estimate the value of θ as 2.505842. Running the algorithm again we estimate its value as 2.528965. [Appendix 5.10]

2.1.2 Stratified Sampling

Here we estimate θ by breaking the integral up and performing Monte Carlo integration on each separately.

$$\theta = \int_a^b \phi(x)dx = \int_{\alpha_0=a}^{\alpha_1} \phi(x)dx + \int_{\alpha_1}^{\alpha_2} \phi(x)dx + ... + \int_{\alpha_{k-1}}^{\alpha_k=b} \phi(x)dx$$

If we consider our function, we find it achieves its maximum value at 34.36266 when $x = 0.01864517$ and the majority of its density is between 0 and 0.3. Therefore, we might wish to divide our integral using values for α_i as given in Appendix 5.11 to obtain an improvement. Using $n = 100,000$ we estimate the value of θ as 2.506135. Running the algorithm again we estimate its value as 2.505751. However, using so many integrals is fairly inefficient.

2.2 Hit or Miss Algorithm

Here we let $U = u_i \sim U(a,b)$, $V = v_i \sim U(0,c)$, $i = 1, ..., n$ and define

$$\tilde{\theta} = c(b-a)\frac{1}{n}\sum_{i=1}^{n} I(v_i \leq \phi(u_i))$$

With $a = 0, b = 1$ and $c = sup_{x \geq 0}\frac{1}{x(1-x)}\exp\left[-\frac{1}{2}\left(3 + \ln\left(\frac{x}{1-x}\right)\right)^2\right] = 34.36266$ [Appendix 5.12]. We therefore estimate $\tilde{\theta}$ as

$$\tilde{\theta} = \frac{34.36266}{n}\sum_{i=1}^{n} I\left(v_i \leq \frac{1}{u_i(1-u_i)}\exp\left[-\frac{1}{2}\left(3 + \ln\left(\frac{u_i}{1-u_i}\right)\right)^2\right]\right) \tag{9}$$

We note that $I(V \leq \phi(U)) \sim \mathrm{Bernoulli}(P(V \leq \phi(U))$ so $\mathrm{E}\left(\tilde{\theta}\right) = \theta$ and

$$\begin{aligned}\mathrm{var}\left(\tilde{\theta}\right) &= c^2\left(b-a\right)^2 \tfrac{1}{n^2} n \mathrm{var}\left(I(V \leq \phi(U))\right) \\ &= \tfrac{c^2}{n}\left(\left(P(V \leq \phi(U))\right)\left(1 - P(V \leq \phi(U))\right)\right) \\ &= \tfrac{c^2}{n}\left(\tfrac{\theta}{c}\left(1 - \tfrac{\theta}{c}\right)\right) \\ &= \tfrac{\theta}{n}\left(c - \theta\right)\end{aligned}$$

Using $n = 100,000$ we estimate the value of θ as 2.545929. Running the algorithm again we estimate its value as 2.525999. [Appendix 5.13 & 5.14]

2.3 Importance Sampling

We instead could consider $\theta = \int \phi(x)f(x)dx = \int \phi(x)\frac{f(x)}{g(x)}g(x)dx = \int \psi(x)g(x)dx$ where $\psi(x) = \frac{\phi(x)f(x)}{g(x)}$. We can therefore estimate θ as

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

where $X_1, ..., X_n \sim g(\cdot)$ and $\psi(x) = \frac{1}{g(x)} \cdot \frac{1}{x(1-x)} \exp\left[-\frac{1}{2}\left(3 + \ln\left(\frac{x}{1-x}\right)\right)^2\right]$.

We aim to find a function $g(\cdot)$ such that it mimics the shape of $h(x) = \phi(x)f(x)$. If we consider several different cases of the beta distribution we might observe that the Beta(1.5,22) distribution takes values for $x = 0 \to 1$ and has a similar shape to that of our probability density function. We are therefore able to estimate θ. The variance of $\hat{\theta}$ is given as

$$\begin{aligned} \text{var}\left(\hat{\theta}\right) &= \frac{1}{n}\text{var}\left(\psi\left(X\right)\right) \\ &= \frac{1}{n}\text{var}\left(\frac{\phi f(X)}{g(X)}\right) \end{aligned}$$

Therefore, we can minimise our variance by using a function g such that it mimics the shape of $h = \phi f$. Using $n = 100,000$ we estimate the value of θ as 2.50104. Running the algorithm again we estimate its value as 2.510881. [Appendix 5.15]

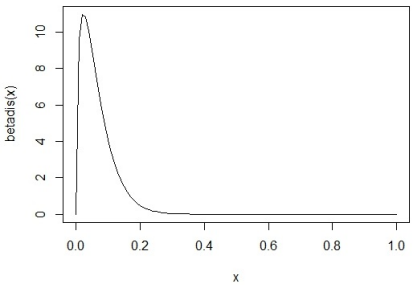

Figure 3: The Beta(1.5,22) distribution [Appendix 5.16]

2.4 Comparison of the different methods

We note that the variance of (9) is always greater than that of (7) and therefore we could argue that we should never use the hit and miss algorithm.

Proof

$$\text{var}\left(\hat{\theta}\right) = \frac{1}{n}\left(\int_0^1 \phi^2(x)dx - \theta^2\right) \leq \frac{1}{n}\left(c\int_0^1 \phi(x)dx - \theta^2\right) = \frac{\theta}{n}\left(c - \theta\right) = \text{var}\left(\tilde{\theta}\right)$$

We are able to improve on this further using the antithetic version, provided corr $\left(\hat{\theta}_1, \hat{\theta}_2\right)$ is large and negative. Furthermore, we could also use stratified sampling to reduce our variance further. Theoretically, we are able to reduce our variance down to zero using importance sampling, provided our function g is such that it perfectly mimics the shape of $h = \phi f$.

3 Metropolis-Hastings Sampler

The Metropolis-Hastings sampler is a type of Markov Chain Monte Carlo method (MCMC), which allows us to simulate a markov chain using an iterative procedure, which has a equilibrium distribution equal to some target density.

3.1 Algorithm

We aim to sample from some density $f(x)$ using a different probability density $q(x, y)$, satisfying $\int_{\mathcal{X}} q(x, y)dy = 1$ for all x. The algorithm proceeds as follows:

1. Start from an arbitrary $X^{(0)}$
2. Given $X^{(n)} = x$, generate a trial value $Y = y$ from the probability density $q(x, y)$.

3. Define $\alpha = \min\left(\frac{f(y)q(y,x)}{f(x)q(x,y)}, 1\right)$. If $\alpha = 1$ then set $X^{(n+1)} = Y$. If $0 < \alpha < 1$ then accept Y with probability α. If Y is accepted then $X^{(n+1)} = Y$; else $X^{(n+1)} = X^{(n)}$.

4. Replace n by $n + 1$ and go to Step 2.

3.2 Implementing the Algorithm

We are required to consider some probability density $q(x, y)$. We shall consider two different choices for $q(x, y)$, adjusting both so as to have an acceptance rate of about 20%. We shall take a starting value of $X^{(0)} = 0.1$ in both cases since this value is close to where the majority of the density is.

3.2.1 Uniform distribution

Here we take our proposed new state y as $y = x + W$ where x is our current state and $W \sim U(-\alpha, \alpha)$. Using $n = 100,000$ we adjust our algorithm and find that for $W \sim U(-0.33, 0.33)$ we estimate our acceptance probability as 0.1943839. [Appendix 5.17]

3.2.2 Normal distribution

Here we again take our proposed new state y as $y = x + W$ where x is our current state and $W \sim N(0, \nu)$. Using $n = 100,000$ we adjust our algorithm and find that for $W \sim N(0, 0.24)$ we estimate our acceptance probability as 0.2017039. [Appendix 5.18]

3.3 Adjusting the starting value

The Metropolis Hastings algorithm requires a large number of steps before it is judged to have converged. Often a number of iterations at the beginning are deleted to allow *burn-in*, a period where it is assumed that the probability density is settling down towards the true $f(x)$. For the purposes of this investigation we shall consider just the case of $q(x, y)$ normal.

We start by analysing the difference between the autocorrelation sequences for the Metropolis Hastings algorithm and the Ratio of Uniforms algorithm. The autocorrelation is defined as the cross-correlation of a signal with itself, which allows us to find patterns which might be hidden within noise. By definition the autocorrelation of a continuous white noise process has a value of 1 at lag 0 and has a value of zero for all other values. If we look at Figure 4 we can see the Ratio of Uniforms algorithm follows this approximate structure. This is unsurprising given that each of the observations are independent. However the Metropolis Hastings algorithm for $X^{(0)} = 0.9$ has a clear structure up until lag 18. We aim to remove this structure so our Metropolis Hastings autocorrelation structure resembles that for the Ratio of Uniforms algorithm. Interestingly the Metropolis Hastings algorithm for $X^{(0)} = 0.4$ has a structure up until lag 15, and has a more linear decline.

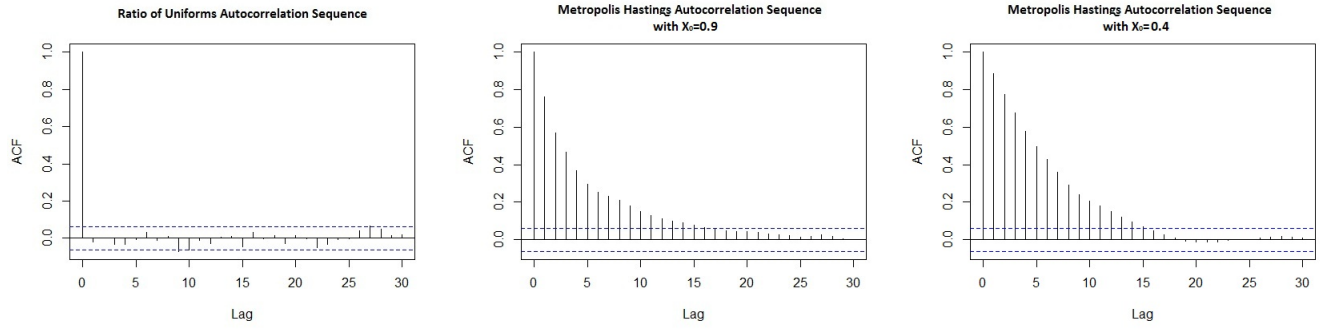

Figure 4: Autocorrelation Sequence for the Ratio of Uniforms algorithm and Metropolis Hastings with $X^{(0)} = 0.9$, $X^{(0)} = 0.4$ and $n = 1000$

We can adjust our Metropolis Hastings algorithm to take every $(\tau+1)^{th}$ point as a random variable from our target density, for example for $\tau = 5$ we'll take the values $6, 11, 16, ...$, we can then compare the autocorrelation function this produces with that obtained using the Ratio of Uniforms algorithm. In Figure 5 we can see that using $\tau = 5$ fails to produce a purely random sequence, however using $\tau = 15$ we are able to produce an autocorrelation sequence similar to that obtained using the Ratio of Uniforms Algorithm. [Appendix 5.20]

We can also consider the path the markov chain takes as we alter the starting value. If we start by investigating $X^{(0)} = 0.9$ we can compare the results with a value such as $X^{(0)} = 0.4$. We might also like to note that if we select a value for $X^{(0)}$ which is outside our range the algorithm produces an error.

In Figure 6 we can see that the markov chain quickly falls from its starting value at $X^{(0)} = 0.9$ to lower values around 0.1. For the subsequent iterations the markov chain appears to be bounded by 0.5. This is in contrast to our results for $X^{(0)} = 0.4$ which remains bounded by 0.6 for the first 5000 iterations. Therefore, we might conclude that $X^{(0)} = 0.4$ reaches the equilibrium distribution faster than for $X^{(0)} = 0.9$, which provides some indication about the

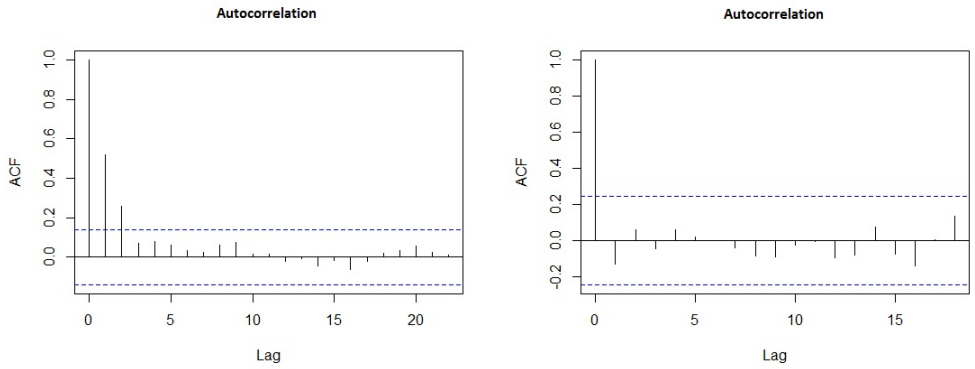

Figure 5: Autocorrelation Sequence for the amended Metropolis Hastings algorithm for $\tau = 5$ and $\tau = 15$ using $X^{(0)} = 0.9$ and $n = 1000$

relative importance of our starting value $X^{(0)}$. This also provides a visualisation of our burn-in time. Consequently, we might wish to discard the first few iterations for the Metropolis Hasting algorithm after which the algorithm produces a purely random Markov chain.

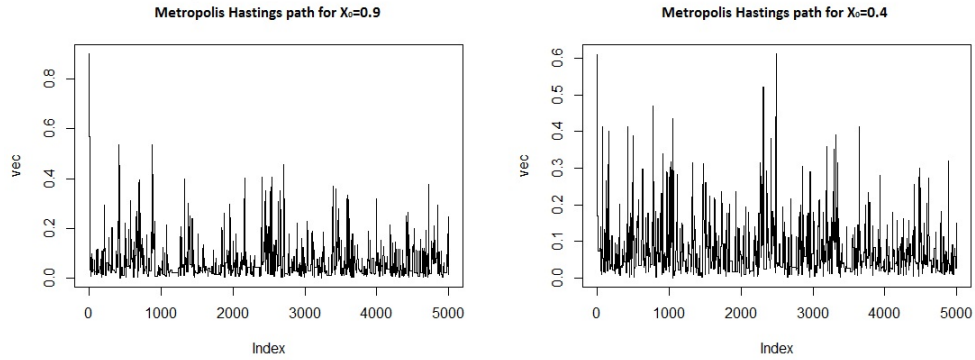

Figure 6: Metropolis Hasting paths for $n = 5000$ and $X^{(0)} = 0.9$ and $X^{(0)} = 0.4$ respectively

We might also like to compare the results from the first 1000 iterations with that obtained by the subsequent 1000 iterations to illustrate the importance of our starting value $X^{(0)}$. In Figure 7 we can see that for $X^{(0)} = 0.9$ the histogram of our random variables takes relatively extreme values, with some as large as 0.9 and the rest heavily concentrated around 0.05. By comparison the subsequent 1000 iterations is more similar to our actual density as plotted in Figure 1. Interestingly for $X^{(0)} = 0.4$, as in Figure 8 we see that the first 1000 and the following 1000 iterations both loosely resemble that of our density. We might consider this somewhat unsurprising given the path of our algorithm appears to settle down quicker for $X^{(0)} = 0.4$ than for $X^{(0)} = 0.9$.

Finally, if we compare the results from the Metropolis Hastings with that obtained when using the Ratio of Uniforms algorithm with $n = 2000$, as in Figure 9, we can see the histogram produced is already beginning to resemble that of our distribution. Therefore, we may take the view that the Metropolis Hastings algorithm has a potential weakness when compared the the Ratio of Uniforms algorithm.

4 Conclusion

In conclusion we have produced several different methods to sample from some specified density. We have explored the technique of Monte Carlo integration, exploring different methods with the objective of variance reduction. We then explored Monte Carlo Markov Chain techniques, in particular the Metropolis Hastings sampler which has the desired distribution in its equilibrium distribution. We have shown that the Metropolis Hastings algorithm requires a burn-in time and by comparing the autocorrelation function with that obtained when using the Ratio of Uniforms algorithm have shown one of its potential weaknesses, as well as demonstrating the importance of which value we choose for $X^{(0)}$.

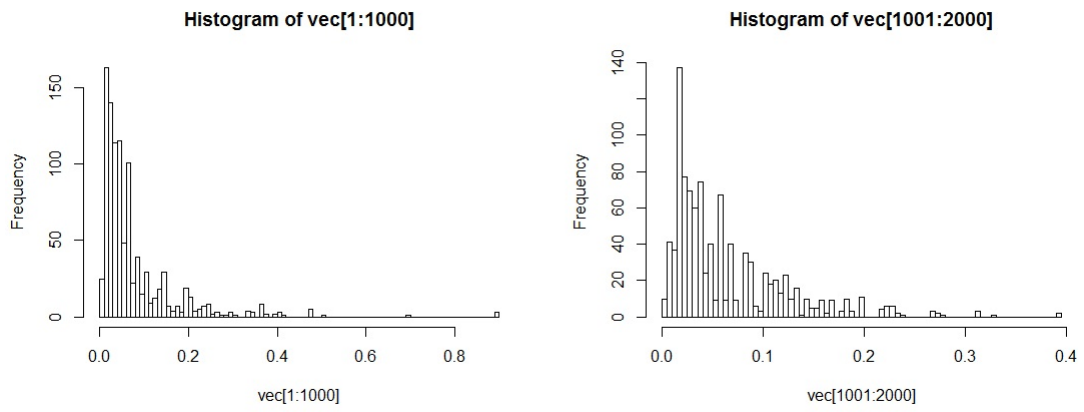

Figure 7: Comparing the first 1000 samples compared to the subsequent 1000 from our density $f(x)$ for $X^{(0)} = 0.9$

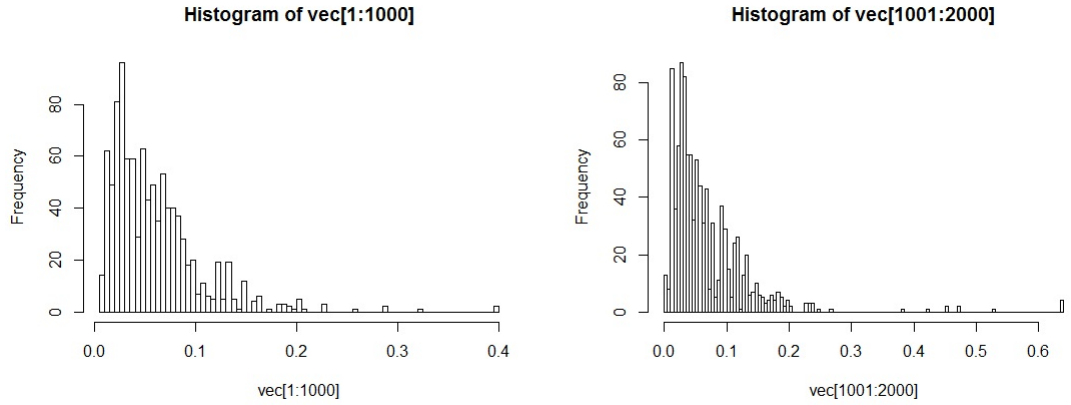

Figure 8: Comparing the first 1000 samples compared to the subsequent 1000 from our density $f(x)$ for $X^{(0)} = 0.4$

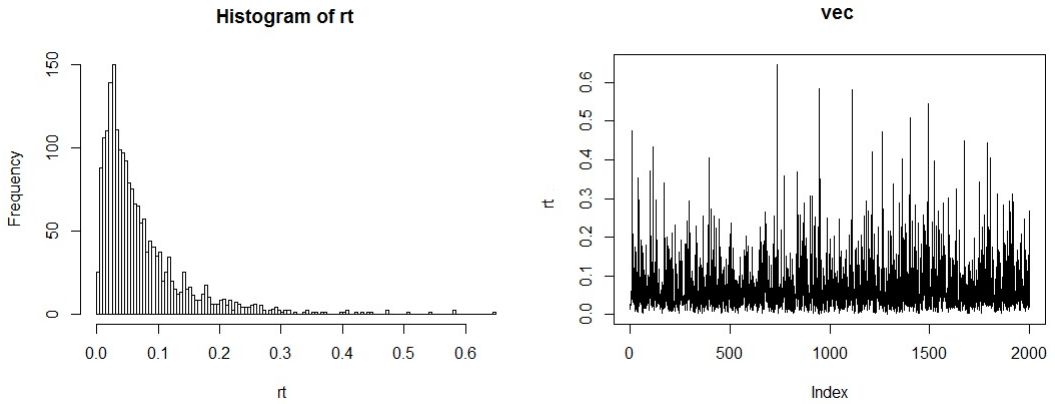

Figure 9: Sampling from our density $f(x)$ using the Ratio of Uniforms algorithm and $n = 200$

5 Appendix

The below code was implemented in R.

5.1 Plotting our density

```
y <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
plot(y)
#Plot of our density f(x) up to proportionality
```

5.2 Numerical approximation

```
y <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
integrate(y,0,1)
```

5.3 Finding a and c

```
y_sqrt <- function(x) { ((1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2))^(1/2) }
optimize(f=y_sqrt,interval=c(0,1),maximum=TRUE)
#Finding the value of a
y_xsqrt <- function(x) { x*((1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2))^(1/2) }
optimize(f=y_xsqrt,interval=c(0,1),maximum=TRUE)
#Finding the value of c
```

5.4 Ratio of Uniforms

```
"RatioofUniforms" <- function(n) {
    u <- runif(n, min=0, max=1)
    value = mean((1/(u*(1-u)))*exp(-0.5*(3+log(u/(1-u)))^2))
    #Approximate the value of the integral h(x)
    a = 5.861968
    b = 0
    c = 0.2865048
    probaccept = (0.5*value)/(a*(c-b))
    #The acceptence probability is given by the above formula
    factor <- 1/probaccept
    ngen = ceiling(n*factor)
    #Generating ngen numbers at a time for rejection
    cat("Generating", format(ngen),"at a time for rejecting","\n")
    count <- 0
    rt <- vector("numeric")
    total <- 0
    while(count < n)
        {
        u1 = runif(ngen)
        u2 = runif(ngen)
        U = a*u1
        V = b + (c-b)*u2
        condition <- (1/((V/U)*(1-(V/U)))*exp(-0.5*(3+log((V/U)/(1-(V/U))))^2))^0.5
        rt <- c(rt, V[U<condition]/U[U<condition])
        count <- length(rt)
        total <- total + ngen
        }
    cat("acceptance probability =", format(count/total),"\n")
    cat("Theoretical acceptance probability =", format(probaccept),"\n")
    rt<-na.omit(rt)
    #Removing the values in rt which give NA
    hist(rt[1:n],100)
    #Plots the first n values from our vector rt as a histogram
    plot(rt[1:n],,"l")
    #Plots the path of the first n values from our vector rt
    acf(rt[1:n])
    #Plots the autocorrelation function for the first n values from our vector rt
    print(rt[1:n])
}
```

5.5 Finding M for g(x) Uniform

```
f1 <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
optimize(f=y,interval=c(0,1),maximum=TRUE)
#Finding the value of M
```

5.6 Uniform Rejection Algorithm

```
"RejectionUniform" <- function(n) {

    M = 34.36266
    probaccept = 1/M
    factor <- 1/probaccept
    ngen = ceiling(n*factor)
    #Generating ngen at a time for rejection
    cat("Generating", format(ngen),"at a time for rejection","\n")
    count <- 0
    rt <- vector("numeric")
    total <- 0
    while(count < n)
        {
        u1 <- runif(ngen, min=0, max=1)
        u2 <- runif(ngen, min=0, max=1)
        condition <- (1/M)*(1/(u1*(1-u1)))*exp(-0.5*(3+log(u1/(1-u1)))^2)
        rt <- c(rt, u1[u2<condition])
        count <- length(rt)
        total <- total + ngen
        }
    cat("acceptance probability =", format(count/total),"\n")
    cat("Theoretical acceptance probability =", format(probaccept),"\n")
    hist(rt,100)
    print(rt)

}
```

5.7 Finding M for g(x) Beta

```
f1_f2 <- function(x)

    {
    (((1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2))/(((x^(1.5-1))*((1-x)^(4-1)))/beta(1.5,4)))
    }

optimize(f=f1_f2,interval=c(0,1),maximum=TRUE)
#Finding the value of M
```

5.8 Beta Rejection Algorithm

```
"RejectionBeta" <- function(n) {

    M = 34.36266
    probaccept = 1/M
    factor <- 1/probaccept
    ngen = ceiling(n*factor)
    cat("Generating", format(ngen),"at a time for rejecting","\n")
    count <- 0
    rt <- vector("numeric")
    total <- 0
    while(count < n)
        {
        u1 <- runif(ngen, min=0, max=1)
        u2 <- runif(ngen, min=0, max=1)
        condition <- (1/M)*(1/(u1*(1-u1)))*exp(-0.5*(3+log(u1/(1-u1)))^2)
        rt <- c(rt, u1[u2<condition])
        count <- length(rt)
        total <- total + ngen
        }
```

```
    cat("acceptance probability =", format(count/total),"\n")
    cat("Theoretical acceptance probability =", format(probaccept),"\n")
    hist(rt,100)
    print(rt)

}
```

5.9 Direct $\phi - f$ method

```
"montecarlo"<-function(n){

    u <- runif(n, min=0, max=1)
    value = mean((1/(u*(1-u)))*exp(-0.5*(3+log(u/(1-u)))^2))
    print(value)

}
```

5.10 Anithetic

```
"montecarloA"<-function(n){

    u <- runif(n, min=0, max=1)
    v <- 1-u
    value = mean(0.5*((1/(v*(1-v)))*exp(-0.5*(3+log(v/(1-v)))^2) +
     (1/(u*(1-u)))*exp(-0.5*(3+log(u/(1-u)))^2)))
    print(value)

}
```

5.11 Stratified Sampling

```
"montecarloS"<-function(n)

    {
    u1 <- runif(n, min=0, max=0.005)
    u2 <- runif(n, min=0.005, max=0.01)
    u3 <- runif(n, min=0.01, max=0.015)
    u4 <- runif(n, min=0.015, max=0.02)
    u5 <- runif(n, min=0.02, max=0.03)
    u6 <- runif(n, min=0.03, max=0.05)
    u7 <- runif(n, min=0.05, max=0.1)
    u8 <- runif(n, min=0.1, max=0.2)
    u9 <- runif(n, min=0.2, max=0.3)
    u10 <- runif(n, min=0.3, max=0.4)
    u11 <- runif(n, min=0.4, max=1)
    value1 = (0.005)*mean((1/(u1*(1-u1)))*exp(-0.5*(3+log(u1/(1-u1)))^2))
    value2 = (0.005)*mean((1/(u2*(1-u2)))*exp(-0.5*(3+log(u2/(1-u2)))^2))
    value3 = (0.005)*mean((1/(u3*(1-u3)))*exp(-0.5*(3+log(u3/(1-u3)))^2))
    value4 = (0.005)*mean((1/(u4*(1-u4)))*exp(-0.5*(3+log(u4/(1-u4)))^2))
    value5 = (0.01)*mean((1/(u5*(1-u5)))*exp(-0.5*(3+log(u5/(1-u5)))^2))
    value6 = (0.02)*mean((1/(u6*(1-u6)))*exp(-0.5*(3+log(u6/(1-u6)))^2))
    value7 = (0.05)*mean((1/(u7*(1-u7)))*exp(-0.5*(3+log(u7/(1-u7)))^2))
    value8 = (0.1)*mean((1/(u8*(1-u8)))*exp(-0.5*(3+log(u8/(1-u8)))^2))
    value9 = (0.1)*mean((1/(u9*(1-u9)))*exp(-0.5*(3+log(u9/(1-u9)))^2))
    value10 = (0.1)*mean((1/(u10*(1-u10)))*exp(-0.5*(3+log(u10/(1-u10)))^2))
    value11 = (0.6)*mean((1/(u11*(1-u11)))*exp(-0.5*(3+log(u11/(1-u11)))^2))
    value = value1+value2+value3+value4+value5+value6+value7+value8+value9+value10+value11
    print(value)

}
```

5.12 Finding c for Hit and Miss

```
y <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
optimize(f=y,interval=c(0,1),maximum=TRUE)
```

5.13 Hit & Miss Algorithm

```
"montecarloHM"<-function(n) {
    u <- runif(n, min=0, max=1)
    v <- runif(n, min=0, max=34.36266)
    w <- (1/(u*(1-u)))*exp(-0.5*(3+log(u/(1-u)))^2)
    value <- 0
    count <- 0
    c=34.36266
        while(count < n)
        {
        value <- sum(ifelse(v<w, 1, 0))
        count <- count + 1
        }
    Answer = c*(1/n)*value
    print(Answer)
}
```

5.14 Faster Hit & Miss Algorithm

We are able to improve the above algorithm, with the following, more efficient code.

```
"montecarloHMF"<-function(n){
    u <- runif(n, min=0, max=1)
    v <- runif(n, min=0, max=34.36266)
    indicator <- vector("numeric")
    c=34.36266
    value = 0
    for(i in 1:n)
        {
           if(v[i]<=(1/(u[i]*(1-u[i])))*exp(-0.5*(3+log(u[i]/(1-u[i])))^2))
           indicator[i]=1
           else
           indicator[i]=0
        value = indicator[i] + value
        }
    Answer = c*(1/n)*value
    print(Answer)
}
```

5.15 Importance Sampling

```
"montecarloIS"<-function(n){
    betadis <- function(x) {((x^(1.5-1))*((1-x)^(22-1)))/beta(1.5,22)}
    u <- rbeta(n,1.5,22)
    value = mean( (1/(u*(1-u)))*exp(-0.5*(3+log(u/(1-u)))^2)/betadis(u))
    print(value)
}
```

5.16 Plotting the Beta(1.5,22) distribution

```
betadis <- function(x) { ((x^(1.5-1))*((1-x)^(22-1)))/beta(1.5,22)  }
plot(betadis)
```

5.17 Metropolis Hastings Uniform distribution

```
mhastU<-function (n,k,x0) {
    vec <- vector("numeric", n)
    alpha <- vector("numeric",n)
    density <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
    x <- x0
    vec[1] <- x
    for (i in 2:n) {
```

```
        q <- runif(1, -k, k)
        y <- x + q
        if(dunif(y,0,1)==0) aprob <- 0 else aprob <- min(1,density(y)/density(x))
        u <- runif(1)
        if (u < aprob) x <- y
        vec[i] <- x
        alpha[i] <- aprob
        }
    hist(vec,100)
    print(vec)
    plot(vec,,"l")
    mean(alpha)
}
```

5.18 Metropolis Hastings Normal distribution

```
mhastN<-function (n,sigma,x0)  {
    vec <- vector("numeric", n)
    alpha <- vector("numeric",n)
    density <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
    x <- x0
    vec[1] <- x
    for (i in 2:n) {
        q <- rnorm(1, 0, sigma)
        y <- x + q
        if(dunif(y,0,1)==0) aprob <- 0 else aprob <- min(1,density(y)/density(x))
        u <- runif(1)
        if (u < aprob) x <- y
        vec[i] <- x
        alpha[i] <- aprob
        }
    hist(vec,100)
    print(vec)
    plot(vec,,"l")
    mean(alpha)
    acf(vec)
    #Plots the autocorrelation function
}
```

5.19 Comparing the first 1000 samples to the subsequent 1000 samples

```
mhastN2<-function (sigma,x0)  {
    n=2000
    vec <- vector("numeric", n)
    alpha <- vector("numeric",n)
    density <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
    x <- x0
    vec[1] <- x
    for (i in 2:n) {
        q <- rnorm(1, 0, sigma)
        y <- x + q
        if(dunif(y,0,1)==0) aprob <- 0 else aprob <- min(1,density(y)/density(x))
        u <- runif(1)
        if (u < aprob) x <- y
        vec[i] <- x
        alpha[i] <- aprob
        }
    hist(vec[1:1000],100)
    #Plots the first 1000 variables as a histogram
    hist(vec[1001:2000],100)
    #Plots the second 1000 variables as a histogram
    hist(vec,100)
```

```
    print(vec)
    plot(vec,,"l")
    mean(alpha)
    acf(vec)
}
```

5.20 Ammended Metropolis Hastings Algorithm

```
mhastlag<-function (n,x0,lag) {
    vec <- vector("numeric", n)
    alpha <- vector("numeric",n)
    density <- function(x) { (1/(x*(1-x)))*exp(-0.5*(3+log(x/(1-x)))^2) }
    x <- x0
    vec[1] <- x
        for (i in 2:n)
        {
        q <- rnorm(1, 0, 0.24)
        y <- x + q
        if(dunif(y,0,1)==0) aprob <- 0 else aprob <- min(1, density(y)/density(x))
        u <- runif(1)
        if (u < aprob) x <- y
        vec[i] <- x
        alpha[i] <- aprob
        }
    v <- vector("numeric")
    n1 = floor(n/lag)-1
    for (j in 1:n1) { v[j]<-vec[j*lag+1] }
    #Producing our new vector v
    hist(v,100)
    plot(v,,"l")
    acf(vec)
    #Plots our original autocorrelation function
    acf(v)
    #Plots our new autocorrelation function
}
```