Advanced Statistical Methods

Project Report

Monte Carlo Methods

Lushaank K – 20170010082

Vineesh S – 20170010132

Sai Santosh C – 20170010134

Monte Carlo Methods

Monte Carlo methods are widely used heuristic techniques which can solve a variety of common problems including optimization and numerical integration problems. These algorithms work by cleverly sampling from a distribution to simulate the workings of a system.

They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo simulation can be used to tackle a range of problems in virtually every field such as finance, engineering, supply chain, and science.

Monte Carlo simulations are named after the gambling hotspot in Monaco, since chance and random outcomes are central to the modelling technique, much as they are to games like roulette, dice, and slot machines.

Application

Consider a game called wheel of fortune in which you get to spin a wheel of different activities named 1-50, If you get a number lesser

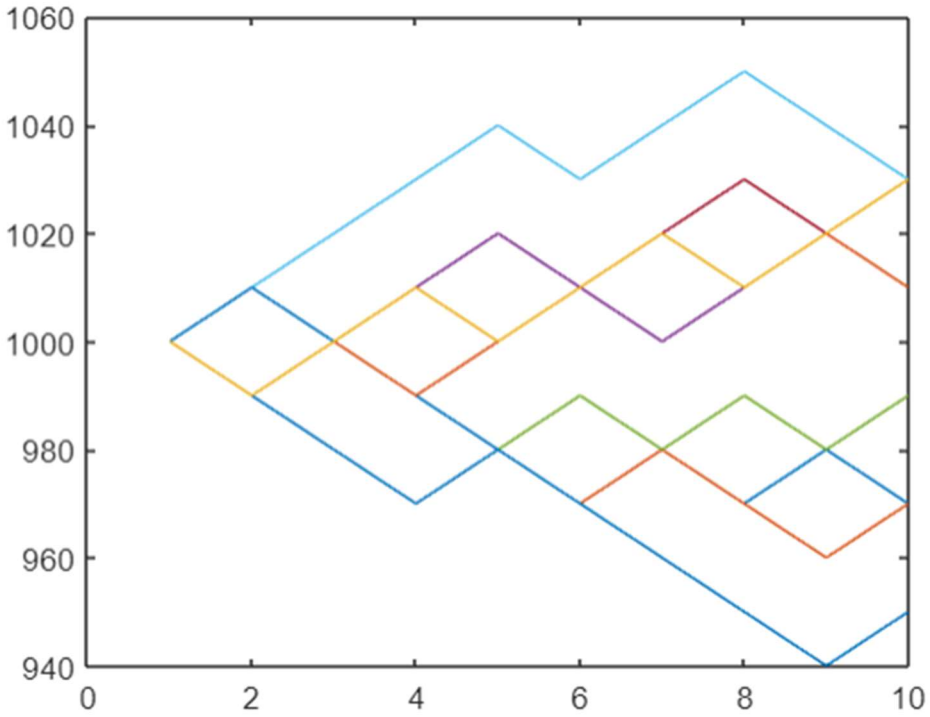
than 26, you lose. But any number greater then 26 and you would win the game.

Prob of you winning = 24/50

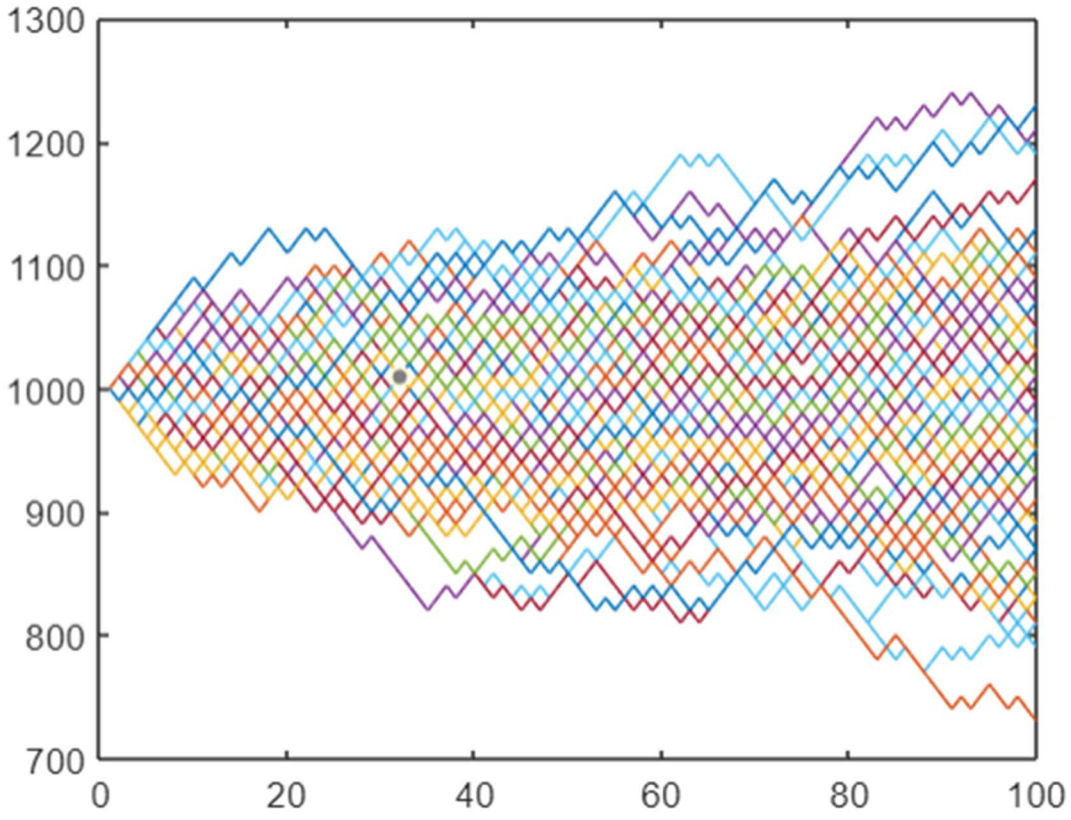
Prob of you losing = 26/50

Expected profit per round = 1*(24/50) – 1*(26/50) = -0.04 = -4%

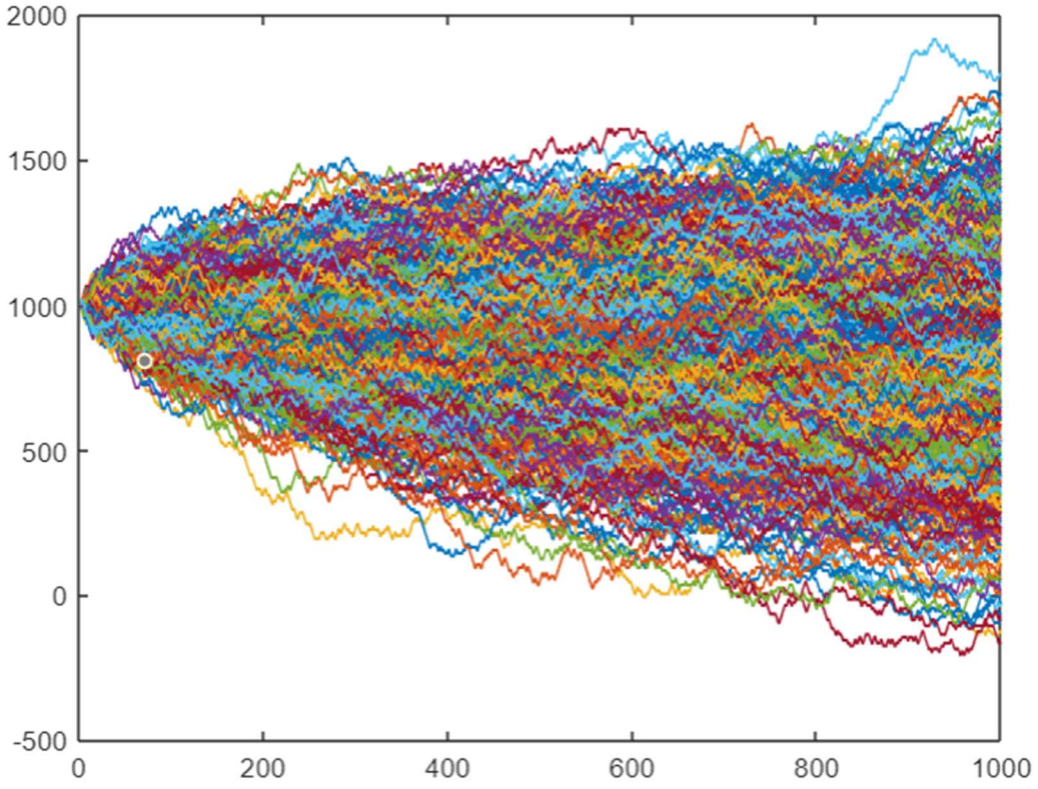
This is simulated in MATLAB and player starts at an amount of $1000 and he gambles $10 every round.



Player starts with $1000 and ends up with $1002



Player starts with $1000 and ends up with $985.4



Player starts with $1000 and ends up with $792.02

From the simulation experiment, we can observe that Jack has a better chance of making a profit (or minimize loss), if he places fewer bets. It can also be observed that the amount lost in every scenario is approximately 4% of the betting amount.

Importance Sampling

Importance sampling is a variance reduction technique that can be used in the Monte Carlo method. The idea behind importance sampling is that certain values of the input random variables in a simulation have more impact on the parameter being estimated than others. It is clear intuitively that we must get some samples from the interesting or an important region. We do this by sampling from a distribution that over weights the important region, hence the name importance sampling. Having oversampled the important region, we have to adjust our estimate somehow to account for having sampled from this other distribution.

Checking how Crude Monte Carlo different from Importance Sampling

Let us take an example of Integrating a hard integral using these methods so f(x) would be

$$\int_0^{\infty} \frac{e^{-1}}{1+(1-x)^3}\,dx$$

For the function we could find average value of the integral(function) It would be “y = 2x” and our algorithm would be

1. Get a random input value from the integration range
2. Evaluate the integrand
3. Repeat Steps 1 and 2 for as long as you like
4. Determine the average of all these samples and multiple by the range

Performing at a value of N=1000 we get an estimate of 0.531 which is not far from Wolfram's approximation of 0.529837

Now the real difference comes in the variance of the two methods in which importance sampling method has a lot lesser variance.

Crude Monte Carlo method gives a variance of 0.25 which is an error of 0.008

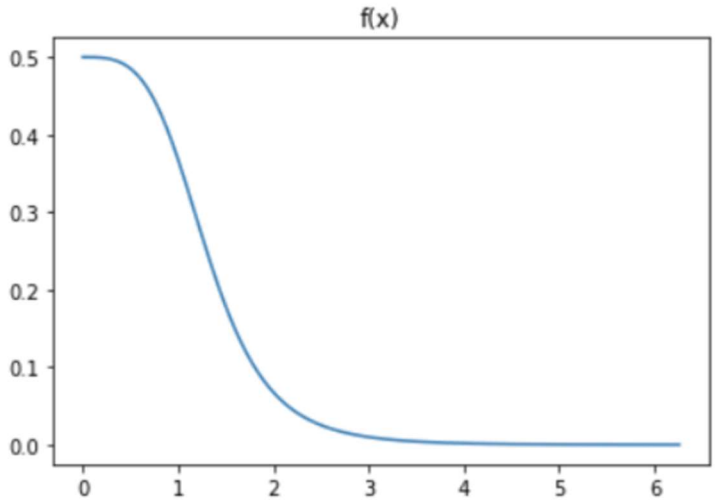
For the importance sampling we get to choose a different function such that

$$\frac{f(x)}{g(x)} \approx k$$

where "k" is some constant

There are a few criteria for g(x) to follow

1. *g(x)* is integrable
2. *g(x)* is non-negative on [a,b]
3. The indefinite integral of *g(x)*, which we'll call *G(x),* has a real inverse
4. The integral of *g(x)* in the range [a,b] must equal 1



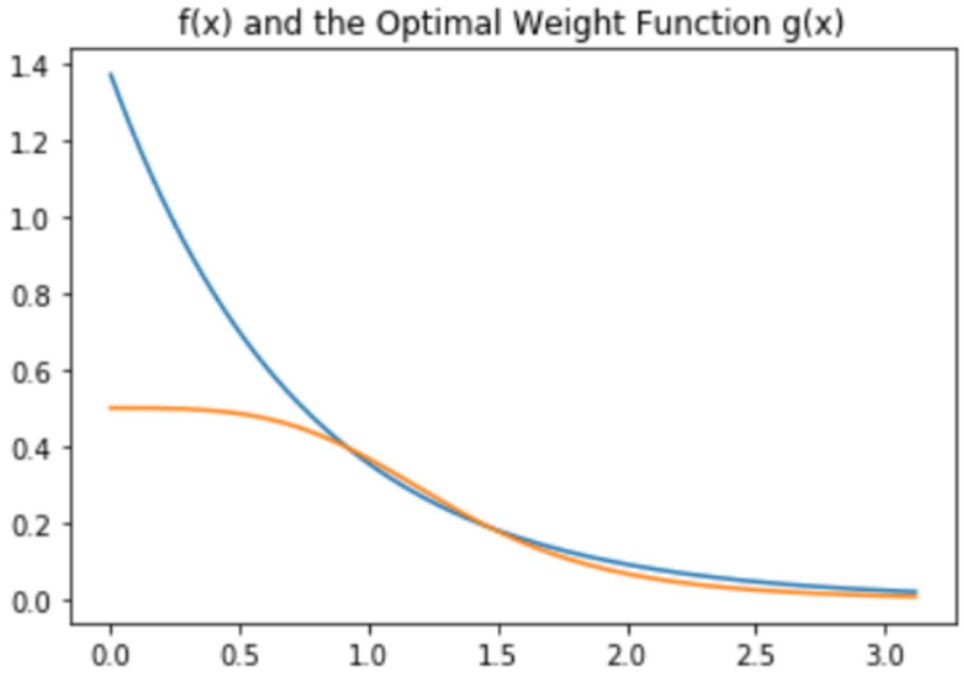
Visualizing suitable g(x) for f(x)

A suitable g(x) would be

$$g(x) = Ae^{-\lambda x}$$

where A = 1

And integral over '0' to 'inf' gives value of G(x) = 1



Finding the optimal value of λ at λ = 1.65

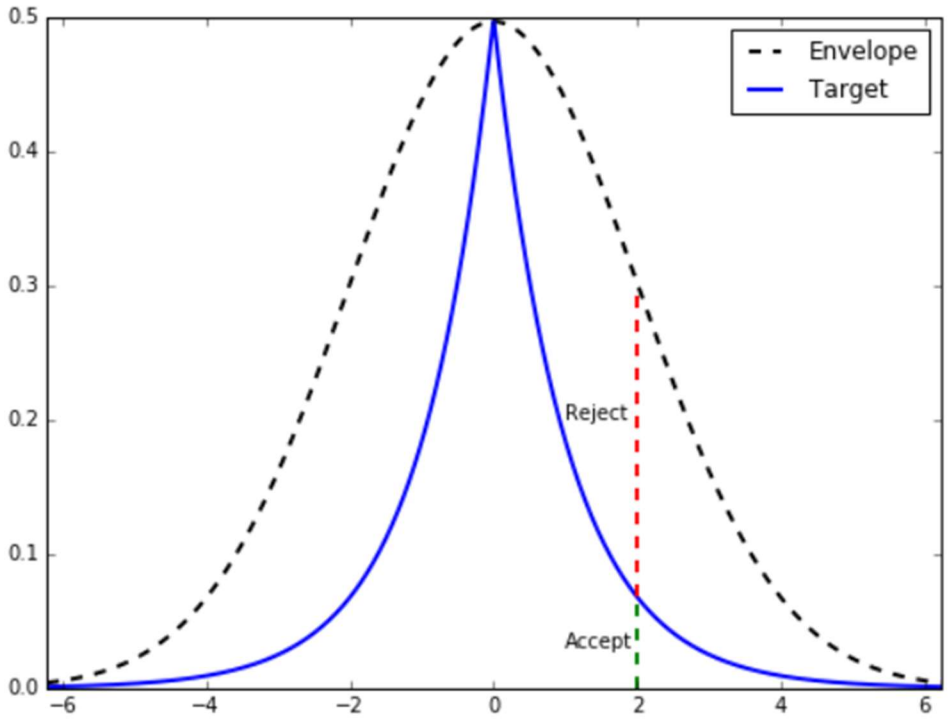
Performing at a value of N=1000 we get an estimate of 0.53 which is not far from Wolfram's approximation of 0.529837

Rejection Sampling

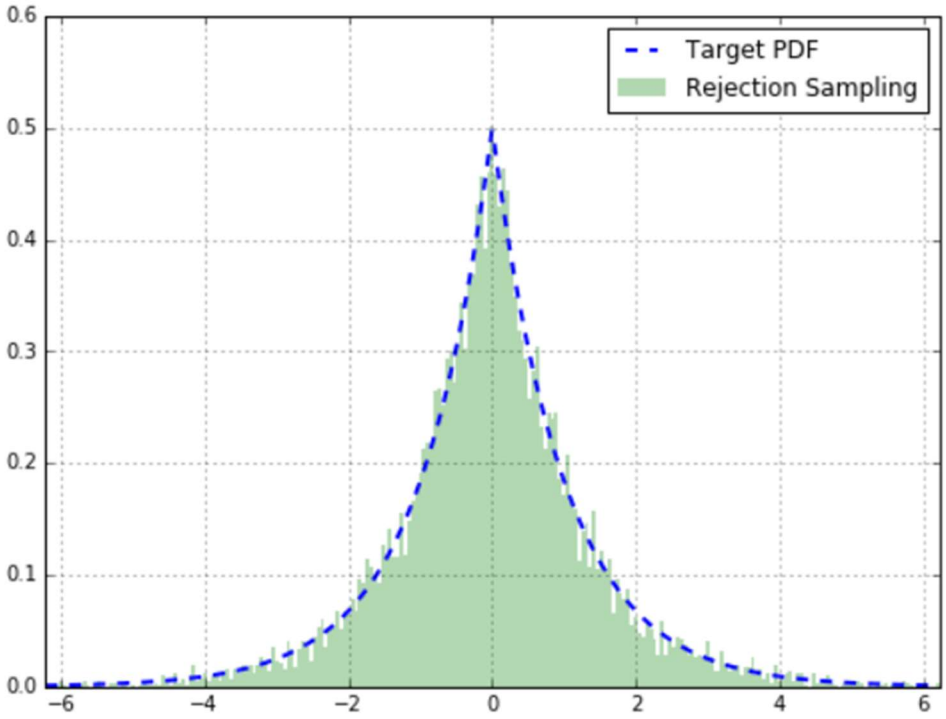
Rejection or Acceptance-Rejection sampling is a way to simulate random samples from an unknown (or difficult to sample from) distribution (called the target distribution) by using random samples from a similar, more convenient distribution. A random subset of the generated samples is rejected; the rest are accepted.

The goal is for the accepted samples to be distributed as if they were from the target distribution. It's usually only used when it's challenging to sample individual distributions within a larger Markov chain.
Our target distribution is double gamma distribution which is f(x) and to choose g(x) we use a normal distribution with mean at 0 and standard deviation of 2. The only restriction is $f(x) < Mg(x)$ for some $M > 1$.
Algorithm could be simply defined as

1. Sample x from g(x).
2. Sample y from U (0, Mg(x)) (uniform distribution).
3. If $y < f(x)$, then accept xx as a sample for f(x), otherwise go to step 1.



Once we find the sample of g(x) at x=2, we draw a uniform distribution with range equal to the height of Mg(x). If it's within the height of the target PDF we accept it, otherwise we reject it.



We conducted rejected sampling for N = 10000 and we could find a good fit for rejection sampler. In general, it is difficult to find an efficient envelope distribution that doesn't have too much "waste". When extending this method to higher dimensions, it becomes even more difficult because the ratio of "actual" space vs "wasted" space tends toward zero as the dimensions increase resulting in a highly inefficient algorithm.

Metropolis – Hastings Algorithm

Metropolis algorithm is method used in MCMC techniques to generate random samples from probability distribution. It is a beautifully simple algorithm for producing samples from distributions that may otherwise be difficult to sample from especially in high dimensional distributions rather than single ones which has another technique.

Let's start with equation of Markov Chain

$$p(x)P(x \to x') = p(x')P(x' \to x)$$

Here p(x) is target distribution and $\mathrm{P}(\mathrm{x}\to\mathrm{x}')$ is the transitional probability going from point " *x* " to point " *x'* ".

We broke $P(x \rightarrow x')$ into two independent steps namely a proposal distribution $g(x \rightarrow x')$ and an acceptance distribution $A(x \rightarrow x')$. Rewriting equations give us this equation,

$$\frac{A(x \rightarrow x')}{A(x' \rightarrow x)} = \frac{f(x')}{f(x)} \frac{g(x' \rightarrow x)}{g(x \rightarrow x')}$$

A typical choice that satisfies above equation is,

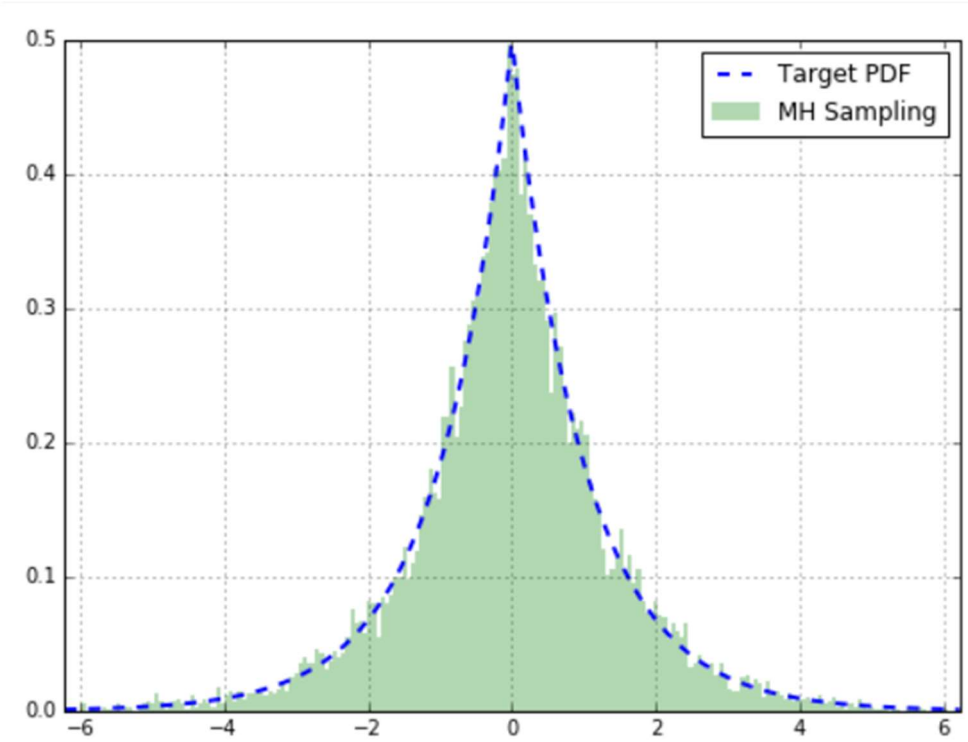
$$A(x \rightarrow x') = min(1, \frac{f(x')g(x' \rightarrow x)}{f(x)g(x \rightarrow x')})$$

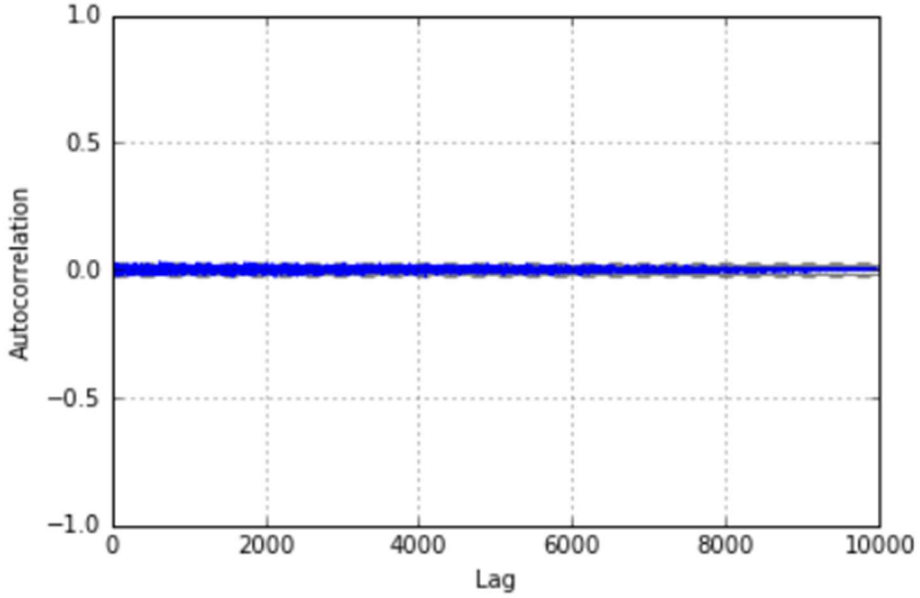
So, overall algorithm would be,

1. Initialize the initial state by picking a random x.
2. Find new x' according to $g(x \rightarrow x')$.
3. Accept x' with uniform probability according to $A(x \rightarrow x')$. If accepted transition to x', otherwise stay in state x.
4. Go to step 2, T times.
5. Save state x as a sample, go to step 2 to sample another point.

Addressing two main problems,

1. Since we randomly choose initial value of x, there might a chance of becoming p(x) small. If it's starting there it might spend a significant amount time before leaving that region with low densities. To overcome this, we generate a bunch of samples and throwing them away.
2. In the transition function of $P(x \rightarrow x')$ it is possible that both x and x' be adjacent samples. Finding correlation would be would lose one key aspect called independence. To overcome this, we take T^{th} sample, and record previous samples.





This is a good approximation to our double gamma distribution. On the Autocorrelation graph, we can see it doesn't diverge much in our sample, indicating they are relatively independent. It is also much faster than rejection (Acceptance-Rejection) sampling in terms of run time of codes.