Applied Statistical Modeling

PROJECT REPORT

Group Members:

1) M Hemanth Reddy (S20170010089)
2) S B Koushik (S20170010131)
3) V Himavanth Reddy (S20170010176)

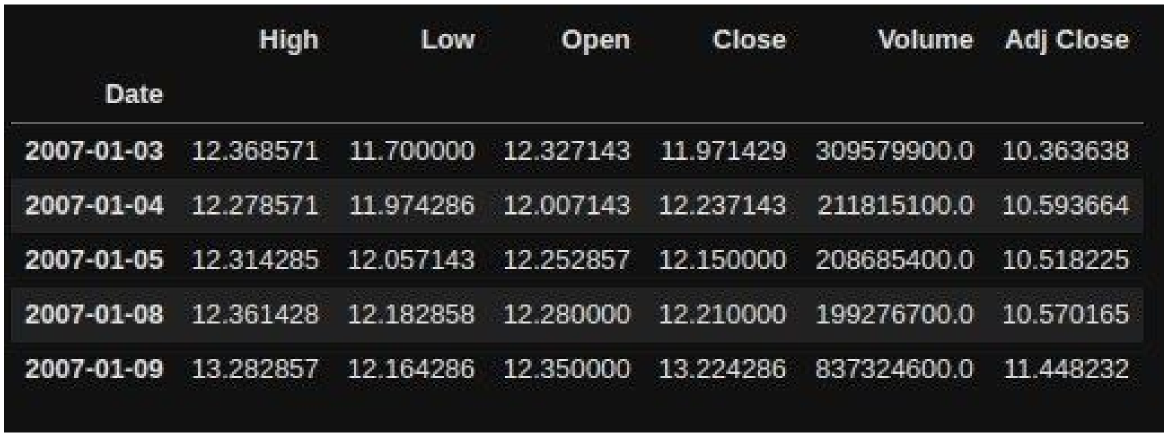
Contents

Monte Carlo Simulation:

- Monte Carlo is a computerized mathematical technique that allows people to account for risk in quantitative analysis and decision making.
- Monte Carlo simulation can be used to tackle a range of problems in virtually every field such as finance, engineering, science etc.
- Monte Carlo simulation furnishes the decision maker with a range of possible outcomes and the probabilities they will occur for any choice of action.
- Monte Carlo simulation performs risk analysis by building models of possible results by substituting a range of values- a probability distribution- for any factor that has inherent uncertainty.
- It then calculates the results over and over each time with a different set of random values.
- Depending on the number of uncertainties and range of values, a Monte Carlo simulation may involve thousands or millions of recalculations before it is complete.
- Monte Carlo simulation produces distribution of possible outcomes values.

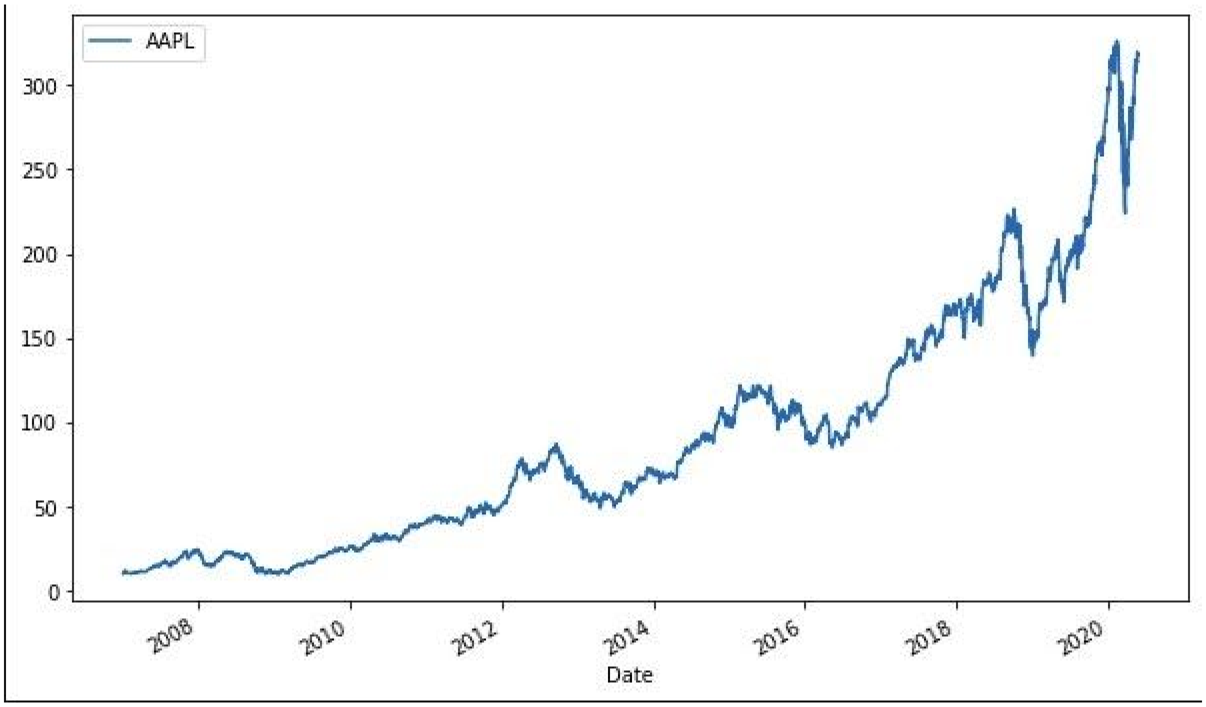
Predicting Stock Prices Using Monte Carlo Simulation:

- We used Pandas library to call a finance API for stock data, we used the Yahoo Finance API to get the stock prices of Apple stock prices for the past 10 years.
- The sample of the data collected is shown in the figure below.

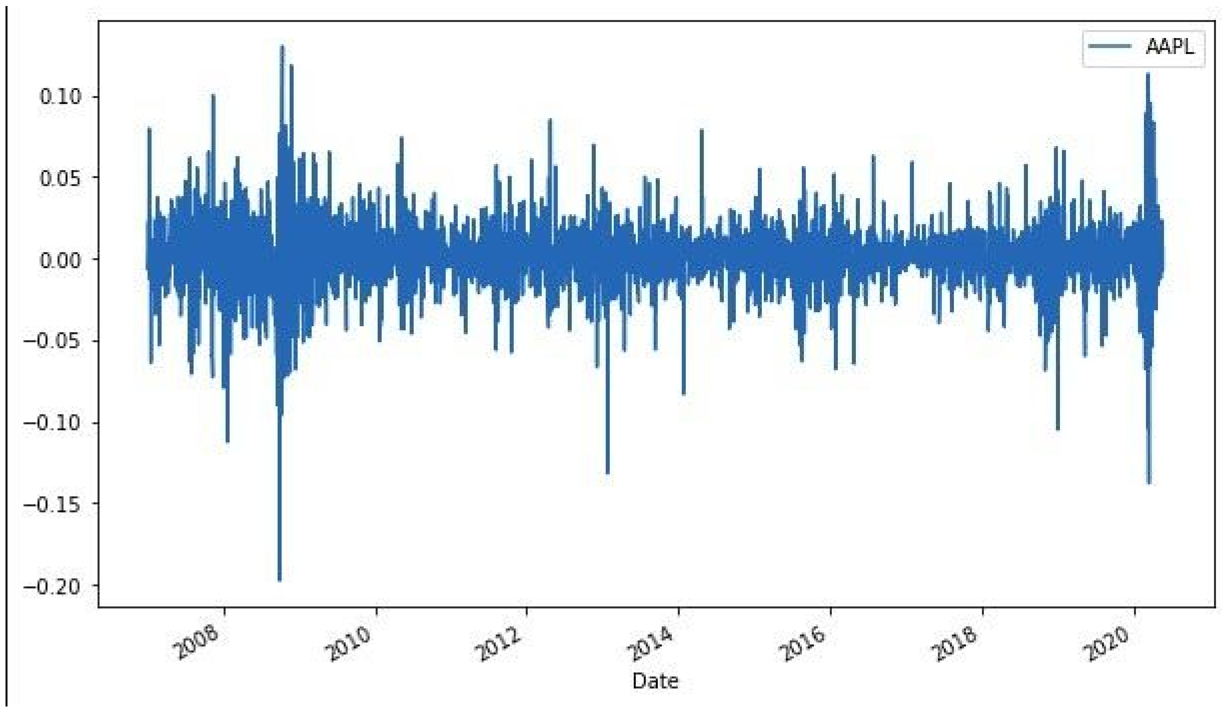


	High	Low	Open	Close	Volume	Adj Close
Date						
2007-01-03	12.368571	11.700000	12.327143	11.971429	309579900.0	10.363638
2007-01-04	12.278571	11.974286	12.007143	12.237143	211815100.0	10.593664
2007-01-05	12.314285	12.057143	12.252857	12.150000	208685400.0	10.518225
2007-01-08	12.361428	12.182858	12.280000	12.210000	199276700.0	10.570165
2007-01-09	13.282857	12.164286	12.350000	13.224286	837324600.0	11.448232

- The graph below shows Apple's closing price, which is gradually increasing.



- The picture below shows the log returns of the company’s stock.



Next we calculate the drift and generate some random values for the iteration using norm.ppf function from the scipy python library.

- <u>Drift: </u> Rate at which average changes.

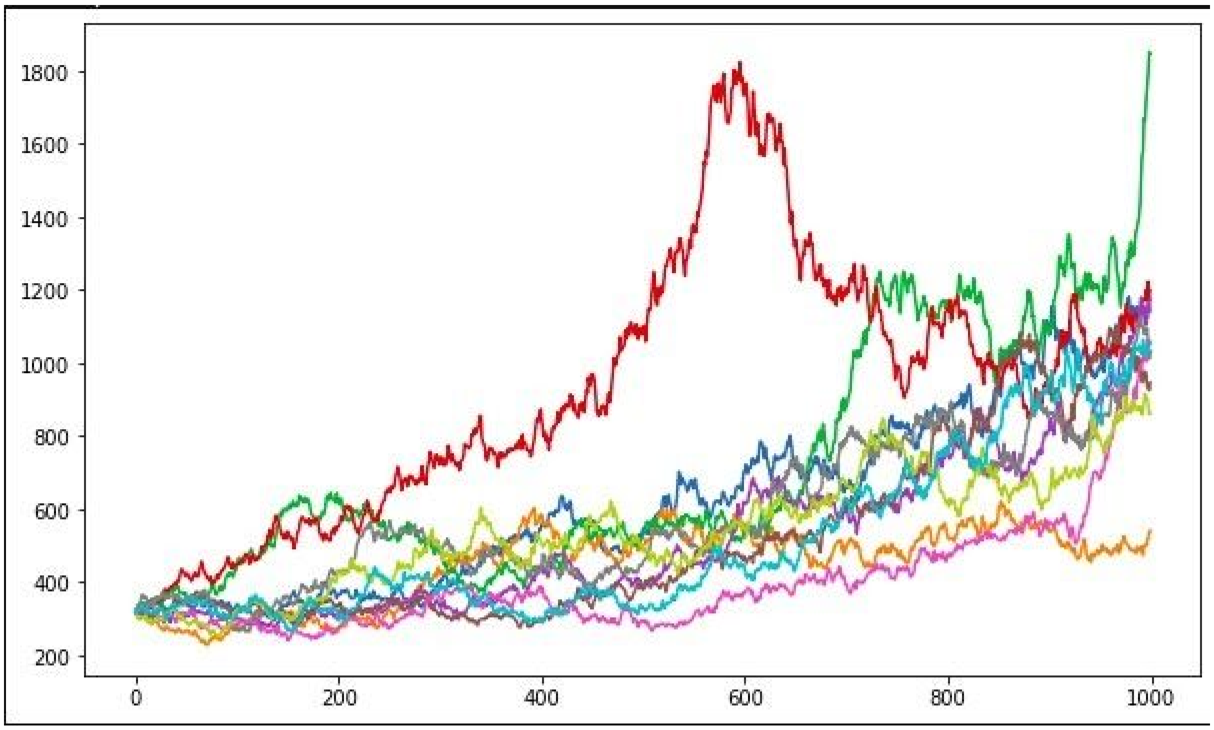
$$drift = mean(of\ returns) - \frac{(variance)}{2}$$

Random values = $\sigma \times NORMINV(RAND())$

Next day’s price = Today’s price $\times\, e^{(drift + stdev \times random\ value)}$

Output:

- We did 10 iterations for 1000 values each.
- The figure below shows the 10 different sets of stock predictions from an initial point.

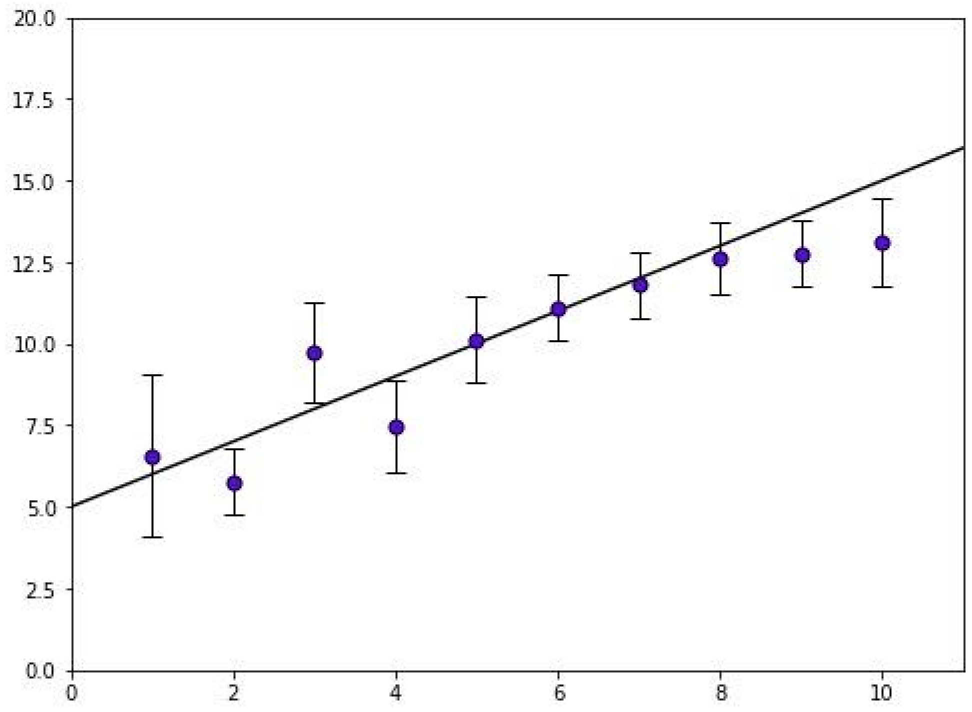


Gibbs Sampling:

- Gibbs Sampling is a Markov Chain Monte Carlo algorithm for obtaining a sequence of observations which are approximately from a specified multivariate probability distribution,when direct sampling is difficult.
- Gibbs sampling algorithm is generally used for high dimensional data such as image processing and micro arrays.
- It is called Monte Carlo because it draws samples from a specified probability distribution.
- Markov Chain comes from the fact that each sample is dependent on the previous sample.
- Gibbs Sampler is based on each parameter's posterior density, conditional on other parameters.
- In order to make use of the algorithm you need to be able to sample from all the conditional distributions for all model's parameters.
- Gibbs sampler is easy to implement, however it is less efficient than direct simulation from the distributions.

Fitting a Straight Line to Data Using Gibbs Sampler:

- Initially we generate mock data to form a straight line, i.e $y=mx+c$(where $m = 1$ and $c = 5$) and add some random error to the point so that we have to find a new straight line.



- To find the posterior distributions of the model parameters m and c, given the data y_i along with x_i and uncertainties σ.

$$P(m, c \mid \{y^i\}) = \frac{P(\{y^i\} \mid m,c) \times P(m,c)}{P(\{y^i\})}$$

$$P(m, c \mid \{y^i\}) \propto P(\{y^i\} \mid m, c) \times P(m, c))$$

- .Our uncertainties are Gaussian, so the likelihood function is as follows:

$$P(\{y^i\} \mid m, c) = \Pi_i \frac{1}{\sqrt{2\pi\sigma_i^2}} exp(- \frac{(y_i - mx_i - c^2)^2}{2\sigma_i^2})$$

- We need to decide on prior m,c and for simplicity, we'll use

$$P(m, c) \propto 1$$

$$P(m, c \mid \{y^i\}) \propto exp(-\Sigma_i \frac{(y_i - mx_i - c)^2}{2\sigma_i^2})$$

- To obtain samples from the joint distribution of our model parameters given our data, Gibbs Sampler relies on sampling from conditional distribution for each parameter in turn.
- This can be very useful when it is very difficult or impossible to sample from a joint distribution of all model parameters for some reason.
- For m:

$$P(m \mid c, \{y^i\}) \propto exp(-\Sigma_i \frac{(y_i^2 + m^2x_i^2 + c^2 + 2mcx_i - 2mx_iy_i - 2cy_i)}{2\sigma_i^2}) \propto exp(-\Sigma_i \frac{(m^2x_i^2 + 2mcx_i - 2mx_iy_i)}{2\sigma_i^2})$$

- We can get this into Gaussian by completing the square inside the exponential to turn our expression of the form $am^2 + bm$ into to the form $a(m-h)^2$, where $h = \frac{-b}{2a}$.

$$h = \frac{\Sigma_i\left(\frac{y_i}{\sigma_i^2}\right) - c\Sigma_i\left(\frac{x_i}{\sigma_i^2}\right)}{\Sigma_i\left(\frac{x_i^2}{\sigma_i^2}\right)}$$

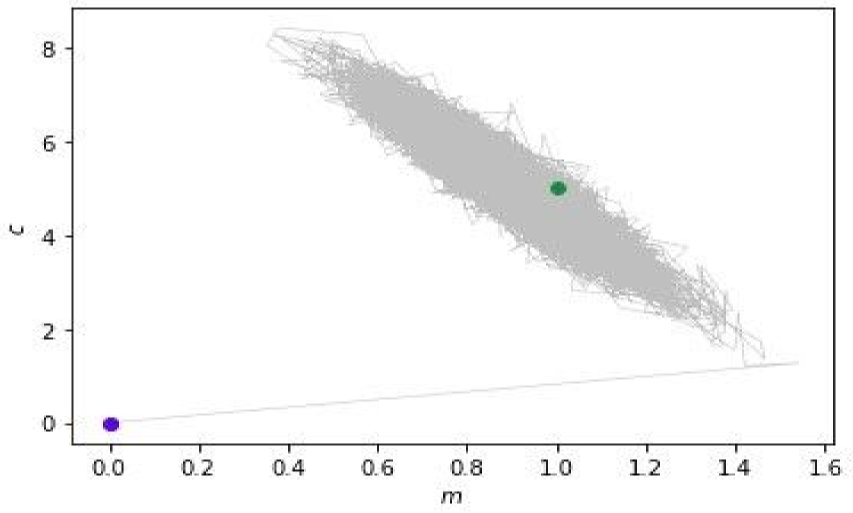
- Distribution is of the form:

$$P(m \mid c, \{y^i\}) \propto exp(-\tfrac{1}{2}\Sigma_i\left(\frac{x_i^2}{\sigma_i^2}\right)(m-h)^2)$$

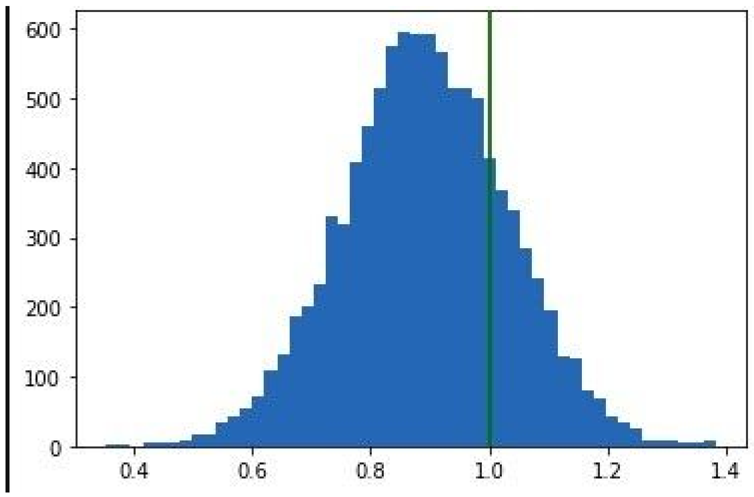
- The above equation is a Gaussian with mean $\mu_m = h$ and standard deviation $\sigma_m = \left(\frac{x_i^2}{\sigma_i^2}\right)^{\frac{-1}{2}}$
- Following the same procedure, we can find for distribution $P(c \mid m, \{y^i\})$

$$\mu_c = h \quad \text{and} \quad \sigma_c = \left(\frac{x_i^2}{\sigma_i^2}\right)^{\frac{-1}{2}}$$

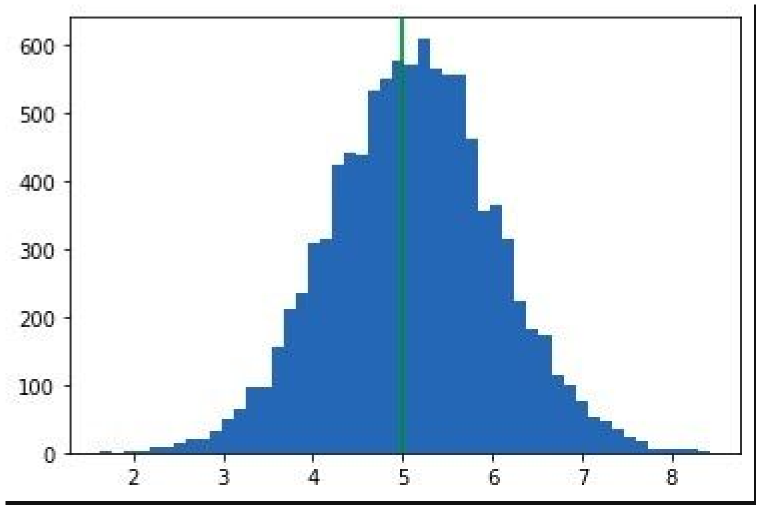
- We'll take the initial value to be (0,0) and sample for 11000 times from each conditional distribution in turn.
- We obtain the figure below after all the iterations,($m \approx 1$ and $c \approx 5$).



- The blue point is the initial point (0,0) and green point is the parameter value we selected in the beginning for our straight line.
- The gray portion is the path(how they vary) by the random samples during the iterations.

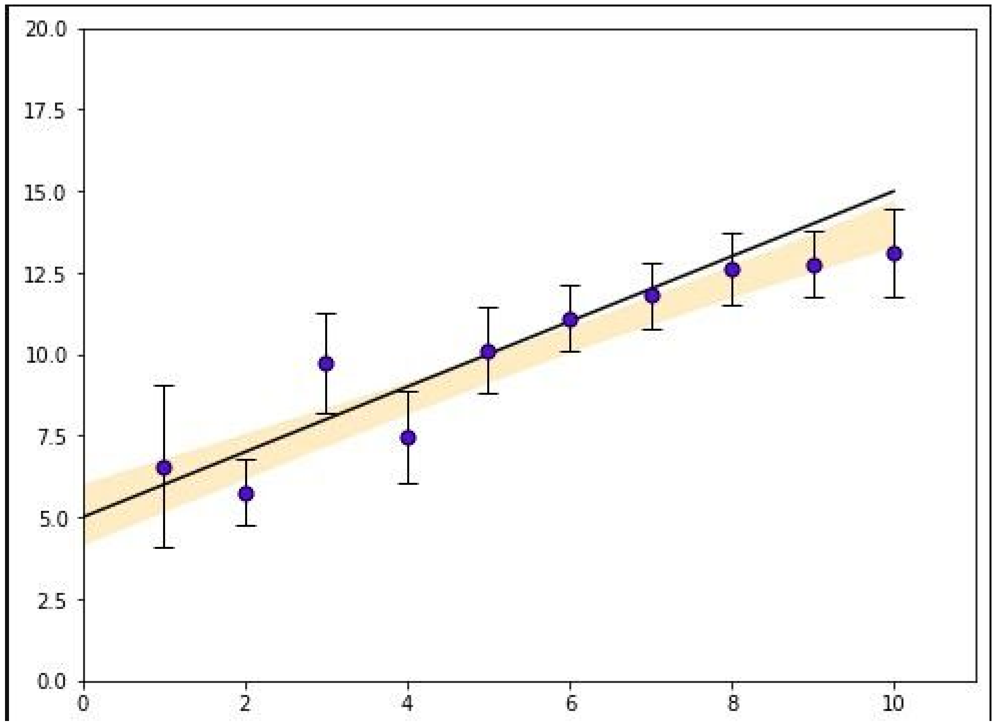


- After leaving the first 500 values of the iterations, we plot the obtained value of m against its frequency.



After leaving the first 500 values of the iterations, we plot the obtained value of c against its frequency.

Output:



Importance Sampling:

- Unlike other sampling techniques, this method is used to find the approximation.
- If we want to calculate an Expectation of function f(x) ,where x~p(x),belongs to some distribution,Then

$$E[f(x))] = \int f(x)p(x)dx \approx \frac{1}{n}\Sigma_i f(x_i)$$

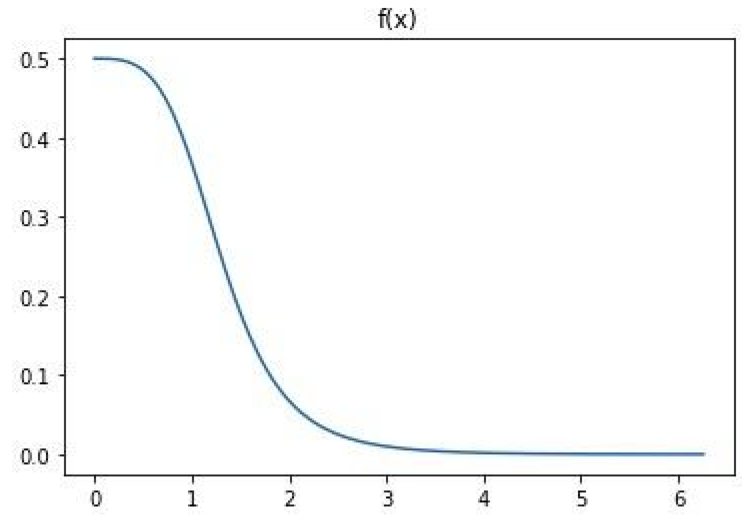
- In the Monte Carlo algorithm, we just take the samples from p-distribution and calculate the average for a large number of iterations.
- But what if we aren't able to sample from the p-distribution.
- In Importance Sampling, the expectation can be estimated with some known and easily sampled distribution.

$$E[f(x))] = \int f(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{n}\Sigma_i f(x_i)\frac{p(x_i)}{q(x_i)}$$

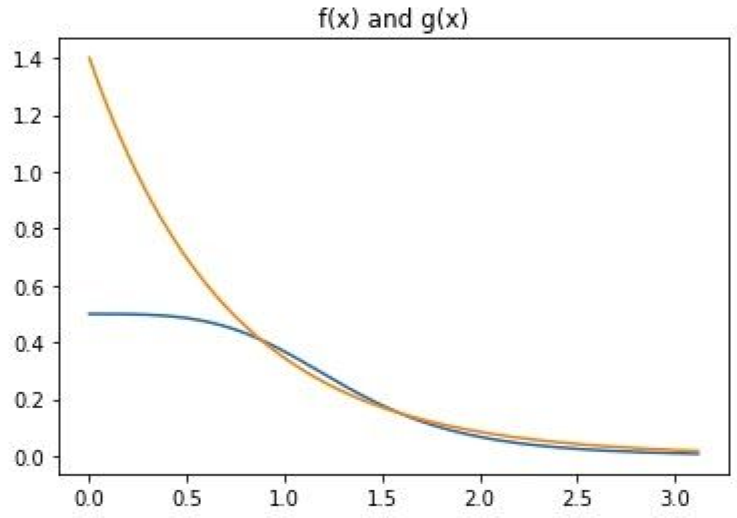
- Here, q(x) is a known probability density function.Now we can take the samples from the q-density function where sampling is possible.
- The q(x) density function has to be as close to as f(x) to reduce the variance estimator.

Implementation of Importance Sampling:

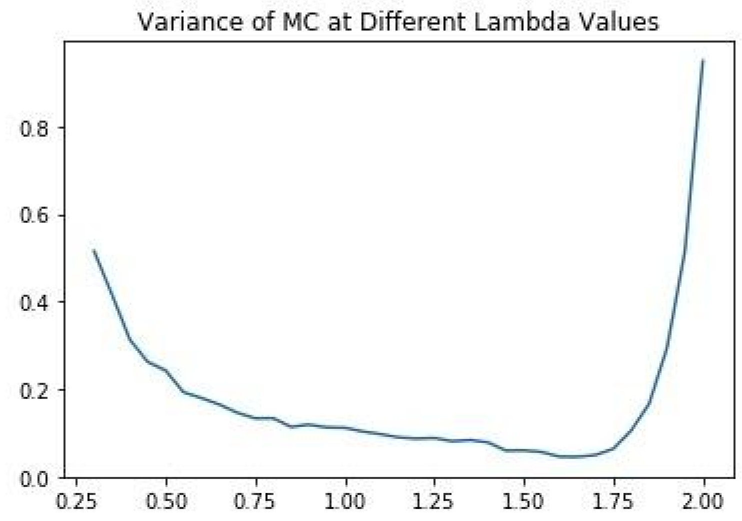
➔ We take the input function f(x) to be $f(x) = \frac{e^{-x}}{(1+(x-1)^2)}$



- Monte Carlo Approximation for 10000 values is : 0.669941.
- Variance of Approximation for 10000 values is : 0.266369.
- Error in Approximation for 10000 values is : 0.005161.
- Now we take q(x) function, i.e q(x) = $ae^{-\lambda x}$ with initial $\lambda = 0.05$ *and* $a = 1.4$



- After iterating through a set of values, substituting each value of a and λ, and finding the variance result, we select a and λ for which the variance is least among all.
- From the picture below you can see how λ values influence the variance for range of values.



- After finding the optimal λ value, we run many simulations on the g(x) density function to get the mean.
- The obtained results for Importance Sampling Approximation: 0.698503.

Importance Sampling Variance: 0.05047.

Importance Sampling Error : 0.002246.

Simulated-annealing

Simulated annealing is a method for finding a good (not necessarily perfect) solution to an optimization problem.

Simulated annealing's strength is that it avoids getting caught at local maxima. We simulated the traveling salesman problem, the salesman is looking to visit a set of cities in the order that minimizes the total number of miles he travels.

We simulated as follows:

- Initially we generated a random tourp plan(the order in which the salesman visits the cities) and computed the cost function of the tour.
- Our cost function is the total number of miles the salesman has to travel if he follows that tour plan.
- Generated a random neighboring tour plan.
- Calculated the new tourplans cost function.
- Depending on the acceptance probability change to the new state or stay in the same state.
- Repeat the above steps until an acceptable solution is found.

$$\text{Acceptance Probability} = e^{\frac{c_{new} - c_{old}}{T}}$$

We took initial values as follows:

T = 1.0

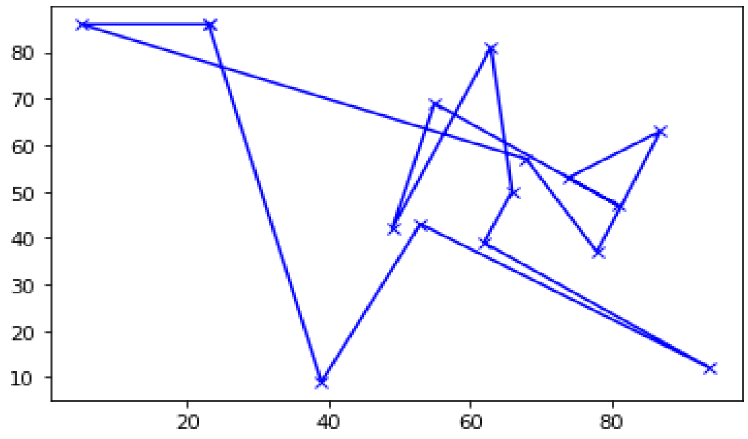
T_min = 0.00001

alpha=0.9

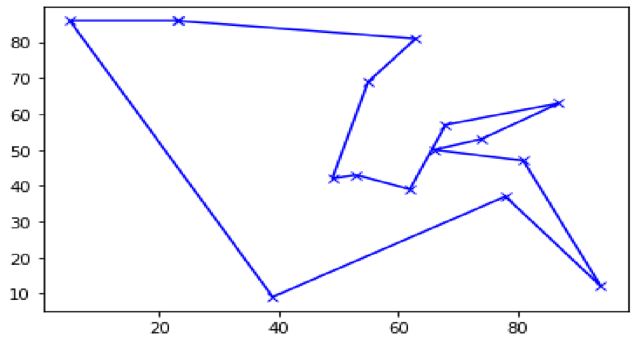
Results as follows for the travelling salesman problem:

Example 1 :

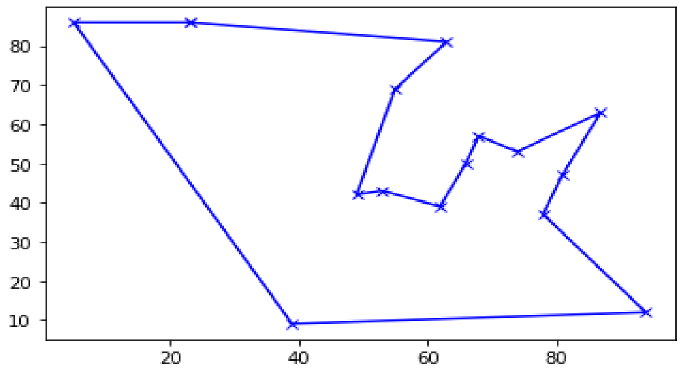
Initial randomly generated state



In the middle of simulation

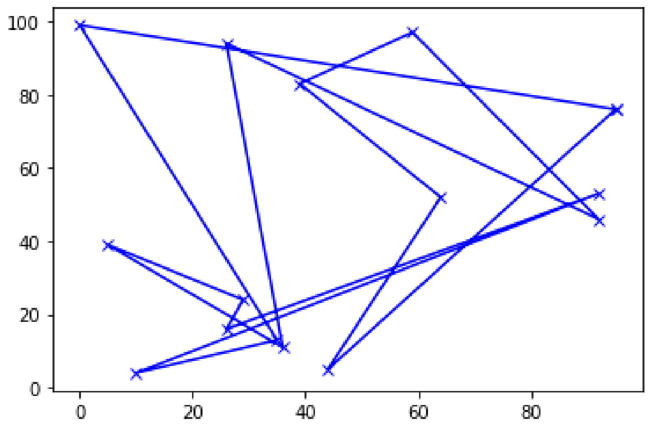


Best solution

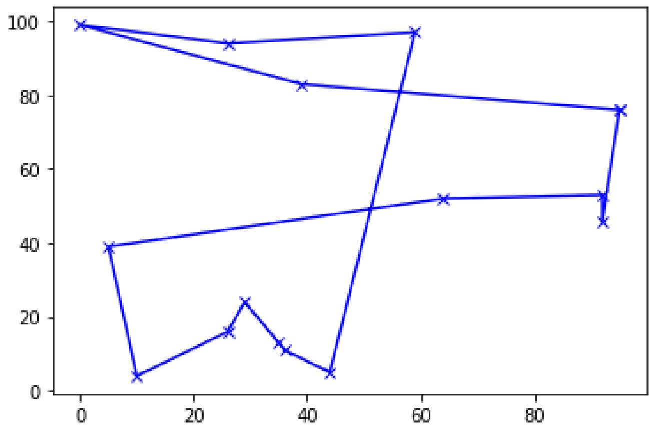


Example 2:

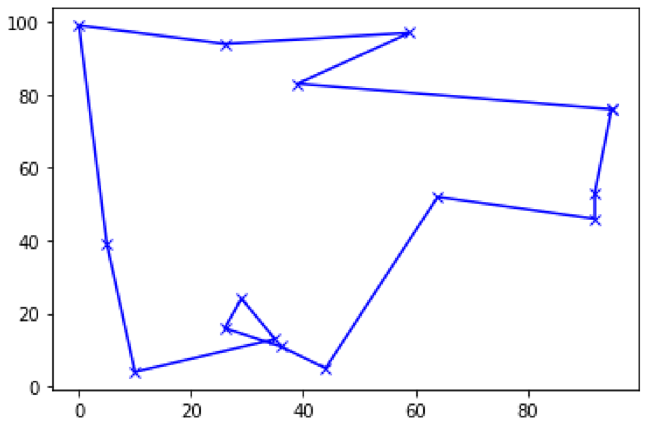
Initial randomly generated state



In the middle of simulation



Near optimum solution



Rejected Sampling

Rejection Sampling is used to draw samples from a complicated target distribution where direct sampling is hard.

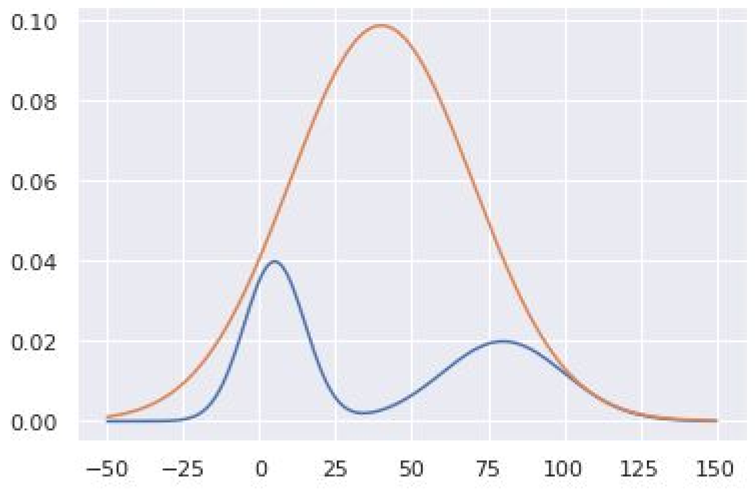
Sampling can be done by using a proposal distribution Q(x) that is easy to sample from.
This Q(x) has to envelope the target distribution P(x)(complicated target distribution).
This Q(x) can be sampled directly under the restriction that $P(x) < MQ(x)$ where $M > 1$ and is an appropriate bound on P(x).
The scaling factor $M = \max(P(x)/Q(x)$ for all x.)

Select samples from scaled proposal distribution and then uniformly select the value from zero to its height at that point.
Now we get samples that are uniform under the proposal distribution and Only accept samples that are under the target distribution.

Our target distribution(P(x)) is combination of two probability distribution functions as below:

1)normal continuous probability distribution function with mean 5 and standard deviation 10.
2)normal continuous probability distribution function with mean 80 and standard deviation 20.

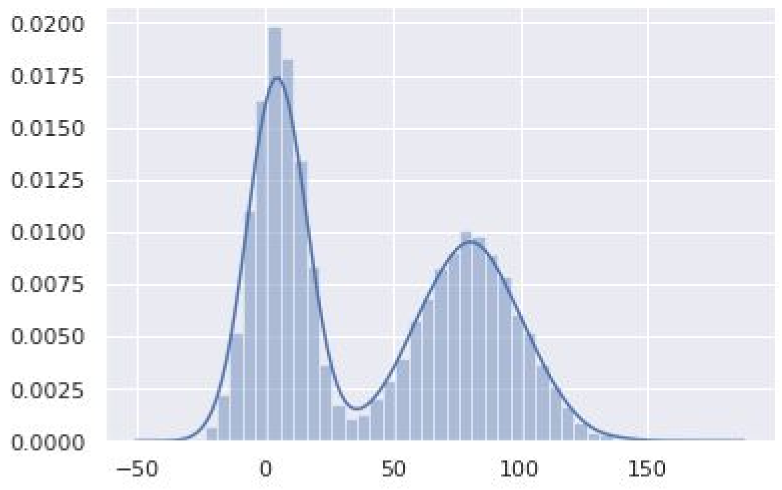
Our proposed distribution(Q(x)) is normal continuous probability distribution function with mean 40 and standard deviation 30.



In the above image the red curve is the proposed distribution Q(x) after scaling, the blue curve is the target distribution P(x).

As we can see from the curves, P(x) is hard to sample, so we are using Q(x) for sampling P(x) easily and Q(x) after scaling just exactly envelops the curve P(x).

Sampling curve



Monte Carlo Metropolis-Hastings :

Metropolis-Hastings algorithm generates candidate state transitions from an alternate distribution, and accepts or rejects each candidate probabilistically.

It is an algorithm which allows us sampling from a generic probability distribution. Generic probability distribution allows us to sample from a generic probability distribution/ Target distribution even if we don't know the normalizing constant.

We construct a sample from the Markov chain whose stationary distribution is the target distribution we are looking for. It consists of picking an arbitrary starting value and then iteratively accepting or rejecting candidate samples drawn from another distribution. It is easy to sample.

Initially We only know :

Sample distribution $P(\theta) \alpha G(\theta)$

*we don't know the normalizing constant

Algorithm :

1) Select an initial value for θo
2) For large number of iterations :
    a) Draw a candidate $\theta^* \sim q(\theta^* \mid \theta i\text{-}1)$ [from the proposal distribution]
    b) Set ratio $\alpha = \frac{Q(\theta^*) \,.\, q(\theta^* \mid \theta_{i-1})}{Q(\theta_{i-1}) \,.\, q(\theta_{i-1} \mid \theta^*)}$
    c) If $\alpha > 1$ => accept θ^* and set $\theta_i \leftarrow \theta^*$

       If $\alpha < 1$ => accept θ^* and set $\theta_i \leftarrow \theta^*$

with probability α.

reject θ^* and set $\theta_i \leftarrow \theta_{i-1}$

with probability $1 - \alpha$.

b) and c) act as a correction since proportional distribution q is not equal to target distribution. At each step in the chain we draw a candidate and decide to move the chain or remain at the same position.

If the proposed move to the chain is advantageous ($\alpha > 1$) : we will move. If not. We move with probability α. In either case, the state moves to high-density points in the distribution with high probability, and to low-density points with low probability. After convergence, the Metropolis-Hastings algorithm describes the full target posterior density, so all points are recurrent.

Our move to the candidate only depends on where the chain currently is. This is a Markov chain.

We must select an appropriate candidate generation distribution Q. It may or may not depend on the previous iterations value of teta.

Simulation :

The dataset is a selection of real estate prices 'p' , with the associated age 'a' of each house. We estimate a linear relationship between the two variables, using the Metropolis-Hastings algorithm.

Linear model:

$$\mu i = \beta 0 + \beta 1 \, . \, ai$$

Sampling distribution:

$$pi \sim N(\mu i, \tau)$$

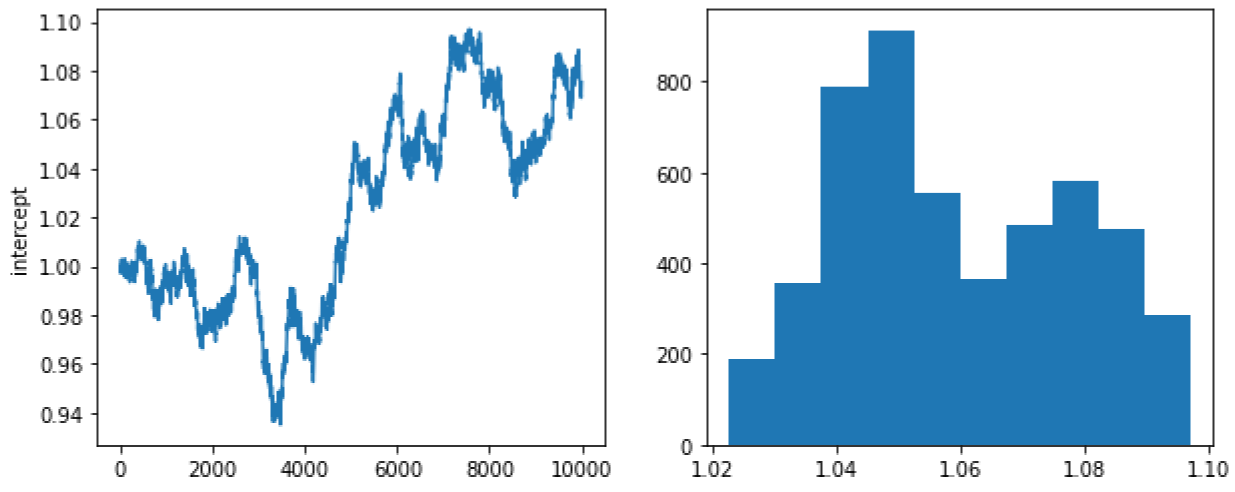
Prior distributions:

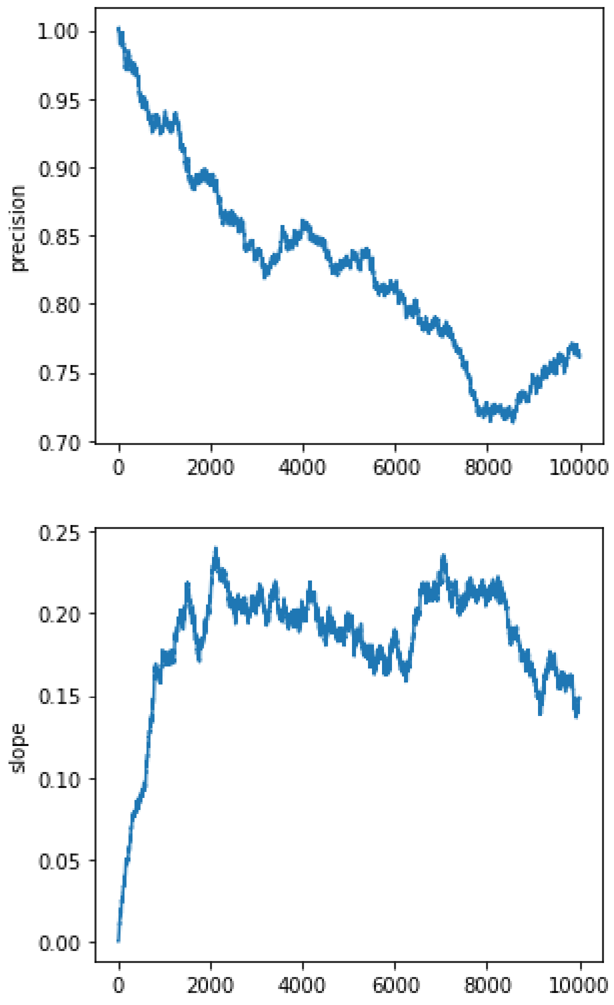
$$gi \sim N(0,10000)$$
$$\tau \sim \text{Gamma}(0.001,0.001)$$

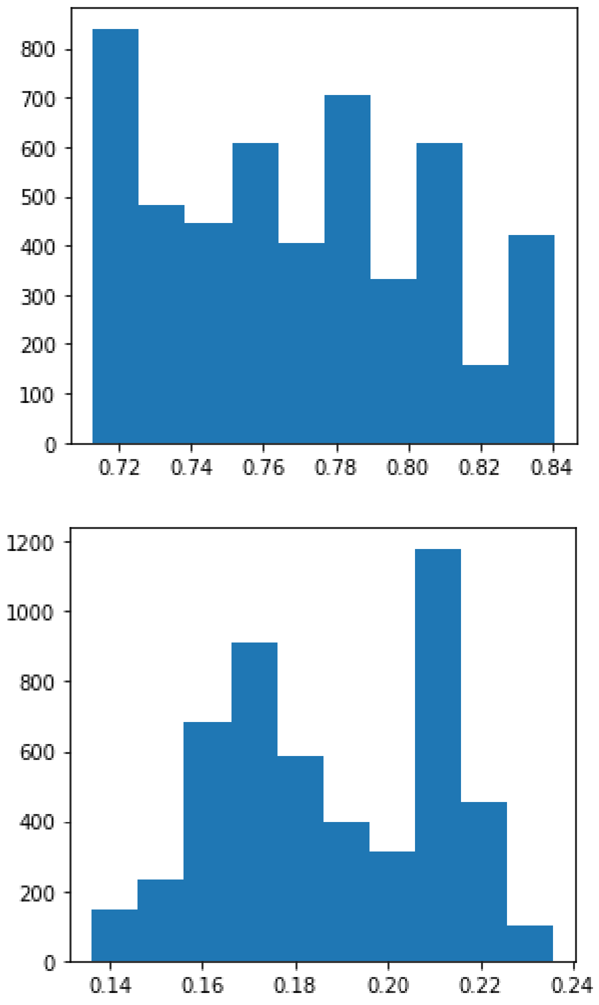
The metropolis function implements a simple random-walk Metropolis-Hastings sampler for this problem. It accepts as arguments:

- the number of iterations to run
- initial values for the unknown parameters
- the variance parameter of the proposal distribution (normal)

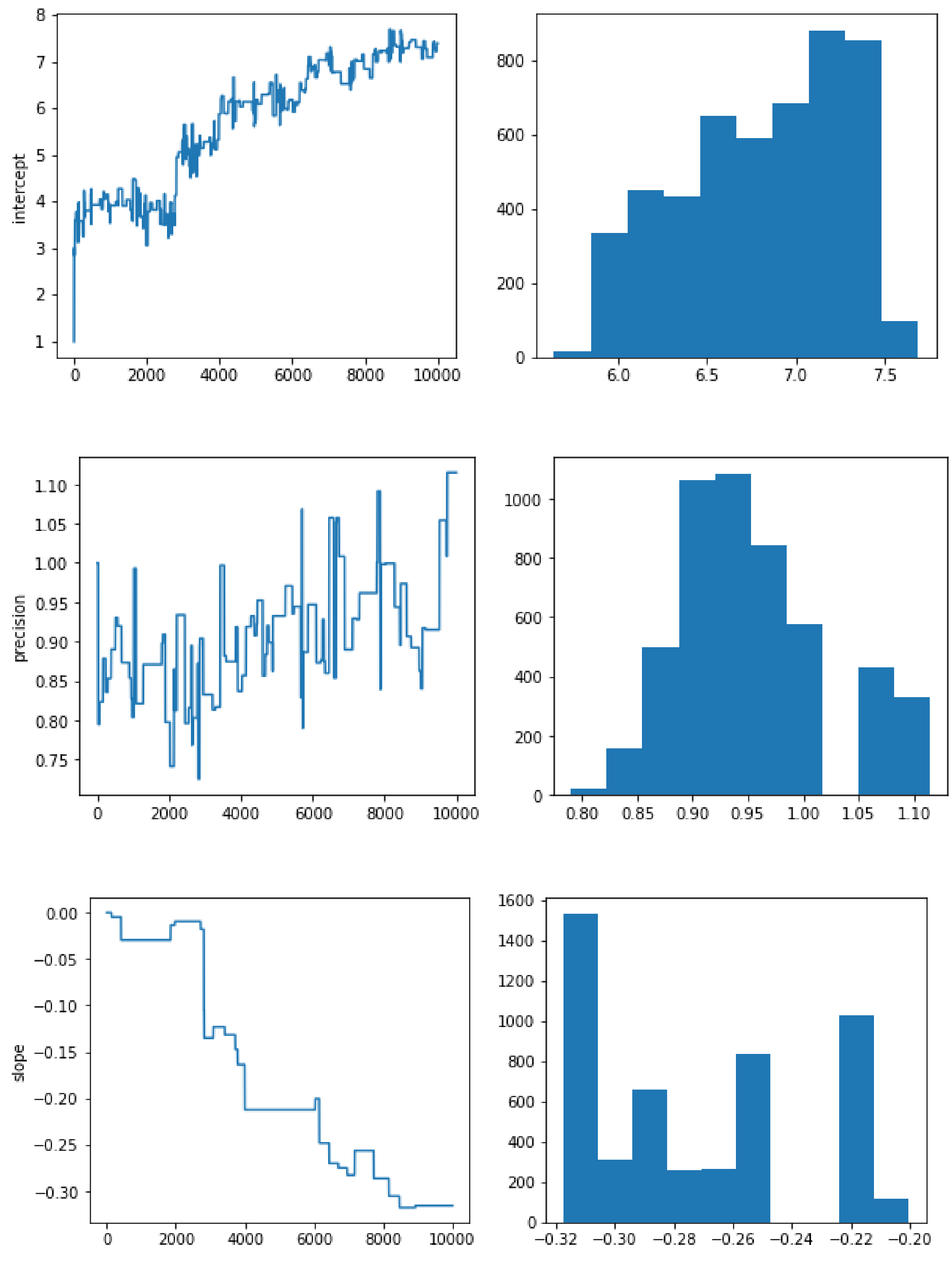
Initially we ran the algorithm with a very small proposal variance and observed that the acceptance rate is way too high (around 0.97).





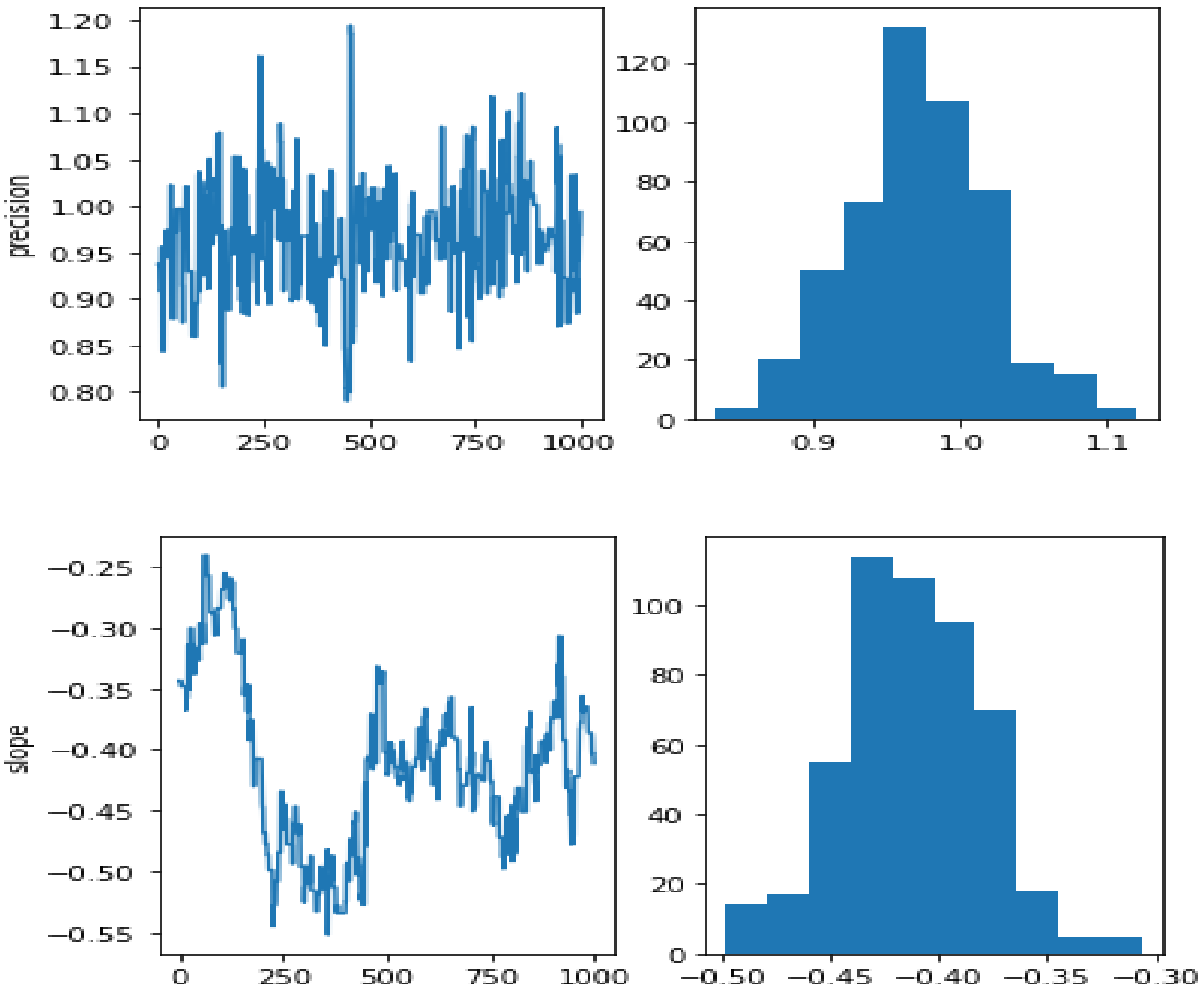


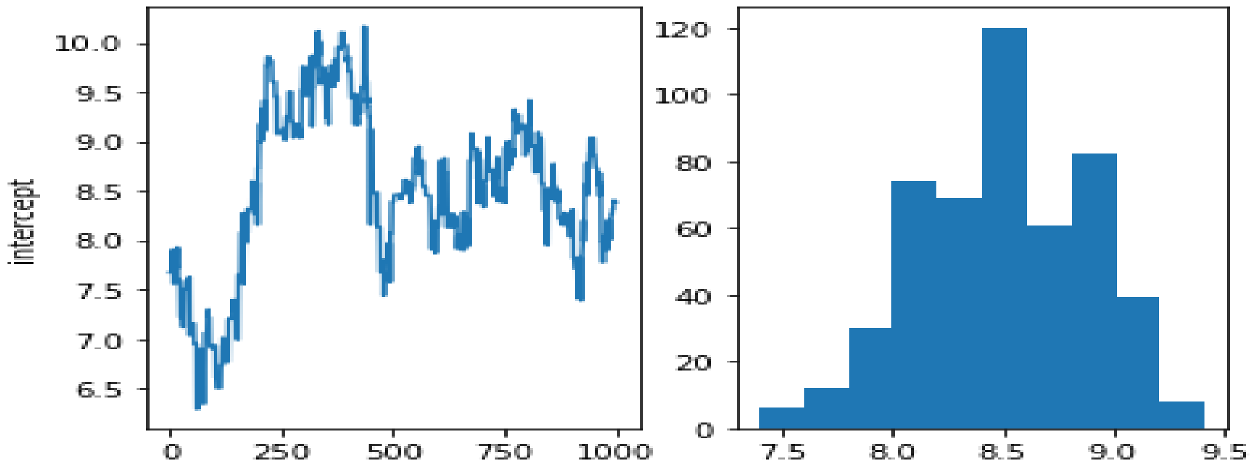
When we ran the algorithm with a very large proposal variance we observed that the acceptance rate is low (around 0.02).


intercept
precision
slope
0
2000
4000
6000
8000
10000

Adaptive Metropolis :

In order to avoid having to set the proposal variance by trial-and-error, we added some tuning logic to the algorithm. It reduced proposal variances by 10% when the acceptance rate is low, and increased it by 10% when the acceptance rate is high.





intercept
10.0
9.5
9.0
8.5
8.0
7.5
7.0
6.5
0
250
500
750
1000
120
100
80
60
40
20
0
7.5
8.0
8.5
9.0
9.5