Monte Carlo simulations and option pricing

by Bingqian Lu
Undergraduate Mathematics Department
Pennsylvania State University
University Park, PA 16802

Project Supervisor: Professor Anna Mazzucato

July, 2011

Abstract

Monte Carlo simulation is a legitimate and widely used technique for dealing with uncertainty in many aspects of business operations. The purpose of this report is to explore the application of this technique to the stock volality and to test its accuracy by comparing the result computed by *Monte Carlo Estimate* with the result of *Black-Schole model* and the Variance Reduction by *Antitheric Variattes*. The mathematical computer softwear application that we use to compute and test the relationship between the sample size and the accuracy of Monte Carlo Simulation is itshapeMathematica. It also provides numerical and geometrical evidence for our conclusion.

0.1 Introduction to Monte Carlo Simulaion

Monte Carlo Option Price is a method often used in Mathematical finance to calculate the value of an option with multiple sources of uncertainties and random features, such as changing interest rates, stock prices or exchange rates, etc.. This method is called Monte Carlo simulation, naming after the city of Monte Carlo, which is noted for its casinos. In my project, I use ***Mathematica***, a mathematics computer software, we can easily create a sequence of random number indicating the uncertainties that we might have for the stock prices for example.

0.2 Pricing Financial Options by Flipping a Coin

A distcrete model for change in price of a stock over a time interval [0,T] is

$$S_{n+1} = S_n + \mu S_n \Delta t + \sigma S_n \varepsilon_{n+1} \sqrt{\Delta t}, \qquad S_0 = s \tag{1}$$

where $S_n = S_{t_n}$ is the stock price at time $t_n = n\Delta t, n = 0, 1, ..., N-1, \Delta t = T/N$, μ is the annual growth rate of the stock, and σ is a measure of the stocks annual price volatility or tendency to fluctuate. Highly volatile stocks have large values of σ. Each term in sequence $\varepsilon_1, \varepsilon_2...$ takes on the value of 1 or -1 depending on the outcoming value of a coin tossing experiment, heads or tails respectively. In other words, for each n=1,2,...

$$\varepsilon_n = \begin{cases} 1 & \text{with probability} = 1/2 \\ -1 & \text{with probability} = 1/2 \end{cases} \tag{2}$$

By using *Mathematica*, it is very easy to create a sequence of random number. With this sequence, the equation (1) can then be used to simulate **a sample path** or **trajectory** of stock prices, $\{s, S_1, S_2, ..., S_N\}$. For our purpose here, it has been shown as a relatively accurate method of pricing options and very useful for options that depend on paths.

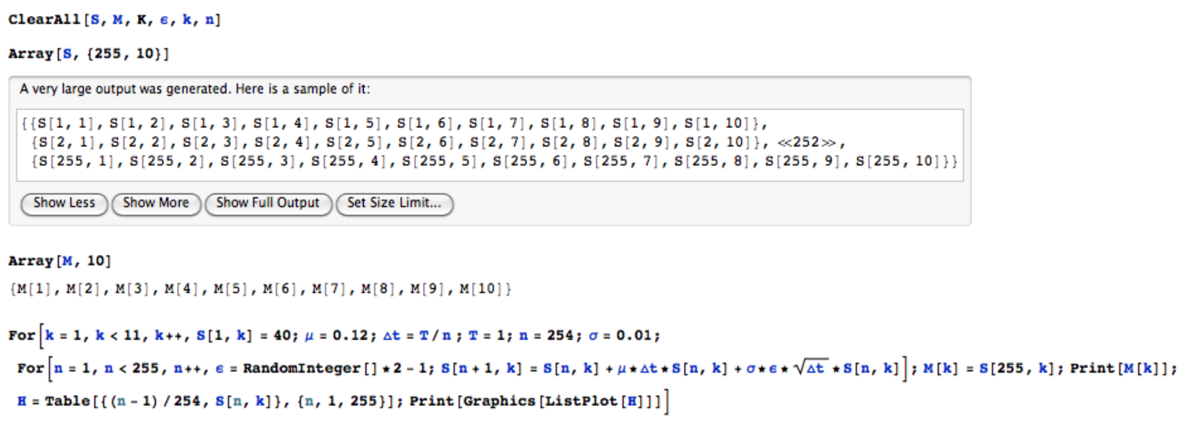
0.3 Proof of highly volatile stocks have large values for σ

0.3.1 $\sigma = 0.01$

Let us simulate several sample trajectories of (1) for the following parameter values and plot the trajectoris: $\mu = 0.12, \sigma = 0.01, T = 1, s = \$40, N = 254$.

The following figures are the graphs that we got for $\sigma = 0.01$ Since we have ε to be the value generated by flipping a coin, it gives us arbitrary values and thus, we have different graphs for parameter $\sigma = 0.01, \mu = 0.12, T = 1, s = \$40, N = 254$

This is another possibile graph.



```
ClearAll[S, M, K, ε, k, n]

Array[S, {255, 10}]
```

A very large output was generated. Here is a sample of it:

```
{{S[1, 1], S[1, 2], S[1, 3], S[1, 4], S[1, 5], S[1, 6], S[1, 7], S[1, 8], S[1, 9], S[1, 10]},
 {S[2, 1], S[2, 2], S[2, 3], S[2, 4], S[2, 5], S[2, 6], S[2, 7], S[2, 8], S[2, 9], S[2, 10]}, «252»,
 {S[255, 1], S[255, 2], S[255, 3], S[255, 4], S[255, 5], S[255, 6], S[255, 7], S[255, 8], S[255, 9], S[255, 10]}}
```

```
Array[M, 10]
{M[1], M[2], M[3], M[4], M[5], M[6], M[7], M[8], M[9], M[10]}

For[k = 1, k < 11, k++, S[1, k] = 40; μ = 0.12; Δt = T / n ; T = 1; n = 254; σ = 0.01;
 For[n = 1, n < 255, n++, ε = RandomInteger[] * 2 - 1; S[n + 1, k] = S[n, k] + μ * Δt * S[n, k] + σ * ε * √Δt * S[n, k]]; M[k] = S[255, k]; Print[M[k]];
 H = Table[{(n - 1) / 254, S[n, k]}, {n, 1, 255}]; Print[Graphics[ListPlot[H]]]]
```

Figure 1: Let M be the value of S_{254} of different trajectories, k is the number of trajectories

0.3.2 $\sigma = 0.7$

Then we repeated the experiment using the value of $\sigma = 0.7$ for the volality and other parameters remain the same.

Similarly, it should have a number of different graphs due to the arbitrary value of ε we generated by Mathematica. The following figures are the graphs that we got for $\sigma = 0.7$

Conclusion: From the two experiments above with the large different σ and constant other parameters, we can tell that the larger the σ, the greater degree of variability in their behavior forthe ε 's it is permissible to use random number generator that creates normally distributed random numbers with mean zero and variance one. Recall that the standard normal distribution has the bell-shape with a standard deviation of 1.0 and standard normal random variable has a mean of zero.

0.4 Monte Carlo Method vs. Black-Scholes Model

0.4.1 Monte Carlo Method and its computing

Monte Carlo Method

In the formular (1), the random terms $S_n\varepsilon_{n+1}\sqrt{\Delta t}$ on the right-hand side can be consider as shocks or distrubances that model functuations in the stock price. After repeatedly simulating stock price trajectories, as we did in the previous chapter, and computing appropriate averages, it is possible to obtain estimates of the price of a **European call option**, a type of

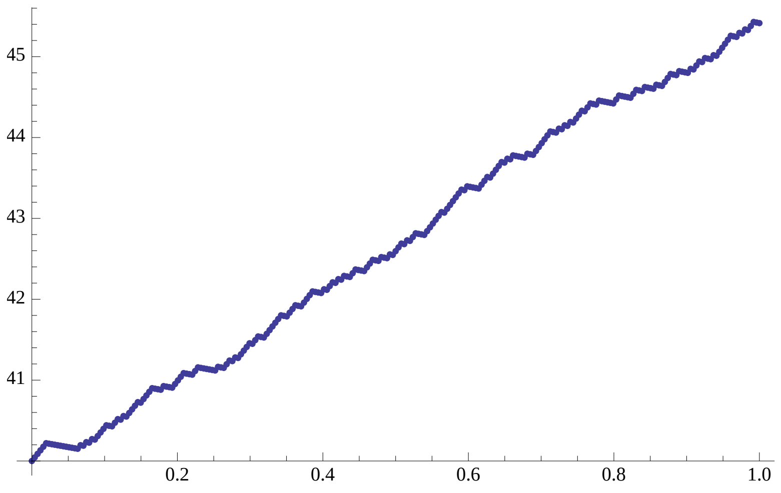


Figure 2: $S_{254} = 45.415$, $\sigma = 0.01$

financial derivative. A statistical simulation algorithm of this type is what we known as **"Monte Carlo method"**

A European call option is a contract between two parties, a holder and a writer, whereby, for a premium paid to the writer, the holder can purchase the stock at a future date T (the expiration date) at a price K (the strike price) agreed upon in the contract. If the buyer elect to exercise the option on the expiration date, the writer is obligated to sell the inderlying stock to the buyer at the price K, the strike price. Thus, the option has a payoff function

$$f(S) = max(S - K, 0) \tag{3}$$

where $S = S(T)$ is the price of the underlying stock at the time T when the option expires. This equation (3) produces one possible option value at expiration and after computing this thousands of times in order to obtain a feel for the possible error in estimating the price. Equation(3) is also known as the value of the option at time T since if $S(T) > K$, the holder can purchase, at price K, stock with market value S(T) and thereby make a profit equal to$S(T) - K$ not counting the option premium. However, on the other hand, if $S(T) < K$, the holder will simply let the option expire since there would be no reason to purchase stock at a price that exceeds the market value.

In other words, the option valuation problem is determine the correct and fair price of the option at the time that the holder and writer enter into the contract. In order to estimate the price call of a call option using a Monte Carlo method, an ensemble

$$\left\{ S_N^{(k)} = S^{(k)}(T), k = 1, ...M \right\} \tag{4}$$

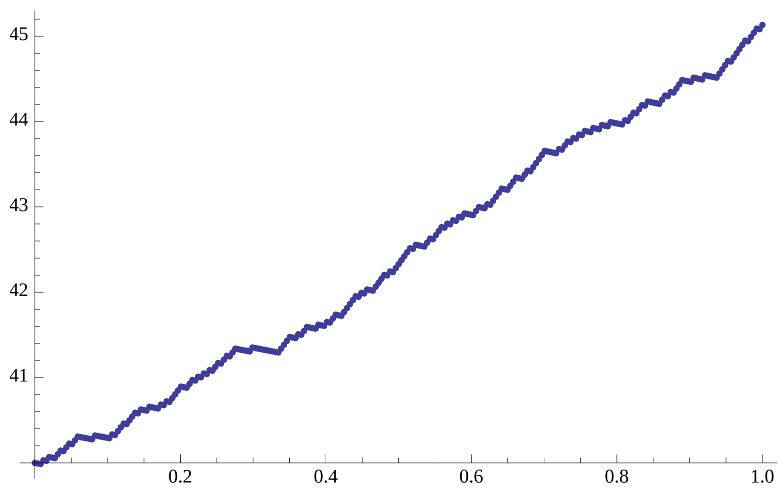


Figure 3: S_{254} =45.1316, $\sigma = 0.01$

of M stock orices at expiration is generated using the difference equation

$$S_{n+1}^{(k)} = S_n^{(k)} + rS_n^{(k)}\Delta t + \sigma S_n^{(k)}\varepsilon_{n+1}^{(k)}\sqrt{\Delta t}, \qquad S_0^{(k)} = s \tag{5}$$

Equation (5) is identical to equation (1) for each$k = 1, ..., M$, except the growth rate μ is replaces by the annual interest r that it costs the writer to borrow money. Option pricing theory requires that the average value of the payoffs $\left\{f(S_N^{(k)}0, k = 1, ..., M\right\}$ be equal to the compounded total return obtained by investing the option premium, $\hat{C}(s)$, at rate r over the life of option,

$$\frac{1}{M}\sum_{k=1}^{M} f(s_N^{(k)}) = (1 + r\Delta t)^N \hat{C}(s). \tag{6}$$

Solving(6) for $\hat{C}(s)$ yields the Monte Carlo estimate

$$\hat{C}(s) = (1 + r\Delta t)^{-N}\left\{\frac{1}{M}\sum_{k=1}^{M} f(s_N^{(k)})\right\} \tag{7}$$

for the option price. So, the Monte Carlo estimate$\hat{C}(s)$ is the present value of the average of the payoffs computed using rules of compound interest.

0.4.2 Computing Monte Carlo Estimate

We use equation (7) to compute a Monte Carlo estimate of the value of a five month call option, in other word $T = \frac{5}{12}$ years, for the following parameter values: $r = 0.06, \sigma = 0.2, N = 254, andK = \$50$. N is the number of times of steps for each trajectories.

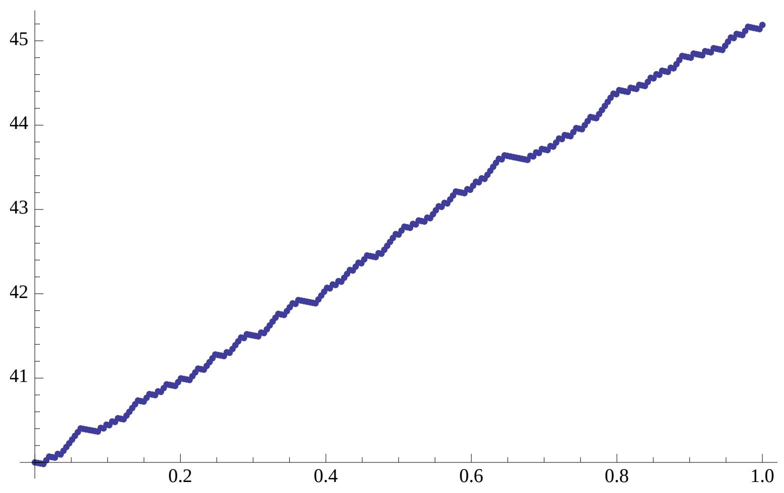


Figure 4: S_{254} =45.1881, $\sigma = 0.01$

0.5 Comparing to the Exact Black-Scholes Formular

Monte Carlo has been used to price standard European options, but as we known that Black-Scholes model is the correct method of pricing these options, so it is not necessary to use Monte Carlo simulation.

Here is the formular for exact Black-Scholes model:

$$C(s) = \frac{s}{2} erfc(\frac{-d_1}{\sqrt{2}}) - \frac{K}{2} e^{-r_T} erfc(frac{-}d_2\sqrt{2}) \tag{8}$$

where

$$d_1 = \frac{1}{\sigma\sqrt{T}}[ln(\frac{s}{k}) + (r + \frac{\sigma^2}{2})T], d_2 = d_1 - \sigma\sqrt{T} \tag{9}$$

and erfc(x) is the complementary error function,

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt \tag{10}$$

Now we insert all data we have to the Black-Schole formula to check the accuracy of our results by comparing the Monte Carlo approximation with the value computed from exact Black-Schole formula. We generated Black-Scholes Model with parameter$r = 0.06, \sigma = 0.2, K = \$50, k = 1, ..., M(whereT = N\Delta t), N = 200$. And we got: $C(40) = 1.01189, C(45) = 2.71716 \, and \, C(50) = 5.49477$. The error of the Monte Carlo Estimate seems to be very large. Thus we repeat the previous procedure and increased our sample size (M) to 50,000 and 100,000. Then, I made a chart to check if the accuracy of Monte Carlo Simulation increases by the increasing of the sample size.

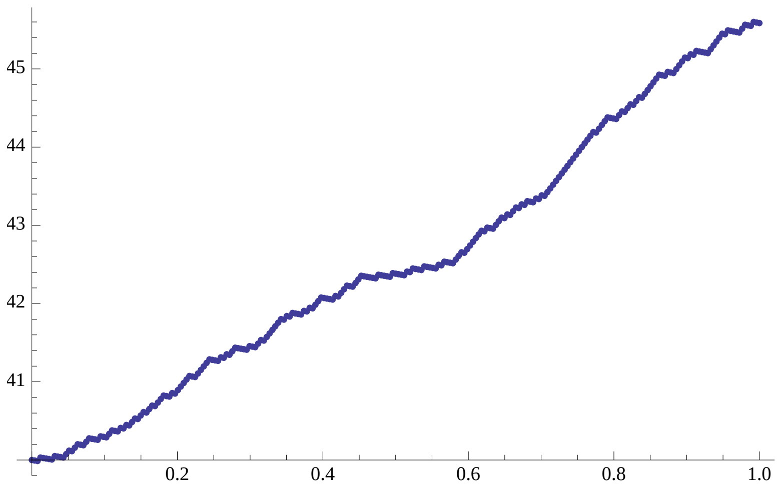


Figure 5: S_{254} =45.5859, $\sigma = 0.01$

Comparison of the accuracy of the Monte Carlo Estimate to the Black-Schole Model

MCE(1,000)	MCE (10,000)	MCE (50,000)	MCE (100,000)	Black-Schole Model
$\hat{C}(40) = 1.67263$	$\hat{C}(40) = 1.61767$	$\hat{C}(40) = 1.76289$	$\hat{C}(40) = 1.7496$	$BS(40) = 1.71179$
$\hat{C}(45) = 4.6343$	$\hat{C}(45) = 4.24609$	$\hat{C}(45) = 4.3014$	$\hat{C}(45) = 4.2097$	$BS(45) = 4.11716$
$\hat{C}(50) = 8.1409$	$\hat{C}(50) = 7.92441$	$\hat{C}(50) = 7.7889$	$\hat{C}(50) = 7.6864$	$BS(50) = 7.49477$

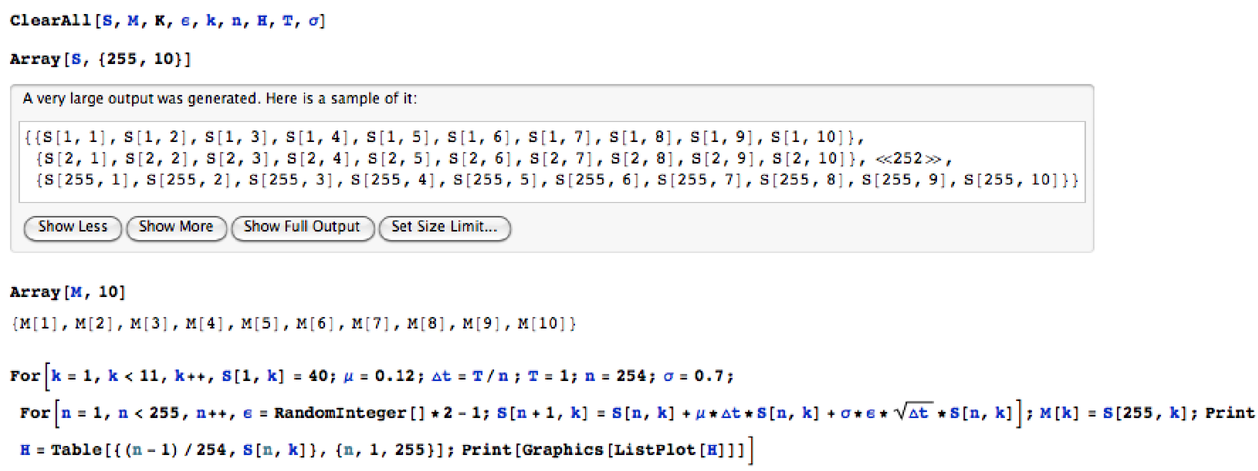
By the comparing Monte Carlo Estimate and the Black-Schole modle, we can tell quite easily that the error gets smaller as the sample size increases. Here we calculated the relative error by the equation

$$\frac{MonteCarloEstimate - Black - Scholemodle}{Black - ScholeModle} \tag{11}$$

The results are

error(1,000)	error (10,000)	error(50,000)	error (100,000)	Black-Schole Model
$E(40) = 0.60593$	$E(40) = 0.30232$	$E(40) = 0.14037$	$E(40) = 0.14037$	$BS(40) = 1.71179$
$E(45) = 0.32330$	$E(45) = 0.19466$	$E(45) = 0.13862$	$E(45) = 0.08833$	$BS(45) = 4.11716$
$E(50) = 0.24272$	$E(50) = 0.07819$	$E(50) = 0.05254$	$E(50) = 0.02157$	$BS(50) = 7.49477$

Conclusion: Monte Carlo Simulation gives the option price is a sample average, thus according to the most elementary principle of statistics, its standard deviation is the standard deviation of the sample divided by the square root of the sample size. So, the error reduces at the rate of 1 over the square root of the sample size. To sum up, the accuracy of Monte Carlo Simulation is increasing by increasing the size of the sample.



```
ClearAll[S, M, K, ε, k, n, H, T, σ]

Array[S, {255, 10}]
```

A very large output was generated. Here is a sample of it:

```
{{S[1, 1], S[1, 2], S[1, 3], S[1, 4], S[1, 5], S[1, 6], S[1, 7], S[1, 8], S[1, 9], S[1, 10]},
 {S[2, 1], S[2, 2], S[2, 3], S[2, 4], S[2, 5], S[2, 6], S[2, 7], S[2, 8], S[2, 9], S[2, 10]}, <<252>>,
 {S[255, 1], S[255, 2], S[255, 3], S[255, 4], S[255, 5], S[255, 6], S[255, 7], S[255, 8], S[255, 9], S[255, 10]}}
```

```
Array[M, 10]
{M[1], M[2], M[3], M[4], M[5], M[6], M[7], M[8], M[9], M[10]}

For[k = 1, k < 11, k++, S[1, k] = 40; μ = 0.12; Δt = T / n ; T = 1; n = 254; σ = 0.7;
 For[n = 1, n < 255, n++, ε = RandomInteger[] * 2 - 1; S[n + 1, k] = S[n, k] + μ * Δt * S[n, k] + σ * ε * √Δt * S[n, k]]; M[k] = S[255, k]; Print
 H = Table[{(n - 1) / 254, S[n, k]}, {n, 1, 255}]; Print[Graphics[ListPlot[H]]]]
```

Figure 6: replaced previous value of sigma with 0.7

0.6 Comparing to the Variance Reduction by Antitheric Variattes

Variance Reduction by Antithetic Variates is a simple and more widely used way to increase the accuracy of the Monte Carlo Simulation. It is the technique used in some certain situations with an additional increase in computational complexity is the method of antithetic variates. In order to achieve greater accuracy, one method of doing so is simple and automatically doubles the sample size with only a minimum increase in computational time. This is called the *antithetic variate* method. Because we are generating obervations of a standard normal random variable which is distributed with a mean of zero, a variance of 1.0 and symmetric, there is an equally likely chance of having drawn the observed value times-1. Thus, for each arbitrary ε we draw, there should be an artificially observed companion observation of $-\varepsilon$ that can be legitimately created by us. This is the antithetic variate.

For each k=1,...,M use the sequence

$$\left\{\varepsilon_1^{(k)},...,\varepsilon_{N-1}^{(k)}\right\} \tag{12}$$

in equation (5) to simulate a payoff $f(S_N^{k+1})$and also use the sequence $\left\{-\varepsilon_1^{(k)},...,-\varepsilon_{N-1}^{(k)}\right\}$ in equation (5) to simulate an associated payoff $f(S_N^{k-})$. Now the payoffs are simulated inpairs $\left\{f(S_N^{k+}),f(S_N^{k-})\right\}$ This is the Mathematica Program that we ran to evaluate the Variance Reduction.

After comparing Monte Carlo Simulation with Variance Reduction by Antithetic Variates, We made a table of data.

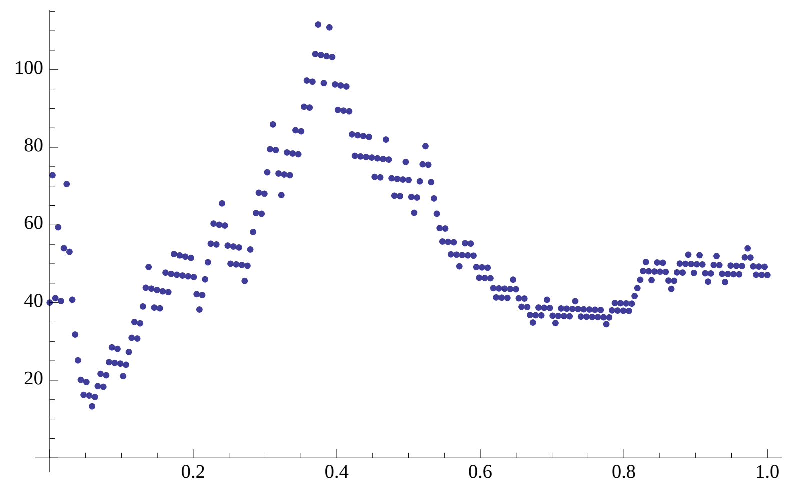


Figure 7: S_{254} =47.084, $\sigma = 0.7$

k(1,000)	k(5,000)	k(50,000)	k (100,000)	BS Model
$V(40) = 1.74309$	$V(40) = 1.66685$	$V(40) = 1.69966$	$V(40) = 1.70674$	$BS(40) = 1.71179$
$V(45) = 4.4789$	$V(45) = 4.305103$	$V(45) = 4.1778$	$V(45) = 4.099835$	$BS(45) = 4.11716$
$V(50) = 7.94642$	$V(50) = 7.67707$	$V(50) = 7.62341$	$V(50) = 7.53716$	$BS(50) = 7.49477$

While the table for the data of Monte Carlo Simulation we get under the same condition is:

MCE(1,000)	MCE (5,000)	MCE (50,000)	MCE (100,000)
$\hat{C}(40) = 1.67263$	$\hat{C}(40) = 1.72486$	$\hat{C}(40) = 1.76289$	$\hat{C}(40) = 1.7496$
$\hat{C}(45) = 4.6343$	$\hat{C}(45) = 4.34609$	$\hat{C}(45) = 4.3014$	$\hat{C}(45) = 4.2097$
$\hat{C}(50) = 8.1409$	$\hat{C}(50) = 7.92241$	$\hat{C}(50) = 7.7889$	$\hat{C}(50) = 7.6864$

By the chart of Monte Carlo Estimate and the Variance Reduction by Antithetic Variates, we can tell quite easily that the error gets smaller as the sample size increases. Here we calculated the relative error by the equation

$$\frac{MonteCarloEstimate - VarianceReductionbyAntitheticVariates}{VarianceReductionbyAntitheticVariates} \tag{13}$$

The results are

error(1,000)	error (50,000)	error (100,000)
$E(40) = 0.0402$	$E(40) = 0.0372$	$E(40) = 0.0251$
$E(45) = 0.0347$	$E(45) = 0.0296$	$E(45) = 0.0268$
$E(50) = 0.0245$	$E(50) = 0.0217$	$E(50) = 0.0198$

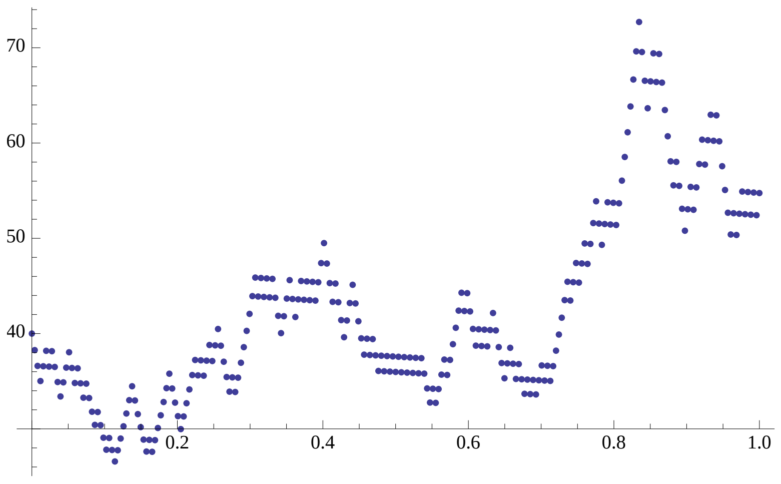


Figure 8: S_{254} =54.7495, $\sigma = 0.7$

0.7 Random Walk

Random Walk

$$d_s = \mu S d_t + \sigma S^{\beta} d_w t \tag{14}$$

this is the random walk, where $d_s is\ S(t_k+\Delta t)-S(t_k)$, so this can be written as

$$S(t_k + \Delta t) - S(t_k) = \mu S(t_k)\Delta t + \sigma S(t_k)^{\beta}\varepsilon_k\sqrt{\Delta t} \tag{15}$$

We randomly picked two different positive numbers, one greater than 1 and one less than one. β as 0.5 and 2 and we made mathematica ran twice while keep every other variables constant. Here are some graphs that we plot using Mathematica.

Figure 19 is a graph for $\beta = 0.5$

In addition, we plot the graphs when $\beta = 2$ as well. During Mathematica's computation, we got some values that are overflowing. Within the values that are available, we plotted some corresponding graphs.

Figure 20 is a graph for the random walk when $\beta = 2$.

By using 1000 as a sample size, we used the same program to compute $\hat{C}_M(10)$, $\hat{C}_M(20)$, $\hat{C}_M(40)$ and $\hat{C}_M(160)$ when $\beta = 0.5$, where $\hat{C}_M(10)$ is the Monte Carlo estimate for sample size 1000, $\hat{C}_M(20)$ is the Monte Carlo simulation for sample size 2000 ans so forth. We get $\hat{C}_M(10) = 3.80332$, $\hat{C}_M(20) = 4.75116$, $\hat{C}_M(40) = 6.09321$ and $\hat{C}_M(160) = 7.63962$. Now we are using the equation

$$\log_2(\frac{\hat{C}_M(10) - \hat{C}_M(20)}{\hat{C}_M(40) - \hat{C}_M(160)}) \tag{16}$$

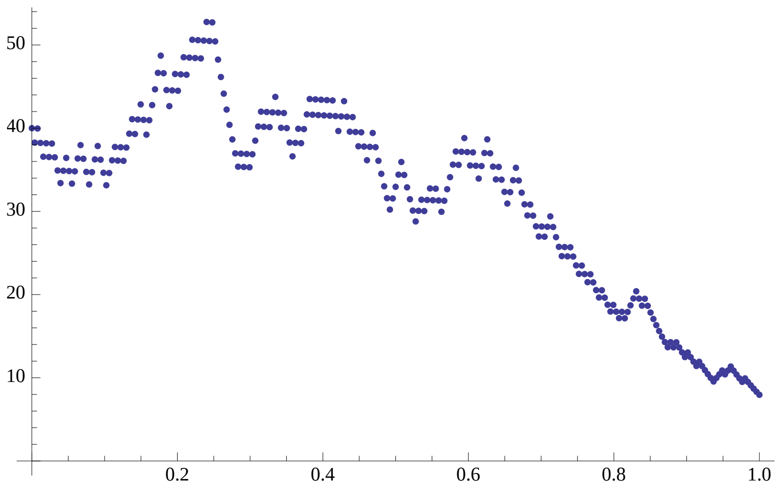


Figure 9: S_{254} =7.95396, $\sigma = 0.7$

to see the error. After plugging the numbers, we get this value of the equation to be -0.67621. This value is very close to $\frac{1}{2}$ as desired.
By the same procedure, we used Mathematica to generate the value for $\hat{C}_M(10)$, $\hat{C}_M(20)$, $\hat{C}_M(40)$ and $\hat{C}_M(160)$ when $\beta = 2$. There are some overflow in the result which gives us trouble for going further and also tells us that when $\beta > 1$, there will be overflow.

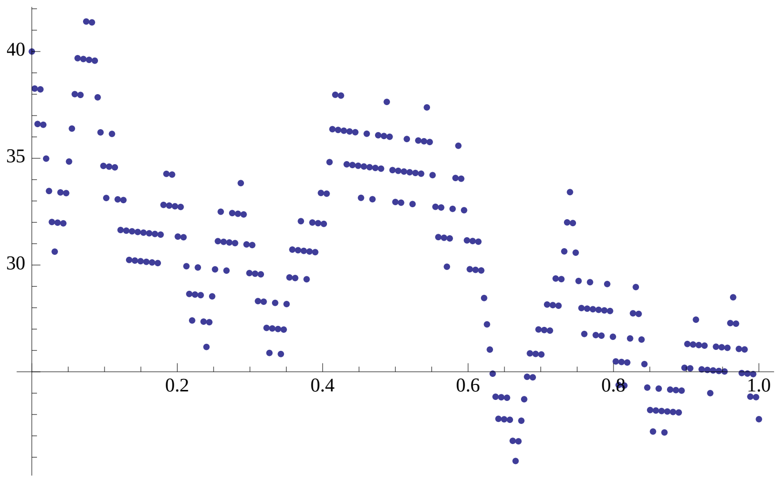
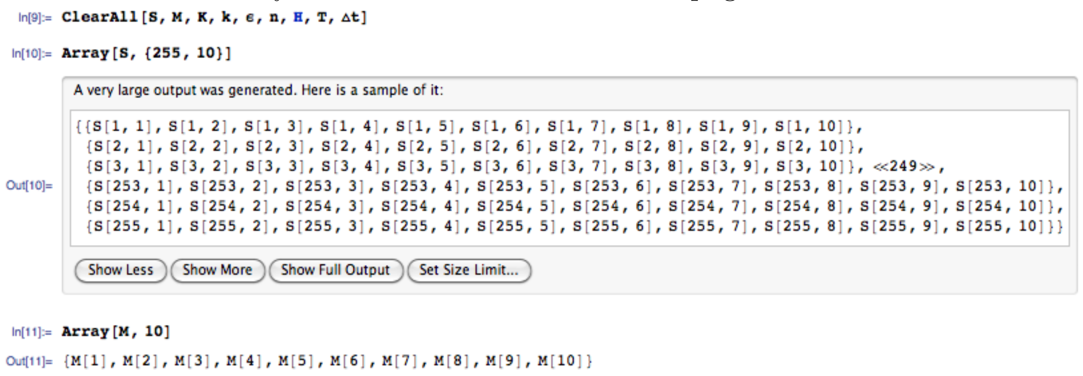


Figure 10: S_{254} =22.7805, σ = 0.7

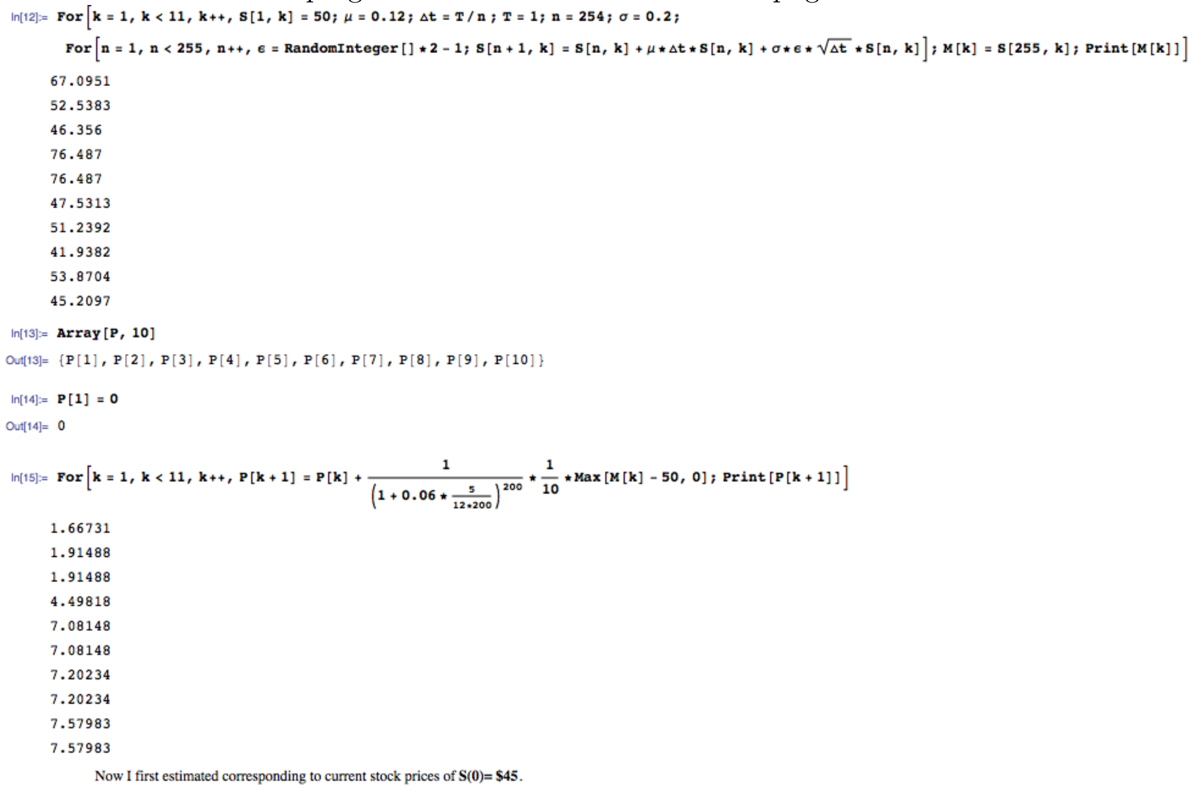
Array of S for monte carlo estimate1.png



```
In[9]:= ClearAll[S, M, K, k, ε, n, H, T, Δt]

In[10]:= Array[S, {255, 10}]

A very large output was generated. Here is a sample of it:

Out[10]= {{S[1, 1], S[1, 2], S[1, 3], S[1, 4], S[1, 5], S[1, 6], S[1, 7], S[1, 8], S[1, 9], S[1, 10]},
  {S[2, 1], S[2, 2], S[2, 3], S[2, 4], S[2, 5], S[2, 6], S[2, 7], S[2, 8], S[2, 9], S[2, 10]},
  {S[3, 1], S[3, 2], S[3, 3], S[3, 4], S[3, 5], S[3, 6], S[3, 7], S[3, 8], S[3, 9], S[3, 10]}, ≪249≫,
  {S[253, 1], S[253, 2], S[253, 3], S[253, 4], S[253, 5], S[253, 6], S[253, 7], S[253, 8], S[253, 9], S[253, 10]},
  {S[254, 1], S[254, 2], S[254, 3], S[254, 4], S[254, 5], S[254, 6], S[254, 7], S[254, 8], S[254, 9], S[254, 10]},
  {S[255, 1], S[255, 2], S[255, 3], S[255, 4], S[255, 5], S[255, 6], S[255, 7], S[255, 8], S[255, 9], S[255, 10]}}

In[11]:= Array[M, 10]

Out[11]= {M[1], M[2], M[3], M[4], M[5], M[6], M[7], M[8], M[9], M[10]}
```

Figure 11: We first array for S in order to make space of memory for the data we will get later. Here, P(k) is the$\hat{C}(k)$

Mathematica program for Monte Carlo Estimate1.png

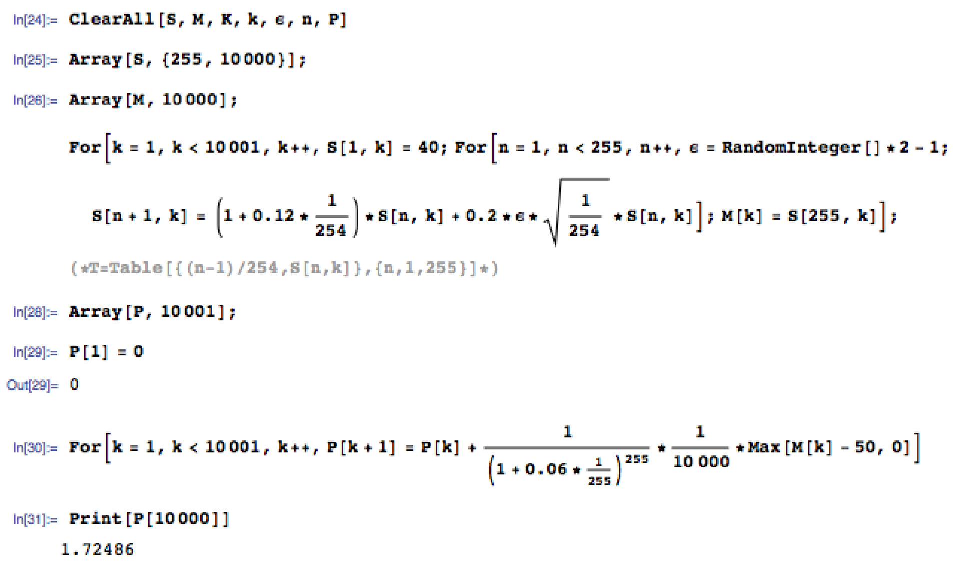


```
In[12]:= For[k = 1, k < 11, k++, S[1, k] = 50; μ = 0.12; Δt = T / n ; T = 1; n = 254; σ = 0.2;
          For[n = 1, n < 255, n++, ϵ = RandomInteger[] * 2 - 1; S[n + 1, k] = S[n, k] + μ * Δt * S[n, k] + σ * ϵ * √Δt * S[n, k]]; M[k] = S[255, k]; Print[M[k]]]
         67.0951
         52.5383
         46.356
         76.487
         76.487
         47.5313
         51.2392
         41.9382
         53.8704
         45.2097

In[13]:= Array[P, 10]
Out[13]= {P[1], P[2], P[3], P[4], P[5], P[6], P[7], P[8], P[9], P[10]}

In[14]:= P[1] = 0
Out[14]= 0

In[15]:= For[k = 1, k < 11, k++, P[k + 1] = P[k] + 1/(1 + 0.06 * 5/(12*200))^200 * 1/10 * Max[M[k] - 50, 0]; Print[P[k + 1]]]
         1.66731
         1.91488
         1.91488
         4.49818
         7.08148
         7.08148
         7.20234
         7.20234
         7.57983
         7.57983
```

Now I first estimated corresponding to current stock prices of **S(0)= $45**.

Figure 12: This is the Matehamatica program we made and data we got for the different S(254), after 10 times of computing, named as M[k]=S[255, k] in mathematica. Again, here P(k) is the $\hat{C}(k)$. So, according to the data we got, there are 10 different P[k+1], which in other word, 10 different Monte Carlo estimate $\hat{C}(k+1)$ after 10 times of computing

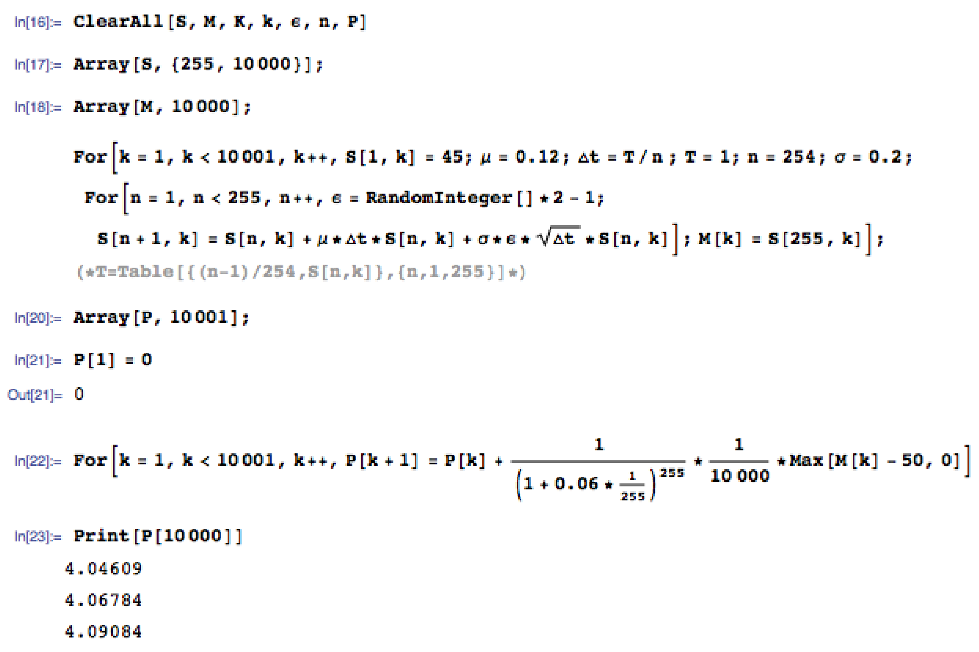
S(0)=40.png



```
In[24]:= ClearAll[S, M, K, k, ϵ, n, P]

In[25]:= Array[S, {255, 10000}];

In[26]:= Array[M, 10000];

         For[k = 1, k < 10001, k++, S[1, k] = 40; For[n = 1, n < 255, n++, ϵ = RandomInteger[] * 2 - 1;

           S[n + 1, k] = (1 + 0.12 * 1/254) * S[n, k] + 0.2 * ϵ * Sqrt[1/254] * S[n, k]]; M[k] = S[255, k]];

         (*T=Table[{(n-1)/254,S[n,k]},{n,1,255}]*)

In[28]:= Array[P, 10001];

In[29]:= P[1] = 0

Out[29]= 0

In[30]:= For[k = 1, k < 10001, k++, P[k + 1] = P[k] + 1/(1 + 0.06 * 1/255)^255 * 1/10000 * Max[M[k] - 50, 0]]

In[31]:= Print[P[10000]]
        1.72486
```

Figure 13: This result is the Monte Carlo estimate corresponding to our current stock prices of $S(0) = s = 40$ after using 254 steps (the number of N) and $M \cong 10,000$ for each trajectories for each Monte Carlo estimate.

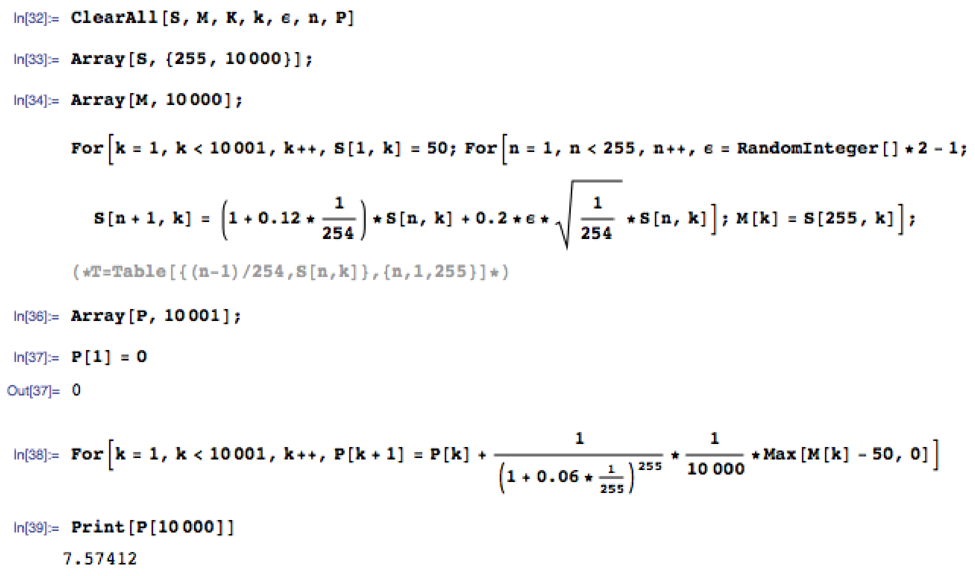
S(0)=45.png



```
In[16]:= ClearAll[S, M, K, k, ϵ, n, P]

In[17]:= Array[S, {255, 10000}];

In[18]:= Array[M, 10000];

         For[k = 1, k < 10001, k++, S[1, k] = 45; μ = 0.12; Δt = T / n ; T = 1; n = 254; σ = 0.2;
          For[n = 1, n < 255, n++, ϵ = RandomInteger[] * 2 - 1;
           S[n + 1, k] = S[n, k] + μ * Δt * S[n, k] + σ * ϵ * √Δt * S[n, k]]; M[k] = S[255, k]];
         (*T=Table[{(n-1)/254,S[n,k]},{n,1,255}]*)

In[20]:= Array[P, 10001];

In[21]:= P[1] = 0

Out[21]= 0

In[22]:= For[k = 1, k < 10001, k++, P[k + 1] = P[k] + 1/(1 + 0.06 * 1/255)^255 * 1/10000 * Max[M[k] - 50, 0]]

In[23]:= Print[P[10000]]
        4.04609
        4.06784
        4.09084
```

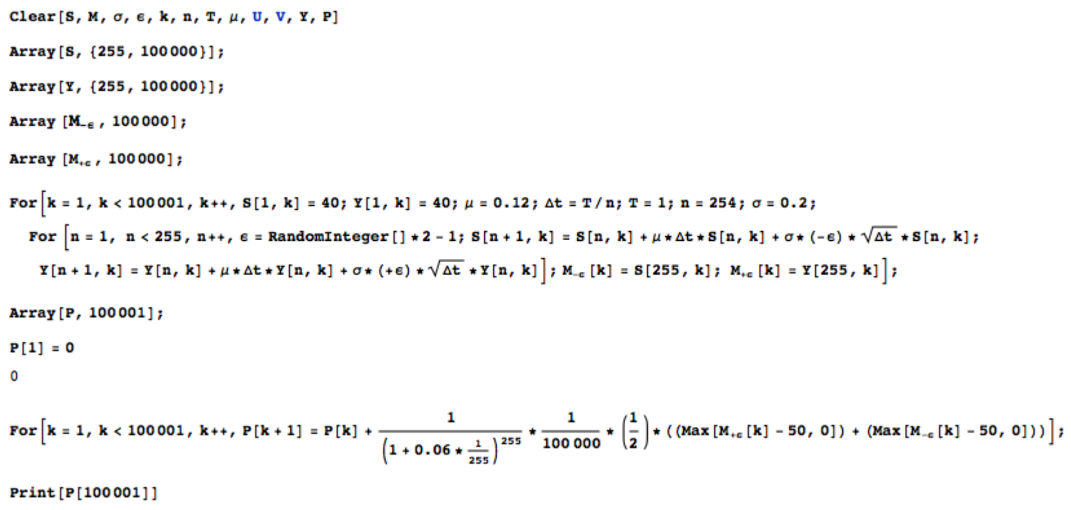
Figure 14: This result is the Monte Carlo estimate corresponding to our current stock prices of $S(0) = s = 45$ after using 254 steps (the number of N) and $M \cong 10,000$ for each trajectories for each Monte Carlo estimate.

S(0)=50.png



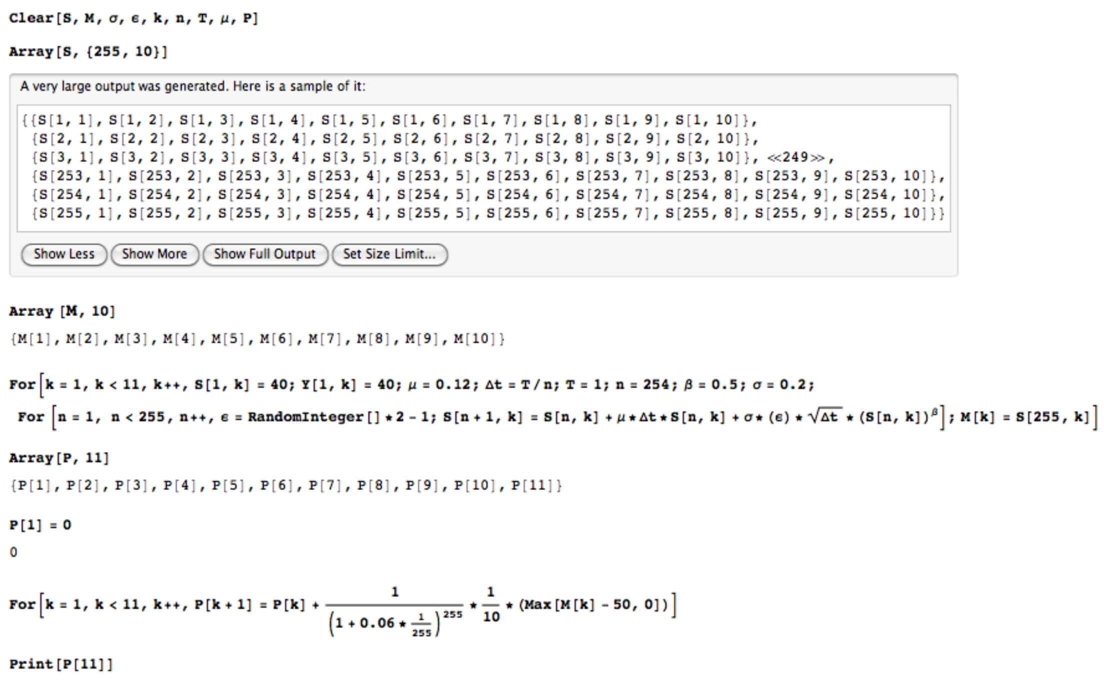
```
In[32]:= ClearAll[S, M, K, k, ε, n, P]

In[33]:= Array[S, {255, 10000}];

In[34]:= Array[M, 10000];

         For[k = 1, k < 10001, k++, S[1, k] = 50; For[n = 1, n < 255, n++, ε = RandomInteger[] * 2 - 1;

            S[n + 1, k] = (1 + 0.12 * 1/254) * S[n, k] + 0.2 * ε * Sqrt[1/254] * S[n, k]]; M[k] = S[255, k]];

         (*T=Table[{(n-1)/254,S[n,k]},{n,1,255}]*)

In[36]:= Array[P, 10001];

In[37]:= P[1] = 0

Out[37]= 0

In[38]:= For[k = 1, k < 10001, k++, P[k + 1] = P[k] + 1/(1 + 0.06 * 1/255)^255 * 1/10000 * Max[M[k] - 50, 0]]

In[39]:= Print[P[10000]]
        7.57412
```

Figure 15: This result is the Monte Carlo estimate corresponding to our current stock prices of $S(0) = s = 50$ after using 254 steps (the number of N) and $M \cong 10,000$ for each trajectories for each Monte Carlo estimate.



```
Clear[S, M, σ, ε, k, n, T, μ, U, V, Y, P]

Array[S, {255, 100000}];

Array[Y, {255, 100000}];

Array [M-ε , 100000];

Array [M+ε , 100000];

For[k = 1, k < 100001, k++, S[1, k] = 40; Y[1, k] = 40; μ = 0.12; Δt = T / n; T = 1; n = 254; σ = 0.2;
  For [n = 1,  n < 255, n++, ε = RandomInteger[] * 2 - 1; S[n + 1, k] = S[n, k] + μ * Δt * S[n, k] + σ * (-ε) * Sqrt[Δt] * S[n, k];
   Y[n + 1, k] = Y[n, k] + μ * Δt * Y[n, k] + σ * (+ε) * Sqrt[Δt] * Y[n, k]]; M-ε [k] = S[255, k];  M+ε [k] = Y[255, k]];

Array[P, 100001];

P[1] = 0

0

For[k = 1, k < 100001, k++, P[k + 1] = P[k] + 1/(1 + 0.06 * 1/255)^255 * 1/100000 * (1/2) * ((Max[M+ε [k] - 50, 0]) + (Max[M-ε [k] - 50, 0]))];

Print[P[100001]]
```

Figure 16: This the program we wrote with Mathematica for the Variance Reduction by Antithetic Variattes. We have S for the positive ε and Y for the negative ε and we get the average of these value by adding them up and divide by 2.



```
Clear[S, M, σ, ε, k, n, T, μ, P]

Array[S, {255, 10}]

A very large output was generated. Here is a sample of it:

{{S[1, 1], S[1, 2], S[1, 3], S[1, 4], S[1, 5], S[1, 6], S[1, 7], S[1, 8], S[1, 9], S[1, 10]},
 {S[2, 1], S[2, 2], S[2, 3], S[2, 4], S[2, 5], S[2, 6], S[2, 7], S[2, 8], S[2, 9], S[2, 10]},
 {S[3, 1], S[3, 2], S[3, 3], S[3, 4], S[3, 5], S[3, 6], S[3, 7], S[3, 8], S[3, 9], S[3, 10]}, <<249>>,
 {S[253, 1], S[253, 2], S[253, 3], S[253, 4], S[253, 5], S[253, 6], S[253, 7], S[253, 8], S[253, 9], S[253, 10]},
 {S[254, 1], S[254, 2], S[254, 3], S[254, 4], S[254, 5], S[254, 6], S[254, 7], S[254, 8], S[254, 9], S[254, 10]},
 {S[255, 1], S[255, 2], S[255, 3], S[255, 4], S[255, 5], S[255, 6], S[255, 7], S[255, 8], S[255, 9], S[255, 10]}}

Array [M, 10]
{M[1], M[2], M[3], M[4], M[5], M[6], M[7], M[8], M[9], M[10]}

For[k = 1, k < 11, k++, S[1, k] = 40; Y[1, k] = 40; μ = 0.12; Δt = T / n; T = 1; n = 254; β = 0.5; σ = 0.2;
 For [n = 1, n < 255, n++, ε = RandomInteger[] * 2 - 1; S[n + 1, k] = S[n, k] + μ * Δt * S[n, k] + σ * (ε) * √Δt * (S[n, k])^β]; M[k] = S[255, k]]

Array[P, 11]
{P[1], P[2], P[3], P[4], P[5], P[6], P[7], P[8], P[9], P[10], P[11]}

P[1] = 0
0

For[k = 1, k < 11, k++, P[k + 1] = P[k] + 1/(1 + 0.06 * 1/255)^255 * 1/10 * (Max[M[k] - 50, 0])]

Print[P[11]]
```

Figure 17: This the program we wrote with Mathematica for different values of β.

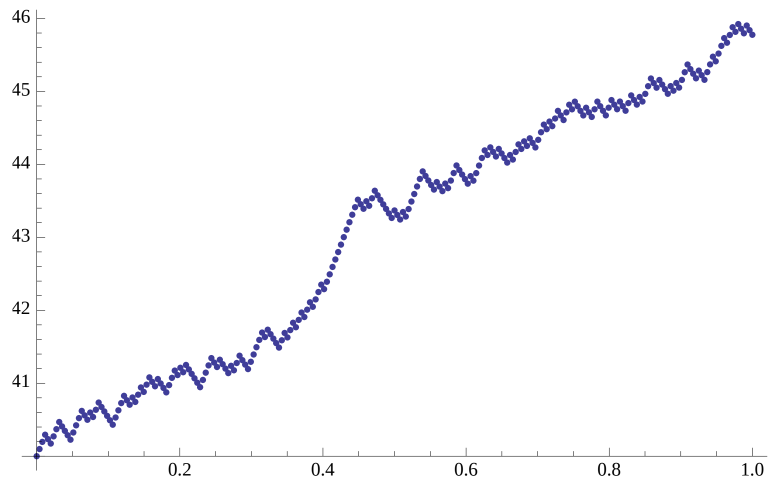


Figure 18: This is the graph when $\beta = 0.5$

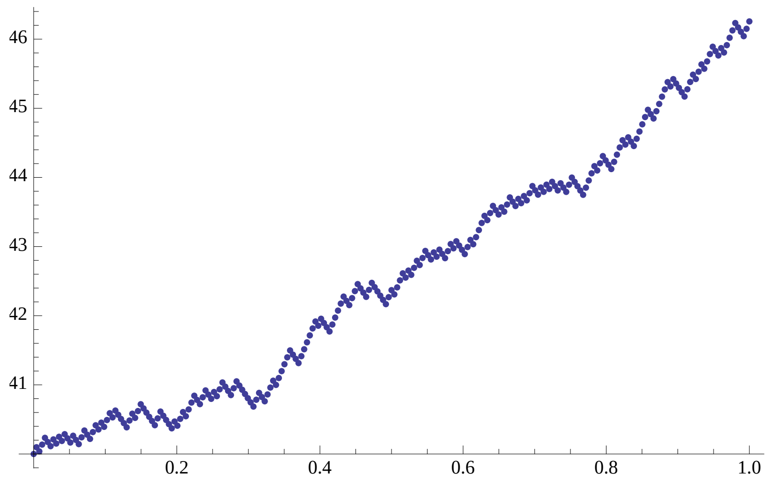


Figure 19: This is how the graph looks like when $\beta = 0.5$

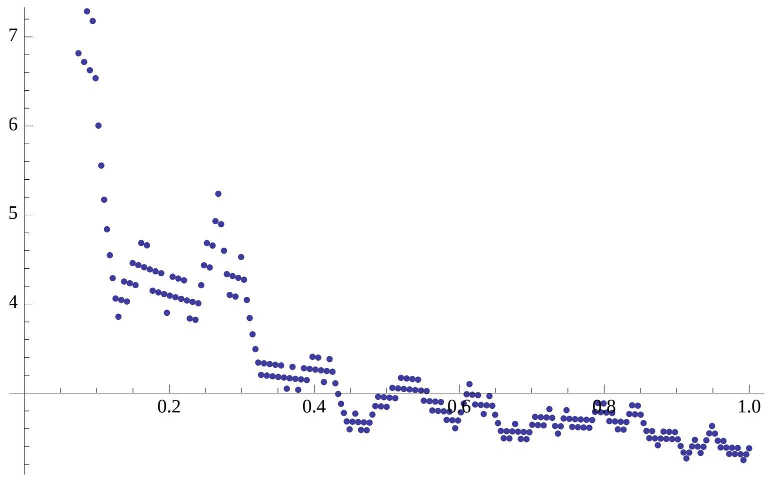


Figure 20: This is the graph when $\beta = 2$

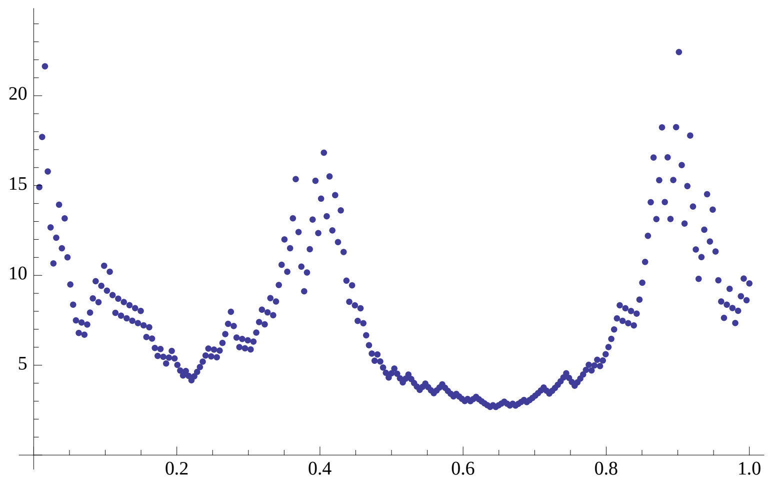


Figure 21: This is how the graph looks when $\beta = 2$