AE567 Project 1 - Akshat Dubey

2 Warmup: life without a CLT

Graphing the Pareto distribution function, its CDF and its inverse

The given probability density function is:

$$p_X(x) = \begin{cases} \frac{\alpha}{x^{\alpha+1}} & \text{if } x \geq 1 \\ 0 & \text{if } x < 1 \end{cases} .$$

where $\alpha = 3/2 = 1.5$.

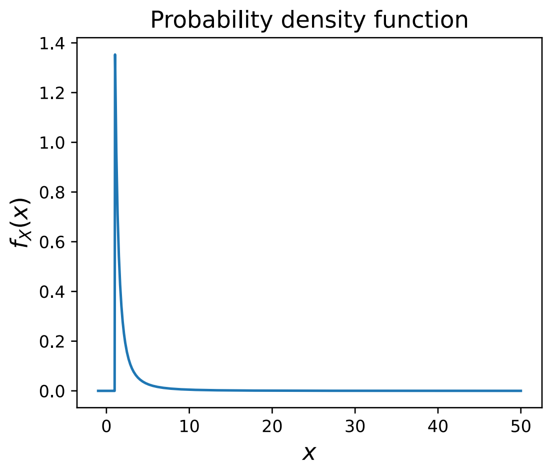

Figure 1: Probability density function

The aim is to find the Expectation of this function via Monte Carlo, but we have no way to sample it. To sample it, we will need to generate its inverse CDF. The first step to getting there is to get its CDF. We can integrate this function from 1 to x to get the cumulative distribution function for $x \geq 1$, and any $x < 1$ will be 0. The CDF is:

$$F_X(x) = \begin{cases} 1 - \frac{1}{x^{\alpha}} & \text{if } x \geq 1 \\ 0 & \text{if } x < 1 \end{cases} .$$

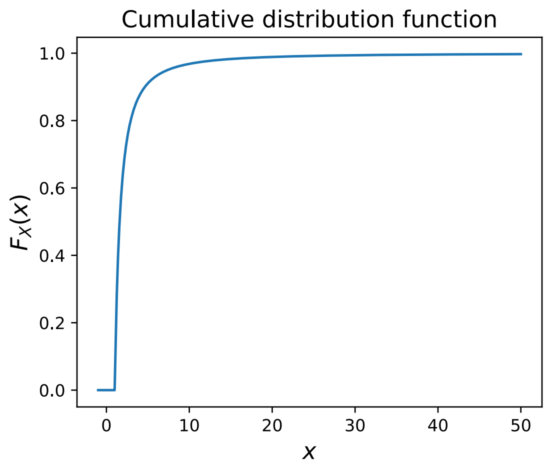

Figure 2: Cumulative Density function

We can sample the distribution using the inverse CDF, so first we need to find the inverse of the CDF. This way we can input a random number in $[0, 1)$ and get a value of x. This inverse can been found analytically, it is good to check against a numerical method. The numerical method does the following steps:

- Generate CDF values for a range of x, the CDF will be in the range $[0, 1)$.
- Given a random number u in $[0, 1)$ that represents the value of the CDF, we find the value of all the x's such that $F_X(x) > u$.
- Take the minimum of these values, this is the value of x that corresponds to the value of the CDF.

For the analytical inverse of the CDF, we can ignore the piecewise condition $x < 1$ since the CDF is only valid in $[0, 1)$.

The analytical inverse CDF is:

$$F_x^{-1}(x) = -\frac{1}{(x-1)^{1/\alpha}}$$

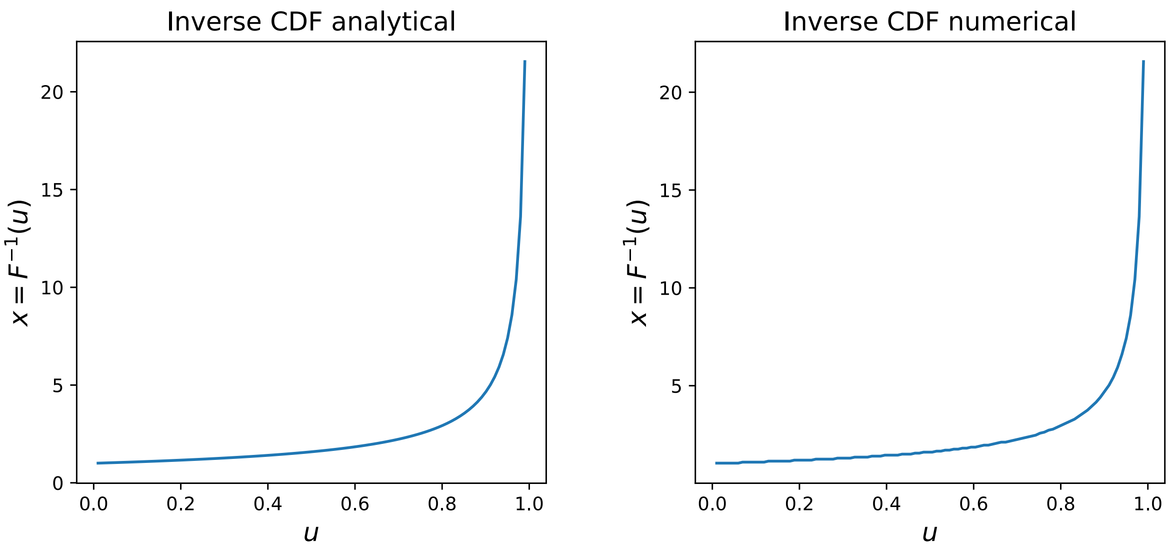

Figure 3: Inverse CDF: analytic vs numerical

Since both results match, we can now go ahead and use the inverse CDF to sample the distribution for use in Monte-Carlo simulations.

It is also useful to analytically calculate other properties of the distribution, such as the true mean and variance so we can compare the Monte Carlo estimate to the true value. The true mean of the distribution is:

$$\mu = \int_1^\infty x \frac{\alpha}{x^{\alpha+1}} dx$$

plugging in values for $\alpha = 3/2$ we get

$$\mu = 3$$

The true variance of the distribution using reference: ProofWiki:

$$
\begin{aligned}
Var[X] &= E[X^2] - E[X]^2 \\
&\text{where} \\
E[X^2] &= \int_1^\infty x^2 \frac{\alpha}{x^{\alpha+1}} dx \\
&= \infty
\end{aligned}
$$

Distribution of the Monte Carlo Estimator for various values of n

To generate a Monte Carlo estimate of the expectation of the distribution, we use n samples of the distribution and take the average of these samples. To see how the distribution of the Monte Carlo estimate changes with n, we can plot the distribution of 500 repeated sequences of Monte Carlo estimate for various values of n with 100001 samples in each sequence.

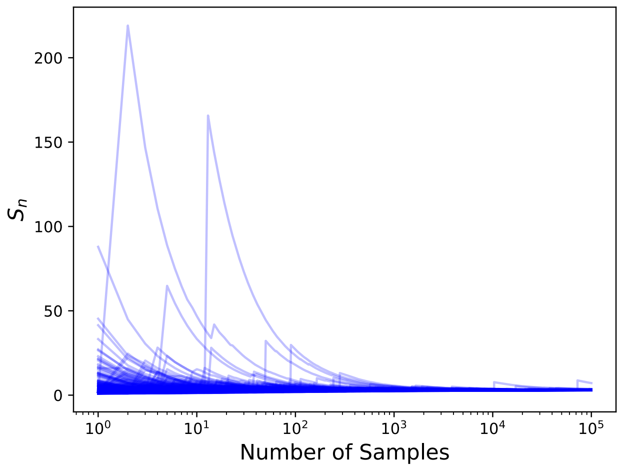

Figure 4: Monte Caro sequences

The distribution of the Monte Carlo estimates for various values of n is shown in the following figure:

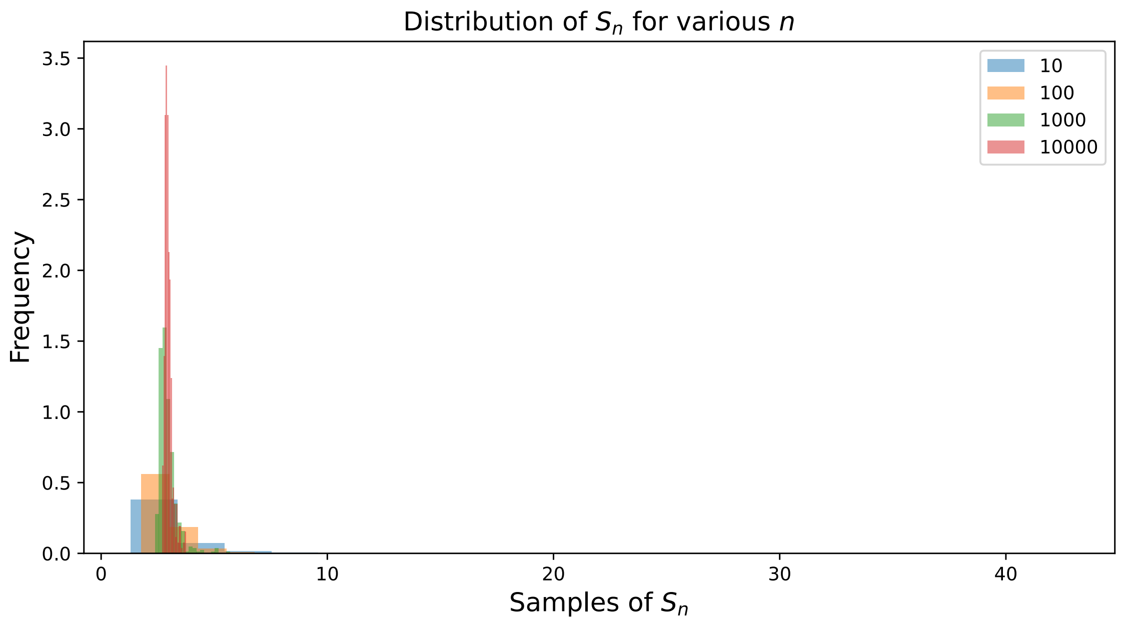

Figure 5: Distribution of the Monte Carlo estimate for various n's

The distribution of the Monte Carlo estimate is not a normal distribution, since it is not symmetric and seems to extend out more on the positive side. However, its expectation does converge to the true mean of the distribution as n becomes larger. Given a large enougn n, the distribution of the Monte Carlo estimate will be centered around the true mean of the distribution. It is also clear that the mean is finite and the probability of it going to infinity is 0. Specifically if X_i is a sequence of i.i.d. random variables with finite mean μ, then the strong law of large numbers states:

$$\mathbb{P}(\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}X_i=\mu)=1$$

Hence, using the strong law of large numbers we can conclude that the partial means converge to the true mean and we can say that the Monte Carlo estimate converges almost surely to the true mean of the distribution.

We can observe that as n increases, the distribution of the Monte Carlo estimate becomes more concentrated around the true expectation of the distribution. This is consistent with the fact that the variance of the Monte Carlo estimate decreases with n. This indeed seems like an asymptotic distribution since the frequency/probability of values drops off as we move away from the mean, approaching 0 but never really reaching it. This makes sense since the analytical variance of the distribution is infinite. Since the analytical variance is infinite, CLT will not apply here.

To get a better idea of the convergence, we can plot the variance of the Monte Carlo estimate as a function of n.

We observe that while the general trend follows $1/n$, there are multiple spikes in the variance througout the range of n. The spikes are not expected for a general case of almost sure convergence, but can be explained by the asymptotic behavior of the sampler we have created, which approaches infinity as the sampling values get close to 1. If for a certain S_n, there happens to be a sample randomly generated that is close to 1, the value of x will be extremely high, causing the variance to spike.

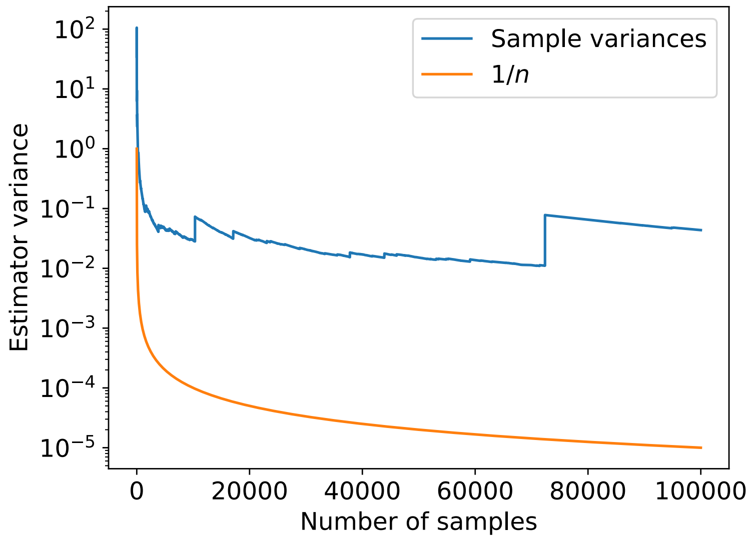

Figure 6: Variance of the Monte Carlo estimate as a function of n

3.1 1-D Bernoulli random walk

2.1.a

A random walk is implemented by:

- Take a random number from a uniform distribution in $[0, 1)$.
- if the number is less than 0.5, take a step = -1, else take a step = 1.
- The walk S is the cumulative sum of these steps, as described in the problem

Visualization of an N-step 1-D Bernoulli random walk, with $n = 1000$ steps.

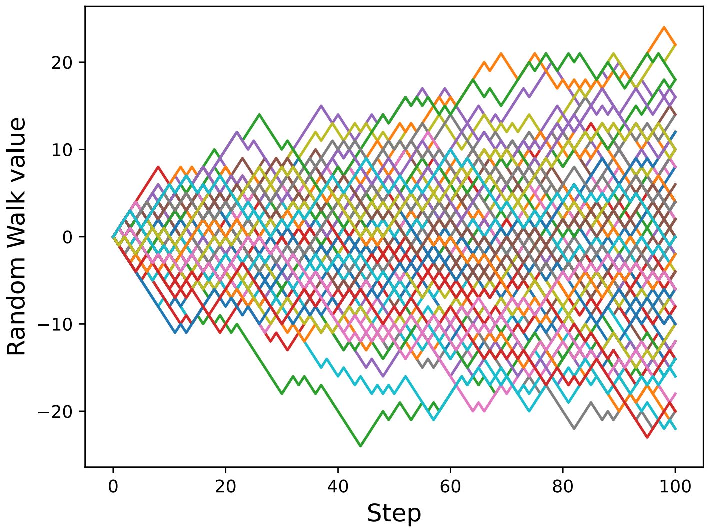

Figure 7: 1-D Bernoulli random walk vs n

2.1.b

To perform a Monte Carlo estimate of $\mathbb{P}(S > 10)$:

- Simulate the random walk 10^5 times with 100 steps.
- Count the number of times the walk exceeds 10 (our indicator function) $1_A(x)$:

$$1_A^{10} = \begin{cases} 1 & \text{if } S > 10 \\ 0 & \text{if } S \leq 10 \end{cases}.$$

- The probability is then the ratio of the number of times the walk exceeds 10 to the total number of trials.

$$\mathbb{P}(S > 10) = \frac{1}{n} \sum_{i=1}^{n} 1_A(x)$$

Monte Carlo estimate with 10^5 trials with 1000 steps in each walk for $\mathbb{P}(S > 10) = 0.13579$

2.1.c

Before performing importance sampling, lets take a look at the distribution of the random walk with 100 steps.

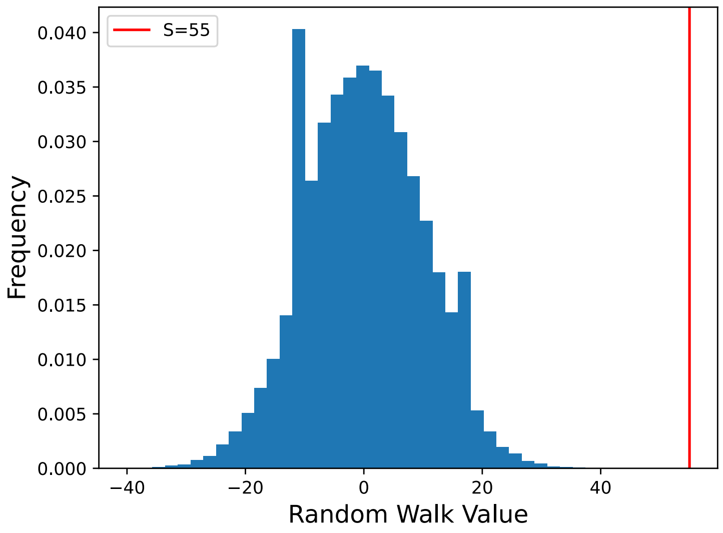

Figure 8: 1-D Bernoulli random walk distribution

The distribution has

- Mean = 0.0838
- Variance = 100.63476096000001

As expected from a random walk, the distribution is normal with a mean ≈ 0 and a standard deviation $\approx$ the number of time steps, which is the same as the number of steps in our case since we are not shrinking the time. However, we see that the probability of the walk exceeding 55 is very low, which makes it a good candidate for importance sampling.

For this problem we change our indicator function to:

$$1_A^{55} = \begin{cases} 1 & \text{if } S > 55 \\ 0 & \text{if } S \leq 55 \end{cases} .$$

Our original distribution is a random walk with a probability of $\mathbb{P}(X_j = 1) = \mathbb{P}(X_j = -1) = 0.5$. If we simply perform Monte Carlo on the original distribution using the new 1_A, we get a probability of 0 with the same number of trials, confirming what we saw in the probability distribution figure above.

To perform the importance sampling, intuitively we need a proposal distribution that has a high probability of exceeding 55. We can just use another random walk with a the probability $\mathbb{P}(X_j = 1) > \mathbb{P}(X_j = -1)$. If we ensure that the expected value of this new distribution is around 55, we can ensure that at least half the samples will exceed 55 so we can get a better estimate. To calculate the new probability for 100 steps, we can use a simple formula to estimate the value of a random walk given a probability p :

$$\sum_{i=1}^{100} (p)(1) + (1-p)(-1) > 55$$

$$100(2p-1) > 55$$

$$p > 0.775$$

So, if the original distribution was $\mathbb{P}(X_j = 1) = \mathbb{P}(X_j = -1) = 0.5$ with PDF as

$$f(S) = p^k(1-p)^{100-k} \quad = 0.5^{100}$$

The proposal distribution can be $\mathbb{P}(X_j = 1) = 0.8, \mathbb{P}(X_j = -1) = 0.2$ with PDF as

$$\pi(S) = p^k(1-p)^{100-k} \quad = 0.8^k(0.2)^{100-k}$$

Where k is the number of $+1$ steps taken in the random walk S.

So we can use $\pi(S)$ to get a better estimate for $S > 55$.

More formally, the expectation can be re-written as:

$$\begin{aligned}
\mathbb{P}(S > 55) &= \mathbb{E}_f[1_A^{55}(S)] \\
&= \int 1_A^{55}(S) f(S) dx \\
&= \int 1_A^{55}(S) \frac{f(S)}{\pi(S)} \pi(S) dx \\
&= \int 1_A^{55}(S) w(S) g(S) dx \\
&= \mathbb{E}_\pi[1_A^{55}(S) w(S)]
\end{aligned}$$

And we can perform Monte Carlo on this new distribution like before, taking samples from the proposal distribution instead of the original distribution to calculate 1_A^{55}. We can then multiply them by the weights. The Importance sampling estimate for $\mathbb{P}(S > 55) = 7.966439961067404e - 09$.

Interestingly, increasing the $\mathbb{P}(X_j = 1)$ too high in the prposal distribution closer to 0.9 actually causes the MC with importance sampling estimate to become worse and stray away further from the analytical estimate. This could be explained by the fact that the proposal distribution gets too far from the original distribution, outputting values that are very high with extremely low probabilities in the original distribution. This makes the weights $w(x) \sim 0$, giving us no additional information. At the extreme, if we set this to 1, then the MC with importance sampling estimate becomes 0, which is not correct. So it makes sense to keep it close to the original distribution at 0.8.

2.1.d

Analytical expression for the probability of the random walk exceeding a threshold T is a simple probability calculation for a binomial distribution:

$$\mathbb{P}(S > T) = \sum_{i=T+1}^{100} \mathbb{P}(S = i)$$

For $\mathbb{P}(S = i)$ we would have j steps with +1 and $100 - j$ steps with -1. Then this must hold true for i, j:

$$\begin{aligned} j - (100 - j) &= i \\ 2j - 100 &= i \\ j &= \frac{i + 100}{2} \end{aligned}$$

Take note that we can only choose an integer from another integer, so values of j that are fractions will be ignored. This will happen when i is odd, and there is no way for this random walk to generate a final step that is odd.

For n steps $\mathbb{P}(S = i)$ becomes:

$$\begin{aligned} \mathbb{P}(S = i) &= \binom{100}{j} p^j (1-p)^{100-j} \\ \textit{substituting} \quad p = (1-p) = 0.5 \\ &= \binom{100}{j} 0.5^{100} \\ \textit{substituting} \quad j = \frac{i+100}{2} \\ &= \binom{100}{\frac{i+100}{2}} 0.5^{100} \end{aligned}$$

Substituting this into $\mathbb{P}(S > T)$:

$$\mathbb{P}(S > T) = \sum_{i=T+1}^{100} \binom{100}{\frac{i+100}{2}} 0.5^{100}$$

When $T = 10$, evaluating this expression gives us $\mathbb{P}(S > 10) = 1.356265e - 01$, which is very close to the Monte Carlo estimate with a difference of only 0.16%.

When $T = 55$, evaluating this expression gives us $\mathbb{P}(S > 55) = 7.95266423689307e - 09$, which is very close to the MC with importance sampling estimate with a difference of only 0.17%.

2.1.e.i

Standard errors for the estimates can be calculated as follows:

Monte Carlo for $\mathbb{P}(S > 10)$: Reference: Notes Page 56

Using $1_A^{10}(S)$ is the indicator function for the event $\mathbb{P}(S > 10)$:

$$Var[S_n] = \frac{1}{n} Var[1_A^{10}(S)]$$
$$std \quad error = \sqrt{\frac{Var[1_A^{10}(S)]}{n}}$$

Or alternatively, using the analytical probability of $p = \mathbb{P}(S > 10)$:

$$std \quad error = \sqrt{\frac{p(1-p)}{n}}$$

Both evaluate to a similar result:

- Using indicator function: $1.083489e - 03$
- Analytical using p : $1.082737e - 03$

MC with Importance Sampling for $\mathbb{P}(S > 55)$: Reference: Notes Page 58

$$Var[S_n^{IS}] = \frac{1}{n} Var[1_A^{55}(S) w(S)]$$
$$std \quad error = \sqrt{\frac{Var[1_A^{55}(S) w(S)]}{n}}$$

$1_A^{55}(S)$ is the indicator function for the event $\mathbb{P}(S > 55)$ and $w(S)$ is the weight for each sample calculated by dividing the original probability by the proposal probability. This evaluates to:

- MC with Importance Sampling standard error = 6.003709e-11

Reference: Notes Page 48

For calculating the 95% confidence intervals using the central limit theorem, we need to first normalize the estimates so they become $\sim \mathcal{N}(0, 1)$ as $n \to \infty$. This is done by subtracting the mean and dividing by the standard deviation to get the normalized estimate, H_n:

$$H_n = \frac{S_n - \mu}{\sigma / \sqrt{n}}$$

A confidence interval is defined as the interval $[-z, z]$ such that $P_X(H_n \in [-z, z]) \geq 1 - \delta$ where z is the z-score for the desired confidence level and $1 - \delta$ is the confidence level. For a 95% confidence interval, $1 - \delta = 0.95$ and $z = 1.96$ for $\mathcal{N}(0, 1)$, which is a standard quantity that can be obtained from various online sources and from `scipy.stats.norm.ppf`. Simplifying the expression for the probability defined above:

$$\begin{aligned}
P_X(H_n \in [-z, z]) &= P_X(-z \leq H_n \leq z) \\
&= P_X(-z \leq \frac{S_n - \mu}{\sigma/\sqrt{n}} \leq z) \\
&= P_X(\frac{-z\sigma}{\sqrt{n}} \leq S_n - \mu \leq +\frac{z\sigma}{\sqrt{n}}) \\
&= P_X(-S_n - \frac{z\sigma}{\sqrt{n}} \leq -\mu \leq -S_n + \frac{z\sigma}{\sqrt{n}}) \\
&= P_X(S_n + \frac{z\sigma}{\sqrt{n}} \geq \mu \geq S_n - \frac{z\sigma}{\sqrt{n}}) \\
&= P_X(S_n - \frac{z\sigma}{\sqrt{n}} \leq \mu \leq S_n + \frac{z\sigma}{\sqrt{n}})
\end{aligned}$$

The standard error calculated earlier is just $\frac{\sigma}{\sqrt{n}}$, which makes it easy to calculate this. The 95% confidence intervals calculated for the Monte Carlo and Importance Sampling estimates are:

- Monte Carlo for $\mathbb{P}(S > 10)$ = [0.1337263621371712, 0.1379736378628288]
- MC with Importance Sampling for $\mathbb{P}(S > 55)$ = [7.848767266908999e-09, 8.084112655225919e-09]

2.1.e.ii

Number of confidence intervals that contain the true value of the probability:

- Monte Carlo: 945
- MC with Importance Sampling: 956

The number of confidence intervals that contain the true value of the probability is very close to the expected value of 950 for both methods for a 95% confidence interval.

2.1.e.iii

To get the empirical 95% band, we can use the 2.5 and 97.5 percentiles of the at each step of the sequence of estimates. To get the CLT 95% confidence interval for the Monte Carlo estimates at each step in the sequence, we can just pick one of the replicas and calculate the confidence interval for that replica. This is because the standard error should be similar for all replicas, and the confidence interval is just a function of the standard error. Take note that for calculating the CLT 95% confidence interval, we take the mean as the analytical mean, and all the calculations are done in the same way as in part **2.1.e.i**.

Plotting an envelope of sequences of Monte Carlo estimates for $\mathbb{P}(S > 10)$ and MC with Importance sampling for $\mathbb{P}(S > 55)$. Black represents the envelope, Blue shaded region represents the empirical 95% band and the dotted red line represents the 95% CLT confidence interval.

Note: I was not able to run the code for more then 100 replicas, hence the graph is not very smooth. However, the general trend is still visible. I have made my conclusions based on what the graph would look like if I had 1000 replicas.

For both Estimators, as the number of trials increases, the bound gets tighter, which is expected as variance for a Monte Carlo estimator decreases as the number of trials increases. The envelope of sequences of Monte Carlo estimates for $\mathbb{P}(S > 10)$ is much wider than the envelope of sequences of MC with Importance sampling for $\mathbb{P}(S > 55)$, which is expected since the standard error for the MC with Importance sampling is much lower than the standard error for the Monte Carlo estimate. We can see that the empirical and CLT 95% envelope are very close to each other, and they get closer as the number of trials increases. This proves that the CLT 95% confidence interval is a good approximation for the empirical 95% band.

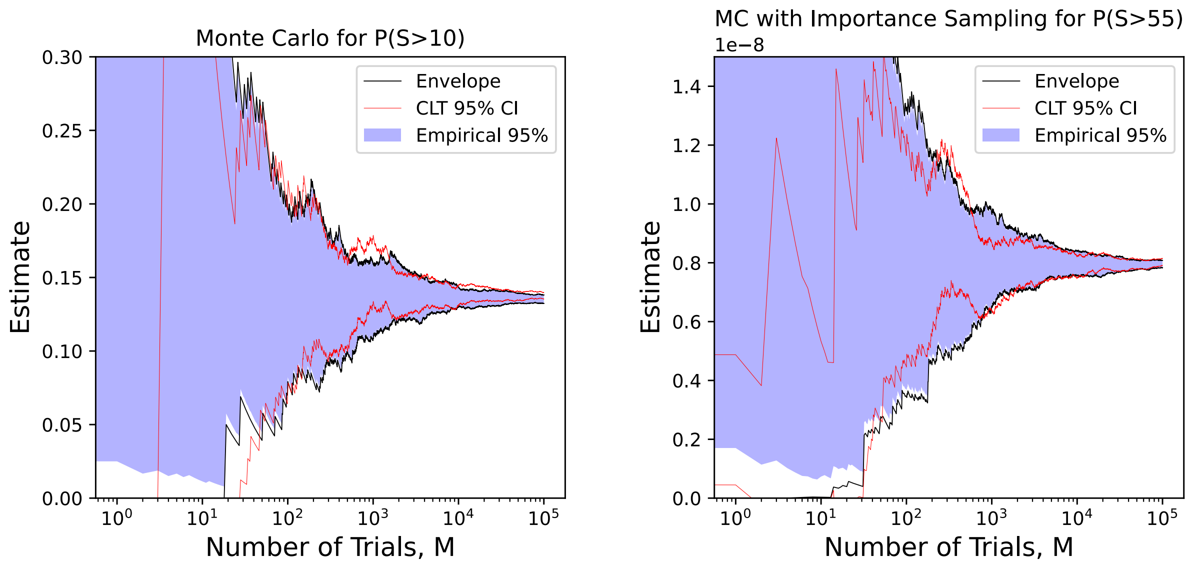

Figure 9: Envelope of sequences of Monte Carlo vs M

3.2 3-D Gaussian random walk

2.2.a

Code implementation for a 3-D Gaussian random walk with $n = 1000$ steps.

```
def gaus_rw_steps(ntrials):
    """sample_generator, Generate random walks for a number of trials"""
    return np.random.randn(ntrials, NSTEPS, 3)


def gaus_rw_test(samples):
    """g_evaluator, Evaluate the indicator function for total distance travelled, S > 10"""
    return np.where(np.linalg.norm(np.sum(samples, axis=1), axis=1) > S, 1, 0)
```

The code above generates a 3-D Gaussian random walk with $NSTEPS = 100$ steps.

`gaus_rw_steps` generates a set of steps for a random walk, taking note that now we are generating steps in 3 dimensions. It sampes from the standard normal distribution, which is a Gaussian distribution with $\mu = 0$ and $\sigma = 1$ and generates a value for each of the 3 dimensions of the walk for a set number of steps.

`gaus_rw_test` evaluates the indicator function for the total distance travelled, $S > 10$. It takes the sum of the steps in each walk for every dimension to get the total distance travelled per dimension, then takes the norm of the total distance travelled to get the total distance travelled in 3 dimensions. It then checks if the total distance travelled is greater than S and returns 1 if it is, 0 otherwise for every sample provided to the function.

The norm we do here is equivalent to $|S|$:

$$|S| = \sqrt{s_1^2 + s_2^2 + s_3^2}$$

where s_1, s_2, s_3 are the total distance travelled in each dimension.

$$s_1 = \sum_{j=1}^{n} x_{1,j}$$

$$s_2 = \sum_{j=1}^{n} x_{2,j}$$

$$s_3 = \sum_{j=1}^{n} x_{3,j}$$

Plotting 20 random walks generated by the code above for 100 steps in 3D:

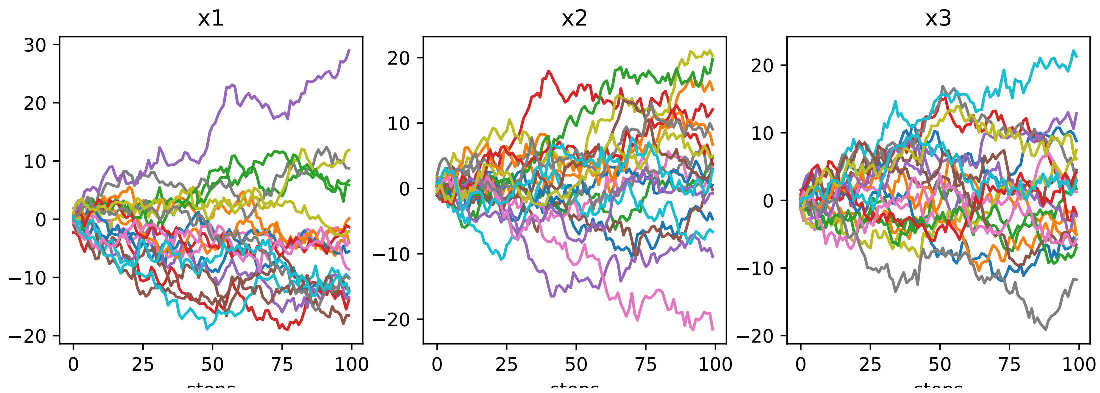

Figure 10: 3D Gaussian Random Walk

2.2.b

Monte Carlo method to estimate the probability given an indicator function has been discussed in the previous exercises, so I will not go through the entire explanation agian. The indicator function here 1_A^{10} is given by:

$$1_A^{10} = \begin{cases} 1 & \text{if } |S| > 10 \\ 0 & \text{if } |S| \leq 10 \end{cases} .$$

We can then generate samples and evaluate them using this indicator function, and then finally find the mean of the result to get our Monte Carlo estimate for the probability $|S| > 10$

Using a simple Monte Carlo method, the probability $\mathbb{P}(|S| > 10)$ is estimated as: 8.010300e-01.

2.2.c

Before we move on with the importance sampling, it makes sense to calculate the exact probability of the event $|S| > 55$ for the 3-D Gaussian random walk. This is given by:

$$\mathbb{P}(|S| > 55) = \mathbb{P}(\sqrt{s_1^2 + s_2^2 + s_3^2} > 55)$$

where s_1, s_2, s_3 are sums of 100 x $\sim \mathcal{N}(0, 1)$

hence giving them a probability density function of $\mathcal{N}(0, 100)$

normalizing them, we get

$$
\begin{aligned}
|S|_{norm} &= \sqrt{\frac{s_1^2 + s_2^2 + s_3^2}{100}} \\
|S|_{norm} &= \frac{\sqrt{s_1^2 + s_2^2 + s_3^2}}{10} \\
|S|_{norm} &= \frac{|S|}{10} \sim \mathcal{N}(0,1)
\end{aligned}
$$

so the probability becomes

$$
\begin{aligned}
\mathbb{P}(|S| > 55) &= \mathbb{P}(\frac{|S|}{10} > \frac{55}{10}) \\
&= \mathbb{P}(|S|_{norm} > 5.5)
\end{aligned}
$$

Now our $|S|_{norm}$ should follow a chi distribution with 3 degrees of freedom, and we can calculate the probability of $|S|_{norm} > 5.5$ using the chi distribution to be $1.222653e - 06$

Tryin to run a simple Monte Carlo estimator for this probability just gives us a value of 0, which means that we would need a very large sample size to get a good estimate for this probability.

For the importance sampling, we need a new proposal distribution with a PDF $\pi(S)$ such that it is > 0 whenever $1_A^{10}(S) f_X(S) \neq 0$.

$$
\begin{aligned}
\mathbb{P}(S > 55) &= \mathbb{E}_f[1_A^{55}(S)] \\
&= \int 1_A^{55}(S) f(S) dx \\
&= \int 1_A^{55}(S) \frac{f(S)}{\pi(S)} \pi(S) dx \\
&= \int 1_A^{55}(S) w(S) g(S) dx \\
&= \mathbb{E}_\pi[1_A^{55}(S) w(S)]
\end{aligned}
$$

Many variations of gaussians were tried for this, in an effort to increase the variance so that there is a higher chance of taking big steps. However, the main issue here is that there is always an equal probability of taking steps in any direction, so over the course of a few steps, they tend to cancel each other out, which makes sense as the mean of this distribution is 0.

A good proposal distribution here would ensure that steps are taken in the same direction, which means that the direction of the steps should be correlated. This would ensure that the steps do not cancel each other out, and the total distance travelled is more likely to be greater than 55.

I was not able to implement such a function in the time given, but I believe that a good proposal distribution would be a multivariate gaussian with a covariance matrix that is not diagonal, and has a high correlation between the dimensions. This would ensure that the steps are taken in the same direction, and the total distance travelled is more likely to be greater than 55.

2.2.d

Using $1_A^{10}(S)$ is the indicator function for the event $\mathbb{P}(|S| > 10)$:

$$
\begin{aligned}
Var[S_n] &= \frac{1}{n} Var[1_A^{10}(S)] \\
std \quad error &= \sqrt{\frac{Var[1_A^{10}(S)]}{n}}
\end{aligned}
$$

Monte Carlo eror for the probability $\mathbb{P}(|S| > 10)$ is estimated as: 9.999900e-06.

4 Multilevel Monte Carlo and Control Variates for Stochastic ODEs

Q: What is the distribution of ΔW_n ?

A: $\Delta W_n \sim \mathcal{N}(0, \Delta t)$

Explanation:

As per property 3, $W(t + \Delta t) - W(t) \sim \mathcal{N}(0, \Delta t)$

And

$$\begin{aligned} \Delta W_n &= W(t_{n+1}) - W(t_n) && (discrete) \\ &= W(t + \Delta t) - W(t) && (original) \\ &\sim \mathcal{N}(0, \Delta t) \end{aligned}$$

Basically here, the increment here is just 1 step of Δt, which makes the variance of the increment of the Weiner process ΔW_n also equal to Δt.

4.1 Geometric Brownian Motion

1

The Geometric Browninan motion is simulated by first generating a sequence of simple brownian motion paths, where each step is $\sim \mathcal{N}(0, 1)$, and each step is added to the previous over time to generate a full path. This is repeated here for 1000 paths, and then we can then use those paths with the Euler-Maruyama method to scale the *diffusion* term of the SDE over time, essentially adding to the variance of the SDE. Upon simulating 1000 paths over T[0, 1] with $\Delta t = 0.01$, and then plugging them into the Euler-Maruyama method, we get the following plot:

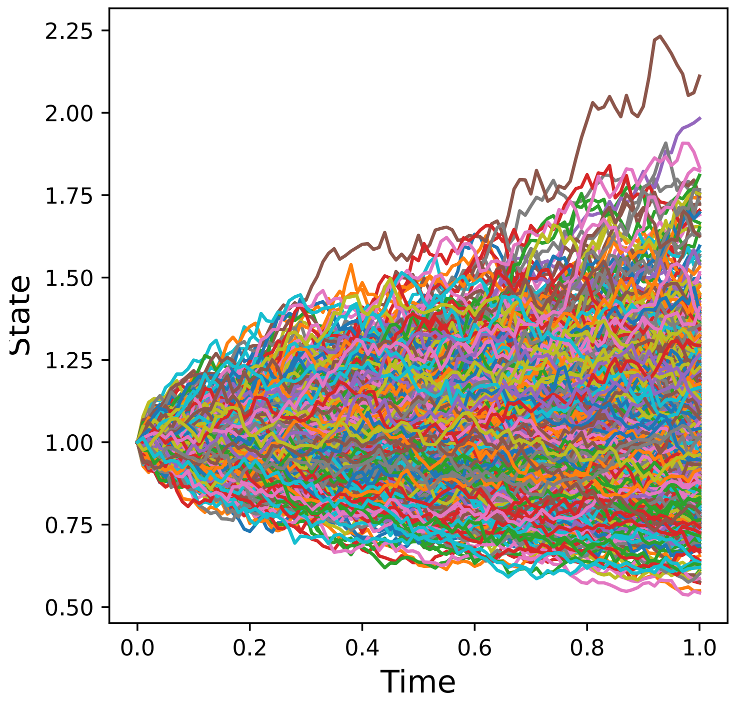

Figure 11: Geometric Brownian Motion

The mean and variance of $Y(1)$ using the simulation with Euler-Maruyama is calculated by simply finding the mean of the paths at $t = 1$ and the variance of the paths at $t = 1$. The mean at $t = 1$ corresponds to a Monte Carlo estimator for the expected value of $Y(1)$. The variance at $t = 1$ is just the variance of the samples, not the variance of the Monte Carlo estimator. The mean and variance of $Y(1)$ using the simulation is:

- Mean: 1.0513250656841955
- Variance: 0.045393795029522364

Instead of simulating using the whole time interval $[0, 1]$ using Euler-Maruyama, we can also use the analytical equation to solve for $Y(1)$ directly by just using the value of all the paths of the brownian motion at $t = 1$. The mean and variance of $Y(1)$ using this method is:

- Mean: 1.051286135353164
- Variance: 0.04547183854880979

And finally, we can also compute the full analytical solution for the mean and variance of $Y(1)$ without using any sampled data, which is:

$$\begin{aligned}
Y_t &= Y_0 exp((\mu - \frac{\sigma^2}{2})t + \sigma W_t) \\
\text{sustituting} \quad & \mu = 0.05, \sigma = 0.2, Y_0 = 1, t = 1 \\
Y(1) &= 1 \times exp((0.05 - \frac{0.2^2}{2}) \times 1 + 0.2 \times W(1)) \\
&= exp(0.03 + 0.2W(1)) \\
&= exp(0.03)exp(0.2W(1)) \\
\text{we know that } W(t) & \text{ is normally distributed with mean 0 and variance } t \\
Y(1) &= exp(0.03)exp(0.2\mathcal{N}(0, 1)) \\
&= exp(0.03)exp(\mathcal{N}(0, 0.04)) \\
\text{using } E[exp(X)] &= exp(\mu + \sigma^2/2) \\
\text{and } Var(exp(X)) &= (exp(\sigma^2) - 1)exp(2\mu + \sigma^2) \\
Y(1) &= exp(0.03)Lognormal(exp(0.04/2), (exp(0.04) - 1)exp(0.04)) \\
&= exp(0.03)Lognormal(1.0202, 0.0425) \\
&= 1.0305Lognormal(1.0202, 0.0425) \\
&= Lognormal(1.0513, 0.0451)
\end{aligned}$$

- Mean: 1.0512710963760241
- Variance: 0.04510288078157963

The mean and variance of $Y(1)$ using all 3 methods are very close, which is a good indicator that the simulation is working as expected.

Now to get the variance of a Monte Carlo estimator of the mean, we can just use the formula for the variance of a sample mean, assuming $\bar{Y}$ is the sample mean of $Y(1)$ after simulating n paths using Euler-Maruyama:

$$\begin{aligned}
\text{Var}(\bar{Y}) &= \frac{\text{Var}(Y)}{n} \\
&= \frac{0.045393795029522364}{1000} \\
&= 4.539379502952236e - 05
\end{aligned}$$

2

The code can be adjusted by

1. Generating a fine sample
2. Coarsening it every 4 steps to generate a coarse sample. Since the coarse sample is generated by picking out one of every 4 steps, they will share the underlying realization of the Wienner process.
3. We can then use the coarse samples to simulate the Geometric Brownian motion using the Euler-Maruyama method, but now using a time step of $4\Delta t$.

Taking a single path from the coarse and fine samples and setting the Δt to 0.05 instead of 0.01 from earlier to make it easier to see the coarsening:

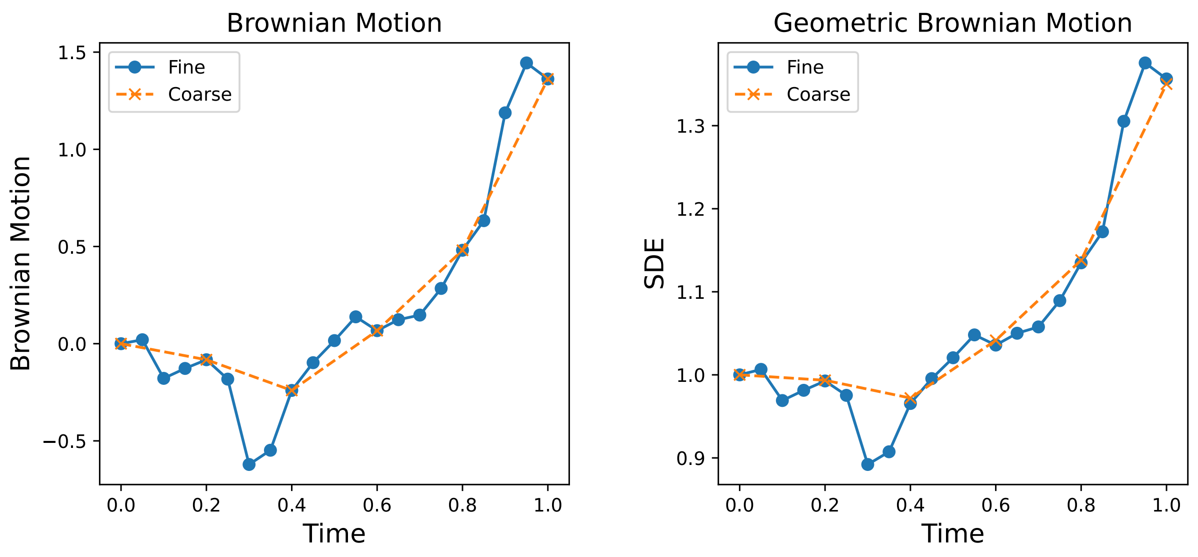

Figure 12: Coarse and fine samples of Brownian motion

We can observe here that for just the Brownian Motion, coarse samples are exactly the same as the fine samples every 4 time steps, which is expected since they share the same underlying realization of the Wienner process.

However, when looking at the Geometric Brownian motion simulated by the Euler-Maruyama method using the coarse and fine samples, we can see that the paths deviate slightly over time, but they still are very close to each other. This is expected since the coarse samples will have numerical integration errors due to the randomness of the stochastic process that will accumulate over time, but the paths will still be close to each other since they share the same underlying realization of the Wienner process.

3

Four level MLMC estimator For a multilevel Monte Carlo estimator, we will need multiple fidelities of the underlying realization that we use to simulate our SDE. Since we have already displayed how to coarsen the samples while still retaining the underlying realization, we will use the same process to generate 4 levels of fidelities, just increasing the time step by a factor of 4 each time we coarsen the samples. We can then use the Euler-Maruyama method to simulate the Geometric Brownian motion using the samples at each level of fidelity of coarseness, and then use the multilevel Monte Carlo estimator ($S_N^{MLMC}(Y)$) to estimate the expectation $Y(1)$.

$$S_N^{MLMC}(Y) = E[Y(1)_2] + \sum_{l=3}^{5} E[Y(1)_l - Y(1)_{l-1}]$$

Where $Y(1)_l$ is the set of values of $Y(1)$ obtained using the Euler-Maruyama method with brownian motion samples at $\Delta t = 4^{-l}$ and $E[Y(1)_l]$ is simply the mean of those sets of values. N is a vector that contains the number of samples to be taken at each level. At every level besides $l = 2$, we will have coarse($l-1$) and fine (l) samples, generated using the same underlying realization for the Brownian motion so the coarse and fine estimators for each level are correlated. Take note that the samples generated at each level will be independent of other levels. In addition, the the number of samples needed for each level can be different as well, which is the point of MLMC as we want to reduce the number of samples taken at the finer levels.

Modifying the code to estimate $E[P]$ instead of $E[Y(1)]$ is just a matter of using the equation shown in the project description calculate the value of P for each value of $Y(1)$ for all the samples of browninan motion at every level and then using the Monte-Carlo method to estimate the final value of $E[P]$.

For each level, the process to generate data for the estimators is as follows:

- Level 2:
  - Generate brownian motion samples using $\Delta t = 4^{-2}$ and the appropriate number of samples.
  - Use Euler-Maruyama to simulate the SDE using Geometric Brownian motion for all samples and get $Y(1)_2$.
  - Use the all the simulated paths to calculate the value of P_2.
  - Calculate the mean of all the values in $Y(1)_2$ and P_2 to get the final estimate for $E[Y(1)_2]$ and $E[P_2]$.
- All other levels
  - Generate fine brownian motion samples using $\Delta t = 4^{-l}$ and the appropriate number of samples
  - Generate coarse brownian motion samples by coarsening the fine samples by a factor of 4.
  - Use Euler-Maruyama to simulate the SDE using Geometric Brownian motion for all samples and get $Y(1)_l$ and $Y(1)_{l-1}$.
  - Use the all the simulated paths to calculate the value of P_l and P_{l-1}.
  - For each simulated path in $Y(1)_l$ and $Y(1)_{l-1}$, calculate the difference between the two and then calculate the mean difference between all the simulated paths to get the estimate for $E[Y(1)_l - Y(1)_{l-1}]$.
  - For each value in P_l and P_{l-1}, calculate the difference between the two and then calculate the mean difference between all the values to get the estimate for $E[P(1)_l - P(1)_{l-1}]$.

Once we have this data for all the levels, we can then use the equation for the multilevel Monte Carlo estimator described earlier to calculate the final estimate for $E[Y(1)]$ and $E[P]$.

For calculating the variance of the MLMC estimator, we can just divide each level's variance by the number of samples used at that level and then sum them up to get the final variance of the MLMC estimator.

$$Var[S_N^{MLMC}(Y)] = \frac{Var(Y(1)_2)}{N_2} + \sum_{l=3}^{5} \frac{Var[Y(1)_l - Y(1)_{l-1}]}{N_l}$$

Where N_l is the number of samples used at level l. For this project, we will use the same number of samples at each level, which is 10^5.

After running the MLMC estimator, we get the following results for $E[Y(1)]$:

- $E[Y(1)]$: 1.0511142682675358
- Difference from analytical solution: 0.01%

Which means that our MLMC estimator is working as expected.

Discussing Variance of the MLMC estimator

After running the MLMC estimator, we get the following results for variance:

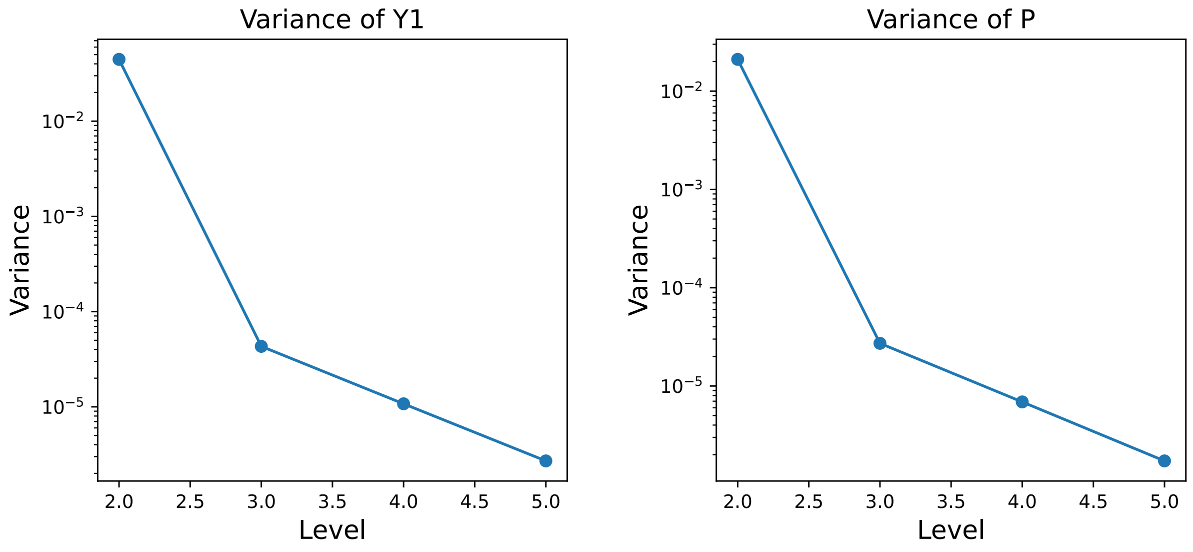

Figure 13: Variance of MLMC estimator for each level

Variance for MLMC estimator:

- $Y(1)$: $4.4887287222994214e-07$
- P : $2.1419784299969563e-07$

We can notice that the variance of each level of the MLMC estimator is decreasing as we go to the finer levels. This is expected, since at levels besides the coarsest level, we are taking the variance of the difference between the coarse and fine values of $Y(1)$, which are generated using the same underlying realiziation of the brownian motion. When we calculate the correlation between the coarse and fine simulations at levels [3, 4, 5], we can see that they are ~ 1 for both $Y(1)$ and P, which explains the low vaiance of the difference between the coarse and fine simulations. This is a good indicator that the MLMC estimator will work well for this problem since it means that we can extract a lot of the same information from low fidelity simulations as we can from high fidelity simulations.

Also, notice that variance reduces as the number of samples used increases and the cost of running a simulation using larger time steps at lower levels is lower. This means that we can use more samples at lower levels to reduce the variance of the MLMC estimator even further without incurring a large additional cost. On the other hand, we can also reduce the overall cost of the MLMC estimator while keeping variance the same by using fewer samples at the higher(finer) levels and balaning it out by taking more samples at the lower(coarser) levels.

Discussing the expectation of the MLMC estimator

After running the MLMC estimator, we get the following results for expectation:

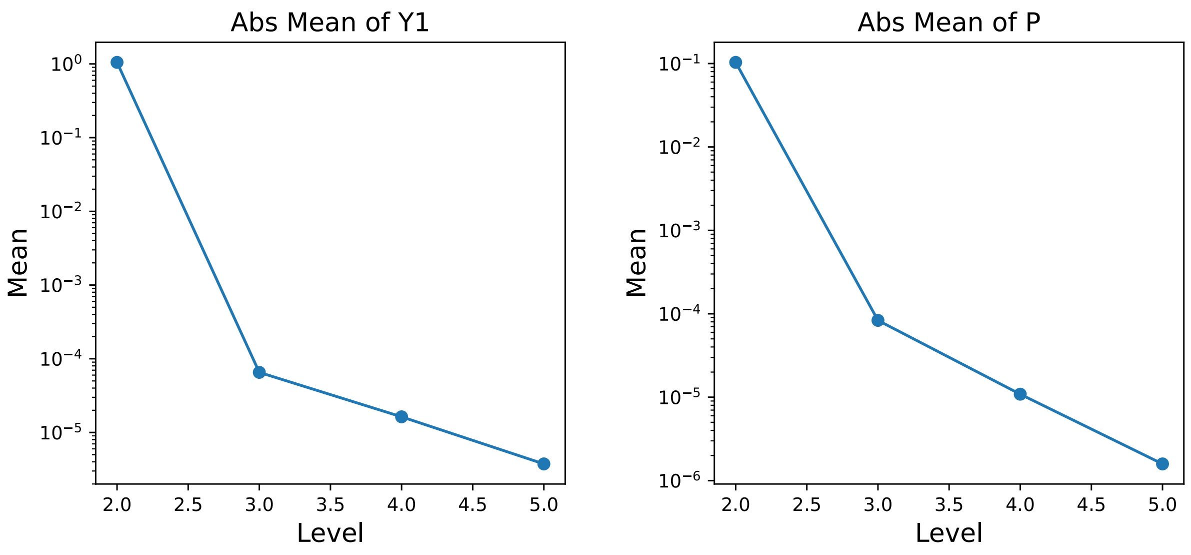

Figure 14: Expectation of MLMC estimator for each level

Expectation for MLMC estimator:

- $Y(1)$: 1.0511142682675358
- P : 0.10490297238106877

We see a similar trend in the expectation of the MLMC estimator as we did with the variance. The expectation of the MLMC estimator is decreasing as we go to the finer levels, which is expected. Beside the coarsest level, we are taking the mean of the difference between the coarse and fine values of $Y(1)$ and P, which are expected to be low because simulations that generate both coarse and fine values of $Y(1)$ and P are generated using the same underlying realization of the brownian motion. We can notice that the coarsest level has almost all of the total expectation, which is a good indicator that we can extract most of the information we need from low fidelity simulations. The higher levels are there to just correct the low fidelity simulations and push the expectation of the MLMC estimator closer to the true value of $Y(1)$ and P.

4

To estimate the theoretical cost, it makes sense to use the payoff function for the stock, P since that is ultimately what we are interested in. However, P is only calculated at the end once we have simulated Y over the total time T so its cost is constant and only changes if we change the number of levels in our MLMC. Hence here we will only consider the cost of $Y(T)$ where T=1. The cost of the MLMC estimator is the sum of the cost of running the simulations at each level. The cost of running the simulations at each level l is proportional to the number of samples needed at each level N_l and the fidelity of the simulation, determined by the inverse of Δt. The number of samples at each level is the same at 10^5 and the inverse of the fidelity of a single simulation at a level is $(\Delta t)_l^{-1} = (4^{-l})^{-1} = 4^l$. Take not that for $l > 2$, we have to add the cost associated with both the coarse and fine simulation:

$$\begin{aligned}
C_{total}^{MLMC} &= C_2 + C_3 + C_4 + C_5 \\
&= N_2(\Delta t)_2^{-1} \sum_{l=3}^{5} N_l((\Delta t)_l^{-1} + (\Delta t)_{l-1}^{-1}) \\
&= 10^5 \times 4^2 + (10^5(4^3 + 4^2) + 10^5(4^4 + 4^3) + 10^5(4^5 + 4^4)) \\
&= 1600000 + (8000000 + 32000000 + 128000000) \\
&= 169600000
\end{aligned}$$

Referencing the notes `multilevel_monte_carlo.pdf` from the course, the problem for finding the optimal sample allocation is an optimization problem which can be solved by using a lagrangian formulation: Take note that in the notes, the summation starts with $l = 0$ but in this project, the summation starts with $l = 2$ because of the convention we have followed till now.

$$L(N, \lambda^2) = \sum_{l=2}^{L} \frac{V_l}{N_l} + \lambda^{-2}(\sum_{l=2}^{L} N_l C_l - C_{total}^{MLMC})$$

Where:

- V_l is the variance of the MLMC estimator for P at level l
  - for $l > 2$: variance of the difference between the coarse and fine estimations
- N_l is the number of samples at level l
- C_{total}^{MLMC} is the total cost of running the simulations at all levels as we calculated earlier
- C_l is the cost of running the simulations at level l as we saw earlier when calculating C_{total}^{MLMC}
- λ is the lagrange multiplier

Solving this for N_l after setting the derivative of L with respect to N_l to 0, we get:

$$N_l^{opt} = \lambda \sqrt{\frac{V_l}{C_l}}$$

And the variance of the estimator (the accuracy ϵ^2) using this allocation is:

$$Var[S_N^{MLMC}(Y)] = \epsilon^2 = \lambda^{-1} \sum_{l=0}^{L} \sqrt{V_l C_l}$$

which gives us

$$\lambda = \epsilon^{-2} \sum_{l=0}^{L} \sqrt{V_l C_l}$$

Using these equations, we can calculate the λ for a given ϵ^2 and then use that to calculate the optimal number of samples N_l at each level.

For example, for a target accuracy(variance) of $\epsilon^2 = 10^{-8}$ for P, we can calculate the optimal number of samples at each level by first calculating λ, using the variance of P at each level. This comes out to be $\lambda = 22955386440.88965$.

Using this λ, we can then calculate the optimal number of samples at each level to make the samples realistic, we take the ceiling of the resulting N_l^{opt}:

- $N_2^{opt} = 2653380$
- $N_3^{opt} = 42486$
- $N_4^{opt} = 10689$
- $N_5^{opt} = 2674$

5

The variance for the highest fidelity Monte Carlo esimate of P would be at $l = 5$ with the fine simulation. Let this be P_5^{fine}. This variance is given by:

$$Var[S_N^{MC}(P_5^{fine})] = \frac{Var[P_5^{fine}]}{N_5}$$

Hence, if we want to achieve a certain variance $Var[S_N^{MC}(P_5^{fine})]$ of ϵ^2 in the highest fidelity Monte Carlo estimate of $P = P_5^{fine}$, we can estimate the Number of samples N_5^{ϵ} needed for this is:

$$N_5^{\epsilon} = \frac{Var[P_5^{fine}]}{\epsilon^2}$$

Using this equation, and using variance of P_5^{fine} calculated using the MLMC estimator we ran earlier, $Var[P_5^{fine}] = 0.021544072463013958$, $N_5^{\epsilon} = 2154408.0$

The overall costs for both Monte Carlo and MLMC would just be their respective costs to achieve the same variance ϵ^2. The cost of the Monte Carlo estimator is just the cost of running the simulation at the highest fidelity level, C_{hf} multiplied by the number of samples needed at that level, which is N_5^{ϵ}. The cost of the MLMC estimator is just the total cost of running the simulations at all levels, which is C_{total}^{MLMC} for a certain optimal sample allocation N^{opt}.

Therefore, to calculate the equivalent number of high-fidelity evaluations for MLMC when compared to a Monte Carlo estimator, we can just divide the cost of the MLMC estimator by the cost of a single run of the high fidelity simulation Monte Carlo estimator. In this case, that would be:

$$\text{Equivalent number of high-fidelity evaluations} = \frac{C_{total}^{MLMC}}{C_{hf}}$$

Plotting Estimator variance vs. Equivalent number of high-fidelity evaluations for both Monte Carlo and Multilevel Monte Carlo:

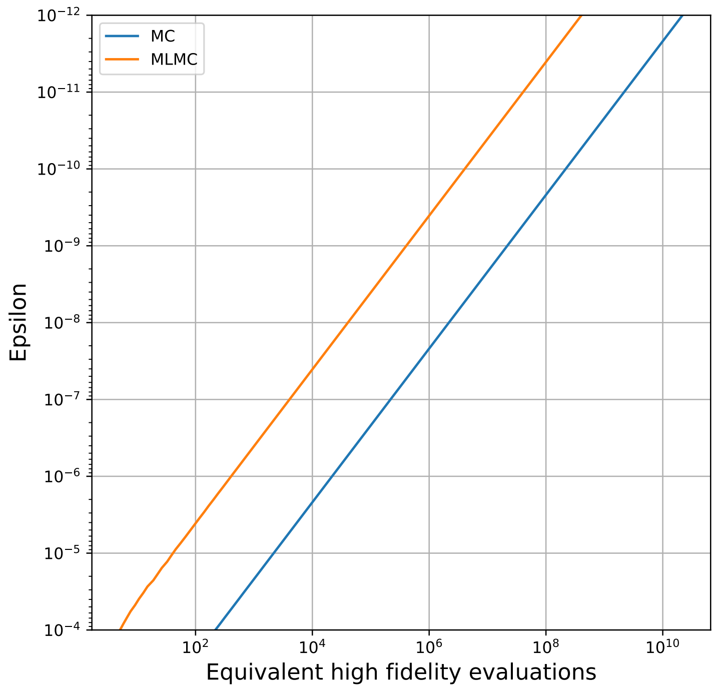

Figure 15: Estimator variance vs. Equivalent number of high-fidelity evaluations

In the graph we can see clearly that as we require higher accuracy, the equivalent number of high-fidelity evaluations for both Monte Carlo and Multilevel Monte Carlo increases. However, we see that to achieve the same accuracy, MLMC takes almost 2 orders of magnitude fewer evaluations than Monte Carlo. This is expected since the problem in question shows high correlation between the coarse and fine simulations, which means that MLMC can extract a lot of the same information from low fidelity simulations as it can from high fidelity simulations, requiring fewer equivalent high fidelity evaluations to achieve the same accuracy.