Monte Carlo Mean Estimation Experiment

Daniel Venter - 2592427v

November 5, 2024

1 Introduction

2 Motivation and Background

3 Analytical Mean

We are able to use the analytical mean to verify the output of the simulations that we will conduct below. We are able to do this due to our chosen function and the given scope of the function.
Let $f(x) = \sin^2(x)$.

The expected value $\mathbb{E}[f(x)]$ is given by:

$$\mathbb{E}[f(x)] = \mathbb{E}\left[\sin^2(x)\right]$$

Using the identity $\sin^2(x) = \frac{1-\cos(2x)}{2}$, we have:

$$f(x) = \sin^2(x) = \frac{1 - \cos(2x)}{2}$$

Thus, the expected value becomes:

$$\mathbb{E}[f(x)] = \frac{1}{2} - \frac{1}{2}\mathbb{E}[\cos(2x)]$$

For a normally distributed x with mean 0 and variance $\sigma^2 = 1$, we use the result:

$$\mathbb{E}[\cos(2x)] = e^{-2\sigma^2} = e^{-2}$$

Substituting $\mathbb{E}[\cos(2x)] = e^{-2}$, we get:

$$\mathbb{E}[f(x)] = \frac{1}{2} - \frac{1}{2}e^{-2}$$

This expression can be simplified using the hyperbolic sine function as:

$$\mu = \mathbb{E}[f(x)] = \frac{\sinh(1)}{e}$$

We can then use a simple Python script, utilising the math module to calculate the analytical mean. Code to calculate the analytical mean:

```
import math

AnalyticalMean = math.sinh(1) / math.exp(1)
```

4 Monte Carlo Simulation

We now calculate the value through classical Monte Carlo simulations, the mean μ following this process:

1. Generate a sample of points from the underlying distribution of x. We do this utilising the numpy module.
2. Compute the function value at each point in the sample.
3. Compute the sample mean of the function values.

```
import numpy as np

# Mathematical Function
def func_f(x):
    return np.sin(x) ** 2

numSamples: int = 1000
sampleData: list = np.random.randn(numSamples, 1)

values: list = func_f(sampleData)

MCMean: float = np.mean(values)
```

Below are two histograms, the left indicating the distribution of our random sample points, with the right indicating the values of the function and the given frequency across our simulation.

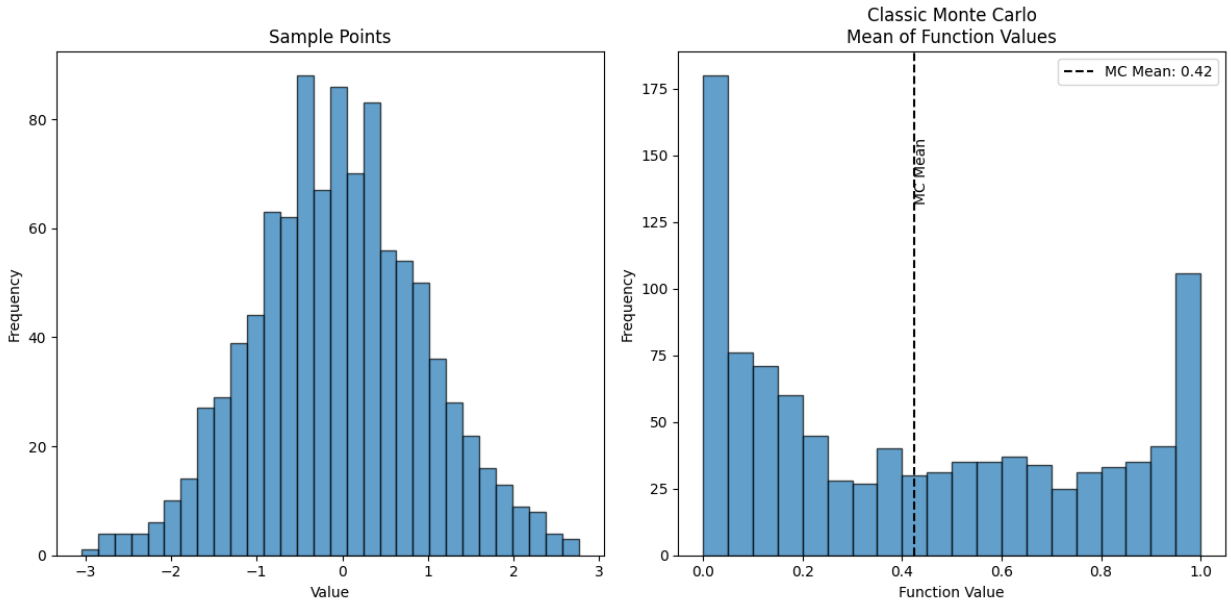

Figure 1: Diagram showing the distribution of the sample points and the Classic Monte Carlo mean distribution.

5 Quantum Monte Carlo Methodology

Given a probability distribution $p(i)$ of dimension $M = 2^m$ for some $m \geq 1$ and a function $f : X \to [0, 1]$ defined on the set of integers $X = \{0, 1, \ldots, M-1\}$, this function implements the algorithm that allows the following expectation value to be estimated:

$$\mu = \sum_{i \in X} p(i) f(i).$$

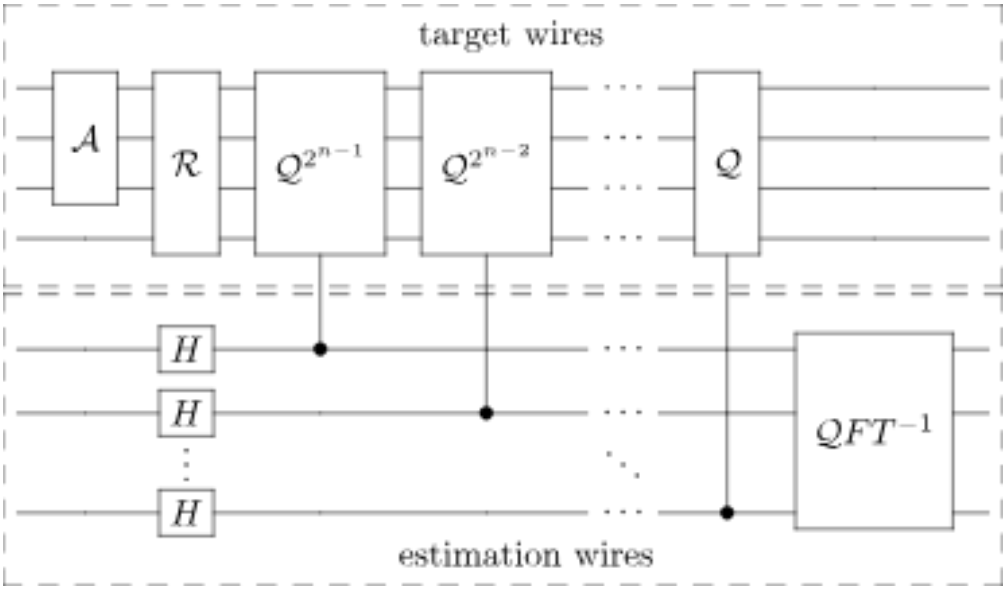

Figure 2: Circuit Diagram for Quantum Monte Carlo

Explanation of the Circuit
First we set some parameters for the quantum simulation. We define the number of qubits for the probability register as 5, this gives us 2^5 values we can encode, and the number of qubits for the estimation as $5 + 1$, one for each initial qubit and 1 to write the final answer to.

Code Block used initialise the data required for the circuit.

```
import pennylane as qml
import numpy as np
from scipy.stats import norm

m = 5
M = 2 ** m

xmax = np.pi  # bound to region [-pi, pi]
xs = np.linspace(-xmax, xmax, M)

probs = np.array([norm().pdf(x) for x in xs])
```

```
probs /= np.sum(probs)

def func(i):
    return np.sin(xs[i]) ** 2
```

We can then use the Quantum Monte Carlo Template by Pennylane to simulate the quantum circuit:

```
n = 10
N = 2 ** n

target_wires = range(m + 1)
estimation_wires = range(m + 1, n + m + 1)

dev = qml.device("default.qubit", wires=(n + m + 1))

@qml.qnode(dev)
def circuit():
    qml.templates.QuantumMonteCarlo(
        probs,
        func,
        target_wires=target_wires,
        estimation_wires=estimation_wires,
    )
    return qml.probs(estimation_wires)

phase_estimated = np.argmax(circuit()[:int(N / 2)]) / N
```

Once the value has been calculated we can use phase estimation to investigate the probabilities of each given value. As evident in Figure 3 there are two possible values with far higher probabilities than the other values. From the figure it is also clear that the value with the higher probability is the value that was estimated.

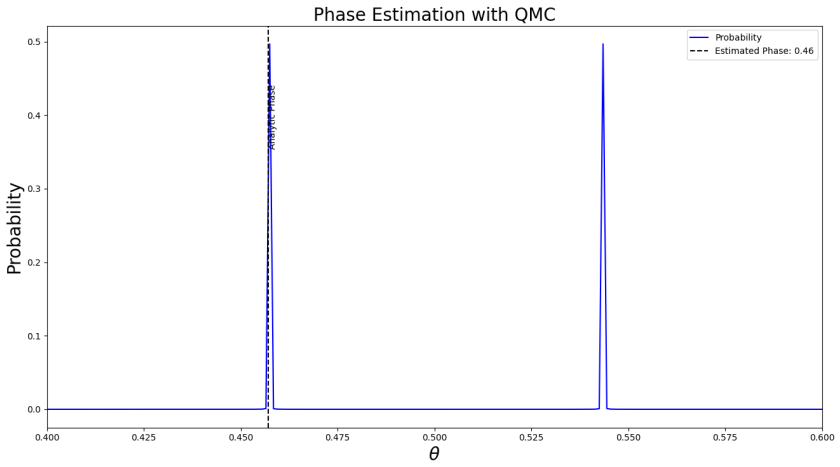

Figure 3: Theta value and probability of value

6 Results

Table of results showing type of calculation method, number of sample required to calculate value if applicable and value rounded to 4 decimal places.

Mean Type	Analytical Mean	Classic Monte Carlo	Discrete Mean	Quantum Monte Carlo
Number Samples	NA	1000	NA	1
Value	0.4323	0.42345	0.4326	0.4327
Absolute Error	0.0	0.00887	0.02053	0.4327
Relative Error	0.0	0.00031	0.00072	0.4327
Mean Squared Estimate	0.0	$7.8776e^{-5}$	$9.6481e^{-8}$	$1.4234e^{-7}$

Table 1: Table of Results for different Methods of Calculating Mean of Given Function

The Classical Monte Carlo simulation's accuracy is heavily dependent on the number of samples.

Mean	Abs Error	Relative Error	MSE	Number Samples
0.9443	0.5119	1.1841	0.2621	1
0.6854	0.2531	0.5854	0.0640	10
0.4249	0.0075	0.0173	0.0001	100
0.4314	0.0010	0.0022	0.0000	1000
0.4334	0.0011	0.0025	0.0000	10000

Table 2: Table of Results for Monte Carlo Sampling Experiment

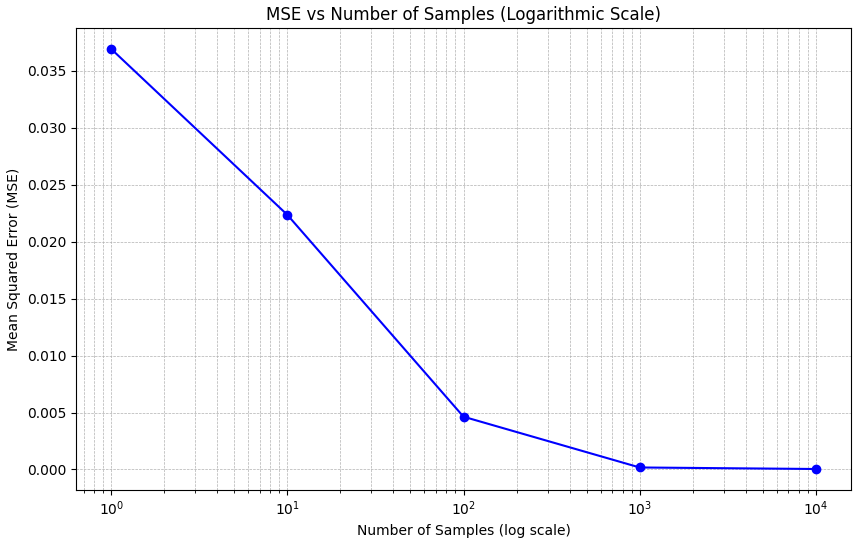

Figure 4: Line Graph inidicating the MSE of Monte Carlo Simulation after number of samples.

7 Conclusion

Given the data in Table 2 and plotted in Figure 4 it is clear that the number of samples has correlation with on the accuracy of the calculation. This is due to as the number of samples increase so do the number of data points taken into account when calculating the mean.
The law of diminishing return is a prevalent as we can see the increase in accuracy between 10^4 and 10^5 samples does not warrant the increased number samples and the computational cost associated with these samples.

8 Appendix

Full Code File with Graph Printing

```
import pennylane as qml
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
import math

np.random.seed(0)

AnalyticalMean = math.sinh(1) / math.exp(1)
print("Analytical Mean", AnalyticalMean)
```

```
def func_f(x):
    return np.sin(x) ** 2

numSamples: int = 1000
sampleData: list = np.random.randn(numSamples, 1)
values: list = func_f(sampleData)
MCMean: float = np.mean(values)

print("MC Mean", MCMean)

fig, axs = plt.subplots(1, 2, figsize=(12, 6))

axs[0].hist(sampleData, bins=30, edgecolor='black', alpha=0.7)
axs[0].set_title("Sample Points")
axs[0].set_xlabel("Value")
axs[0].set_ylabel("Frequency")

axs[1].hist(values, bins=20, edgecolor='black', alpha=0.7)
axs[1].axvline(MCMean, color='black', linestyle='--', label=f'MC Mean: {MCMean:.2f}
axs[1].text(MCMean, max(np.histogram(values, bins=20)[0]) * 0.8, 'MC Mean', rotatio
axs[1].set_title("Classic Monte Carlo\nMean of Function Values")
axs[1].set_xlabel("Function Value")
axs[1].set_ylabel("Frequency")
axs[1].legend()

plt.tight_layout()
plt.show()

m = 5
M = 2 ** m

xmax = np.pi  # bound to region [-pi, pi]
xs = np.linspace(-xmax, xmax, M)

probs = np.array([norm().pdf(x) for x in xs])
probs /= np.sum(probs)

def func(i):
    return np.sin(xs[i]) ** 2

DiscreteMean = np.sum(func_f(xs) * probs)
print("Discrete Mean", DiscreteMean)

n = 10
N = 2 ** n
```

```
target_wires = range(m + 1)
estimation_wires = range(m + 1, n + m + 1)

dev = qml.device("default.qubit", wires=(n + m + 1))

@qml.qnode(dev)
def circuit():
    qml.templates.QuantumMonteCarlo(
        probs,
        func,
        target_wires=target_wires,
        estimation_wires=estimation_wires,
    )
    return qml.probs(estimation_wires)


qmc_probs = circuit()
phase_estimated = np.argmax(qmc_probs[:int(N / 2)]) / N
phase_estimated_value = (1 - np.cos(np.pi * phase_estimated)) / 2
print("Phase Estimated", phase_estimated_value)




theta_values = np.linspace(0, 1, len(qmc_probs))
plt.figure(figsize=(8, 6))
plt.plot(theta_values, qmc_probs, label="Probability", color='blue')

plt.axvline(phase_estimated, color='black', linestyle='--', label=f'Estimated Phase
plt.text(phase_estimated, max(qmc_probs) * 0.8, 'Analytic Phase', rotation=90, vert


plt.title("Phase Estimation with QMC", fontsize = 20)
plt.xlabel(r"$\theta$", fontsize = 20)
plt.xlim(0.4, 0.6)
plt.ylabel("Probability" , fontsize = 20)
plt.legend()


plt.show()
```