# Computer Intensive Methods Final Paper: Weighted Monte Carlo

Vincent Jia
vjia@uwaterloo.ca

December 18, 2024

## Contents

# 1 Introduction

## 1.1 Motivation for Weighted Monte Carlo

In *Large Sample Properties of Weighted Monte Carlo Estimators* [2] by Glasserman and Yu, the authors discuss the idea of Weighted Monte Carlo (WMC) Estimators. From hereafter, we will refer to Weighted Monte Carlo as WMC. In their paper, Glasserman and Yu discuss two cases for WMC. The first case is the "unbiased" case. The second case is the "biased" case. We will only talk about the "unbiased" case in this paper. Using WMC in the "unbiased" case reduces variance when compared to naive Monte Carlo.

Let me explain the intuition behind WMC. I'll speak a bit abstractly. Let us say that we want to sample from some random variable $Y$. Let's say that if we were to directly sample from $Y$, we would get a variance $\sigma^2$. However, if we were to sample from $\omega Y$ where $\omega$ is a "weight", then we could get a variance lower than $\sigma^2$. We will see this idea more concretely in the numerical examples.

## 1.2 Our Implementation Goals

We will replicate part of Glasserman and Yu's results for the unbiased case for WMC. Specifically, we will replicate the case for where $h := \frac{\omega^2}{2}$ where $h$ is an objective function. We call the case where $h = \frac{\omega^2}{2}$ as the "quadratic" case. We will not replicate the cases where $h = -\log(\omega)$ ("log" case) and $h = \omega \log(\omega)$ ("entropy" case). This is not from lack of trying. To apply WMC in the "log" and "entropy" cases involves applying nonlinear least squares to solve a Lagrangian equation. I have tried unsuccesfully to implement nonlinear least squares.

For now, we will apply WMC to pricing an Asian option under arithmetic payoff. We will compare naive MC to WMC when the number of time steps in the Asian Option are $2, 4, 20$ and $40$. In their paper, Glasserman and Yu provide results for when the time steps are $1, 2$ and $4$. They provide results for only these small number of time steps, because as the time steps increase, the benefit to WMC diminishes. However, an Asian option with only 4 time steps is unrealistic. Thus, finding the results for when time steps equal 20 and 40 will be interesting.

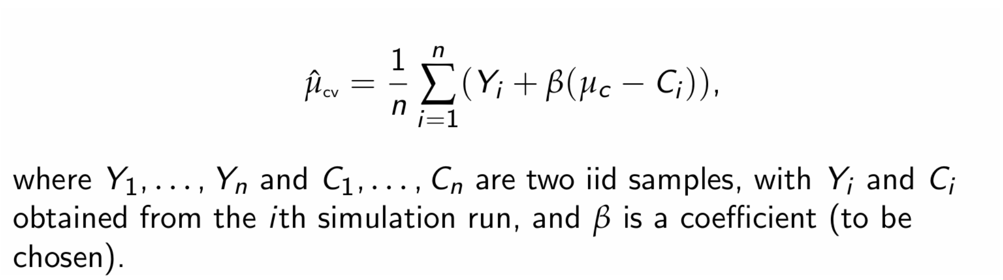
We will also attempt to apply "Array RQMC" on top of MC and WMC. "Array RQMC" is the idea of applying RQMC methods along an array. We will see that using "Array RQMC" in this specific case will actually increase the variance. This runs contrary to our expectation. Perhaps there is an interesting reason why.

# 2 Methodology for Project

## 2.1 WMC Connection to Past Variance Reduction Techniques

We can think of Weighted Monte Carlo as analogous to Control Variates and Importance Sampling.

The definition of Control Variates is:



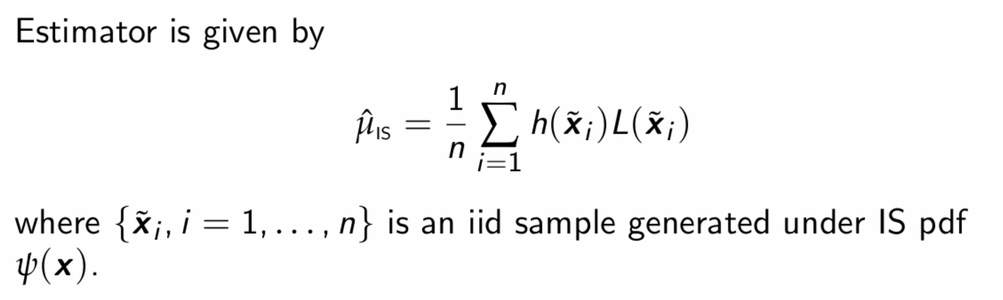
$$\hat{\mu}_{\text{cv}} = \frac{1}{n}\sum_{i=1}^{n}(Y_i + \beta(\mu_c - C_i)),$$

where $Y_1, \ldots, Y_n$ and $C_1, \ldots, C_n$ are two iid samples, with $Y_i$ and $C_i$ obtained from the $i$th simulation run, and $\beta$ is a coefficient (to be chosen).

Here, we are trying to estimate some $E[Y]$. Here, we also know that $E[C] = \mu_c$. So the term $\frac{1}{n}\sum_{i=1}^{n}(\mu_C - C_i)$ has expectation equal to 0. The reason we have the $\beta(\mu_c - C_i)$ term is because if $\text{Cov}(Y, C) > 0$, then if $Y$ goes up, then $-C$ will "drag" $Y$ down. Thus, we reduce the variance. We will a similar set up for the WMC case.

According to Glasserman and Yu, WMC is like control variates, except that WMC forces simulated values from the control variate to agree with the theoretical value, even over a finite number of replications.

From importance sampling, we know the definition is:



Estimator is given by

$$\hat{\mu}_{\text{IS}} = \frac{1}{n}\sum_{i=1}^{n} h(\tilde{\boldsymbol{x}}_i)L(\tilde{\boldsymbol{x}}_i)$$

where $\{\tilde{\boldsymbol{x}}_i, i = 1, \ldots, n\}$ is an iid sample generated under IS pdf $\psi(\boldsymbol{x})$.

Under importance sampling, we try to estimate $\mu = E[h(\boldsymbol{X})]$ where $\boldsymbol{X}$ has density $\phi(x)$. Here, $L(\tilde{x}_i) = \frac{\phi(\tilde{x}_i)}{\psi(\tilde{x}_i)}$. Basically, the idea is not to sample from $\phi(x)$, but to sample from $\psi(\tilde{x}_i)$. By sampling from $\psi(\tilde{x}_i)$, we can possibly get a variance reduction. This is a similar idea to WMC where $\hat{Y}_{WMC} = \frac{1}{n}\sum_{i=1}^{n}\omega_{i,n}Y_i$. Instead of sampling from $Y_i$, we sample from $\omega_{i,n}Y_i$.

## 2.2 WMC in Detail

Now, we will discuss the WMC method in detail. Glasserman and Yu apply WMC in the multivariate case, so we will be working primarily with vectors and matrices.

Below is the setup for WMC:

Now let $h\colon \Re \to \Re \cup \{+\infty\}$ be a strictly convex function and consider the optimization problem

$$\min_{\omega_{1,n},\ldots,\omega_{n,n}} \sum_{i=1}^{n} h(\omega_{i,n}) \tag{5}$$

subject to

$$\frac{1}{n}\sum_{i=1}^{n} \omega_{i,n} = 1, \tag{6}$$

$$\frac{1}{n}\sum_{i=1}^{n} \omega_{i,n} X_i = c_X \tag{7}$$

for some fixed $c_X \in \Re^d$.

Once you find the optimal $\omega_{1,n}...\omega_{n,n}$, you can solve

$$\hat{Y}_{WMC} = \frac{1}{n}\sum_{i=1}^{n} \omega_{i,n} Y_i$$

Let me explain what is going on. We have a nonlinear optimization problem. As before, we have three choices for $h$. These are $\frac{w^2}{2}$, $-\log(w)$ and $\omega\log(\omega)$. Luckily, the case where $h = \frac{w^2}{2}$ has an easy to solve solution. We can think of each $\omega_{i,n}$ as a scalar value with a value around 1. There are $n$ of these $\omega$'s where $i$ goes from 1 to $n$. We want to find the optimal $\omega_{1,n}...\omega_{n,n}$ to minimize $\sum_{i=1}^{n} h(\omega_{i,n})$ while also meeting the two constraints. The two constraints are:

$$\frac{1}{n}\sum_{i=1}^{n} \omega_{i,n} = 1$$

and

$$\frac{1}{n}\sum_{i=1}^{n} \omega_{i,n} X_i = c_X$$

Let us interpret the two constraints. $\frac{1}{n}\sum_{i=1}^{n} \omega_{i,n} = 1$ can be rewritten as $\sum_{i=1}^{n} \omega_{i,n} = n$. We can now say, that the summation of all $\omega_{i,n}$ elements must equal the number of $\omega$'s. Loosely speaking, we can say that we "force" the expectation of $\omega_{i,n}$ to be exactly equal to 1.

The second condition makes more sense once you realize that $X_i$ and $c_X$ are both vectors. $X_i$ has a dimension of $d \times 1$ and $c_X$ is the 0 vector of dimension $d \times 1$. Sometimes, we will refer to $d$ as $m$ to be consistent with the paper. Just know I'm talking about the same thing. Loosely speaking, we can say that we "force" the expectation of $X_i$ to be the 0 vector. In this way, we can think of $X_i$ as like a control variate.

Once you finally have the optimal $\omega_{1,n}...\omega_{n,n}$, you can plug in those optimal $\omega$'s into $\hat{Y}_{WMC} = \frac{1}{n}\sum_{i=1}^{n}\omega_{i,n}Y_i$. In our particular case, we can think of $X_i$ as a stock path for an Asian option, $Y_i$ as the payoff for a particular stock path, and $w_{i,n}$ as some scalar adjusting factor for $Y_i$. By multiplying $Y_i$ by some ideal $\omega_{i,n}$'s, we should be able to reduce the variance of $\hat{Y}_{WMC}$.

## 2.3 Array RQMC in Detail

The concept of Array RQMC is to apply RQMC continuously over a dataset instead of generating all the random point at once. This way, the RQMC points stay "fresh" because sequences like the sobol and halton generate points more effectively at lower dimensions.

Credit to this idea goes to my tutor, Gavin Orok. A formal defintion for Array-RQMC is given below. The definition comes from *Array-RQMC for Option Pricing Under Stochastic Volatility Models* [1] by Abdellah, L'Ecuyer and Puchhammer.

**Algorithm 1** : Array-RQMC Algorithm for Our Setting

```
for $i = 0,\dots,n-1$ do $X_{i,0} \leftarrow x_0$;
for $j = 1,2,\dots,\tau$ do
    Sorting: Compute an appropriate permutation $\pi_j$ of the $n$ chains, based on
        the $h(X_{i,j-1})$, to match the $n$ states with the RQMC points;
    Randomize afresh the RQMC points $\{\mathbf{U}_{0,j},\dots,\mathbf{U}_{n-1,j}\}$;
    for $i = 0,\dots,n-1$ do $X_{i,j} = \varphi_j(X_{\pi_j(i),j-1}, \mathbf{U}_{i,j})$;
return the average $\hat{\mu}_{\text{arqmc},n} = \bar{Y}_n = (1/n)\sum_{i=0}^{n-1} g(X_{i,\tau})$ as an estimate of $\mu_y$.
```

In our particular case, I will make our method more concrete. For the sake of clarity, let's say we have 4 time steps in our Asian option. Let's call this $m = 4$. At time $m = 1$, I will generate a vector of RQMC points with dimension $n \times 1$ (row $\times$ column) where $n$ is the number of paths I want to sample. I will generate these $U$'s where $U \sim Unif(0,1)$ in a deterministic way. Then, I will move onto $m = 2$, and "refresh" my RQMC method by generating another $n \times 1$ vector. Notice how I didn't attempt to generate $n \times m$ points immediately. I will now "concatenate" my $n \times 1$ vector from time $m = 1$ and the $n \times 1$ vector from time $m = 2$ to create a $m \times 2$ matrix. I will continue to generate new $n \times 1$ vectors until time $m = 4$.

I will pass these $U$'s into an inversion function to produce the realized $z$ values of a $N(0,1)$. I will pass these $z$'s into some code which will use GBM to generate a stock price. Then, I will **order the data** in terms from smallest to largest. The idea behind this is to "spread out" the stock paths. This should theoretically reduce the variance compared to standard MC. Then, we can either choose to use the standard MC payoff function, or the weighted WMC payoff function.

## 3 Author's Results

The author uses MC and WMC to price an Asian option using the arithmetic payoff. Specifically, they use all the standard assumptions of generating an Asian option.

The typical conditions are displayed below.

$$Y = e^{-rT} \max\{0, \bar{S} - K\},$$

We want to find discounted arithmetic payoff.

$$\bar{S} = \frac{1}{m} \sum_{j=1}^{m} S(t_j)$$

$\bar{S}$ is the average stock price over the stock path.

$$S(t) = S(0) \exp\left(\left[r - \tfrac{1}{2}\sigma^2\right]t + \sigma W(t)\right),$$

We generate stock paths using GBM.

$$W(t_i) = W(t_{i-1}) + \sqrt{t_i - t_{i-1}} Z_i,$$

We can discretize the stock using Euler approximation.

The most interesting condition they have is their choice of $X_i$. Remember, that $E[X_i] = c_X$ where $c_X$ is the 0 vector of dimension $m \times 1$.

$$S(t_i) - S(0)e^{rt_i}.$$

Our $X_i := S(t_i) - S(0)e^{rt_i}$ Input conditions are $S0 = 50, r = 0.05, \sigma = 0.2, K = 54$ and $T = 5$.

To find $Var(\hat{Y}_{WMC})$, they actually run whatever function that generates "pricing" for the Asian option 100 times. They call these 100 runs as "100 batches". So, we use $n$ simulations and $m$ time steps to price the Asian option 1 time. Then, we use $n$ simulations and $m$ time steps 99 more times. So, we have a vector of length 100 which only contains 100 mean discounted payoff values generated from 100 different MC (or WMC) function runs. Then, we take the variance of that vector to get $Var(\hat{Y}_{WMC})$.

Below is a graph of the author's results.

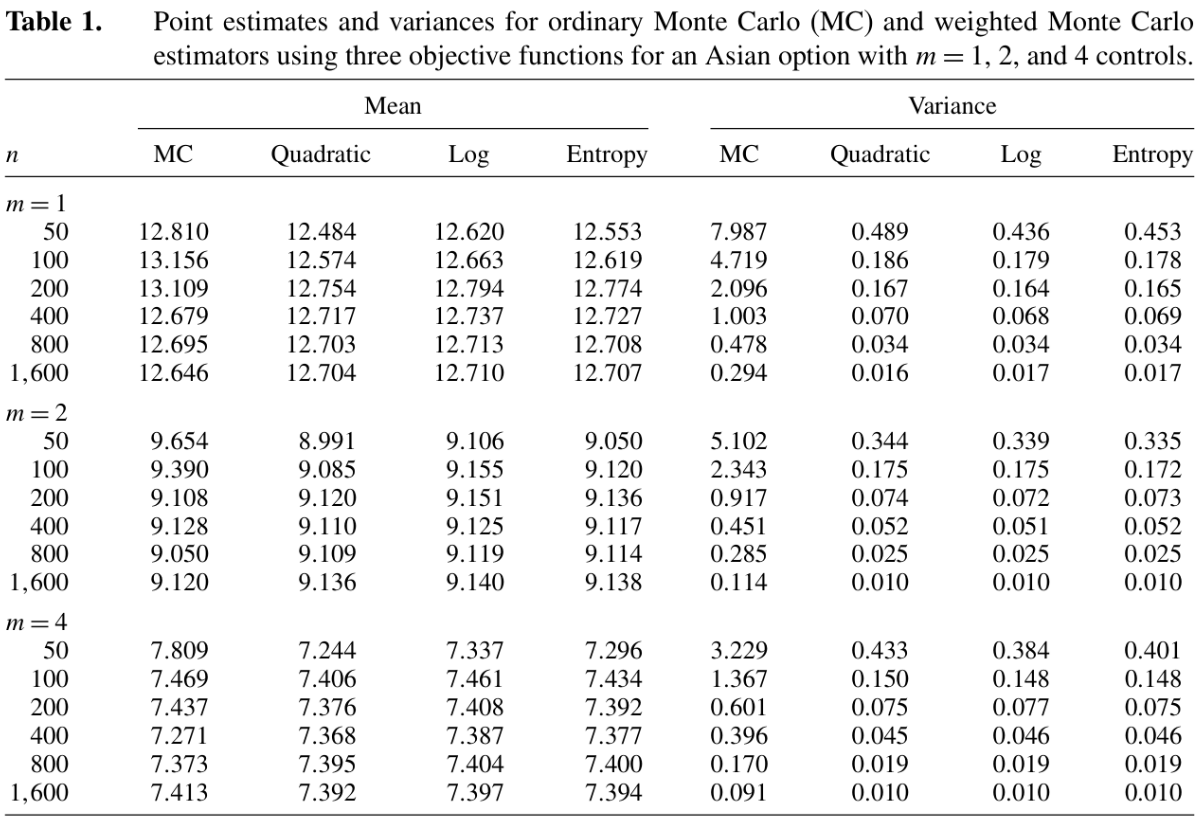


**Table 1.** Point estimates and variances for ordinary Monte Carlo (MC) and weighted Monte Carlo estimators using three objective functions for an Asian option with $m = 1$, 2, and 4 controls.

| | Mean | | | | Variance | | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | MC | Quadratic | Log | Entropy | MC | Quadratic | Log | Entropy |
| $m = 1$ | | | | | | | | |
| 50 | 12.810 | 12.484 | 12.620 | 12.553 | 7.987 | 0.489 | 0.436 | 0.453 |
| 100 | 13.156 | 12.574 | 12.663 | 12.619 | 4.719 | 0.186 | 0.179 | 0.178 |
| 200 | 13.109 | 12.754 | 12.794 | 12.774 | 2.096 | 0.167 | 0.164 | 0.165 |
| 400 | 12.679 | 12.717 | 12.737 | 12.727 | 1.003 | 0.070 | 0.068 | 0.069 |
| 800 | 12.695 | 12.703 | 12.713 | 12.708 | 0.478 | 0.034 | 0.034 | 0.034 |
| 1,600 | 12.646 | 12.704 | 12.710 | 12.707 | 0.294 | 0.016 | 0.017 | 0.017 |
| $m = 2$ | | | | | | | | |
| 50 | 9.654 | 8.991 | 9.106 | 9.050 | 5.102 | 0.344 | 0.339 | 0.335 |
| 100 | 9.390 | 9.085 | 9.155 | 9.120 | 2.343 | 0.175 | 0.175 | 0.172 |
| 200 | 9.108 | 9.120 | 9.151 | 9.136 | 0.917 | 0.074 | 0.072 | 0.073 |
| 400 | 9.128 | 9.110 | 9.125 | 9.117 | 0.451 | 0.052 | 0.051 | 0.052 |
| 800 | 9.050 | 9.109 | 9.119 | 9.114 | 0.285 | 0.025 | 0.025 | 0.025 |
| 1,600 | 9.120 | 9.136 | 9.140 | 9.138 | 0.114 | 0.010 | 0.010 | 0.010 |
| $m = 4$ | | | | | | | | |
| 50 | 7.809 | 7.244 | 7.337 | 7.296 | 3.229 | 0.433 | 0.384 | 0.401 |
| 100 | 7.469 | 7.406 | 7.461 | 7.434 | 1.367 | 0.150 | 0.148 | 0.148 |
| 200 | 7.437 | 7.376 | 7.408 | 7.392 | 0.601 | 0.075 | 0.077 | 0.075 |
| 400 | 7.271 | 7.368 | 7.387 | 7.377 | 0.396 | 0.045 | 0.046 | 0.046 |
| 800 | 7.373 | 7.395 | 7.404 | 7.400 | 0.170 | 0.019 | 0.019 | 0.019 |
| 1,600 | 7.413 | 7.392 | 7.397 | 7.394 | 0.091 | 0.010 | 0.010 | 0.010 |

We will replicate their results for the MC and Quadratic cases for $m = 2, 4, 20, 40$ and $n = 50, 100, 200, 400, 800, 1600$. We will set our seed to equal 1.

We will also use Array RQMC + MC, and Array RQMC + Quadratic to find mean and variance.

## 4 Implementation of Unbiased Weighted Monte Carlo

### 4.1 Naive MC

The implementation is just standard Monte Carlo. We replicate the author's steps. The table is below.

The results from this table are consistent with our expectations. Doing a quick comparison with Glasserman and Yu's results, we see that when $m = 2$, the estimator mean is around 9 which is consistent. When $m = 4$, the estimator mean is around 7.4 which is consistent. Note that there may be significant fluctuation in means for lower values of $n$.

If we check the variances of the estimator for $m = 4$ and $n = 1600$, the value of Glasserman is 0.091 when compared to our value of 0.0744. Thus, we can be fairly confident of our results.

As $m$ goes up for fixed $n$, it brings down the estimator mean. It also seems to bring down the estimator variance.

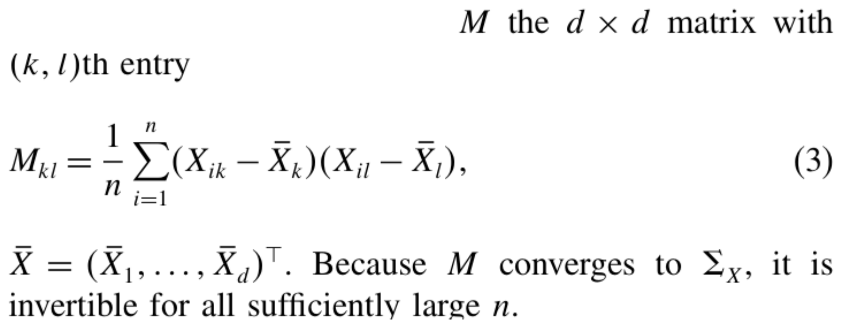
Table 1: Estimator Variance and Mean for MC Naive

| m | n | Estimator Variance | Estimator Mean |
|---|---|---|---|
| 2 | 50 | 3.79001955 | 9.168272 |
| 2 | 100 | 2.05900847 | 9.138770 |
| 2 | 200 | 0.98370064 | 8.989967 |
| 2 | 400 | 0.43450794 | 9.173137 |
| 2 | 800 | 0.27266944 | 9.123541 |
| 2 | 1600 | 0.10845126 | 9.110839 |
| 4 | 50 | 2.79219431 | 7.416072 |
| 4 | 100 | 1.12651593 | 7.389664 |
| 4 | 200 | 0.54029758 | 7.201625 |
| 4 | 400 | 0.31464199 | 7.420949 |
| 4 | 800 | 0.16998806 | 7.428411 |
| 4 | 1600 | 0.07447959 | 7.421046 |
| 20 | 50 | 1.69244885 | 5.957413 |
| 20 | 100 | 1.04393688 | 6.197413 |
| 20 | 200 | 0.45737854 | 5.882276 |
| 20 | 400 | 0.21851138 | 6.094708 |
| 20 | 800 | 0.09194367 | 6.047816 |
| 20 | 1600 | 0.04687347 | 6.069586 |
| 40 | 50 | 1.45060982 | 5.559180 |
| 40 | 100 | 0.97422895 | 5.739095 |
| 40 | 200 | 0.38003939 | 5.948968 |
| 40 | 400 | 0.17581510 | 5.880639 |
| 40 | 800 | 0.08424982 | 5.872872 |
| 40 | 1600 | 0.04660557 | 5.883458 |

As $n$ goes up for fixed $m$, it brings down the estimator variance. The estimator mean seems to stay the same for fixed $m$.

## 4.2 Quadratic WMC

We can implement Quadratic WMC as given by the paper. To implement Quadratic WMC, we need to get two pieces of definition from the paper.

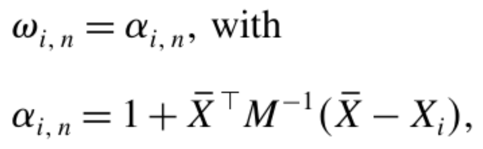


$M$ the $d \times d$ matrix with $(k, l)$th entry

$$M_{kl} = \frac{1}{n}\sum_{i=1}^{n}(X_{ik} - \bar{X}_k)(X_{il} - \bar{X}_l), \tag{3}$$

$\bar{X} = (\bar{X}_1, \ldots, \bar{X}_d)^\top$. Because $M$ converges to $\Sigma_X$, it is invertible for all sufficiently large $n$.

and



$$\omega_{i,n} = \alpha_{i,n}, \text{ with}$$

$$\alpha_{i,n} = 1 + \bar{X}^\top M^{-1}(\bar{X} - X_i),$$

.

We can think of $M$ as the population covariance matrix of $X$. $X$ is defined as

$$\begin{pmatrix} X_1^T \\ X_2^T \\ \ldots \\ X_n^T \end{pmatrix}$$

Remember that $X_i$ has dimension $d \times 1$. $X$ has dimension $n \times d$.

We can also think of $\bar{X}$ as

$$\begin{pmatrix} \bar{X}_1 \\ \bar{X}_2 \\ \ldots \\ \bar{X}_d \end{pmatrix}$$

which has dimension $d \times 1$.

We can think of $\alpha_{i,n}$ as the optimal weights $\omega_{i,n}$ Thus, $\alpha_{i,n}$ is a scalar. $\alpha_{i,n} = 1 + (1 \times d)(d \times d)((d \times 1) - (d \times 1)) = 1 + (1 \times 1)$

The reason why we know that $\alpha_{i,n}$ is the optimal weights is because Glasserman and Yu have already solved the Lagrangian ahead of time. The full Lagrangian is below:

$$L_n = \sum_{i=1}^{n} h(\omega_{i,n}) + \lambda_n \left( n - \sum_{i=1}^{n} \omega_{i,n} \right) + \mu_n^\top \left( nc_X - \sum_{i=1}^{n} \omega_{i,n} X_i \right),$$

Once we know the optimal weights $\omega_{i,n}$ we can multiply them with $Y_i$ to get a variance reduction for $\hat{Y}_{WMC}$. This is exactly what happens. We will produce the table of our results.

Table 2: Estimator Variance and Mean for WMC Quadratic

| m | n | Estimator Variance | Estimator Mean |
|---|---|---|---|
| 2 | 50 | 0.405731368 | 9.112102 |
| 2 | 100 | 0.173372702 | 9.045266 |
| 2 | 200 | 0.086855009 | 9.102073 |
| 2 | 400 | 0.045322944 | 9.105772 |
| 2 | 800 | 0.022382942 | 9.129274 |
| 2 | 1600 | 0.013399178 | 9.124070 |
| 4 | 50 | 0.322233264 | 7.170073 |
| 4 | 100 | 0.134517246 | 7.288438 |
| 4 | 200 | 0.077816520 | 7.318689 |
| 4 | 400 | 0.033249194 | 7.367866 |
| 4 | 800 | 0.022335230 | 7.383396 |
| 4 | 1600 | 0.008883326 | 7.397125 |
| 20 | 50 | 0.478766310 | 5.804806 |
| 20 | 100 | 0.144027107 | 5.955799 |
| 20 | 200 | 0.075986750 | 5.975846 |
| 20 | 400 | 0.027742546 | 6.030832 |
| 20 | 800 | 0.014505018 | 6.016559 |
| 20 | 1600 | 0.006418078 | 6.042081 |
| 40 | 50 | 1.529387619 | 5.530129 |
| 40 | 100 | 0.177362166 | 5.622770 |
| 40 | 200 | 0.068007583 | 5.815830 |
| 40 | 400 | 0.027743799 | 5.815602 |
| 40 | 800 | 0.013004473 | 5.861808 |
| 40 | 1600 | 0.007161703 | 5.871090 |

When comparing the results with the author's results, we see that the results are consistent. Under Quadratic, when $m = 2$, the mean should be around 9. When $m = 4$, the mean should be around 7.4.

More importantly, we see for the same $m$ and $n$, we get a variance reduction when compared to MC. For example, in my own results, for $m = 2, n = 50$ for the MC case, my variance is 3.79001955. In Quadratic WMC, for $m = 2, n = 50$, the variance is 0.405731368. That's a 9.34 times improvement in variance reduction!

The numbers I have are consistent with Glasserman's numbers. If we check for $m = 4, n = 1600$, Glasserman has a variance of 0.010. I have a variance of 0.0088. I can be confident that our results are correct.

Once again, as $m$ increases for fixed $n$, our variance decreases. As $n$ increases for fixed $m$, our variance decreases.

# 5 Implementation of Array RQMC

In this part, we attempt to use Array RQMC on both naive MC and Quadratic WMC to see if we get a variance reduction. Unfortunately, it seems we actually get a significant variance increase. It also seems our mean increases as well.

## 5.1 Array RQMC for Naive MC

The methodology that I used to implement Array RQMC is previousy discussed under Section 2.3. Basically, the idea is that if we use RQMC to generate new $U$ values (which we convert to $z$ values) for us at each $m$, generate stock paths by GBM, and then sort the states (the stock price) by order statistic, then we should theoretically get a variance reduction. However, this is not the case. Nevertheless, the table is provided below.

We can see that we have a dramatic rise in variance. The variance is even worse than undre standard monte carlo. For $m = 2, m = 50$ our estimator variance under Array RQMC is 13.4755. Under standard monte carlo it was 3.79001955. That is a 3.5 times increase in variance.

Needless to say, this is not a promising result. Also, the estimator means are higher across the board. Under Quadratic WMC, when $m = 2$, the mean is around 9. When $m = 4$, the mean is around 7.2. When $m = 20$ under Quadratic WMC, the mean should be around 6. When $m = 40$, the mean should be around 5.8.

However, for every value of $m$, the estimator mean for the Array RQMC case is too high when compared to Quadratic WMC.

Still, we notice some results consistent with the previous two cases. When $m$ increases for a fixed $n$, the mean goes down. Also, generally, as $n$ increases for a fixed $m$, the variance goes down. However, it does not look like increasing $m$ for a fixed $n$ brings down the variance any more.

Table 3: Estimator Variance and Mean for Array RQMC for Naive MC

| m | n | Estimator Variance | Estimator Mean |
|---|---|---|---|
| 2 | 50 | 13.475547 | 12.185765 |
| 2 | 100 | 15.373606 | 12.352895 |
| 2 | 200 | 14.925381 | 11.931201 |
| 2 | 400 | 13.407273 | 12.131953 |
| 2 | 800 | 13.014995 | 11.950640 |
| 2 | 1600 | 12.722616 | 11.402868 |
| 4 | 50 | 18.956665 | 10.598926 |
| 4 | 100 | 16.356938 | 10.410837 |
| 4 | 200 | 15.394073 | 10.442938 |
| 4 | 400 | 14.001324 | 9.736835 |
| 4 | 800 | 17.841461 | 9.979602 |
| 4 | 1600 | 14.824843 | 10.210608 |
| 20 | 50 | 14.524741 | 8.579490 |
| 20 | 100 | 14.970256 | 8.356536 |
| 20 | 200 | 18.470329 | 8.645753 |
| 20 | 400 | 9.793831 | 7.978929 |
| 20 | 800 | 17.168708 | 8.775239 |
| 20 | 1600 | 20.321972 | 8.924531 |
| 40 | 50 | 14.173867 | 8.195568 |
| 40 | 100 | 10.995356 | 7.908746 |
| 40 | 200 | 15.881786 | 8.497024 |
| 40 | 400 | 15.958305 | 7.955584 |
| 40 | 800 | 14.235283 | 8.330343 |
| 40 | 1600 | 11.658487 | 7.796541 |

## 5.2 Array RQMC for Quadratic WMC

We will also show the results when applying Array RQMC to the Quadratic WMC case. The results here are interesting, as they display some strange behaviour not seen in the previous 3 cases.

The results from Table 4 are in complete contrast to the previous three tables. As both $m$ and $n$ increase, the mean seems to be approaching 0. Also, the scale of the variance parameters is at a much higher level than even Table 3. For $m = 2, n = 50$, we see the estimated variance is 1824!

Table 4: Estimator Variance and Mean for Array RQMC for Quadratic WMC

| m | n | Estimator Variance | Estimator Mean |
|---|---|---|---|
| 2 | 50 | 1824.360003 | 15.612299 |
| 2 | 100 | 17.029395 | 10.506377 |
| 2 | 200 | 10.922693 | 10.872469 |
| 2 | 400 | 9.092345 | 10.895261 |
| 2 | 800 | 9.069794 | 11.382139 |
| 2 | 1600 | 7.745692 | 10.526334 |
| 4 | 50 | 14.771005 | 8.510381 |
| 4 | 100 | 18.499830 | 8.339080 |
| 4 | 200 | 100.624947 | 9.640378 |
| 4 | 400 | 15.588097 | 8.151835 |
| 4 | 800 | 26.541303 | 7.470421 |
| 4 | 1600 | 56.092048 | 8.620066 |
| 20 | 50 | 141.420819 | 7.112579 |
| 20 | 100 | 14.425798 | 5.316447 |
| 20 | 200 | 12.752759 | 4.107030 |
| 20 | 400 | 24.354169 | 3.928106 |
| 20 | 800 | 42.859040 | 4.077186 |
| 20 | 1600 | 33.169460 | 3.588980 |
| 40 | 50 | 17.743436 | 6.189155 |
| 40 | 100 | 16.886376 | 4.577776 |
| 40 | 200 | 14.586074 | 3.987503 |
| 40 | 400 | 24.981329 | 3.450903 |
| 40 | 800 | 26.086711 | 2.021397 |
| 40 | 1600 | 40.205280 | 1.311672 |

There also seems to no strong relationship between $m$, $n$ and the estimator variance. If you look carefully, it seems that variance decreases as $m$ increases. However, for the same $m$, increasing the $n$ does not necesarily decrease the variance. The variance at $m = 4, n = 800$ is 26.5 while at $m = 4, n = 1600$ the variance increases to 56.09.

Overall, the results from this combination of Array RQMC and Quadratic WMC are not very promising.

## 6 Conclusion

We have looked at *Large Sample Properties of Weighted Monte Carlo Estimators* and applied the technique of WMC. We have seen for the case where we go from naive MC to Quadratic WMC, we get a significant variance reduction.

We have also looked at applying Array RQMC. However, the results are not promising because they seem to increase both the mean and variance. More study is need to see if applying Array RQMC can reduce variance in my specific circumstance, or if my implementation was flawed.

## References

[1] Amal Abdellah, Pierre L'Ecuyer, and Florian Puchhammer. Array-rqmc for option pricing under stochastic volatility models. *Proceedings of the 2019 Winter Simulation Conference*, 2019.

[2] Paul Glasserman and Bin Yu. Large sample properties of weighted monte carlo estimators. *Operations Research*, 2005.

# Clean Final Project V3

Vincent Jia

2024-12-19

```r
# clears the environment
rm(list = ls())
```

## Naive MC

```r
calculate_asian_estimator_statistics_mc <- function(batches = 100, n = 1600,
                                                    S0 = 50, r = 0.05, sigma = 0.2,
                                                    K = 54, T = 5, m = 4)
{
  calculate_asian_price_mc <- function(n, S0, r, sigma, K, T, m)
  {
    delta <- T / m
    payoffs <- numeric(n)

    for (i in 1:n)
    {
      Z <- rnorm(m)
      stockpath <- numeric(m)
      stockpath[1] <- S0 * exp((r - 0.5 * sigma^2) * delta + sigma * sqrt(delta) * Z[1])

      for (j in 2:m)
      {
        stockpath[j] <- stockpath[j-1] * exp((r - 0.5 * sigma^2) * delta +
                                             sigma * sqrt(delta) * Z[j])
      }
      average_price <- mean(stockpath)
      payoffs[i] <- exp(-r * T) * max(average_price - K, 0)
    }
    return(mean(payoffs))
  }

  sample_means <- numeric(batches)
  for (i in 1:batches) {
    sample_means[i] <- calculate_asian_price_mc(n, S0, r, sigma, K, T, m)
  }

  estimator_variance <- var(sample_means)
  estimator_mean <- mean(sample_means)

  return(list(
    estimator_variance = estimator_variance,
    estimator_mean = estimator_mean))
}
```

```r
set.seed(1)
m_values <- c(2, 4, 20, 40)
n_values <- c(50, 100, 200, 400, 800, 1600)

# Initialize an empty data frame to store results
my_df1 <- data.frame(m = numeric(), n = numeric(), estimator_variance = numeric(),
                     estimator_mean = numeric())

for (m in m_values)
{
  for (n in n_values)
  {
    stats <- calculate_asian_estimator_statistics_mc(m = m, n = n)
    my_df1 <- rbind(my_df1,
                    data.frame(m = m,
                               n = n,
                               estimator_variance = stats$estimator_variance,
                               estimator_mean = stats$estimator_mean))
  }
}

# Print the results
print(my_df1)
```

```
##     m    n estimator_variance estimator_mean
## 1   2   50         3.79001955       9.168272
## 2   2  100         2.05900847       9.138770
## 3   2  200         0.98370064       8.989967
## 4   2  400         0.43450794       9.173137
## 5   2  800         0.27266944       9.123541
## 6   2 1600         0.10845126       9.110839
## 7   4   50         2.79219431       7.416072
## 8   4  100         1.12651593       7.389664
## 9   4  200         0.54029758       7.201625
## 10  4  400         0.31464199       7.420949
## 11  4  800         0.16998806       7.428411
## 12  4 1600         0.07447959       7.421046
## 13 20   50         1.69244885       5.957413
## 14 20  100         1.04393688       6.197413
## 15 20  200         0.45737854       5.882276
## 16 20  400         0.21851138       6.094708
## 17 20  800         0.09194367       6.047816
## 18 20 1600         0.04687347       6.069586
## 19 40   50         1.45060982       5.559180
## 20 40  100         0.97422895       5.739095
## 21 40  200         0.38003939       5.948968
## 22 40  400         0.17581510       5.880639
## 23 40  800         0.08424982       5.872872
## 24 40 1600         0.04660557       5.883458
```

## Quadratic WMC

```r
calculate_asian_estimator_statistics_quadratic <- function(batches = 100,
                                                           n = 1600, S0 = 50, r = 0.05,
                                                           sigma = 0.2, K = 54, T = 5, m = 4) {
  calculate_asian_price_quadratic <- function(S0, r, K, T, sigma, n, m) {
    delta <- T / m

    # Initialize all matrices
    stockpaths_matrix <- matrix(NA, nrow = n, ncol = m)
    risk_free_growth_matrix <- matrix(NA, nrow = n, ncol = m)
    control_variate_matrix <- matrix(NA, nrow = n, ncol = m)
    payoffs <- numeric(n)

    for (sample_index in 1:n) {
      Z <- rnorm(m)
      stockpath <- numeric(m)
      stockpath[1] <- S0 * exp((r - (0.5 * sigma^2)) * delta + sigma * sqrt(delta) * Z[1])

      for (time_step in 2:m) {
        stockpath[time_step] <- stockpath[time_step - 1] * exp((r - (0.5 * sigma^2)) * delta
                                                               + sigma * sqrt(delta) * Z[time_step])
      }
      stockpaths_matrix[sample_index, ] <- stockpath
      average_price <- mean(stockpath)
      payoffs[sample_index] <- exp(-r * T) * max(average_price - K, 0)
    }

    # Risk-free growth paths
    time_vector <- seq(delta, T, by = delta)
    risk_free_growth_path <- S0 * exp(r * time_vector)
    risk_free_growth_matrix <- matrix(rep(risk_free_growth_path, times = n),
                                      nrow = n, byrow = TRUE)

    # Control Variate Matrix
    control_variate_matrix <- stockpaths_matrix - risk_free_growth_matrix

    # Population covariance matrix and mean vector
    pop_cov_matrix <- cov(control_variate_matrix) * (nrow(control_variate_matrix) - 1) /
      nrow(control_variate_matrix)
    mean_pop_cov_vec_untransposed <- colMeans(control_variate_matrix)
    mean_pop_cov_vec <- matrix(mean_pop_cov_vec_untransposed, nrow = m, ncol = 1)

    # Optimal weights
    optimal_weights <- numeric(n)
    inv_pop_cov_matrix <- solve(pop_cov_matrix)

    for (sample_index in 1:n) {
      X_i <- matrix(control_variate_matrix[sample_index, ], nrow = m, ncol = 1)
      optimal_weights[sample_index] <- 1 + mean_pop_cov_vec_untransposed %*%
        inv_pop_cov_matrix %*% (mean_pop_cov_vec - X_i)
    }
    weighted_payoffs <- optimal_weights * payoffs
    return(mean(weighted_payoffs))
  }

  sample_means <- numeric(batches)
  for (batch_index in 1:batches) {
    sample_means[batch_index] <- calculate_asian_price_quadratic(S0, r, K, T, sigma, n, m)
  }
  estimator_variance <- var(sample_means)
  estimator_mean <- mean(sample_means)

  return(list(
    estimator_mean = estimator_mean,
    estimator_variance = estimator_variance
  ))
}
```

```r
set.seed(1)
m_values <- c(2, 4, 20, 40)
n_values <- c(50, 100, 200, 400, 800, 1600)

# Initialize an empty data frame to store results
my_df2 <- data.frame(m = numeric(), n = numeric(), estimator_variance = numeric(),
                     estimator_mean = numeric())

for (m in m_values)
{
  for (n in n_values)
  {
    stats <- calculate_asian_estimator_statistics_quadratic(m = m, n = n)
    my_df2 <- rbind(my_df2,
                    data.frame(m = m,
                               n = n,
                               estimator_variance = stats$estimator_variance,
                               estimator_mean = stats$estimator_mean))
  }
}

# Print the results
print(my_df2)
```

```
##     m    n estimator_variance estimator_mean
## 1   2   50        0.405731368       9.112102
## 2   2  100        0.173372702       9.045266
## 3   2  200        0.086855009       9.102073
## 4   2  400        0.045322944       9.105772
## 5   2  800        0.022382942       9.129274
## 6   2 1600        0.013399178       9.124070
## 7   4   50        0.322233264       7.170073
## 8   4  100        0.134517246       7.288438
## 9   4  200        0.077816520       7.318689
## 10  4  400        0.033249194       7.367866
## 11  4  800        0.022335230       7.383396
## 12  4 1600        0.008883326       7.397125
## 13 20   50         0.478766310       5.804806
## 14 20  100         0.144027107       5.955799
## 15 20  200         0.075986750       5.975846
## 16 20  400         0.027742546       6.030832
## 17 20  800         0.014505018       6.016559
## 18 20 1600         0.006418078       6.042081
## 19 40   50         1.529387619       5.530129
## 20 40  100         0.177362166       5.622770
## 21 40  200         0.068007583       5.815830
## 22 40  400         0.027743799       5.815602
## 23 40  800         0.013004473       5.861808
## 24 40 1600         0.007161703       5.871090
```

## Array RQMC for Naive MC

```r
calculate_asian_estimator_statistics_mc_rqmc <- function(batches = 100, S0 = 50,
                                                         r=0.05, sigma=0.2, T=5, K=54,
                                                         n=1600, m=4) {

  calculate_asian_price_mc_rqmc <- function(S0, r, sigma, T, K, n, m) {
    delta <- T / m  # Time increment


    full_rqmc_matrix <- matrix(NA, nrow = n, ncol = m)
    for (i in 1:m) {
      u <- ghalton(n, d = 1, method = "generalized")
      full_rqmc_matrix[, i] <- u
    }

    z_matrix <- qnorm(full_rqmc_matrix)

    stockpaths_matrix <- matrix(NA, nrow = n, ncol = m + 1)
    stockpaths_matrix[, 1] <- S0 # Set initial stock price

    # Calculate stock paths
    for (i in 1:n) {
      for (j in 2:(m + 1)) {
        stockpaths_matrix[i, j] <- stockpaths_matrix[i, j - 1] * exp(
          (r - 0.5 * sigma^2) * delta + sigma * sqrt(delta) * z_matrix[i, j - 1]
        )
      }
    }

    # Remove the initial column for calculations
    stockpaths_matrix <- stockpaths_matrix[, -1]
    # sort matrix by column
    stockpaths_matrix_sorted <- apply(stockpaths_matrix, 2, sort)

    row_averages <- rowMeans(stockpaths_matrix_sorted)
    payoffs <- pmax(row_averages - K, 0)
    mean_payoff <- mean(payoffs)
    return(mean_payoff)
  }

  # Initialize a vector to store results
  sample_means <- numeric(batches)

  # Call the calculate function `batches` times
  for (i in 1:batches) {
    sample_means[i] <- calculate_asian_price_mc_rqmc(S0, r, sigma, T, K, n, m)
  }

  estimator_variance = var(sample_means)
  estimator_mean = mean(sample_means)
  return(list(estimator_variance = estimator_variance,
           estimator_mean = estimator_mean))
}
```

```
set.seed(1)
m_values <- c(2, 4, 20, 40)
n_values <- c(50, 100, 200, 400, 800, 1600)

# Initialize an empty data frame to store results
my_df3 <- data.frame(m = numeric(), n = numeric(), estimator_variance = numeric(),
                     estimator_mean = numeric())

for (m in m_values)
{
  for (n in n_values)
  {
    stats <- calculate_asian_estimator_statistics_mc_rqmc(m = m, n = n)
    my_df3 <- rbind(my_df3,
                    data.frame(m = m,
                               n = n,
                               estimator_variance = stats$estimator_variance,
                               estimator_mean = stats$estimator_mean))
  }
}

# Print the results
print(my_df3)
```

```
##     m    n estimator_variance estimator_mean
## 1   2   50          13.475547      12.185765
## 2   2  100          15.373606      12.352895
## 3   2  200          14.925381      11.931201
## 4   2  400          13.407273      12.131953
## 5   2  800          13.014995      11.950640
## 6   2 1600          12.722616      11.402868
## 7   4   50          18.956665      10.598926
## 8   4  100          16.356938      10.410837
## 9   4  200          15.394073      10.442938
## 10  4  400          14.001324       9.736835
## 11  4  800          17.841461       9.979602
## 12  4 1600          14.824843      10.210608
## 13 20   50          14.524741       8.579490
## 14 20  100          14.970256       8.356536
## 15 20  200          18.470329       8.645753
## 16 20  400           9.793831       7.978929
## 17 20  800          17.168708       8.775239
## 18 20 1600          20.321972       8.924531
## 19 40   50          14.173867       8.195568
## 20 40  100          10.995356       7.908746
## 21 40  200          15.881786       8.497024
## 22 40  400          15.958305       7.955584
## 23 40  800          14.235283       8.330343
## 24 40 1600          11.658487       7.796541
```

## Array RQMC for Quadratic WMC

```r
calculate_asian_estimator_statistics_quadratic_rqmc <- function(batches = 100,
                                                               n = 1600, S0 = 50,
                                                               r = 0.05, sigma = 0.2,
                                                               K = 54, T = 5, m = 4)
{
  calculate_asian_price_quadratic_rqmc <- function(n, S0, r, sigma, K, T, m)
  {
    delta <- T / m  # Time increment
    full_rqmc_matrix <- matrix(NA, nrow = n, ncol = m)

    for (i in 1:m){
      u <-  ghalton(n, d = 1, method = "generalized")
      full_rqmc_matrix[, i] <- u
    }

    z_matrix <- qnorm(full_rqmc_matrix)

    stockpaths_matrix <- matrix(NA, nrow = n, ncol = m + 1)
    stockpaths_matrix[, 1] <- S0 # Set initial stock price

    for (i in 1:n) {
      for (j in 2:(m + 1)) {
      stockpaths_matrix[i, j] <- stockpaths_matrix[i, j - 1] * exp(
        (r - 0.5 * sigma^2) * delta + sigma * sqrt(delta) * z_matrix[i, j - 1])
      }
    }
    # Remove the initial column for calculations
    stockpaths_matrix <- stockpaths_matrix[, -1]
    # sort matrix by column
    stockpaths_matrix_sorted <- apply(stockpaths_matrix, 2, sort)


    # calculate payoff on sorted stockpaths
    row_averages <- rowMeans(stockpaths_matrix_sorted)
    payoffs <- pmax(row_averages - K, 0)
    time_vector <- seq(delta, T, by = delta)
    risk_free_growth_path <- S0 * exp(r * time_vector)
    risk_free_growth_matrix <- matrix(rep(risk_free_growth_path, times = n), nrow = n, byrow = TRUE)

    # Control Variate Matrix
    # stockpaths_matrix_sorted
    control_variate_matrix <- stockpaths_matrix_sorted - risk_free_growth_matrix

    # Population covariance matrix and mean vector
    pop_cov_matrix <- cov(control_variate_matrix) * (nrow(control_variate_matrix) - 1) / nrow(control_va
    mean_pop_cov_vec_untransposed <- colMeans(control_variate_matrix)
    mean_pop_cov_vec <- matrix(mean_pop_cov_vec_untransposed, nrow = m, ncol = 1)

    # Optimal weights
    optimal_weights <- numeric(n)
    inv_pop_cov_matrix <- solve(pop_cov_matrix)

    for (sample_index in 1:n) {
      X_i <- matrix(control_variate_matrix[sample_index, ], nrow = m, ncol = 1)
      optimal_weights[sample_index] <- 1 + mean_pop_cov_vec_untransposed %*%
        inv_pop_cov_matrix %*% (mean_pop_cov_vec - X_i)
    }
    weighted_payoffs <- optimal_weights * payoffs
    return(mean(weighted_payoffs))
  }

  sample_means <- numeric(batches)

  for (i in 1:batches) {
    sample_means[i] <- calculate_asian_price_quadratic_rqmc(n, S0, r, sigma, K, T, m)
  }

  estimator_variance = var(sample_means)
  estimator_mean = mean(sample_means)
  return(list(estimator_variance = estimator_variance,
           estimator_mean = estimator_mean))


}
```

```
set.seed(1)
m_values <- c(2, 4, 20, 40)
n_values <- c(50, 100, 200, 400, 800, 1600)

# Initialize an empty data frame to store results
my_df4 <- data.frame(m = numeric(), n = numeric(), estimator_variance = numeric(),
                     estimator_mean = numeric())

for (m in m_values)
{
  for (n in n_values)
  {
    stats <- calculate_asian_estimator_statistics_quadratic_rqmc(m = m, n = n)
```

```r
    my_df4 <- rbind(my_df4,
                    data.frame(m = m,
                               n = n,
                               estimator_variance = stats$estimator_variance,
                               estimator_mean = stats$estimator_mean))
  }
}

# Print the results
print(my_df4)
```

```
##      m    n estimator_variance estimator_mean
## 1    2   50        1824.360003      15.612299
## 2    2  100          17.029395      10.506377
## 3    2  200          10.922693      10.872469
## 4    2  400           9.092345      10.895261
## 5    2  800           9.069794      11.382139
## 6    2 1600           7.745692      10.526334
## 7    4   50          14.771005       8.510381
## 8    4  100          18.499830       8.339080
## 9    4  200         100.624947       9.640378
## 10   4  400          15.588097       8.151835
## 11   4  800          26.541303       7.470421
## 12   4 1600          56.092048       8.620066
## 13  20   50         141.420819       7.112579
## 14  20  100          14.425798       5.316447
## 15  20  200          12.752759       4.107030
## 16  20  400          24.354169       3.928106
## 17  20  800          42.859040       4.077186
## 18  20 1600          33.169460       3.588980
## 19  40   50          17.743436       6.189155
## 20  40  100          16.886376       4.577776
## 21  40  200          14.586074       3.987503
## 22  40  400          24.981329       3.450903
## 23  40  800          26.086711       2.021397
## 24  40 1600          40.205280       1.311672
```