Optimization of Policy Evaluation and Policy Improvement Methods in Portfolio Optimization using Quasi-Monte Carlo Methods

by

Gavin Jeffrey Orok

A thesis

presented to the University Of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Quantitative Finance

Waterloo, Ontario, Canada, 2024

Abstract

Machine learning involves many challenging integrals that can be estimated using numerical methods. One application of these methods which has been explored in recent work is the estimation of policy gradients for reinforcement learning. They found that for many standard continuous control problems, the numerical methods RQMC and Array-RQMC that used low-discrepancy point sets improved the efficiency of both policy evaluation and policy gradient-based policy iteration compared to standard Monte Carlo. We extend this work by investigating the application of these numerical methods to model-free reinforcement learning algorithms in portfolio optimization, which are of interest because they do not rely on complex model assumptions that pose difficulties to other analytical methods. We find that RQMC significantly outperforms MC under all conditions for policy evaluation and that Array-RQMC outperforms both MC and RQMC in policy iteration with a strategic choice of the reordering function.

Acknowledgements

I would like to thank my supervisor Christiane Lemieux for mentoring me as I pursued my research over the last 8 months and ffor sponsoring my research.

This research was funded by the NSERC-CGSM grant and GRS funding from the University of Waterloo.

A negligible amount of code (15 lines) used in the final notebooks of the thesis was generated using the AI tools Google Bard and GitHub Copilot.

Dedication

To Mom, Dad, Max and Mikaela

Table of Contents

List of Figures

List of Abbreviations

- API: Application Programming Interface
- Array-RQMC: Array-Randomized Quasi-Monte Carlo
- CAD: Canadian Dollar
- CDF: Cumulative Distribution Function
- GB: Gigabyte
- GBM: Geometric Brownian Motion
- MC: Monte Carlo
- MDP: Markoov Decison Process
- MSE: Mean Squared Error
- MVN: Multivariate Normal
- QMC: Quasi-Monte Carlo
- RAM: Random Access Memory
- RL: Reinforcement Learning
- RQMC: Randomized Quasi-Monte Carlo
- TN: Truncated Normal

1 Introduction

This paper investigates to what extent the randomized Quasi-Monte Carlo methods can be used to optimize the efficiency of reinforcement learning in the financial problem of portfolio optimization.

Machine learning involves many challenging integrals that can be estimated using numerical methods. Some examples of this are the use of Monte Carlo methods for estimating policy gradients and variational inference in reinforcement learning [1]. Under certain conditions, Quasi-Monte Carlo methods that use low-discrepancy point sets instead of using independent uniformly sampled point sets improve on the efficiency of Monte Carlo [2]. The improvement using Quasi-Monte Carlo in the context of reinforcement learning has been explored in a low-dimensional context with empirical experiments on various continuous control problems in a recent paper. In particular, as a variation of Markov Chain Monte Carlo methods, the Array-QMC method was implemented which allows for a dimensional reduction to improve performance [3].

As a direction for potential future research, the authors mention it would be useful to formally characterize when Quasi-Monte Carlo will show improvements over Monte Carlo [3]. In general, the benefits of Quasi-Monte Carlo over Monte Carlo tend to vanish in higher dimensions [4]. Often when performing portfolio optimization major trading firms will work with databases involving thousands of securities, each of which introduces degrees of freedom in reinforcement learning that increase the dimension of the action space. This research seeks to quantify the difference in the performance in this context with very large numbers of securities and determine if there is a significant advantage in a context that reflects the real-world financial industry. In particular, we will focus on the problem of portfolio optimization.

This paper will begin by covering the fundamentals of the numerical methods and standard market models. Then, the paper will explain our problem of interest, the fundamentals of reinforcement learning, the financial environment in which our experiments take place, and the policy evaluation and iteration algorithms that will be used to compare the numerical methods. Then we will present and discuss our findings from running these algorithms.

This paper is accompanied by two Google Colab notebooks that may be used with credit to the author.

2 Background

2.1 Quasi-Monte Carlo Methods

In this section we will discuss the numerical methods for computing challenging integrals that we will compare in this research. Usually they are discussed in the context of a unit hypercube $[0,1]^d, d \geq 1$. We will present it in this context to be consistent with the literature but this can be easily generalized to other domains. An introduction to QMC can be seen in the survey paper [2].

Monte Carlo methods (referred to as MC hereafter) are numerical methods that calculate a value that is challenging to derive theoretically by running repeated random independent experiments and calculating the average result. In computing integrals with this numerical technique, MC sampling assumes the points are iid uniform, so the formula for the approximation of the integral is given by:

$$\int_{[0,1]^d} f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i), x_i \sim U([0,1]^d)(iid).$$

As can be seen from Figure 1, when sampling MC point sets we often see areas of the domain being neglected (gaps) or having too much concentration of sampled points (clusters). Quasi-Monte Carlo (hereafter referred to as QMC) aims to avoid the irregularities that occur when sampling iid points from the domain [5]. It uses low-discrepancy point sets to achieve this, which are more evenly spread out over the domain than what is typically sampled from MC. By themselves, QMC methods are deterministic; Randomized Quasi-Monte Carlo methods (hereafter referred to as RQMC) apply randomization to the point set.

Among other applications, QMC and RQMC can be used to estimate integrals similarly to MC, by sampling a low-discrepancy point set from the domain and then calculating the average value of the function at those points:

$$\int_{[0,1]^d} f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i), D = \{x_1, ..., x_N\} \subseteq [0,1]^d \text{ is low-discrepancy.}$$

Definition 2.1.1: The **Mean Squared Error** (MSE) of an estimator $\hat{X}$ for an unknown quantity μ is defined to be $\mathbb{E}[(\mu - \hat{X})^2]$.

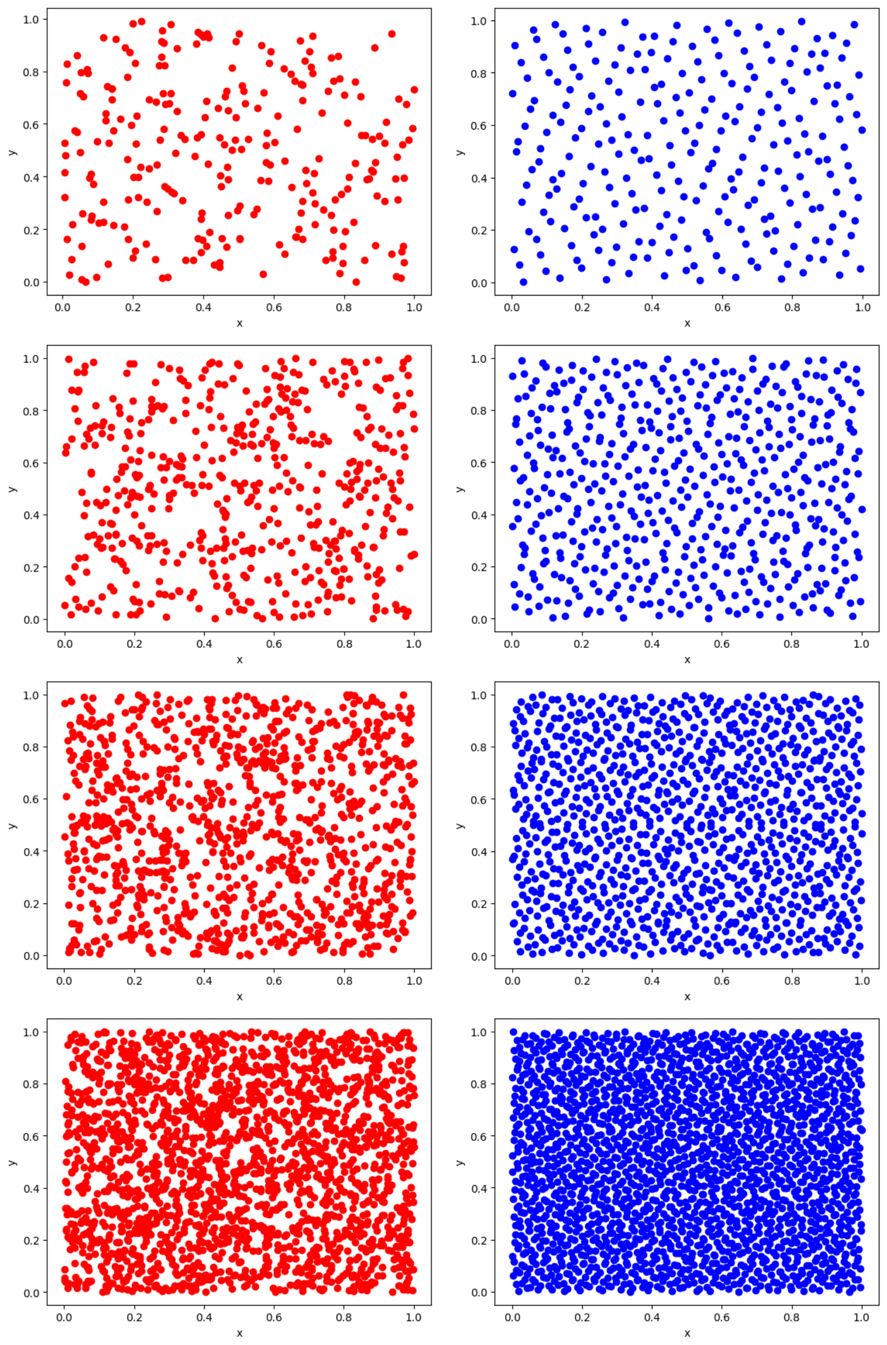


Figure 1: The red dots are MC point sets in the domain $[0,1]^2$ sampled with 256, 512, 1024, and 2048 total points. Beside each is the same-sized random linear scrambled Sobol' point set, with fewer clusters and smaller gaps.

The MSE can be decomposed as $\mathbb{E}[(\mu - \hat{X})^2] = Var(\hat{X} - \mu) + [\mathbb{E}(\mu - \hat{X})]^2$, which is the variance (a smaller variance indicates greater precision) plus the square of the bias (a smaller bias indicates a more accurate estimator) [6].

In our situation, the MC and QMC averages are the estimators and the theoretical value of the integral is the quantity we are trying to estimate. To determine which estimator, MC or RQMC, works best for a problem that involves closely estimating an integral we usually compare their MSEs. Since both estimators are known to be unbiased [7], this boils down to finding the estimator with a smaller variance.

Remark 2.1.2: It is not immediately obvious when RQMC shows an improvement over MC in terms of its variance, so for a given problem and number of points we usually need to evaluate results empirically to see which method has the smaller variance. Exceptions to this include some niche cases, for example, when the function is monotone in each coordinate in two dimensions [8], and results by Art Owen which showed the asymptotic variance of scrambled net estimators which can give some intuition for where QMC will do better [9].

For our experiments, following the methods in paper [3] we employ RQMC and Array-RQMC with scrambled Sobol' point sets.

The Sobol' sequence is used to generate a low-discrepancy point set to use with RQMC and Array-RQMC methods. The points are required to be a power of 2 to be stable and satisfy a quadrature property. These Sobol' point sets are widely used in financial applications [10].

Each coordinate of the Sobol' point set is intended to maximize the speed of convergence. However with increasing dimension the quality of the Sobol' point set deteriorates [11]. To address this problem, there is a wealth of research on techniques that seek to reduce the effective dimension of f. One example commonly used for financial applications is the Brownian Bridge technique [12].

By itself the Sobol' sequence is deterministic. We apply a technique called a random linear scramble to randomize the point set [3].

The Sobol' sequence is a specific example of a digital net in base 2.

Algorithm 2.1.3: We follow these steps to generate a digital net in base 2:

1. For $N = 2^l$ points in d dimensions, we choose d matrices $C_1, ..., C_d \in \{0,1\}^{L\times l}, L \geq l$ (generating matrices).
2. For $1 \leq i \leq N$ we obtain the binary representation of $i : i = b_1^{(i)} + b_2^{(i)}2 + ... + b_l^{(i)}2^{l-1}$.
3. We set the vector $\hat{u}_j^{(i)} = C_j[b_1^{(i)}, ..., b_l^{(i)}]^T$ mod 2 (in each coordinate) for each $j = 1, ..., d$.
4. We convert back to decimal: $u_j^{(i)} = \Sigma_{k=1}^{L} \hat{u}_{j,k}^{(i)} 2^{-k}, j = 1, ..., d$.
5. The concatenation $u^{(i)} = [u_1^{(i)}, ..., u_d^{(i)}]$ gives the coordinates of the low-discrepancy point set.

The Sobol' points can be constructed through this procedure by choosing particular generating matrices [3].

To apply a random linear scramble to the point set, we multiply each matrix C_j used for generating the sequence on the left by a randomly selected nonsingular lower triangular matrix with entries in $\{0, 1\}$ [3]. Another scrambling method that can be employed is Owen's nested uniform scrambling [13]. We can also apply a digital shift, which involves adding a randomly selected vector in $[0, 1]^d$ mod 2 over the binary vectors [2].

The Python module scipy.stats.qmc we use in our experiments applies a linear matrix scramble and a digital shift [14].

The **Array-RQMC** methods are a variation of RQMC that were designed specifically for applications involving estimating integrals of functions f of Markov chains [15]. With a Markov chain with many timesteps, we see degrading quality of the Sobol point set and applications with the scipy module are not supported for dimensions above 21201 [14]. Array-RQMC allows for a dimensional reduction that can allow for higher-quality estimation of the integral for high-dimensional integrands.

With Array-RQMC, we simulate a series of Markov chains $\{X_0^{(i)}, X_1^{(i)}, X_2^{(i)},$
$...\}, 1 \leq i \leq N$ in parallel. At every step of the Markov chain we refresh a new RQMC point set with dimension equal to that of the state space and number of points equal to the number of simulated Markov chains, and reorder the states using a sorting function h to match them with the RQMC points used to advance each chain. This reordering introduces negative dependence between the chains that gives a better estimation of the expectation over all realizations. The power of Array-RQMC comes from the generation of a new randomized Sobol' point set at each time step and from

the strategic choice of the reordering function [16].

Array-RQMC allows for a simulation of Markov chains that will better reflect the theoretical distribution than independently sampled trajectories. They have been used in finance for applications including option pricing which require simulating independent trajectories of assets to find the expected option payoff [17].

Algorithm 2.1.4: This the most commonly presented algorithm in the literature for the Array-RQMC procedure:

We will let d be the dimension of the state space, f be the function that we are finding the expectation for, h be the sorting function, N be the number of sampled Markov chains, T be the number of steps simulated in each Markov chain, x_0 be the initial value of the Markv chains, and $\varphi(s, u)$ be the function that advances the Markov chain in state s by the RQMC point u.

```
Inputs: $d, f, h, N, T, x_0, \varphi$
  for $i = 1, \ldots, N$ do
    $X_0^{(i)} \leftarrow x_0$
  end for
  for $t = 1, \ldots, T$ do
    Get $N$ scrambled Sobol' points $u_t^{(1)}, \ldots, u_t^{(N)}$ of dimension $d$
    Get the permutation $\beta$ of $1, ..., N$ where $h(X_t^{(\beta(1))}) \leq ... \leq h(X_t^{(\beta(N))})$
    for $i = 1, \ldots, N$ do
      $X_t^{(i)} = \varphi(X_{t-1}^{(\beta(i))}, u_t^{(i)})$
    end for
  end for
  Return $\frac{1}{N} \sum_{i=1}^{N} f(X_T^{(i)})$
```

Remark 2.1.5: The literature is not consistent about whether the RQMC points should be reordered as well to minimize the variance of the estimator, and the optimal choice of reordering function depends on the application. For example the algorithm presented in [18] does not show the permutation being applied to the RQMC points, but in [19] it is stated that the

RQMC points must be permuted to match them with the permuted states. We explore which choices work best in the policy iteration experiments for our portfolio optimization applications of interest in section 4.2.

We model our choices for the sorting function after the procedure described in [18]. For a randomized Sobol' point set of size $N = 2^l$, we first sort the point set by the values of the first coordinate and split it in half. For each half, we sort the Sobol' points by the values of the second coordinate and split in half. We repeat this process recursively until we reach the final coordinate of the point sets, where we sort the sublists in increasing order of that index. Figure 2 gives an example of this for 8 states and RQMC points. In our approaches we do not necessarily go in order of the indices, but may go in order of a list of indices in the point sets.

In the event that the number of dimensions d is greater than l where $N = 2^l$, we need to apply a dimensional reduction to retain the most important information in each point. We will explore the dimensional reduction which results in the best performance in section 4.2 as well.

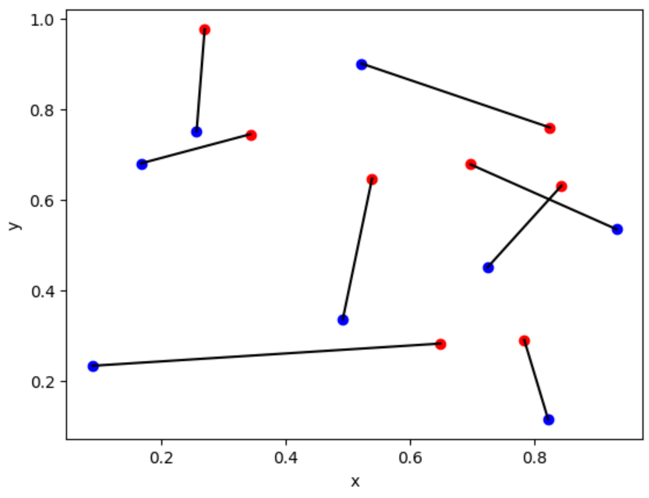


Figure 2: The red points represent a set of 8 states from the same step of 8 parallel Markov processes and the blue points represent 8 points from a random linear scrambled Sobol' sequence in 2 dimensions. The black lines correspond to the matching done by splitting the points into two groups based on increasing values of the x coordinate, then arranging each group in increasing order of the y coordinate. This matching assumes both the states and Sobol' points are permuted.

2.2 Stock Market Fundamentals

Further context on stock market fundamentals may be found in references such as [20].

While we use reasonable parameters based on observations from real-life financial markets, we do not carefully calibrate parameters used to the overall financial market. This is a separate substantial task on its own and these parameters vary with the macroeconomic conditions in the overall financial market.

We assume that a year consists of 250 trading days.

We model an investor with a cash account and a trading account. We assume that there are no fees for holding these accounts, and money can be readily deposited and withdrawn with no penalty. We also permit balances in these accounts to become negative without an overdraft fee applying.

We assume the investor holds an account in the currency CAD, so exchange rate fluctuations can be disregarded. We assume a consistent inflation rate of 2% per year compounded continuously applies to the investment to match the Bank of Canada's targeted annual inflation rate.

Definition 2.2.1: For the purposes of this simulation, the **risk-free interest rate** is defined as the continuously compounded interest rate that will be accumulated on a positive balance of funds placed into a savings account.

The **lending rate** is defined to be the continuously compounded interest rate that will be charged on an owed balance of funds.

Unlike money in the risky assets, there is no volatility that could result in a fall in value for the risk-free asset.

We assume that the lending rate for a negative cash balance is higher than the the risk-free interest rate that is accumulated on savings, to match the traditional business model of banks. We use a lending rate of 4% compounded continuously and an interest rate of 1% compounded continuously for this study. We choose an interest rate that is smaller than the inflation rate to incentivize investing in the financial market.

Definition 2.2.2: A **transaction cost** in stock trading is a cost that is deducted from a trader's account when executing a trade.

For this study, we assume a transaction cost of \$0.50 CAD per transaction, in the value of the currency at the time of the trade. We deduct a transaction cost for each asset that is traded in a simulation. For trades of fractional shares of assets that are less than 0.001, we do not deduct a transaction cost.

Definition 2.2.3: Short selling refers to a trade that places a bet against the stock that involves borrowing the stock, selling it and then repurchasing it later on to return it.

The profit or loss from taking a short position is given by: $S_0 - S_T - t_c$ where S_0 is the initial price of the asset, S_T is the sale price of the asset, and t_c is the transaction cost of the asset. So the worse the stock performs, the more the short seller earns. We assume other costs including lending fees and dividends are negligible in our simulation. We will indicate short selling with a negative position in that asset.

2.3 Mathematical Modelling of Financial Markets

In order to have a realistic model of the financial market, we need to model relationships between similar assets. These relationships can include:

1. Assets whose prices tend to move in the same direction (positively correlated), e.g. two assets from the same sector;
2. Assets whose prices tend to move in opposite directions compared to each other (negatively correlated), e.g. fossil fuel and airline stocks; and,
3. Assets whose prices tend to move independently of each other (uncorrelated).

In our code, we define a correlation matrix Σ to encode the correlations between the assets in the model. In the case where all standard deviations are 1, this is the same as the covariance matrix. We separate the different sectors into blocks of five assets that are adjacent to each other in the indices of the matrix.

Definition 2.3.1: The **Cholesky Factor** of a positive definite matrix Σ is the unique lower triangular matrix A such that $AA^T = \Sigma$ [21].

Definition 2.3.2: We say that a random vector $Z = [Z_1, ..., Z_d]^T$ is **multivariate normal** with mean vector $\vec{\mu} = [\mu_1, ..., \mu_d]^T$ and covariance matrix $\Sigma = [\sigma_{i,j}]_{1 \leq i,j \leq d}$ if it has the joint density function

$$\phi(\vec{x}) = \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1}(\vec{x}-\vec{\mu})}.$$

As an important property, for all $1 \leq i \leq d$ we have that the coordinate Z_i of Z is normal with mean μ_i and variance $\sigma_{i,i}$, and for all $1 \leq i, j \leq d$ we have that $Cov(Z_i, Z_j) = \sigma_{i,j}$ [22].

Furthermore, we have the property that for d-dimensional $Z \sim MVN(\vec{\mu}, \Sigma)$ that for any $M \in \mathbb{R}^{d \times d}$ we have that $Cov(MZ) = M\Sigma M^T$ [22]. In particular if $Z \sim MVN(\vec{0}, I)$ and A is the Cholesky Factor of a covariance matrix Σ then $AZ \sim MVN(\vec{0}, \Sigma)$.

We use several financial models to simulate the price fluctuations of assets. These models all use normal random variables to advance the stochastic processes encoding the prices. To encode the correlations, we multiply the vector $Z = [Z_1, ..., Z_d]^T$ encoding the independent standard normal random variables on the left by the Cholesky factor [21].

Definition 2.3.3: A **Brownian Motion** W_t is a stochastic process indexed by time $t \geq 0$, defined:

1. $W_0 = 0$,
2. The mapping $t \to W_t$ is almost surely continuous,
3. For all times $t_1 \leq t_2, W_{t_2} - W_{t_1} \sim N(0, t_2 - t_1)$,
4. For all times $t_1 < t_2 \leq t_3 < t_4$ we have that $W_{t_4} - W_{t_3}$ and $W_{t_2} - W_{t_1}$ are independent random variables.

As a consequence of these conditions, $W_t = W_t - W_0 \sim N(0, t)$ [23].

Definition 2.3.4: We define the **Geometric Brownian Motion (GBM)** stock price model by $S_t = S_0 e^{(\mu - \frac{\nu^2}{2})t + \nu W_t}$ where $S_t \geq 0$ is the price of the asset at time $t \geq 0$, $\mu, \nu \in \mathbb{R}$ so that $\nu > 0$, and W_t is a Brownian Motion. We refer to the parameter μ as the drift of the GBM and ν as the volatility of the GBM.

We define the **Heston** stock price model by $S_t = S_0 e^{(\mu - \frac{v_t}{2})t + \sqrt{v_t} W_t^S}$ where S_t is the price of the asset at time t, $\mu \in \mathbb{R}$, v_t is a stochastic process defined by the dynamics $dv_t =$

$\kappa(\theta - v_t)dt + \xi\sqrt{v_t}dW_t^v$, and W_t^S and W_t^v are Brownian Motions with correlation ρ [23]. We refer to θ as the long-run volatility of the Heston.

The GBM model is easy to implement and derive theoretical results for, but has the strong assumption of constant volatility. These assumptions are reasonable for short time horizons, but for longer time horizons the Heston model is preferred as it accounts for the fluctuation of volatility [24].

Definition 2.3.5: We define a financial shock as a major unexpected movement of the price of an asset that is caused by external factors [25].

While in realistic scenarios the price is not adjusted instantly in response to major news, under the assumptions of the efficient market hypothesis the price readjusts automatically upon the public revelation of major news updates to its new equilibrium price.

We will model financial shocks with a Compound Poisson Process.

Definition 2.3.6: A **Poisson process** counts the number of occurrences of independent events with an identical and constant arrival rate where two events cannot occur simultaneously. We will denote a Poisson process $N(t)$ and define its probability distribution as:

$$P(N(t) = k) = \frac{(\lambda t)^k}{k!}e^{-\lambda t}, k = 0, 1, 2, ...$$

for $t \geq 0$ and a constant rate $\lambda > 0$.

A **Compound Poisson Process** is a process $Q(t) = \sum_{i=1}^{N(t)} Y_i$ where $N(t)$ is a Poisson counting process, and the Y_i's are identically distributed random variables that are independent from each other and also independent from the process $N(t)$. We usually employ a normal model for the random variables Y_i in the Compound Poisson Processes [26].

Remark 2.3.7: Say we have K Compound Poisson Processes $Q^{(i)}(t), 1 \leq i \leq K$. To model the effects of these processes on a Geometric Brownian motion, we adjust the GBM pricing model as follows by adding the processes in the exponential:

$$S_t = S_0 e^{(\mu - \frac{\nu^2}{2})t + \sigma W_t + Q^{(1)}(t) + ... + Q^{(K)}(t)}.$$

In the literature usually only one Compound Poisson Process is added [26]. We will use three to model linked jumps caused by market-wide shocks, sector-specific shocks and asset-specific shocks.

For our experiments we include jumps with the GBM but not the Heston.

2.4 Sharpe Ratio and Classification of Assets

Definition 2.4.1: We will refer to a market with risky assets $A_1, ..., A_d$ and risk-free asset A_{rf} as the **standard market model**.

We define a **portfolio** $\omega \in \mathbb{R}^{d+1}$ in the standard market model to be a vector such that $\omega^T \vec{1} = 1$.

Say asset $i, 1 \leq i \leq d+1$ in the standard market model has a time t price described by the process $S_t^{(i)}$, for $0 \leq t \leq T$. Then the value of the portfolio at time t is given by $\Pi_\omega(t) = \omega^T S_t$ where $S_t = [S_t^{(1)}, ..., S_t^{(d+1)}]^T$ [27].

We allow weights to be outside $[0, 1]$ to permit short selling. We will mainly work with dynamic portfolios $\omega_t, t \geq 0$, which may be rebalanced at various points in time under the condition that for any t we have $\omega_t^T \vec{1} = 1$.

Definition 2.4.2: We define the **simple return** of a portfolio ω over time period $[t_1, t_2], 0 \leq t_1 < t_2$ to be the ratio $Ret(t_1, t_2) = \frac{\Pi_\omega(t_2) - \Pi_\omega(t_1)}{\Pi_\omega(t_1)}$ [27].

Definition 2.4.3: The **Sharpe ratio** of a portfolio ω in the standard market model is defined as $S = \frac{E(Ret_\omega) - r_f}{\sigma_{Ret_\omega}}$ where $E(Ret_\omega)$ is the expected simple return of the portfolio, r_f is the expected simple return of the risk-free asset and σ_{Ret_ω} is the volatility (standard deviation) of the simple return of the portfolio.

By the Sharpe ratio of a risky asset in the standard market model we refer to the Sharpe ratio of a portfolio where all wealth has been invested into that asset.

The numerator of the Sharpe ratio is the excess return (alpha) of the portfolio, and the denominator is the measure of the risk of the portfolio. A higher Sharpe ratio indicates the capacity for higher return per unit of risk [27].

Lemma 2.4.4: For a Geometric Brownian Motion with constant initial value S_0, drift μ and volatility $\nu > 0$, the expectation of a simple return of the process over a time horizon of T years is given by $e^{\mu T} - 1$ and the variance of the simple return $Ret(0, T)$ is given by $e^{2\mu T}(e^{\nu^2 T} - 1)$.

Proof. The GBM is distributed as $S_T \sim Lognormal(\hat{\mu}, \hat{\nu}^2)$ where $\hat{\mu} = \ln S_0 + (\mu - \frac{\nu^2}{2})T$ and $\hat{\nu}^2 = \nu^2 T$. So, by known results [27] we have

$$\mathbb{E}(S_T) = e^{\hat{\mu}+\frac{\hat{\nu}^2}{2}} = e^{\ln S_0 + \mu T} = S_0 e^{\mu T}$$

and

$$Var(S_T) = e^{2\hat{\mu}+\hat{\nu}^2}(e^{\hat{\nu}^2} - 1) = S_0^2 e^{2\mu T}(e^{\nu^2 T} - 1)$$

So for a simple return,

$$\mathbb{E}(Ret(0,T)) = \mathbb{E}\frac{S_T - S_0}{S_0} = \frac{\mathbb{E}(S_T)}{S_0} - 1 = e^{\mu T} - 1$$

and

$$Var(Ret(0,T)) = Var\frac{S_T - S_0}{S_0} = \frac{1}{S_0^2}Var(S_T - S_0) = \frac{1}{S_0^2}Var(S_T) = e^{2\mu T}(e^{\nu^2 T} - 1).$$

□

Proposition 2.4.5: For a risky asset $i, 1 \leq i \leq d$ in our standard market model with time t price described by a GBM with drift μ and whose Sharpe ratio is c over $[0, T]$, the volatility of the GBM over $[0, T]$ is given by the expression $\nu = \sqrt{\frac{1}{T}\log(1 + \frac{(e^{\mu T} - e^{r_f T})^2}{c^2 e^{2\mu T}})}$.

Proof. If the Sharpe ratio is c, for a simple return $R(0, T)$ that means that

$$\frac{\mathbb{E}(R(0,T)) - [(e^{r_f})^T - 1]}{\sqrt{Var(R(0,T))}} = c$$

$$\implies \frac{(e^{\mu T} - e^{r_f T})^2}{c^2} = e^{2\mu T}(e^{\nu^2 T} - 1)$$

$$\implies 1 + \frac{(e^{\mu T} - e^{r_f T})^2}{c^2 e^{2\mu T}} = e^{\nu^2 T}$$

$$\implies \sqrt{\frac{1}{T}\log(1 + \frac{(e^{\mu T} - e^{r_f T})^2}{c^2 e^{2\mu T}})} = \nu.$$

□

We can use Lemma 3.2 to derive the volatility when the Sharpe ratio and drift of a risky asset are known. We will assume a risk-neutral investor who is indifferent between portfolios with an equal Sharpe ratio. So, this will allow us to determine values of μ and ν to define assets of equal viability in the eyes of the agent. We will use this in tiering of assets later on.

Normally the volatility can be simply calibrated from market data, but this is not appropriate in a context where the assets are synthetic.

3 Methodology

3.1 Reinforcement Learning

Definition 3.1.1: The problem of **portfolio optimization** refers to strategically choosing the weights in a portfolio ω in the standard market model so that a desired metric of the portfolio is maximized [28]. For example, an investor may seek to maximize the simple return of the portfolio or maximize its Sharpe ratio to be resistant to financial shocks. We will focus on dynamic portfolio optimization problems which involve rebalancing the weights as the portfolio performs rather than choosing set weights at the beginning of the investment period.

In our experiments, we will explore a reinforcement learning (RL) approach to portfolio optimization.

Definition 3.1.2: Reinforcement learning is a machine learning technique done under the following framework:

1. There is an agent operating in its environment.
2. At any point in time, the agent exists in a certain state of the world in its environment.
3. The agent takes actions to move between different states of the world.
4. The agent receives a reward for taking a given action in a certain state and ending up at another state.
5. The goal of the agent is to develop a policy over time to choose an action when in a given state that will maximize its long-term reward [29].

In our problem of interest the states are the positions the agent holds in the assets of the standard market model, the actions are the changes the agent makes to the positions in the assets, and the reward is an indication of the performance of the portfolio that we adjust (we will describe this further in Section 3.2).

It is true that often machine learning researchers try to avoid reinforcement learning when possible because it can be noisy, so we need to run experiments many times to get reliable results. While the problem of portfolio optimization can be approached with other means, RL is useful in this context because its model-free algorithms do not need to rely on model assumptions to make decisions in complex financial environments, which is a challenge for other analytical methods for making financial decisions [29].

We now describe the mathematical framework of RL problems.

Definition 3.1.3: A **Markov Decision Process** (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, P_a, R_a)$ where:

1. $\mathcal{S}$ is a set of Markov chain states,
2. $\mathcal{A}$ is a set of actions that agents take to transition between the states,
3. P_a is a probability function where $P_a(s, s') = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability of being in state s' at time $t + 1$ given that the state of an agent at time t is s and the action taken at time t is a, $s, s' \in \mathcal{S}, a \in \mathcal{A}$.
4. $R_a(s, s')$ is the immediate reward from taking action a in state s and ending up in state s', $s, s' \in \mathcal{S}, a \in \mathcal{A}$ [29].

Definition 3.1.4: A **policy** for an RL problem with MDP $(\mathcal{S}, \mathcal{A}, P_a, R_a)$ is a function $\pi : \mathcal{S} \to \mathcal{A}$ which selects an action for an agent given a state of the environment.

Definition 3.1.5: There are two main types of reinforcement learning problems:

1. A **discrete control** reinforcement learning problem involves finitely or countably infinitely many possible actions for any given state.
2. A **continuous control** reinforcement learning problem involves uncountably many possible actions for any given state [29].

We will focus on continuous control problems in this research. Most often when working with continuous state spaces as in our problem of interest, the actions are vectors sampled from multivariate Gaussian distributions. For our application of interest, we will use a truncated multivariate normal policy with coordinates that indicate the number of units that are invested in each risky asset of the market on each rebalancing day.

Optimization of a policy in the context of a discrete control problem involves looking at the maximum projected value over a given set of possible parameter sets for the policy. This is possible due to the discrete nature of the action space.

In a continuous control problem, it is not possible to take a maximum. Instead, optimization techniques are employed with the loss function, a function which maps the parameter set to a value representing the cost for the agent to follow the policy with those parameters, or a reward function, a function which maps the parameter set to a value representing the benefit for the agent to follow the policy with those parameters. In particular, we usually use gradient descent to search for the parameters that minimize a loss function or gradient ascent to search for the parameters that maximize a reward function [29].

In our portfolio optimization problem of interest, we will parametrize the truncated normal policy with its mean vector corresponding to the number of units that are invested into each risky asset. The mean vector determines on average how much is being invested into each asset at each rebalancing point, which determines the weights in the portfolio that we are trying to optimize.

Algorithm 3.1.6: Gradient ascent is an algorithm defined as follows:

We will let f be the multivariable reward function being optimized, $\alpha > 0$ be the **learning rate** of the algorithm, $\epsilon > 0$ be the tolerance level for ending gradient ascent, and θ_0 be the initial value of the parameter set being optimized.

```
Inputs: $f, \alpha, \epsilon, \theta_0$
  Calculate the gradient $\nabla f(\theta_0)$
  Set $\theta_1 \leftarrow \theta_0 + \alpha \nabla f(\theta_0)$
  $t \leftarrow 1$
  while $||\theta_t - \theta_{t-1}|| > \epsilon$ do
      Calculate the gradient $\nabla f(\theta_t)$
      Set $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla f(\theta_t)$
      $t \leftarrow t + 1$
  end while
```

In general, higher values of α result in larger steps of the algorithm in the domain as it runs, and a lower tolerance level will require more time for the policy to converge.

Example 3.1.7: For non-convex optimization problems, gradient ascent is not guaranteed to find the global optimum of a loss or reward function.

Consider the reward function $y = -3x^4 + 16x^3 - 18x^2$. We will demonstrate this has two local maxima at $x = 0$ and $x = 3$, but the local maximum at $x = 3$ attains a much larger y-value.

Differentiating, $y' = -12x^3 + 48x^2 - 36x = -12x(x^2 - 4x + 3) = -12x(x - 3)(x - 1)$. This has roots at $x = 0, 1, 3$. Next, we evaluate the second derivative: $y'' = -36x^2 + 96x - 36$.

Note that $y''(0) = -36$, $y''(1) = 96$ and $y''(3) = -72$. So there are two local maxima at $x = 0, 3$ where the function is concave down. However, $y(0) = 0$ while $y(3) = 27$. So the global maximum lies at $x = 3$ (since the function starts low and ends low).

As shown in Figure 3, applying gradient descent from the initial point $x = -1$ plotted in red on the left with a small learning rate will result in the point plotted in blue being chosen as the global maximum of the loss function, which unfortunately misses the actual global maximum. Figure 4 shows an example of where the global maximum is attained.

Definition 3.1.8: We say an RL algorithm is **policy-based** if it builds an explicit representation of a policy π and improves it iteratively [29]. We will focus on this type of RL algorithm in this research.

By the value of an RL policy we typically refer to the quantity $V^\pi = \mathbb{E}_{s_t,a_t}[Q^\pi(s_t, a_t)]$ where $Q^\pi(s_t, a_t)$ is a function that assigns a value to taking the action a_t in state s_t by following the policy π in a Markov Decision Process at time t. We say an RL algorithm is **value-based** if it builds a representation of a value function which it uses to assign a quantity that indicates the relative value of a policy. This can then be used to derive the policy. Examples of how this could be applied to financial applications may be found in section 4 of [29].

We say an RL algorithm is **model-free** if it reacts to feedback from an unknown environment and does not learn the underlying transition probabilities and reward function from the environment [29].

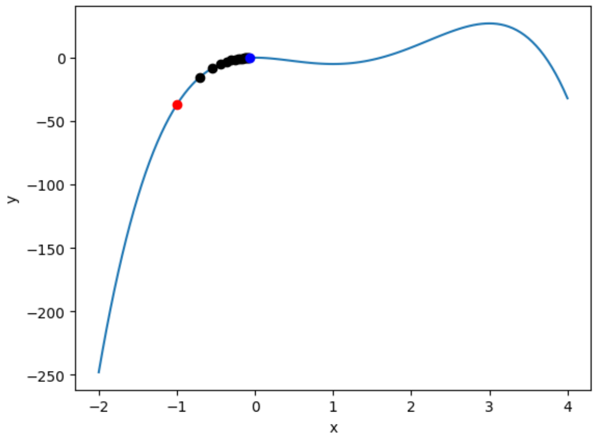


Figure 3: gradient ascent algorithm applied to $y = -3x^4 + 16x^3 - 18x^2$ with the starting point $x = -1$ plotted in red, intermediate points plotted in black, and the terminal point plotted in blue. This choice of initial point does not attain the global max.

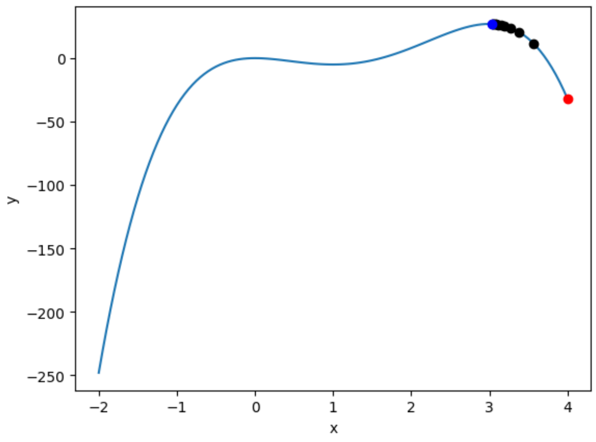


Figure 4: gradient ascent algorithm applied to $y = -3x^4 + 16x^3 - 18x^2$ with the starting point $x = 4$ plotted in red, intermediate points plotted in black, and the terminal point plotted in blue. This choice of initial point attains the global max.

We say an RL algorithm is **on-policy** if it estimates the performance of the policy using data that was collected from following the same policy. We say it is **off-policy** if it estimates performance using a different policy [29].

The **REINFORCE** RL algorithm [29] is a policy-based, model-free algorithm which uses Monte Carlo methods to approximate policy gradients in applying gradient ascent or descent (depending on the loss or reward function) to adjust parameters of a policy.

We will focus on the REINFORCE method through this research, and in particular how it can be improved by using RQMC instead of MC.

On-policy learning is online, which means it refines performance as a policy runs. A particular example of this would be collecting performance data of a previous policy and using it to update the parameters of the policy based on the rewards from its environment. Off-policy is primarily offline, and can flexibly use data from other policies to learn a different policy.

The REINFORCE algorithm is considered off-policy because it uses gradient ascent after completing a run with a policy to update the policy parameters.

The paper [3] whose work we extend through this paper employed several different RL methods based on the application, and focused on value-based methods that learned parameters through a variety of means. Ultimately, we decided that to minimize the fine-tuning of additional model parameters for an initial exploration of RL methods in a financial context, the simple REINFORCE algorithm is the most suitable choice that have been shown to be effective in studies such as [30]. In further extensions to this work, it would be appropriate to consider the state of the art actor-critic methods as well.

3.2 Portfolio Optimization Environment

For this experiment, we used a reinforcement learning environment to model an investor trading assets over a period of time. We incorporate many factors that can be adjusted to test their effects on the algorithms' performances.

We specify the default conditions of the trading problem to be the following:

We have an investor with both a bank account and a trading account. The investment either occurs over T total trading days with rebalances equally spread out over the investment period. In our experiments, we use T = 250, 1250, or 3750 and have 25, 50, or 125 rebalances. We assume fractional shares of every asset can be traded up to a level of precision that is consistent with floating point decimals we will work with in Python; this is practically consistent with online trading platforms that are available to Canadian investors.

The trading strategy is self-financing, meaning there are no cash injections after time 0 while the agent invests.

The investor is able to invest in a set of 2, 5, 10, 25, 50, 100, or 1000 assets in the trading account. Assets are organized into tiers based on their Sharpe ratio. The Sharpe ratios used for the simulation are -1, -0.5, 0.5, 1, and 1.5.

The underlying model of price fluctuations may be a GBM or a Heston. As shown from Proposition 2.4.5, under the assumption of a GBM pricing model we can determine the volatilities directly from the drift, time horizon, risk-free rate, and Sharpe ratio. As the assets are synthetic, we do this to ensure the assets have the desired Sharpe ratios to have a variety of different performances from the assets in the environment. We use a time horizon of one year for these calculations. For the parameters of the Heston, we use this ν as the long-run variance, $\xi = 0.04, \rho = 0.1$ and $\kappa = 3$.

By default, we will refer to assets that move according to GBMs that have Brownian Motions with a correlation of 0.7 as being strongly positively correlated, assets that have Brownian Motion movements that have a correlation of -0.7 as being strongly negatively correlated, and assets that have Brownian Motions with a correlation of 0.3 as being weakly positively correlated. These values are in widely agreed-upon ranges for acceptable values for these correlation strengths [31]; we will adjust these values in section 4 to see if they significantly impact the relative performances of the numerical methods.

The GBM model may or may not include jumps. If jumps are included, we assign separate Compound Poisson processes for 1) the entire market, 2) separate sectors, and 3) individual assets with the same frequency λ of 1/1500 in the implementation described in 2.3.7. This is a simplifying assumption but further analysis would require intense market calibration.

For our simulation, we choose the random variables Y_i to be distributed as follows, where Z is the standard normal distribution:

$$Y_i = -0.1 + 0.5Z \sim N(-0.1, 0.5).$$

Then, the exponential of each Compound Poisson process simplifies as follows:

$$e^{\Sigma_{i=1}^{N(t)} Y_i} = \Pi_{i=1}^{N(t)} e^{Y_i} = \Pi_{i=1}^{N(t)} X_i$$

where each X_i is defined

$$X_i = e^{-0.1+0.5Z} \sim Lognormal(-0.1, 0.5).$$

In defining the shocks this way, we ensure that shocks will never cause the price to become negative. Also, negative price shocks have a greater probability of occurring than positive shocks.

We organize assets into sectors, where all assets in one sector are strongly positively correlated with each other and are all affected by one set of jumps, in a Compound Poisson Process that is separate from asset-specific and jumps that affect all assets (if jumps are included). This reflects common price movements of similar assets in response to a shock, e.g. in response to a global pandemic which puts a ban on international travel, all airline stocks may see a collective price fall. We make the simplifying assumption that all sectors have identical frequencies of sector-specific jumps, which does not reflect reality (e.g. the utilities sector tends to be significantly less volatile than the technology sector).

We model the environments with different numbers of assets as follows:

1. In the 2-asset case, we have a Sharpe -1 asset with drift -0.5 and Sharpe 1.5 asset with drift 0.4. We assume these assets are strongly negatively correlated. This is the smallest case for which we can have correlation between two assets in the market.

2. In the 5-asset case, we have a Sharpe -1, -0.5, 0.5, 1, and 1.5 asset. For the negative Sharpe ratios we assign a drift of -0.5 and for the positive Sharpe ratios we assign a drift of 0.4. We assume all assets are in the same sector. This reflects a sector-specific trading strategy.

3. In the 10-asset case, we have two sectors each with a Sharpe -1, -0.5, 0.5, 1, and 1.5 asset where for the negative Sharpe ratios we assign a drift of -0.5 and for the positive Sharpe ratios

we assign a drift of 0.4. We assume these two sectors are strongly negatively correlated.

4. In the 25-asset case, we have five sectors each with a Sharpe -1, -0.5, 0.5, 1, and 1.5 asset where for the negative Sharpe ratios we assign a drift of -0.5 and for the positive Sharpe ratios we assign a drift of 0.4. We assume the first two sectors are strongly negatively correlated, the third and the fourth sectors are weakly positively correlated, and all other pairs of sectors are uncorrelated.

5. In the 50-asset case, we have five sectors each with two Sharpe -1, -0.5, 0.5, 1, and 1.5 assets. For the negative Sharpe ratio assets we assign drifts of -0.5 and -0.2, and for the positive Sharpe ratio assets we assign drifts of 0.2 and 0.4. We assume the first two sectors are strongly negatively correlated, the third and the fourth sectors are weakly positively correlated, and all other pairs of sectors are uncorrelated.

6. In the 100-asset case, we have five sectors each with four Sharpe -1, -0.5, 0.5, 1, and 1.5 assets. For the negative Sharpe ratio assets we assign drifts of -0.5, -0.4, -0.3 and -0.2, and for the positive Sharpe ratio assets we assign drifts of 0.2, 0.27, 0.33, and 0.4. We assume the first two sectors are strongly negatively correlated, the third and the fourth sectors are weakly positively correlated, and all other pairs of sectors are uncorrelated.

7. In the 1000-asset case, we have five sectors each with forty Sharpe -1, -0.5, 0.5, 1, and 1.5 assets. For the negative Sharpe ratio assets we assign drifts between -0.5 and -0.2 that vary by equal increments, and for the positive Sharpe ratio assets we assign drifts between 0.2 and 0.4 that vary by equal increments. We assume the first two sectors are strongly negatively correlated, the third and the fourth sectors are weakly positively correlated, and all other pairs of sectors are uncorrelated.

While in the real-world market there are market makers available to allow reasonable liquidity of the assets, there are only finitely many shares that exist of a stock and even fewer that can be exchanged on a given trading day. To account for this, we impose (arbitrary) limits of 50000 on how many shares can be bought and sold in one day. To account for this in the simulation, we use -50000 and 50000 as the endpoints of the coordinates of the truncated normal distribution from which we draw the actions for buying and selling on a rebalancing day. We will discuss this further in section 3.3.

On each day, the agent receives one of the following rewards: a modified simple return over the period from the last rebalancing date, the arctangent of this modified simple return, or the arctangent of the difference in the Sharpe ratio (calculated empirically with respect to the daily returns since time 0). Since the arctangent function is a monotone increasing transformation, higher simple returns will still mean higher rewards and vice versa, and values between -1 and 1 will be a very similar value before and after applying the arctangent transformation.

Since we are allowing the agent to short sell assets and the portfolio value may become negative, just using a simple return would result in the algorithm penalizing increases in value when the previous portfolio value is negative. To avoid this, instead of just using the simple return we take the absolute value of the initial price in the denominator, in the modified simple return $\tilde{Ret}(t_1, t_2) = \frac{\Pi_\omega(t_2) - \Pi_\omega(t_1)}{|\Pi_\omega(t_1)|}$ where $t_1 < t_2$ are consecutive rebalancing days.

The main idea behind this choice of reward is the arctangent function is bounded above by $\pi/2$ and bounded below by $-\pi/2$. This prevents the agent's trading strategy from moving in an uncontrolled way in one direction. While fine-tuning the model with just simple returns it was found that the algorithm would frequently fail to produce an adequate strategy due to moving too far in the direction of a positive reward and reaching the imposed buying or selling limits on the shares of the assets in the market. Similar strategies are used by the authors of the paper [32] with a hyperbolic tangent squashing function, and in the paper we extend which uses a tanh-Gaussian policy [3].

For the difference in the Sharpe ratio, the idea behind this is similar to previous research which applied reinforcement learning to portfolio optimization which used the differential Sharpe ratio [28]. This is an additive reward that accounts for both volatility and return. In our case rather than taking the derivative, we look at the difference in Sharpe ratio over small time intervals. We apply the arctangent function in this case too to avoid numerical issues from values getting too large.

We assume all assets start with the same price at time zero. This is a simplifying assumption that can reduce the level of complexity of the market: generally more expensive stocks tend to be value investments which are significantly less volatile in their price movements than small-cap stocks. Furthermore, fewer shares of these value stocks can be readily purchased because they are more expensive, which means they can have less leverage than less expensive assets. However, as we include stocks that have a variety of risk levels and drift parameters in the market, this is a reasonable simplification of the market conditions as the problem still boils down to detecting undervalued and overvalued assets.

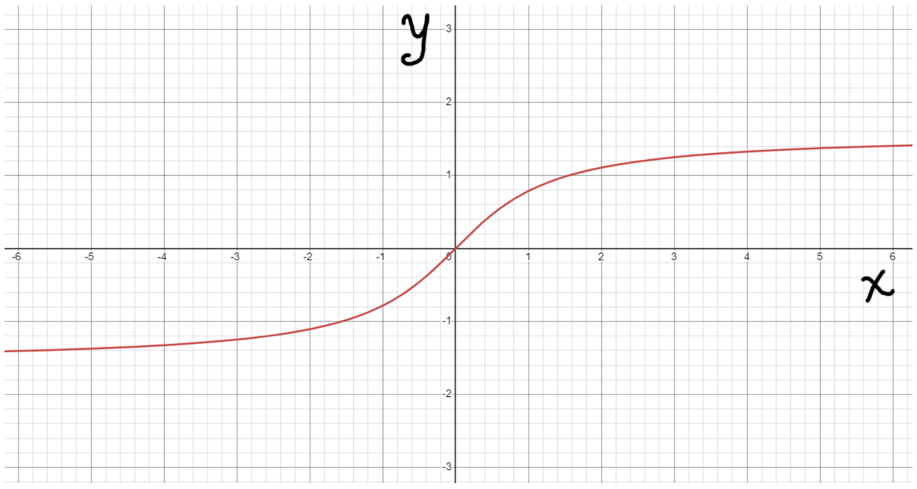


Figure 5: This is a plot of the function $y = tan^{-1}(x)$ from Desmos. Note it is monotone increasing with diminishing slope, and bounded above and below.

Remark 3.2.1: The 10-asset model is intended to serve as a microcosm for a larger stock market: there are separate groups of assets which move together in different directions and have different long-term performances. The 25-, 50-, 100- and 1000-asset cases further increase complexity but just expand on the traits developed through the 10-asset model.

3.3 Policy Evaluation

Definition 3.3.1: For two vectors $\vec{x}, \vec{y} \in \mathbb{R}^d, d \geq 1$ we say that $\vec{x} \leq \vec{y}$ if this inequality applies in each coordinate.

Definition 3.3.2: For $\vec{a} \leq \vec{b} \in \mathbb{R}^d, \vec{\theta} \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d}$ a covariance matrix, $d \geq 1$ we define the **multivariate truncated normal distribution** $TN(\vec{\theta}, \Sigma, \vec{a}, \vec{b})$ to be the distribution with the following probability density:

$$\phi(\vec{x}) = \frac{\frac{1}{(2\pi)^{d/2}} \det(\Sigma)^{-1/2} exp(-\frac{1}{2}(\vec{x}-\vec{\theta})^T \Sigma^{-1}(\vec{x}-\vec{\theta}))}{F(\vec{b}) - F(\vec{a})}$$

where F is defined as the cdf of the multivariate normal distribution $MVN(\vec{\theta}, \Sigma)$ and $\vec{x}$ is any vector such that $\vec{a} \leq \vec{x} \leq \vec{b}$. We refer to $\vec{\theta}$ as the mean vector of the truncated normal distribution and Σ as the covariance matrix [33].

The multivariate normal distribution in d dimensions normally has the support of $\mathbb{R}^d$. For the truncated normal distribution, we restrict the support to the d-dimensional rectangle in $\mathbb{R}^d$ between $\vec{a}$ and $\vec{b}$ and rescale the MVN pdf so the area under the graph is 1.

We will use the coordinates of the truncated normal distribution to represent the number of units invested into each asset in the market.

As mentioned in section 3.2 we keep the left and right endpoints of the truncated normal constant at -50000 and 50000 respectively, so a maximum of 50000 units of each asset may be shorted or longed each day.

We will assume as a relaxing assumption that the covariance matrix is a scalar multiple of the identity; this is a reasonable assumption in the framework of a portfolio optimization problem as it would represent an investor who analyzes the performance of stocks independently when selecting how much to invest in each, and will have the same willingness to consider a variety of choices with each investment. We will investigate the effects of adjusting this constant variance in sections 4.1 and 4.2.

For our simplified initial dive into this problem, we do not optimize over the covariance matrix yet and iterate on the mean vector for units invested into the assets.

We follow the vanilla policy gradient approach presented in [3].

Definition 3.3.3: Policy evaluation is the means of determining how useful a policy is by measuring performance with an objective function [3]. For the vanilla policy gradient methods we employ, we do this by determining the expected future discounted rewards of the process and comparing the values.

We define the **value** of a policy as being the value $V^{\pi_\theta} = \mathbb{E}_{s_t,a_t}[Q^{\pi_\theta}(s_t, a_t)]$ where $Q^{\pi_\theta}(s_t, a_t)$ is a function that assigns a value to taking the action a_t in state s_t by following the policy π_θ in a Markov Decision Process at time t. This expectation is taken with respect to the visitation frequency for the states as actions are sampled from the policy.

For a vanilla policy gradient method, we sample states and rewards by collecting trajectories from running a fixed policy. Then we can estimate the value of the objective function as $\hat{V}^{\pi} \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_{reb}} \gamma^t R(s_t^{(i)}, a_t^{(i)})$ where $0 < \gamma < 1$ is the discount factor that describes to what extent a future reward deteriorates in value if it is rewarded one period later, and where we sample a total of N trajectories of length T_{reb} $(s_1^{(i)}, a_1^{(i)}, ..., s_{T_{reb}}^{(i)}, a_{T_{reb}}^{(i)}), i = 1, ..., N$ [3]. For our simulations, we do not include the discount factor because we already account for time value of money and inflation in the environment.

Algorithm 3.3.4: We define the MC policy evaluation algorithm as follows:

We will let $\vec{a}$ be the vector of the left endpoints of the coordinates of the multivariate truncated normal, $\vec{b}$ be the vector of the right endpoints of the coordinates of the multivariate truncated normal, d be the number of risky assets available for trading, N be a power of two encoding the number of simulated Markov chains, R be the chosen reward function of the RL environment, T_{reb} be the total number of rebalancing points, $\vec{\theta}$ be the variable representing the mean vector of the multivariate truncated normal policy, $\pi_{\vec{\theta}}$ represent the parametrized policy, σ be the constant standard deviation of each coordinate of the multivariate truncated normal policy, and $\varphi(s, a)$ represent the function that advances the Markov Chain in state s using action a.

```
Inputs: $\vec{a}, \vec{b}, d, N, R, T_{reb}, \vec{\theta}, \pi_{\vec{\theta}}, \sigma, \varphi$
  for $i = 1, \ldots, N$ do
      $X_0^{(i)} \leftarrow \vec{0}$
  end for
  for $t = 1, \ldots, T_{reb}$ do
      for $i = 1, \ldots, N$ do
          Generate $a_t^{(i)}$ from $\pi_{\vec{\theta}}$
          $X_t^{(i)} = \varphi(X_{t-1}^{(i)}, a_t^{(i)})$
          Store the reward $R_t^{(i)}$
      end for
```

```
    end for
    Return $\hat{V}^{\pi_{\vec{\theta}}} = \frac{1}{N}\Sigma_{i=1}^{N}\Sigma_{t=1}^{T_{reb}}R_t^{(i)}$
```

Normally, we use MC to sample iid actions at given states. To apply RQMC, we sample N scrambled Sobol' point sets of dimension $T_{reb} \times d$ where d is the number of assets in the market. At time step $t, 1 \leq t \leq T_{reb}$ for trajectory $i, 1 \leq i \leq N$ we use the coordinates $(t-1) \times d + 1$ to $t \times d$ of the Sobol' point to sample the action with an inverse of the CDF [3].

Remark 3.3.5: Another option instead of sampling a high-dimensional Sobol' point set at the beginning of the algorithm is to sample new RQMC points at every time step. This choice of algorithm retains unbiasedness with respect to the uniformity but loses low-discrepancy over $[0,1]^{T_{reb} \times d}$ [3].

Algorithm 3.3.6: We define the RQMC policy evaluation algorithm as follows:

We will let $\vec{a}$ be the vector of the left endpoints of the coordinates of the multivariate truncated normal, $\vec{b}$ be the vector of the right endpoints of the coordinates of the multivariate truncated normal, d be the number of risky assets available for trading, F^{-1} represent the inverse CDF of the policy $\pi_{\vec{\theta}}$, N be a power of two encoding the number of simulated Markov chains, R be the chosen reward function of the RL environment, T_{reb} be the total number of rebalancing points, $\vec{\theta}$ be the variable representing the mean vector of the multivariate truncated normal policy, σ be the constant standard deviation of each coordinate of the multivariate truncated normal policy, and $\varphi(s,a)$ represent the function that advances the Markov Chain in state s using action a.

```
Inputs: $\vec{a}, \vec{b}, d, F^{-1}, N, R, T_{reb}, \vec{\theta}, \sigma, \varphi$
    for $i = 1, \ldots, N$ do
        $X_0^{(i)} \leftarrow \vec{0}$
    end for
    Get $N$ scrambled Sobol' points $u^{(1)}, \ldots, u^{(N)}$ of dimension $d \times T_{reb}$
    for $t = 1, \ldots, T_{reb}$ do
        for $i = 1, \ldots, N$ do
```

```
            $\omega_t^{(i)} \leftarrow [u_{(t-1)\times d+1}^{(i)}, \ldots, u_{t\times d}^{(i)}]$
            $a_t^{(i)} = F^{-1}(\omega_t^i | \vec{a}, \vec{b}, \vec{\theta}, \sigma^2 I)$
            $X_t^{(i)} = \varphi(X_{t-1}^{(i)}, a_t^{(i)})$
            Store the reward $R_t^{(i)}$
        end for
    end for
    Return $\hat{V}^{\pi_{\vec{\theta}}} = \frac{1}{N}\Sigma_{i=1}^{N}\Sigma_{t=1}^{T_{reb}} R_t^{(i)}$
```

3.4 Policy Evaluation Notebook

For the implementation of the algorithms in this paper, we ran code in the free versions of online Python 3 Google Colab notebooks with 12.7 GB of system RAM and 107.7 GB of disk RAM. We used two separate notebooks, one for policy evaluation and one for policy iteration. We implemented the reinforcement learning environment described in section 3.2 using Gymnasium [34].

We generated most pseudorandom numbers using the np Python library. The remaining pseudorandom numbers are used in the scipy generator for RQMC point sets. To initialize this generator, we generate a random integer with the np library and feed this random integer as the seed in the RQMC generator. This seed will be consistent for consistent initializations of the np random seed.

We vectorized our code with operations from the np library to maximize efficiency. In the event that the execution of code needs to be interrupted, we set up a loop at the beginning that burns the random numbers that would otherwise be used in the code. The remainder of a seed can be run by resetting the updated sets of parameters to feed into the algorithm and entering the number of iterations that need to be skipped to continue producing results.

We had the system generate plain text files and graphs from the data and save them in the online Google Drive folder, and issue a Telegram notification when a new set of data was available to download. In the event the code is interrupted, the output can be copied and pasted into a plain text document and then the additional content can be appended on later. For the amount of output that would be produced from running on one dataset, the number of lines would never exceed the limit at which some lines in this output would become inaccessible.

Before running, if the notification feature is desired it will be necessary to have access to Telegram and create a bot. Then, those credentials will need to be added into the code at the spots indicated by comments. But this is optional, and also other notification software can be used.

For the policy evaluation experiments, we conducted 30 independent experiments and we either averaged over the squared error relative to a ground-truth value to compute the MSE or calculated the variance. For computing the ground truth value to compare to, as the results could not be evaluated theoretically we calculated a MC estimate on a set of 2^{14} Markov Chains, similarly to what is done in [3] for the MuJoCo tasks.

We compare MC against RQMC to determine which numerical method is the most appropriate for valuation of policies under different conditions. Each policy considered is truncated normal with a covariance matrix of $\sigma^2 I, \sigma > 0$ as in section 3.3 and a mean vector which depends on the policy keyword.

We considered three different policies:

1. The **Do Nothing** policy, which has a zero mean vector (on average nothing is invested into every asset);
2. The **One Over N** policy, which invests 10 units at every rebalancing point into every asset;
3. The **Long-Short Equity** policy, which longs 25 units of one asset and shorts 25 units of another asset where the two assets are in negatively correlated sectors (for environments with at least 10 assets).

For policy evaluation we specify the following hyperparameters in the functions when running:

1. The constant policy standard deviation σ;
2. A vector with the set of values N to iterate over, with each N being the total number of realizations of the stochastic processes for a given iteration.

We specify the following parameters in the code to alter the external conditions of the environment:

1. The values indicating a strong and weak correlation between two assets;

2. The transaction limit for buying and selling the shares each day (endpoints of the truncated normal distribution);
3. A Boolean indicating whether or not jumps are present if a GBM is used;
4. The initial cash budget the agent begins with at time 0;
5. The time horizon of the trading simulation T;
6. The total number of equally spaced rebalancing points T_{reb};
7. The total number of risky assets in the market d;
8. The underlying pricing model (GBM or Heston);
9. The reward functions for the two algorithms (either the modified simple return, arctangents of modified simple returns or the arctangents of Sharpe ratio differences);
10. An indicator of the dataset selected;
11. A keyword indicating which policy is being evaluated (either the Do Nothing, One Over N or Long-Short Equity strategy).

And we specify the following computational parameters to allow the code to be run in chunks and picked up on later:

1. A keyword indicating if the observed data will consist of the MSEs or variances;
2. The set of squared error average, maximum and minimum data so far that is used to generate a graph of the results if the MSE is used;
3. The set of variances so far that is used to generate a graph of the results if the variance is used;
4. Boolean values indicating whether to generate graphs of the assets in the dataset and a summary graph of the results.

Before beginning running the code, the financial environment conditions are initialized, the dataset for the financial assets is generated, and the mean vector of the policy is initialized based on the policy selected.

Each iteration occurs over one value of N, which is the number of trajectories sampled for both MC and RQMC. When all the values of N have been processed the algorithm terminates. If the MSE is plotted, as the algorithm runs for both MC and RQMC it stores the MSE, the maximum squared error and the minimum squared error over the set of 30 independent trials. This was done instead of using the mean and standard deviation to give some idea of the variability of data while avoiding domain errors caused by the log scale of the graph and standard deviations frequently exceeding the MSE. If the variance is analyzed instead there are not uncertainty bars.

A single iteration of the code run in the notebook consists of the following:

1. Compute the ground-truth value with MC policy evaluation.

2. Compute the average policy value 30 times independently using both MC and RQMC policy evaluation.

3. Compute the squared error compared to the ground-truth value for each trial and keep track of the MSE, maximum and minimum.

3.5 Policy Iteration

With policy evaluation, we took a fixed policy and critiqued its performance using a value function. On the other hand, policy iteration is the means of improving a policy through iterations of updates that aim to maximize its expected future rewards [3].

Theorem 3.5.1: By the policy gradient theorem [35], the gradient of the value function with respect to the optimized parameters $\vec{\theta}$ is given by

$$\nabla V^{\pi_{\vec{\theta}}} = \mathbb{E}_{s,a}[Q^{\pi_{\vec{\theta}}}(s,a)\nabla_{\vec{\theta}} \log \phi(a)].$$

where $\pi_{\vec{\theta}}$ is the policy that is being improved, $Q^{\pi_{\vec{\theta}}}$ is the sum of future rewards under this policy, ϕ is the pdf of the policy $\pi_{\vec{\theta}}$, and where the expectation is taken with respect to the visitation frequency of the states s as actions a are sampled from the policy $\pi_{\vec{\theta}}$.

For a vanilla policy gradient policy iteration method, we assume a constant form of the distribution and use iteration to refine its parameters. We do this by collecting trajectories of actions and states $(s_1^{(i)}, a_1^{(i)}, \dots, s_{T_{reb}}^{(i)}, a_{T_{reb}}^{(i)}), i = 1, \dots, N$ as in the policy evaluation methods to use these actions and rewards for updating the vector $\vec{\theta}$ using gradient ascent [3].

Lemma 3.5.2: For a truncated normal policy $TN(\vec{\theta}, \Sigma, \vec{a}, \vec{b})$ we have the following formula for the gradient of its pdf ϕ with respect to $\vec{\theta}$: $\nabla_{\vec{\theta}} \log \phi(\vec{x}) = \Sigma^{-1}(\vec{x} - \vec{\theta})$.

Proof.

$$\nabla_{\vec{\theta}} \log \phi(\vec{x}) = \nabla_{\vec{\theta}} \log \frac{\frac{1}{(2\pi)^{d/2}} \det(\Sigma)^{-1/2} exp(-\frac{1}{2}(\vec{x} - \vec{\theta})^T \Sigma^{-1} (\vec{x} - \vec{\theta}))}{F(\vec{b}) - F(\vec{a})}$$

$$= \nabla_{\vec{\theta}} \log \frac{1}{(2\pi)^{d/2}} \det(\Sigma)^{-1/2} - \nabla_{\vec{\theta}} \frac{1}{2}(\vec{x} - \vec{\theta})^T \Sigma^{-1} (\vec{x} - \vec{\theta}) - \nabla_{\vec{\theta}} \log(F(\vec{b}) - F(\vec{a}))$$

$$= 0 - \frac{1}{2} 2\Sigma^{-1}(\vec{x} - \vec{\theta}) \times -1 - 0 = \Sigma^{-1}(\vec{x} - \vec{\theta}).$$

□

In the case where $\Sigma = \sigma^2 I$ for $\sigma > 0$ we have $\Sigma^{-1} = \frac{1}{\sigma^2} I$ so $\nabla_{\vec{\theta}} \log \phi(\vec{x}) = \frac{1}{\sigma^2}(\vec{x} - \vec{\theta})$.

By the policy gradient theorem, we have that $\nabla V^{\pi_{\vec{\theta}}} = \mathbb{E}_{s,a}[Q^{\pi_{\vec{\theta}}}(s, a) \nabla_{\vec{\theta}} \log \pi_{\vec{\theta}}(s)]$ where $Q^{\pi_{\vec{\theta}}}(s, a)$ can be approximated by $R(\tau)$, the sum of rewards in a realization τ of the Markov Chain where the action a is taken in state s.

So we will update the vector $\vec{\theta}$ after every iteration k by $\vec{\theta}_k = \vec{\theta}_{k-1} + \alpha R(\tau) \frac{1}{\sigma^2}(\vec{x} - \vec{\theta}_{k-1})$, as shown in [35].

Algorithm 3.5.3: We define the MC policy iteration algorithm, following the vanilla policy gradient algorithm, as follows:

We will let $\vec{a}$ be the vector of the left endpoints of the coordinates of the multivariate truncated normal, $\vec{b}$ be the vector of the right endpoints of the coordinates of the multivariate truncated normal, d be the number of risky assets available for trading, K be the number of algorithm iterations, N be a power of two encoding the number of simulated Markov chains, R be the chosen reward function of the RL environment, T_{reb} be the total number of rebalancing points, $\pi_{\vec{\theta}}$ represent the paramterized policy, σ be the constant standard deviation of each coordinate of the multivariate truncated normal policy, and $\varphi(s,a)$ represent the function that advances the Markov chain in state s using action a.

```
Inputs: $\vec{a}, \vec{b}, d, K, N, R, T_{reb}, \pi_{\vec{\theta}}, \sigma, \varphi$
  $\vec{\theta}_0 \leftarrow \vec{0}$
  for $1 \leq k \leq K$ do
      for $i = 1, \ldots, N$ do
          $X_0^{(i)} \leftarrow \vec{0}$
      end for
      for $t = 1, \ldots, T_{reb}$ do
          for $i = 1, \ldots, N$ do
              Sample $a_t^{(i)}$ from $\pi_{\vec{\theta}_{k-1}}$
              $X_t^{(i)} \leftarrow \varphi(X_{t-1}^{(i)}, a_t^{(i)})$
              Store the reward $R_t^{(i)}$ and action $a_t^{(i)}$
          end for
      end for
      $\vec{\theta}_k \leftarrow \vec{\theta}_{k-1} + \alpha \frac{1}{N} \Sigma_{i=1}^{N} \Sigma_{t=1}^{T_{reb}} R(\tau^{(i)}) \frac{1}{\sigma^2}(a_t^{(i)} - \vec{\theta}_{k-1}),$ where $R(\tau^{(i)}) := \Sigma_{t=1}^{T_{reb}} R_t^{(i)}$
  end for
  Return $\vec{\theta}_K$ as the updated mean vector
```

For the RQMC policy iteration algorithm we apply RQMC in the same way as in the policy evaluation algorithm, for sampling the actions.

Algorithm 3.5.4: We define the RQMC policy iteration algorithm as follows:

We will let $\vec{a}$ be the vector of the left endpoints of the coordinates of the multivariate truncated normal, $\vec{b}$ be the vector of the right endpoints of the coordinates of the multivariate truncated normal, d be the number of risky assets available for trading, F^{-1} represent the inverse CDF of the policy $\pi_{\vec{\theta}}$, K be the number of algorithm iterations, N be a power of two encoding the number of simulated Markov chains, R be the chosen reward function of the RL environment, T_{reb} be the total number of rebalancing points, σ be the constant standard deviation of each coordinate of the multivariate truncated normal policy, and $\varphi(s, a)$ represent the function that advances the Markov chain in state s using action a.

```
Inputs: $\vec{a}, \vec{b}, d, F^{-1}, K, N, R, T_{reb}, \sigma, \varphi$
  $\vec{\theta}_0 \leftarrow \vec{0}$
  **for** $1 \leq k \leq K$ **do**
      **for** $i = 1, \ldots, N$ **do**
          $X_0^{(i)} \leftarrow \vec{0}$
      **end for**
      Get $N$ scrambled Sobol' points $u^{(1)}, \ldots, u^{(N)}$ of dimension $d \times T_{reb}$
      **for** $t = 1, \ldots, T_{reb}$ **do**
          **for** $i = 1, \ldots, N$ **do**
              $\omega_t^{(i)} \leftarrow [u_{(t-1)\times d+1}^{(i)}, \ldots, u_{t\times d}^{(i)}]$
              $a_t^{(i)} = F^{-1}(\omega_t^i | \vec{a}, \vec{b}, \vec{\theta}_{k-1}, \sigma^2 I)$
              $X_t^{(i)} = \varphi(X_{t-1}^{(i)}, a_t^{(i)})$
              Store the reward $R_t^{(i)}$ and action $a_t^{(i)}$
          **end for**
      **end for**
      $\vec{\theta}_k \leftarrow \vec{\theta}_{k-1} + \alpha \frac{1}{N} \Sigma_{i=1}^{N} \Sigma_{t=1}^{T_{reb}} R(\tau^{(i)}) \frac{1}{\sigma^2} (a_t^{(i)} - \vec{\theta}_{k-1}),$ where $R(\tau^{(i)}) := \Sigma_{t=1}^{T_{reb}} R_t^{(i)}$
  **end for**
  Return $\vec{\theta}_K$ as the updated mean vector
```

For Array-RQMC, we test three different algorithms to determine which performs best. First, we have the standard Array-RQMC algorithm, which permutes only the states and which selects the first ten assets for its dimensional reduction. As a slight variation of this method, we also test the effect of permuting the RQMC points as well in the results in section 4.2.

Algorithm 3.5.5: We define the Array-RQMC FIRST TEN policy iteration algorithm as follows:

We will let $\vec{a}$ be the vector of the left endpoints of the coordinates of the multivariate truncated normal, $\vec{b}$ be the vector of the right endpoints of the coordinates of the multivariate truncated normal, d be the number of risky assets available for trading, F^{-1} represent the inverse CDF of the policy $\pi_{\vec{\theta}}$, K be the number of algorithm iterations, N be a power of two encoding the number of simulated Markov chains, R be the chosen reward function of the RL environment, T_{reb} be the total number of rebalancing points, σ be the constant standard deviation of each coordinate of the multivariate truncated normal policy, and $\varphi(s, a)$ represent the function that advances the Markov chain in state s using action a.

```
Inputs: $\vec{a}, \vec{b}, d, F^{-1}, K, N, R, T_{reb}, \sigma, \varphi$
  $\theta_0 \leftarrow \vec{0}$
  for $1 \leq k \leq K$ do
    for $i = 1, \ldots, N$ do
      $X_0^{(i)} \leftarrow \vec{0}$
    end for
    $\beta_1 \leftarrow (1, 2, ..., N)$
    for $t = 1, \ldots, T_{reb}$ do
      Get $N$ scrambled Sobol' points $u_t^{(1)}, \ldots, u_t^{(N)}$ of dimension $d$
      for $i = 1, \ldots, N$ do
        $a_t^{(i)} = F^{-1}(u_t^i | \vec{a}, \vec{b}, \vec{\theta}_{k-1}, \sigma^2 I)$
        $X_t^{(i)} = \varphi(X_{t-1}^{(\beta_t(i))}, a_t^{(i)})$
        Store the reward $R_t^{(i)}$ and action $a_t^{(i)}$
      end for
      Get the permutation $\beta_{t+1}$ of $X_t^{(i)}, 1 \leq i \leq N$ based on a split sort on the coordinates
  $(1, ..., \min(d, 10))$ in order
    end for
    $\vec{\theta}_k \leftarrow \vec{\theta}_{k-1} + \alpha \frac{1}{N} \Sigma_{i=1}^{N} \Sigma_{t=1}^{T_{reb}} R(\tau^{(i)}) \frac{1}{\sigma^2} (a_t^{(i)} - \vec{\theta}_{k-1}),$ where $R(\tau^{(i)}) := \Sigma_{t=1}^{T_{reb}} R_t^{(i)}$
  end for
  Return $\vec{\theta}_K$ as the updated mean vector
```

For the BEST TEN algorithm, we employ an ϵ-greedy approach for multi-armed bandit problems, as discussed in [36]. The exploitation occurs by identifying what the agent has learned to be the most influential coordinates to achieving high rewards through the gradient ascent algorithm by using MC (and not RQMC) to perform the first step of the policy iteration algorithm and then choosing at most ten (or the number of assets, if there are fewer than ten) coordinates to prioritize based on decreasing absolute value of the coordinates. Because the ordering of assets in the market is arbitrary and all assets begin with the same price, this attempts to prioritize the assets it finds to be most effective to invest in when applying an unbiased method for estimating the policy gradients, as the higher absolute values indicate they were influential in increasing the rewards during gradient ascent.

In the context of a multi-armed bandit problem, we exploit the coordinates identified during the first step of the policy iteration algorithm 95% of the time and choose a random set of coordinates 5% of the time.

We apply MC instead of RMC to allow the algorithm to be extended to very high dimensions like the 1000-asset case that is programmed into the environment. We do $K-1$ iterations after, to allow for a fair comparison with the other algorithms (as it will sample the same number of actions, states and rewards as them when improving its policy).

Remark 3.5.6: While this particular method is not used in the literature, the literature does mention many examples of sorting functions chosen inventively to improve performance of the algorithm, such as in [16]. We argue this formulation is consistent with other creative application-dependent choices of sorting functions.

Algorithm 3.5.7: We define the Array-RQMC BEST TEN policy iteration algorithm as follows:

We will let $\vec{a}$ be the vector of the left endpoints of the coordinates of the multivariate truncated normal, $\vec{b}$ be the vector of the right endpoints of the coordinates of the multivariate truncated normal, d be the number of risky assets available for trading, F^{-1} represent the inverse CDF of the policy $\pi_{\vec{\theta}}$, K be the number of algorithm iterations, N be a power of two encoding the number of simulated Markov chains, R be the chosen reward function of the RL environment, T_{reb} be the total number of rebalancing points, σ be the constant standard deviation of each coordinate of the multivariate truncated normal policy, and $\varphi(s, a)$ represent the function that advances the Markov chain in state s using action a.

```
Inputs: $\vec{a}, \vec{b}, d, F^{-1}, K, N, R, T_{reb}, \sigma, \varphi$
  $\vec{\theta}_0 \leftarrow \vec{0}$
  Apply one iteration of MC policy iteration as above and get the vector $\vec{\theta}_1$
  Order $\vec{\theta}_1$ in decreasing absolute value of its coordinates, call this $\vec{\theta}'_1$
  Let $C_{best}$ be the set of $\min(d, 10)$ first indices from $\vec{\theta}_1$ in the reordered vector $\vec{\theta}'_1$
  for $2 \leq k \leq K$ do
    for $i = 1, \dots, N$ do
      $X_0^{(i)} \leftarrow \vec{0}$
    end for
    $\beta_1 \leftarrow (1, 2, ..., N)$
    for $t = 1, \dots, T_{reb}$ do
      Get $N$ scrambled Sobol' points $u_t^{(1)}, \dots, u_t^{(N)}$ of dimension $d$
      for $i = 1, \dots, N$ do
        $a_t^{(i)} = F^{-1}(u_t^{(i)} | \vec{a}, \vec{b}, \vec{\theta}_{k-1}, \sigma^2 I)$
        $X_t^{(i)} = \varphi(X_{t-1}^{(\beta_t(i))}, a_t^{(i)})$
        Store the reward $R_t^{(i)}$ and action $a_t^{(i)}$
      end for
      $r \leftarrow$ Random integer between 1 and 100
      if $r > 95$ then
        Get the permutation $\beta_{t+1}$ of $X_t^{(i)}, 1 \leq i \leq N$ based on a split sort on $\min(d, 10))$ random coordinates
      else
        Get the permutation $\beta_{t+1}$ of $X_t^{(i)}, 1 \leq i \leq N$ based on a split sort on the coordinates in $C_{best}$ in order
      end if
    end for
    $\vec{\theta}_k \leftarrow \vec{\theta}_{k-1} + \alpha \frac{1}{N} \Sigma_{i=1}^{N} \Sigma_{t=1}^{T_{reb}} R(\tau^{(i)}) \frac{1}{\sigma^2} (a_t^{(i)} - \vec{\theta}_{k-1}),$ where $R(\tau^{(i)}) := \Sigma_{t=1}^{T_{reb}} R_t^{(i)}$
  end for
```

Return $\vec{\theta}_K$ as the updated mean vector

3.6 Policy Iteration Notebook

For the policy iteration experiments, we conducted 15 independent trials for each algorithm. We stop after 15 iterations in experiments rather than waiting until convergence.

The algorithm for Array-RQMC does not match exactly with the code, since the RQMC points are transformed into their corresponding actions outside the loop. This is done so that the vectorized operations can be applied outside the loop the minimal number of times, to speed up the algorithm. There is no consequence of this change since the inverse truncated normal distribution is applied to each coordinate separately as they are independent and the inverse CDF is monotone increasing, so it does not matter that they are separated then reassembled into the actions.

At the beginning of one of the independent experiments, the mean vectors of the truncated normal policies are initialized to the zero vector. As the algorithm runs, we append the complete series of returns to a vector that stores all of the relevant data. We compute the means and standard deviations from this data directly when graphing the results. We initialize different experiments with different seeds, which are fed into pseudorandom number generators to produce different sequences of random numnbers.

For policy iteration we specify the following hyperparameters in the functions when running:

1. The learning rate α;
2. The constant standard deviation σ;
3. The total number of realizations of the stochastic processes N;
4. The dimensional reduction and sorting strategy for Array-RQMC, including a Boolean indicating whether or not the RQMC points are permuted and whether FIRST TEN or BEST TEN is employed.

We specify the following parameters in the code to alter the external conditions of the environment:

1. The values indicating a strong and weak correlation between two assets;

2. The transaction limit for buying and selling the shares each day (endpoints of the truncated normal distribution);
3. A Boolean indicating whether or not jumps are present;
4. The initial cash budget the agent begins with at time 0;
5. The time horizon of the trading simulation T;
6. The total number of equally spaced rebalancing points T_{reb};
7. The total number of assets in the market d;
8. The underlying pricing model (GBM or Heston);
9. The reward functions for the three algorithms (either the arctangents of modified simple returns or the arctangents of the Sharpe ratio differences);
10. Frequency of new data generation (either all iterations are run on the same dataset or every iteration is run on a new dataset);
11. If all iterations are run on a new dataset, an indicator of whether or not to link the datasets (have the prices at the end of the previous iteration's dataset being the starting prices of the next iteration's dataset).
12. If all iterations are run on the same dataset, an indicator of the dataset selected.

And we specify the following computational parameters to allow the code to be run in chunks and picked up on later:

1. Rewards collected so far for the numerical methods;
2. The number of seeds out of 15 that still need to be completed;
3. The number of iterations that were previously completed if the code was interrupted in the middle of a seed;
4. A Boolean indicating if just Array-RQMC is run or if MC and RQMC are included as well;

5. A set of the prioritized coordinates C_{best} from BEST TEN if it was interrupted in the middle of a seed;

6. The mean vectors produced for each numerical method run;

7. The set of data collected over all 15 random seeds generated so far that is used to generate a graph of the results;

8. Boolean values indicating whether to generate graphs of the assets in the dataset and a summary graph of the results.

Before beginning the iterations, the environment conditions are initialized and financial datasets to be used are generated.

To quantify how effectively the policy iteration is working, we have two performance measures that we print out as the algorithm runs:

1. RQMC policy evaluation. We choose RQMC for this task when possible because as we will show in section 4.1 it tends to outperform MC policy evaluation under all environment conditions. For the 1000-asset environment we specify for MC policy evaluation to be applied instead due to the dimension of the Sobol' point set otherwise exceeding what is supported by the scipy library.

2. The total simple portfolio return over the duration of the trading strategy. We also printed out the simple return generated from a $1/n$ portfolio strategy as a benchmark to compare the algorithms against in the universe of assets worked with.

4 Results

4.1 Data Collected for Policy Evaluation

When reporting our results we assume the following default parameters (when we differ from these defaults we will explicitly state this in the text and titles of the graphs):

1. The time horizon is $T = 250$ trading days (1 year).
2. There are $T_{reb} = 125$ rebalances.
3. There are $d = 10$ risky assets in the market.
4. There are $N = 2^{10}$ simulations for each numerical method (in cases where we do not iterate over increasing N in the graphs).
5. The policy assessed is the Do Nothing policy
6. The pricing model is the GBM with jumps.
7. The modified simple return over the rebalancing return is the reward, with no squashing from the arctangent function.
8. The strong correlation is 0.7 and the weak correlation is 0.3.
9. The database selected is generated from the first of the set of 30 random seeds worked over.
10. The value of σ for the correlation matrix $\Sigma = \sigma^2 I$ is 25.

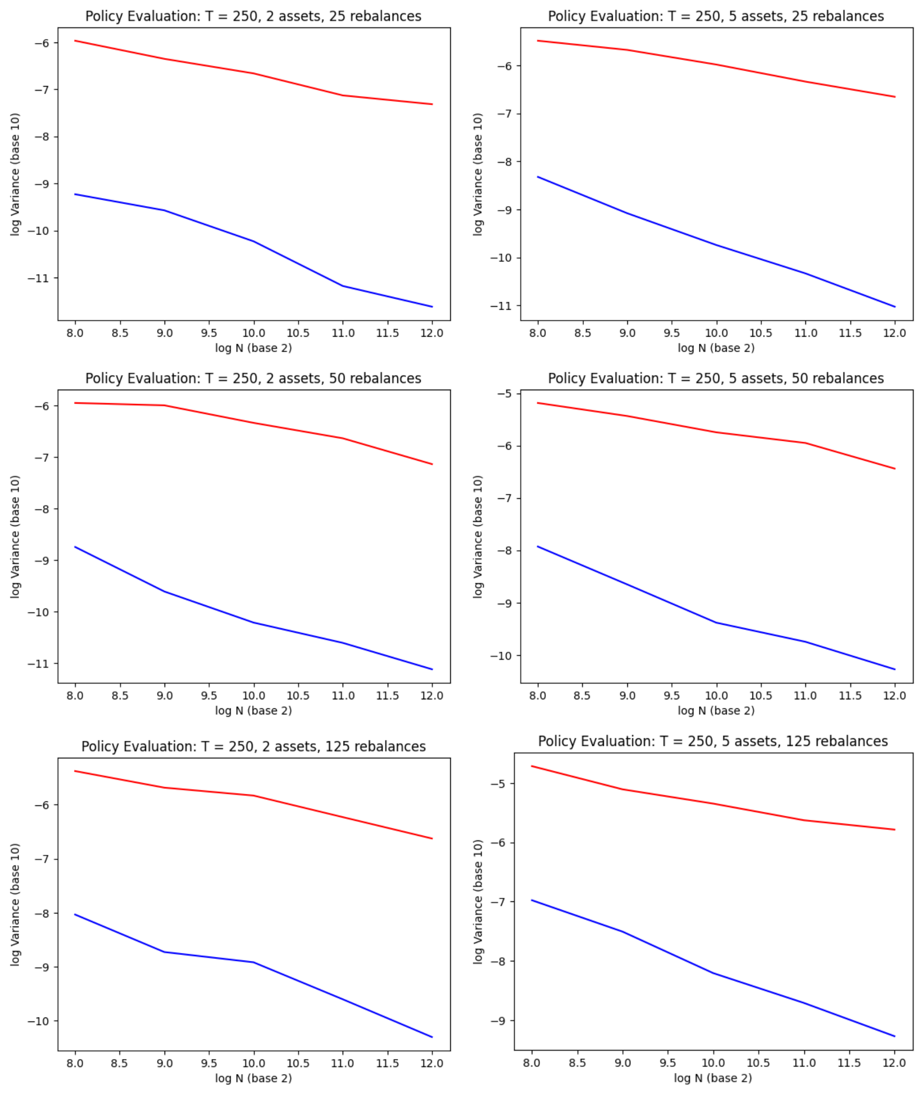


Figure 6: For each plot MC is graphed in red and RQMC is graphed in blue. The trend lines indicate the log of the variance of the estimators.

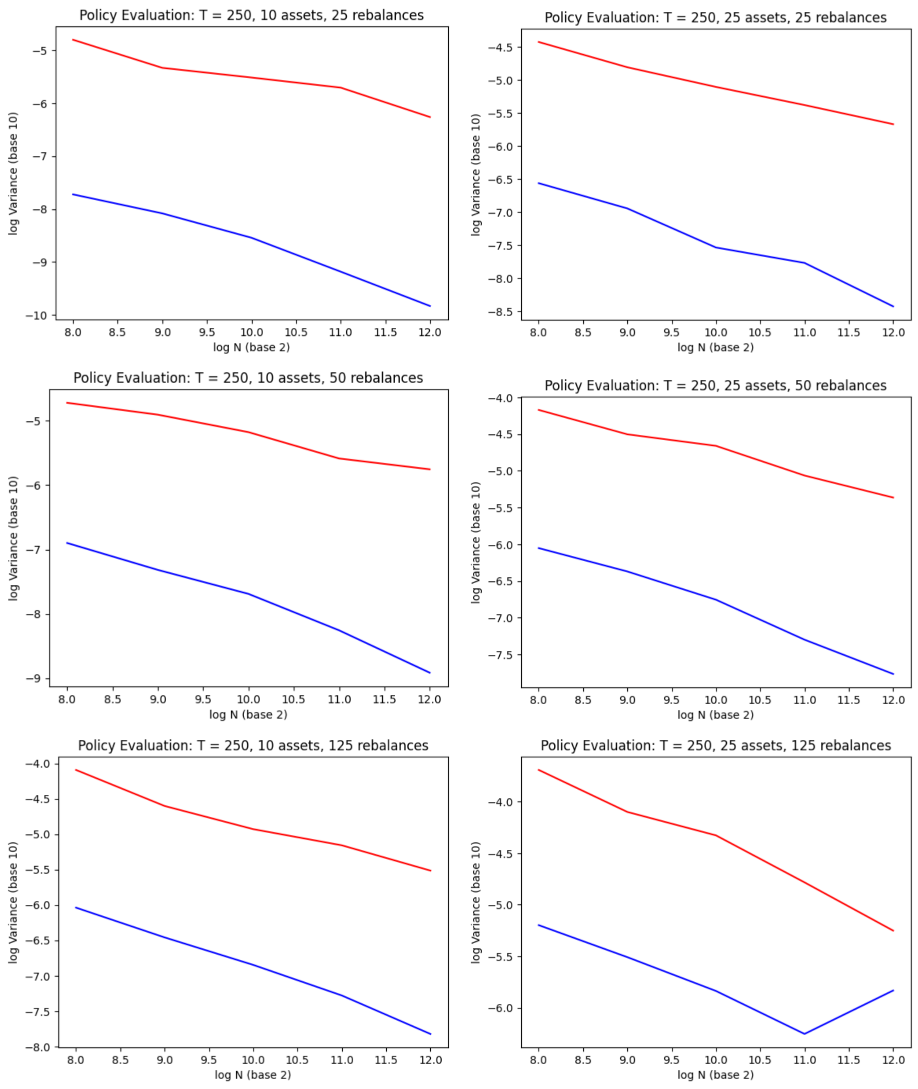


Figure 7: For each plot MC is graphed in red and RQMC is graphed in blue. The trend lines indicate the log of the variance of the estimators.

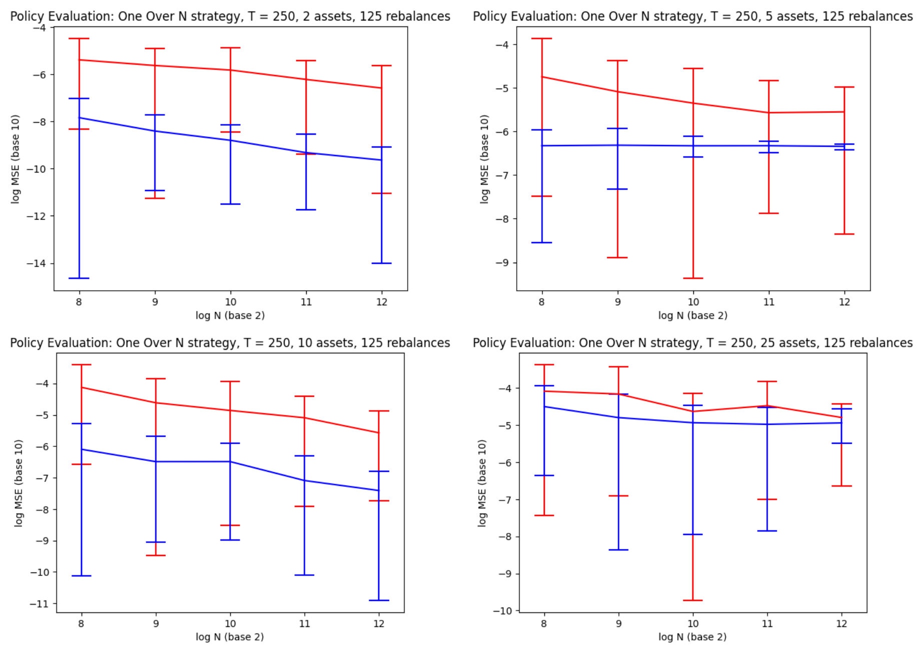


Figure 8: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. The 5-asset case shows a flattening performance of RQMC, this is likely due to the squared bias dominating the variance for this case, as is demonstrated by the maxima and minima converging very close together for higher number of points.

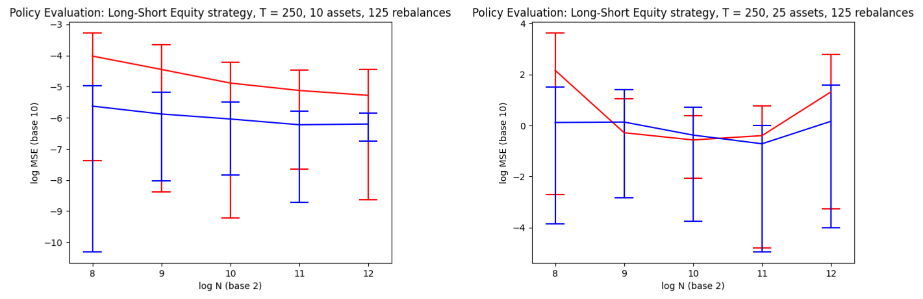


Figure 9: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. For this choice of reward function we see very poor policy evaluation performance for the 25-asset case.

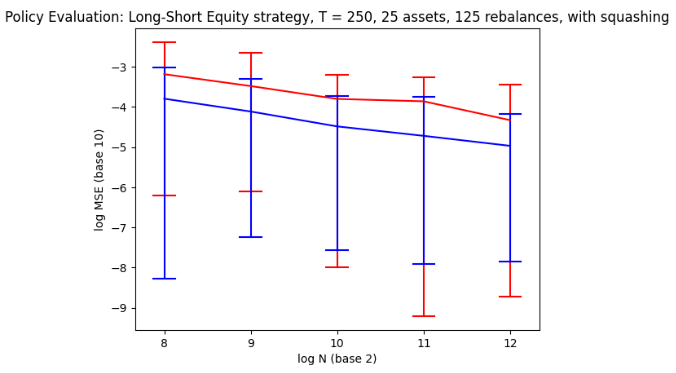


Figure 10: For this plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. For this choice of reward function we see better policy evaluation performance for the 25-asset case.

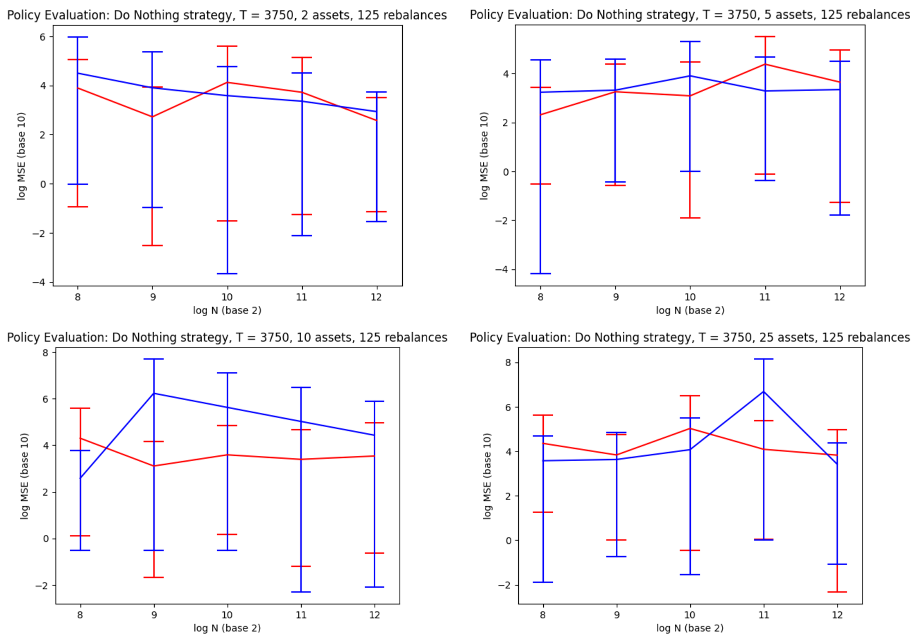


Figure 11: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. For this choice of reward function we see very poor policy evaluation performance for all cases.

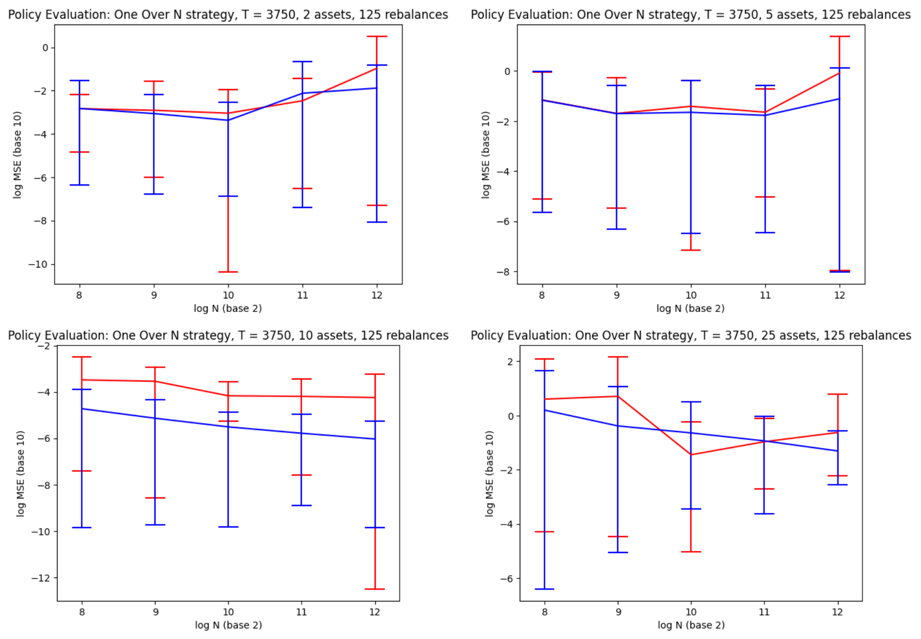


Figure 12: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. For this choice of reward function we see very poor policy evaluation performance for all cases except the 10-asset case.

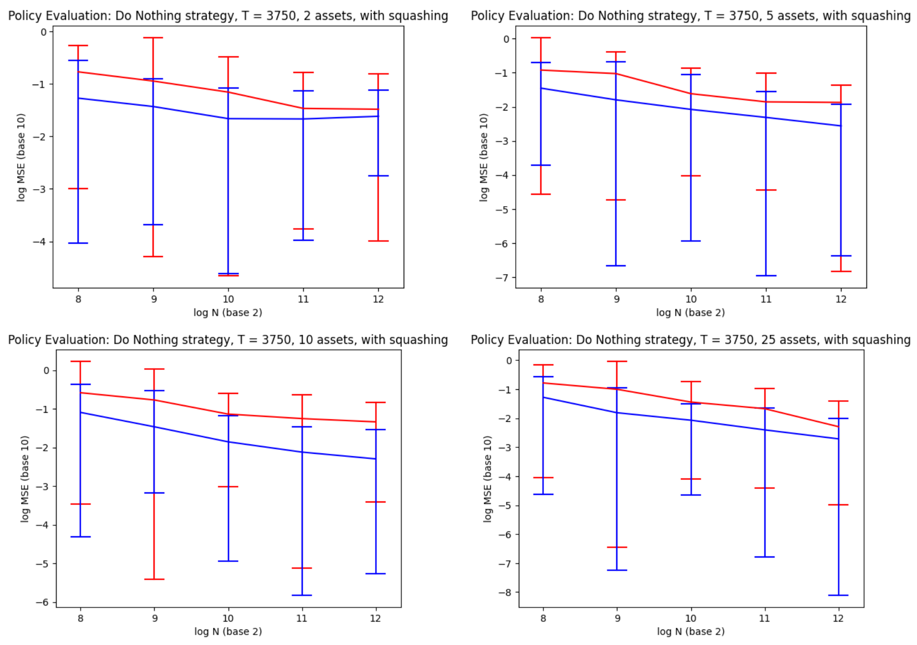
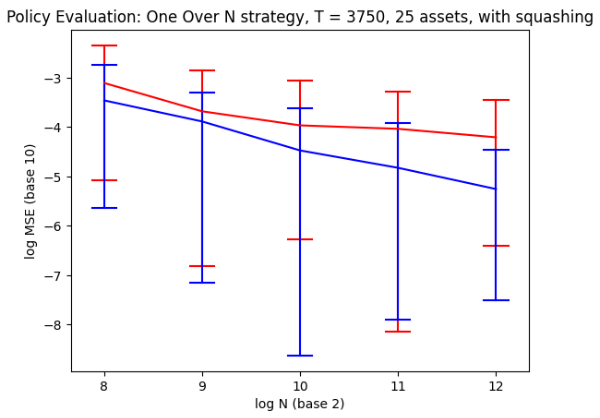
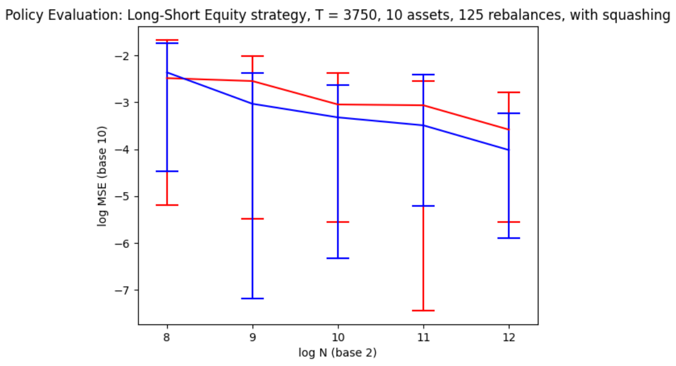


Figure 13: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. For this choice of reward function we see much better policy evaluation performance.





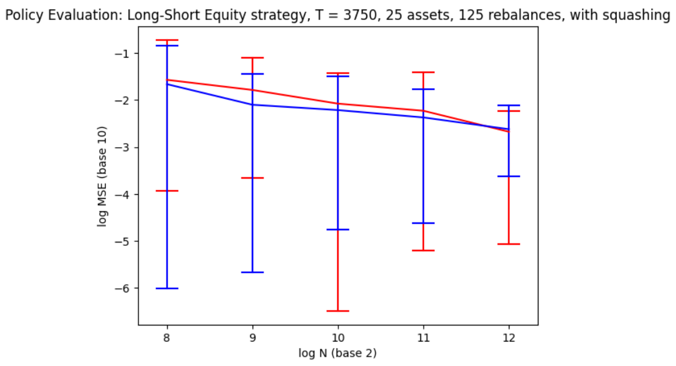
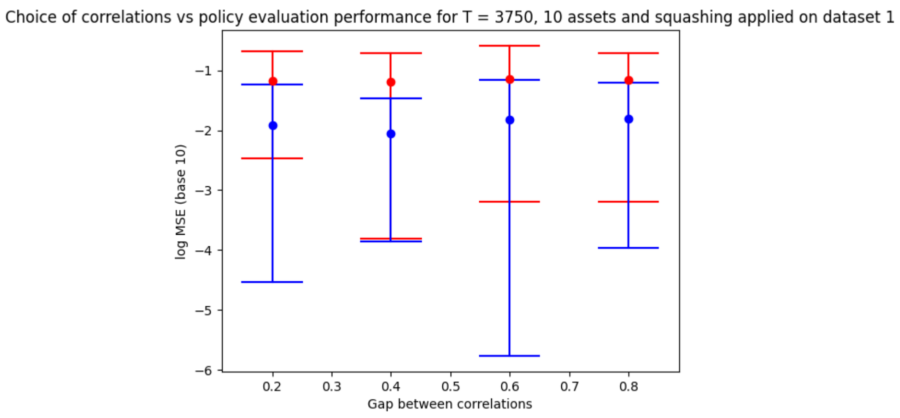
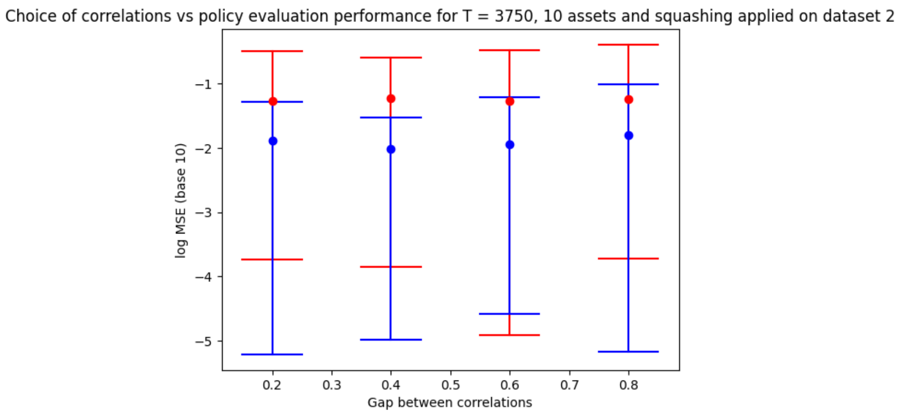


Figure 14: For this plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE. For this choice of reward function for the long-term One Over N policy we see much better policy evaluation performance.





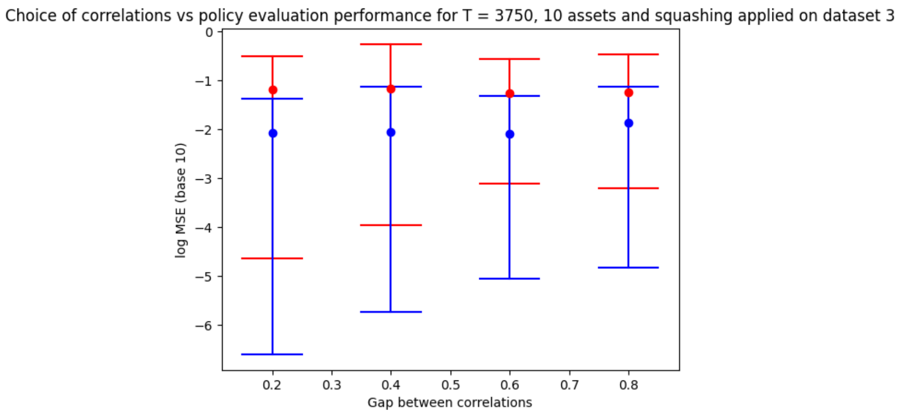


Figure 15: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random seeds, the lower bounds represent the minimum squared error, and the MSEs are plotted with large dots. These experiments used $N = 2^{10}$. The pairs of correlations looked at were 0.9 and 0.1, 0.8 and 0.2, 0.7 and 0.3, and 0.6 and 0.4.

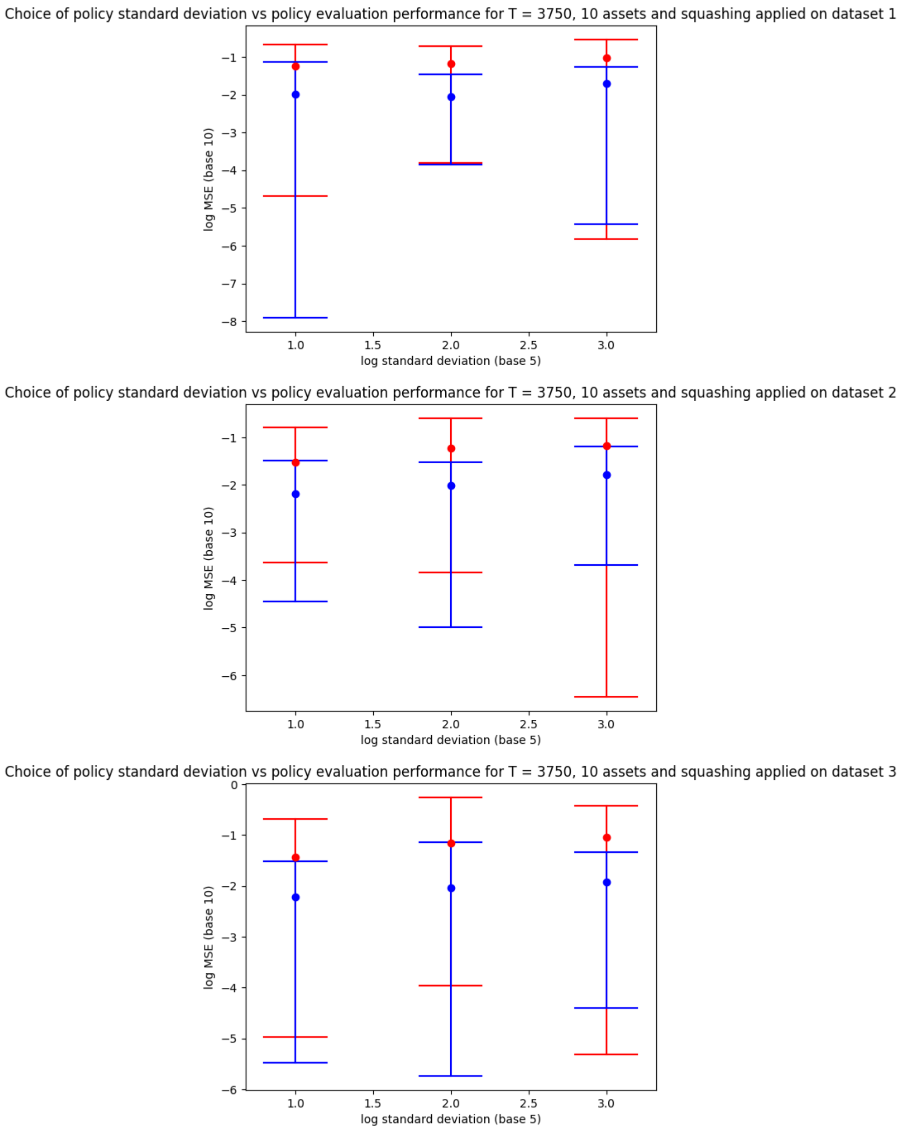
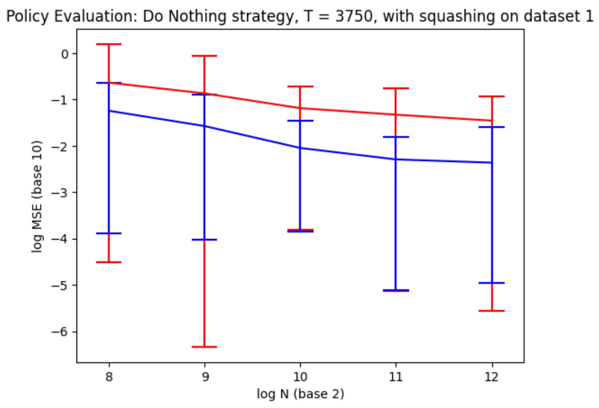
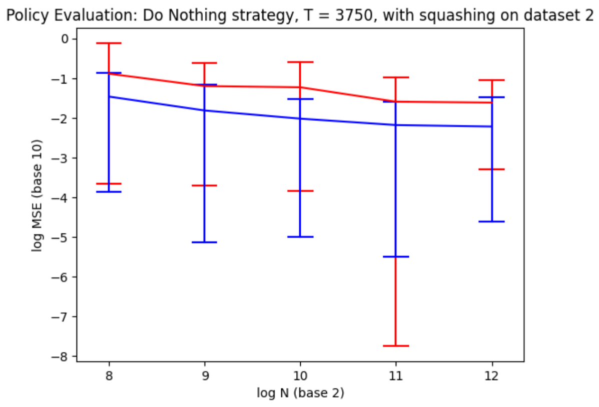


Figure 16: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random seeds, the lower bounds represent the minimum squared error, and the MSEs are plotted with large dots. These experiments used $N = 2^{10}$.





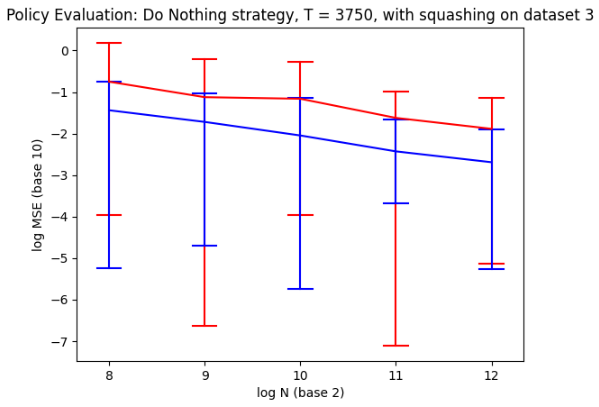


Figure 17: For this plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE.

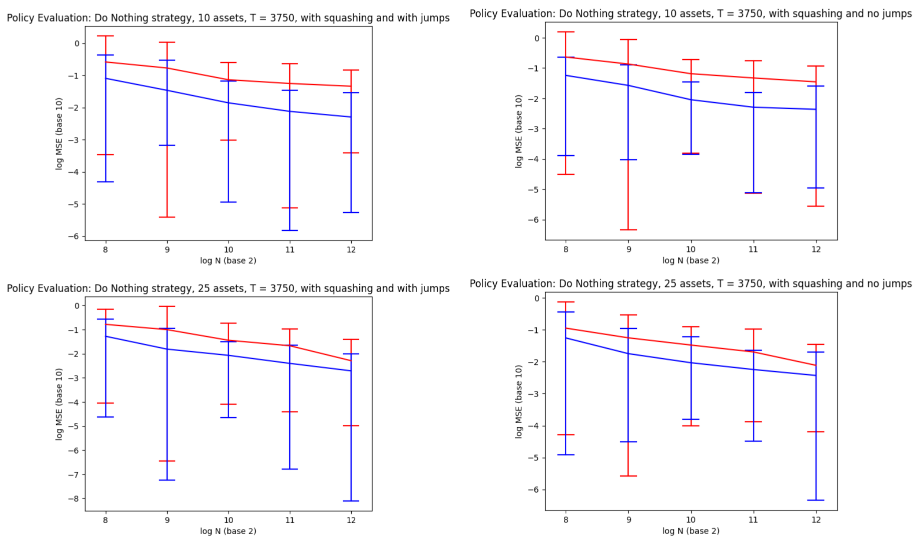


Figure 18: For this plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE.

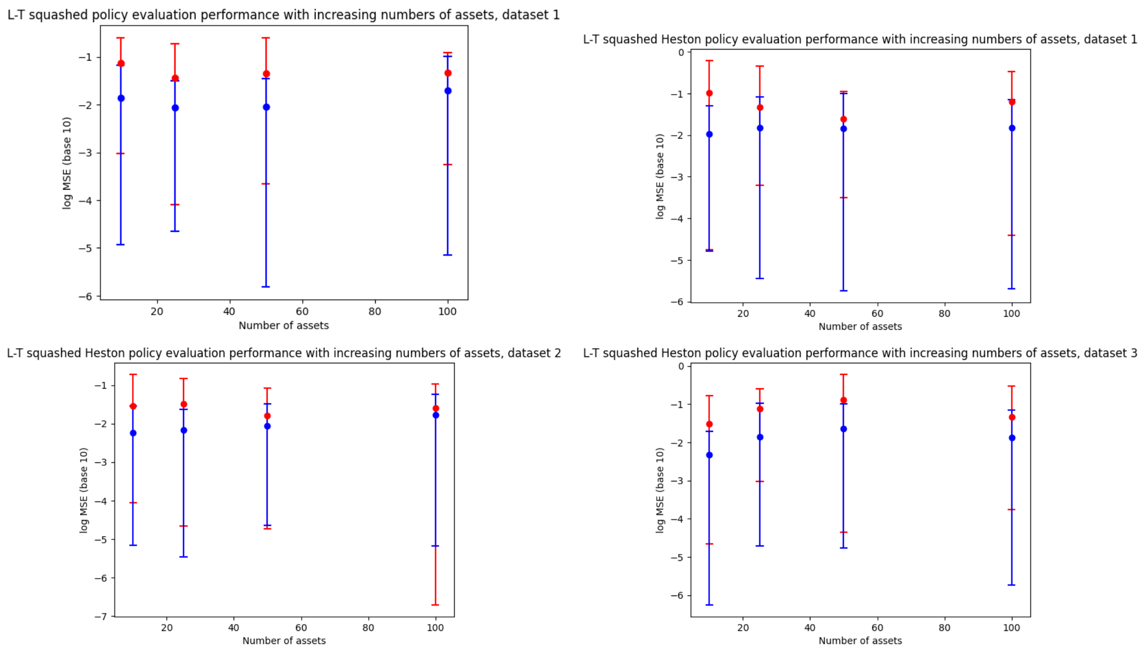


Figure 19: For each plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random seeds, the lower bounds represent the minimum squared error, and the MSEs are plotted with large dots. These experiments used $N = 2^{10}$.

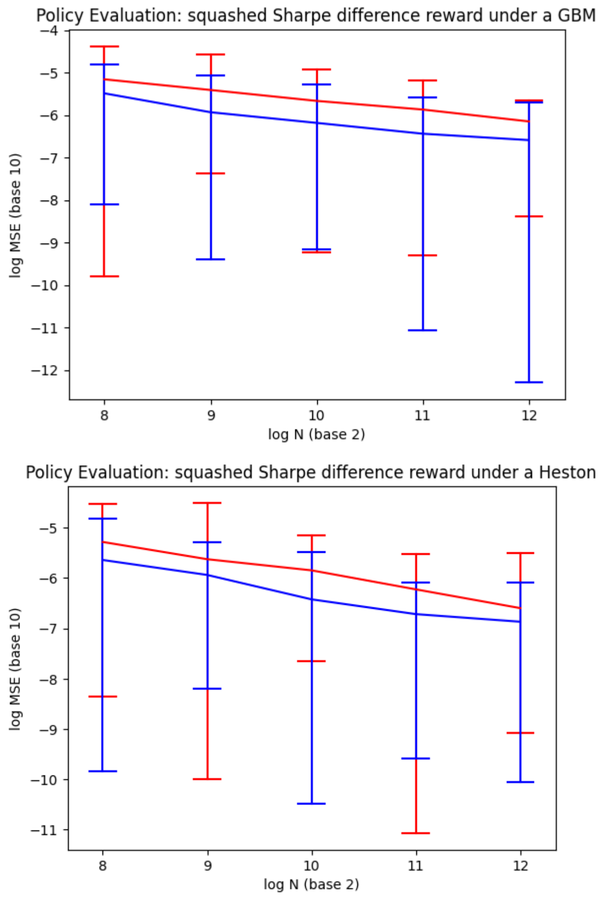


Figure 20: For this plot MC is graphed in red and RQMC is graphed in blue. The upper bounds represent the maximum squared error from the 30 random experiments, the lower bounds represent the minimum squared error, and the trend line shows the MSE.

We make the following observations about the results from the graphs:

1. RQMC consistently outperforms MC in all cases where the algorithms perform well by having a lower variance or MSE. From our graphs we note it seems to converge to zero at a similar logarithmic rate as MC as the number of sampled trajectories (and therefore states and actions) increases.

2. For the long-term time horizons in particular, we see that using a modified simple return the MSEs are significantly higher than what is acceptable for the values provided for the policies. This problem is fixed by squashing the reward with the arctangent function. As an example, see Figures 11 and 13.

3. From Figures 6 and 7, we see that increasing the number of rebalancing points for a policy tends to increase the variance of the short-term Do Nothing policy.

4. From Figure 16, we see that increasing the standard deviation of the truncated normal policy increases the MSE. This makes sense as a policy with a higher standard deviation should have results that deviate to a greater extent from the ground-truth value of the policy.

5. From Figure 19, we see that our results for small number of assets scale well to high-dimensional state spaces. In addition, we remark a small increasing trend in the MSE as the number of assets increases, but this is challenging to quantify exactly as the dataset changes as the number of assets changes (the graphs are over three choices of datasets for each number of assets). Further analysis on datasets that are very consistent between the numbers of assets would be required to determine if this trend exists.

6. From Figures 15, 17, 18, 19 and 20 we see that the choice of correlations, the selected dataset, the presence of jumps, and the use of the Heston model do not significantly impact the results from the experiments.

7. From Figure 20 we note that RQMC also outperforms MC for a choice of the arctangent of the difference in the Sharpe ratio as the reward, with relative performances similar to what we see with the squashed modified simple return reward function.

Remark 4.1.1: While it does appear that the policy evaluation methods perform better on the short-term cases than the long-term cases due to the smaller MSEs, the larger MSEs in the long-term cases are a consequence of the higher values of the policies over a longer time horizon. We do not find a significant difference in performance based on the time horizon alone.

4.2 Data Collected for Policy Iteration

When reporting our results we assume the following default parameters (when we differ from these defaults we will explicitly state this in the text and titles of the graphs):

1. The learning rate is $\alpha = 10$.
2. The time horizon is $T = 250$ trading days (1 year).
3. There are $T_{reb} = 125$ rebalances.
4. There are $d = 10$ risky assets in the market.
5. There are $N = 2^{10}$ simulations for each numerical method.
6. The pricing model is the GBM with jumps.
7. The modified simple return over the rebalancing return is the reward, with squashing from the arctangent function.
8. The strong correlation is 0.7 and the weak correlation is 0.3.
9. The database selected is generated from the first of the set of 15 random seeds worked over.
10. The value of σ for the correlation matrix $\Sigma = \sigma^2 I$ is 25.
11. A consistent dataset is used for all 15 iterations.

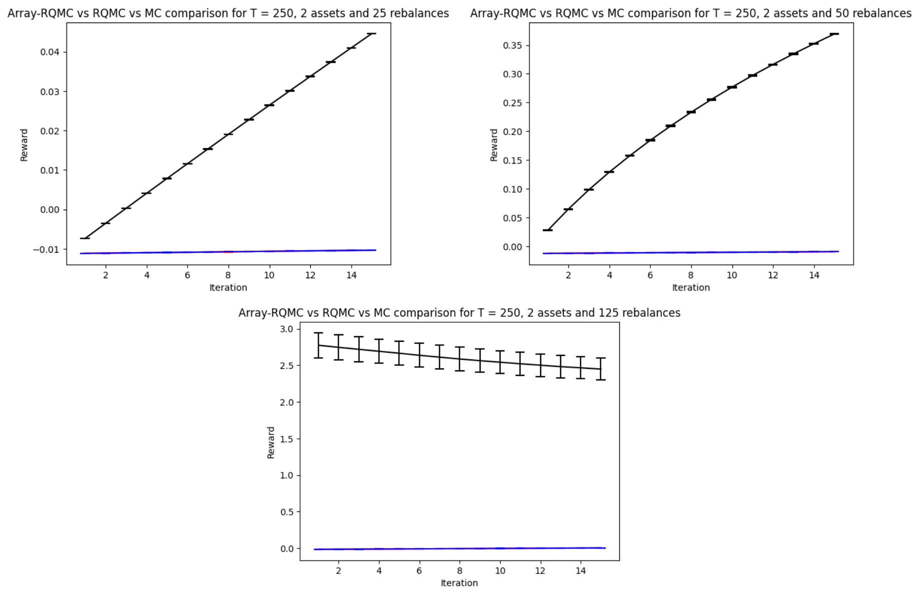


Figure 21: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

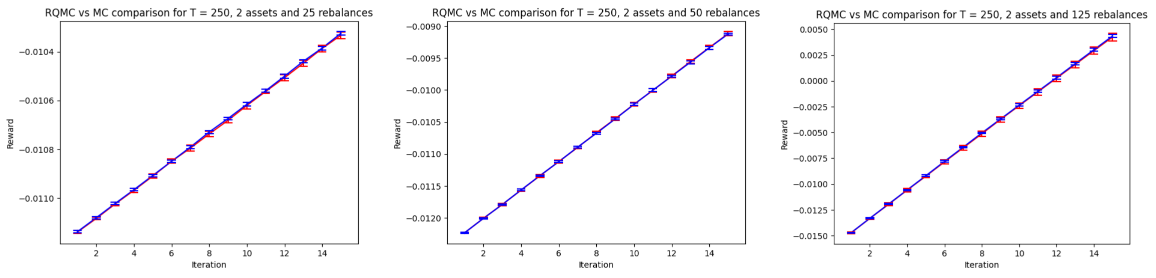


Figure 22: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

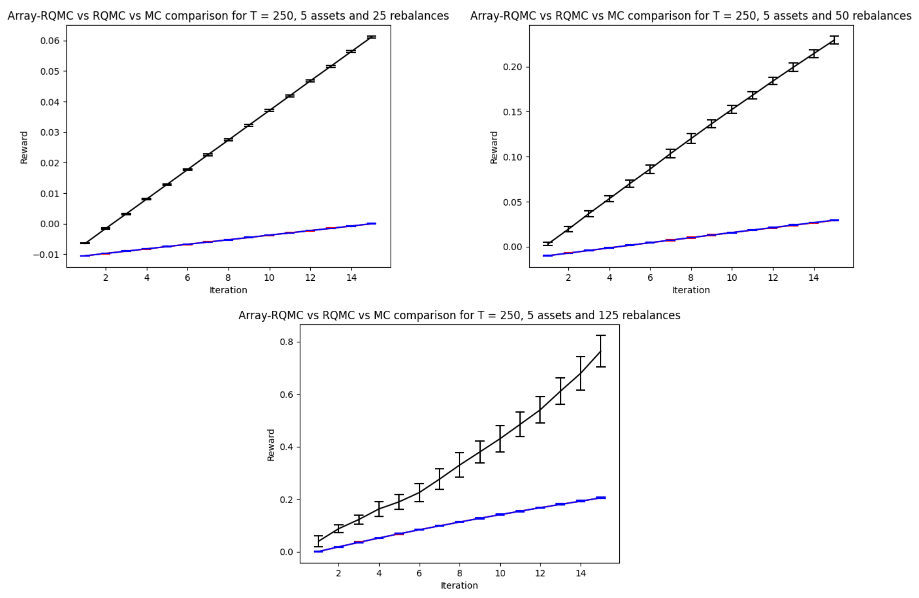


Figure 23: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

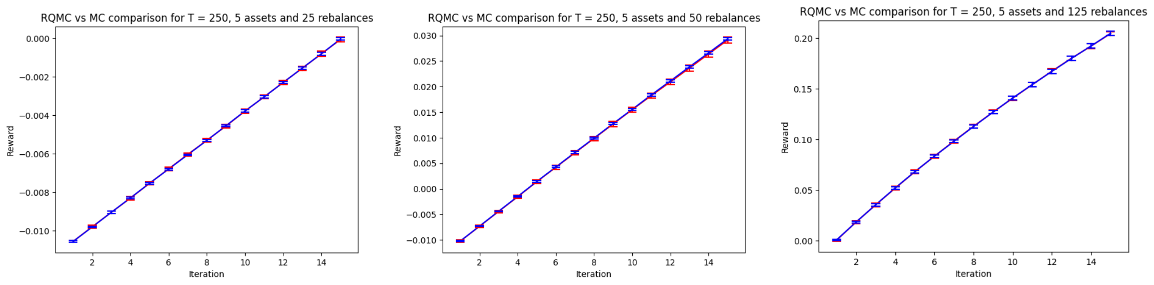


Figure 24: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

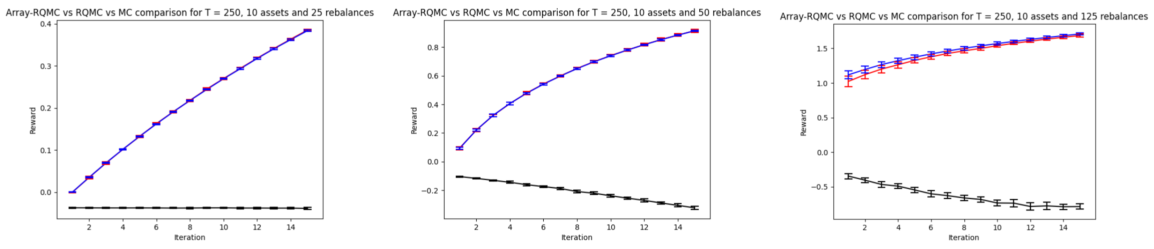


Figure 25: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

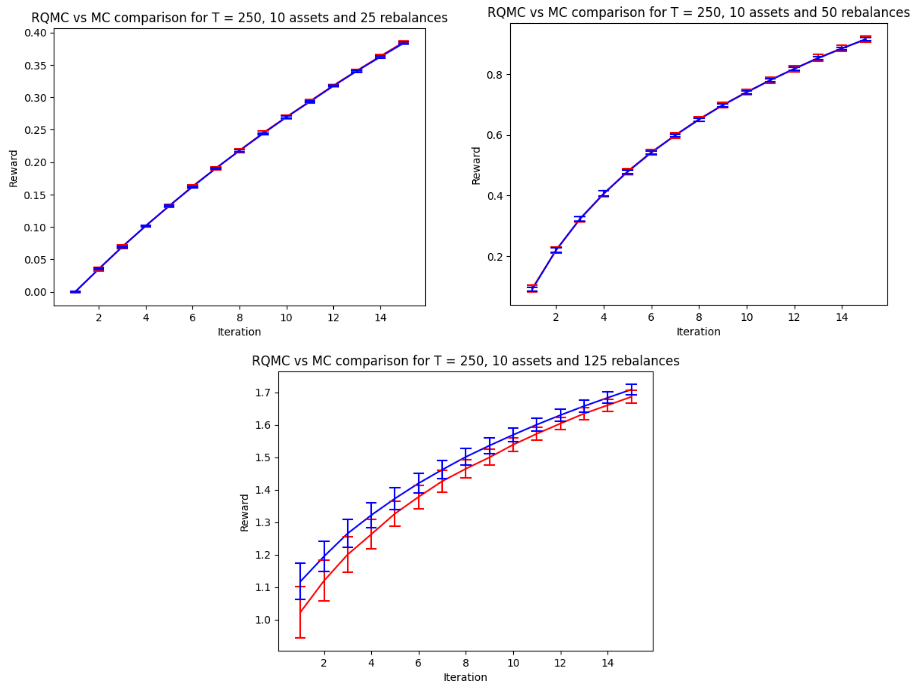


Figure 26: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

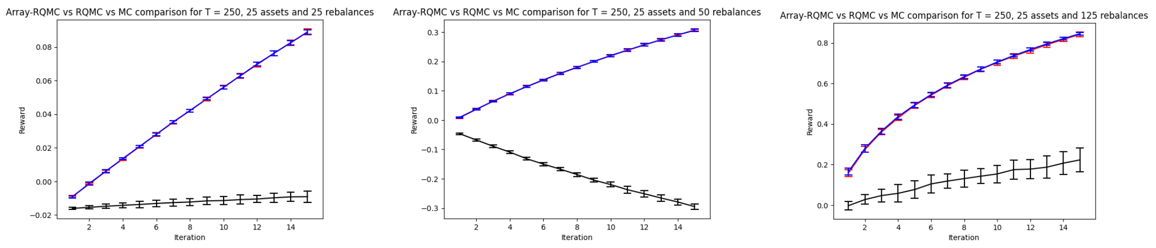


Figure 27: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

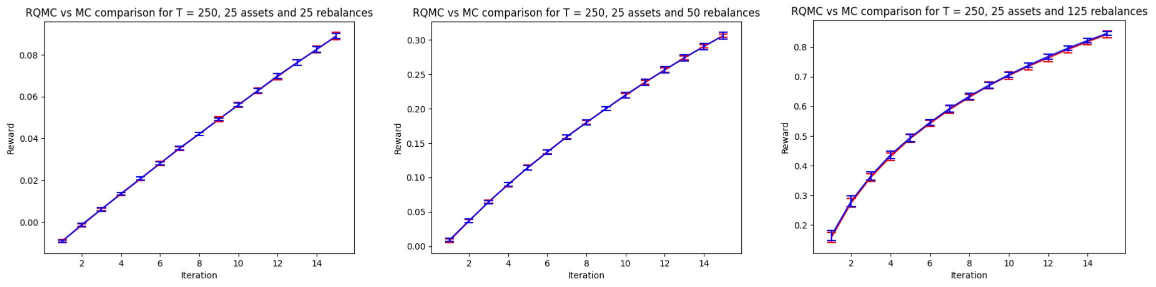


Figure 28: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

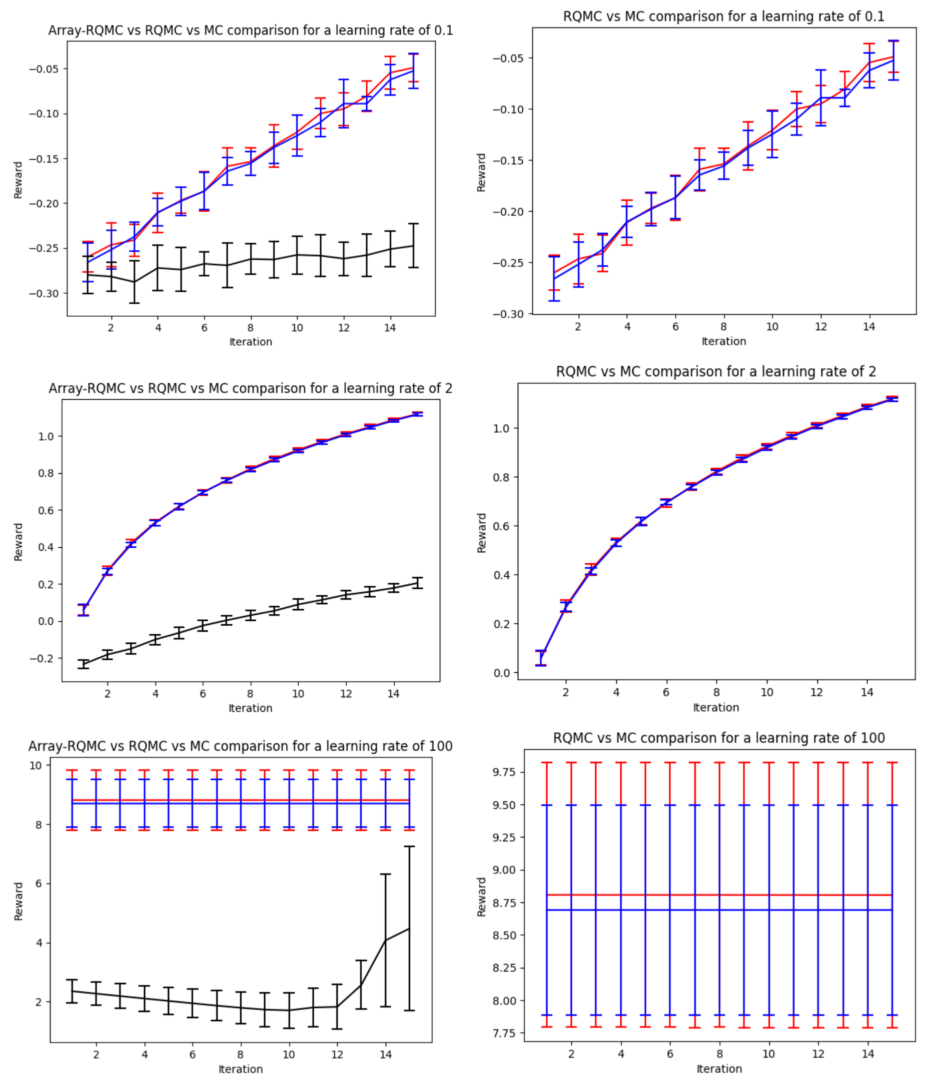


Figure 29: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

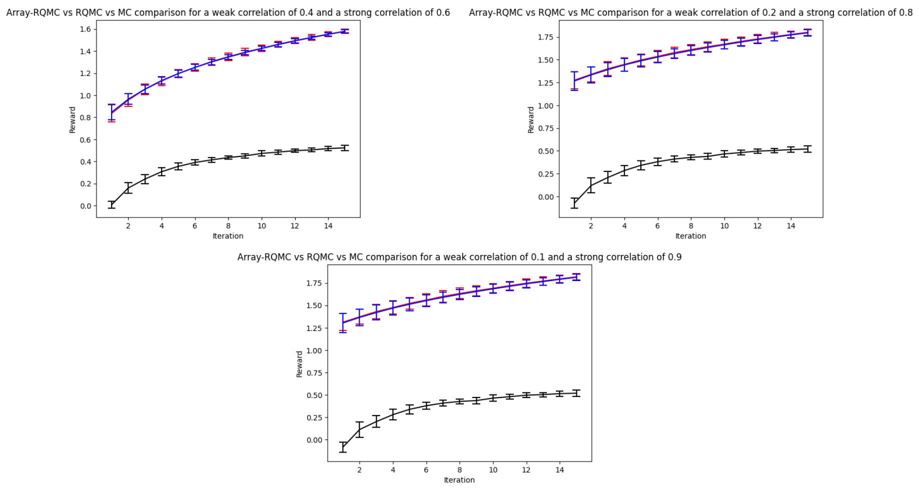


Figure 30: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

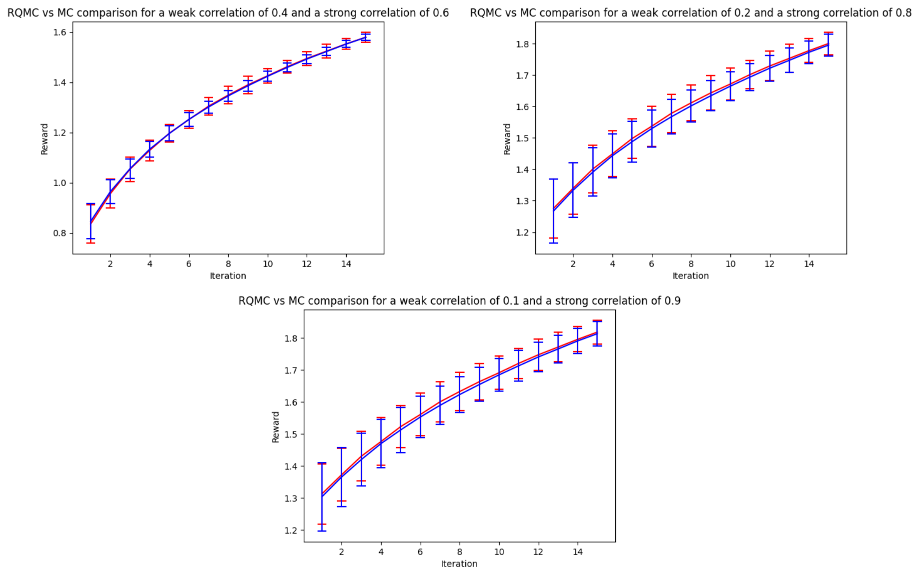


Figure 31: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

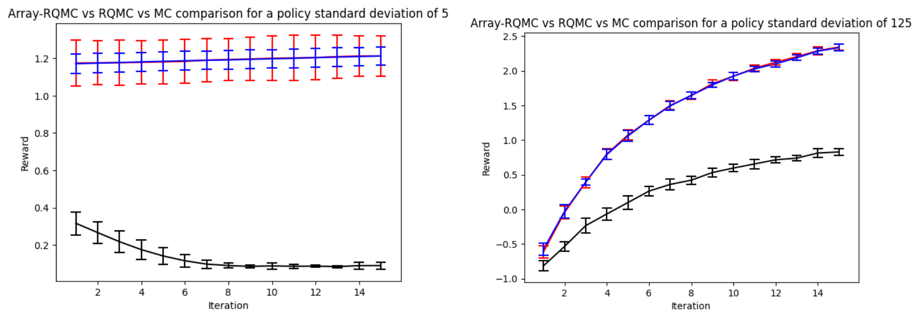


Figure 32: For these plots MC is graphed in red, RQMC is graphed in blue, and FIRST TEN with RQMC points not permuted is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

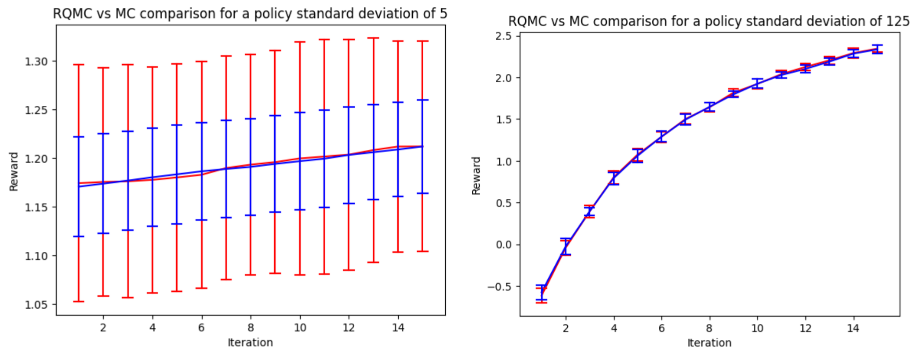


Figure 33: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

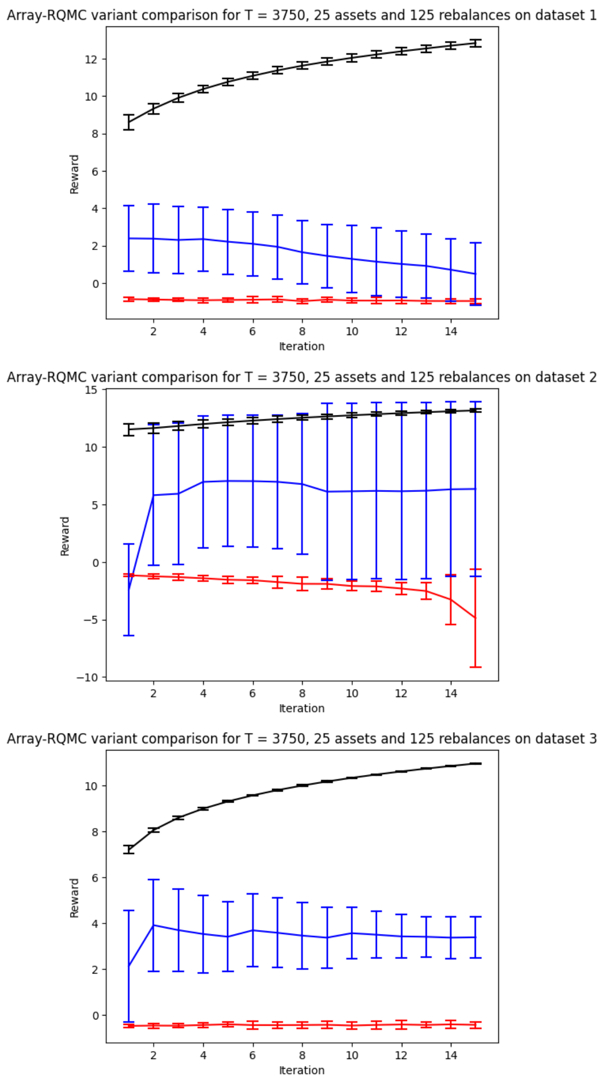
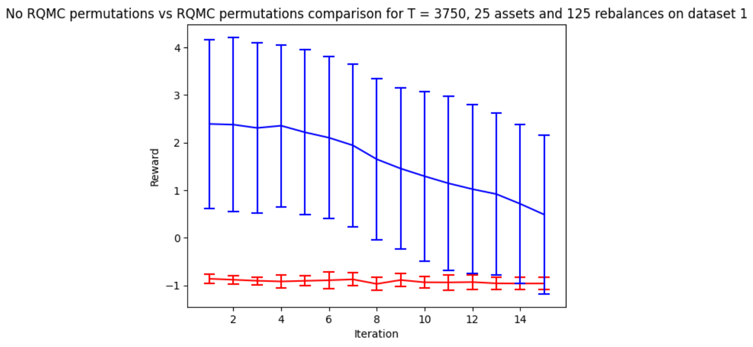
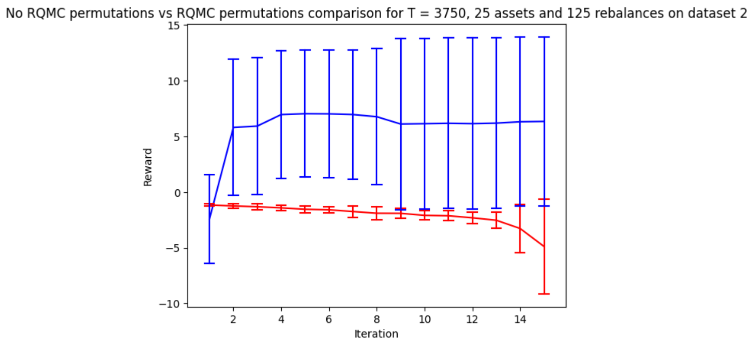


Figure 34: For these plots the FIRST TEN variant with no permutations of the RQMC points is graphed in red, the version with RQMC points permuted is graphed in blue, and the BEST TEN algorithm is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.





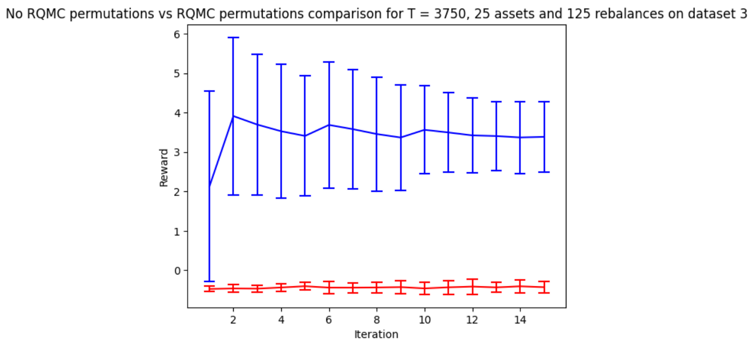
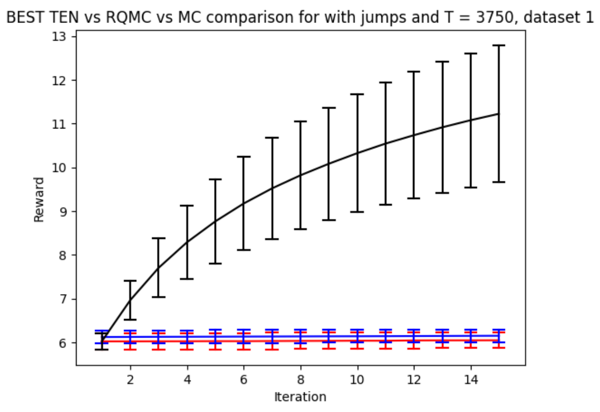
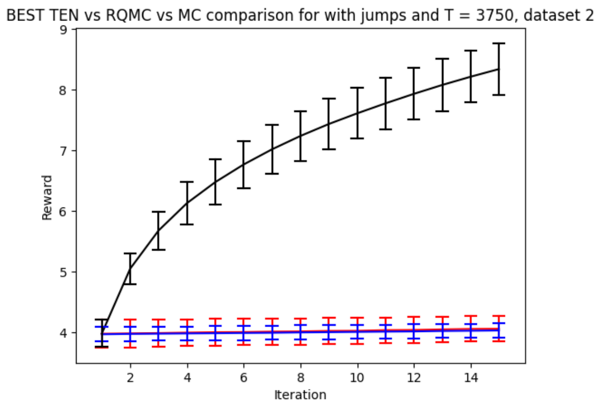


Figure 35: For these plots the FIRST TEN variant with no permutations of the RQMC points is graphed in red and the version with RQMC points permuted is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.





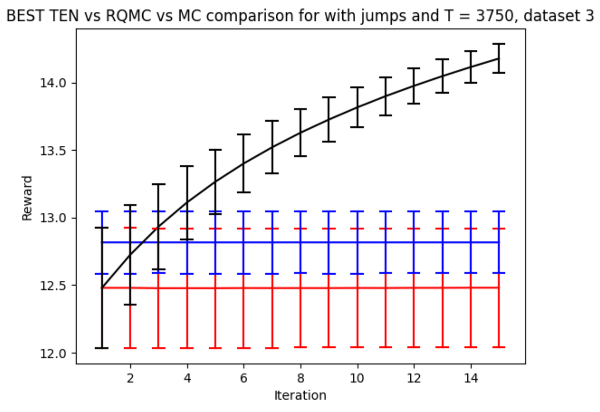
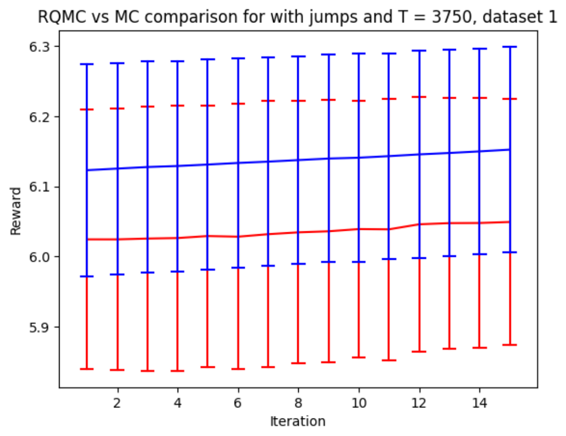
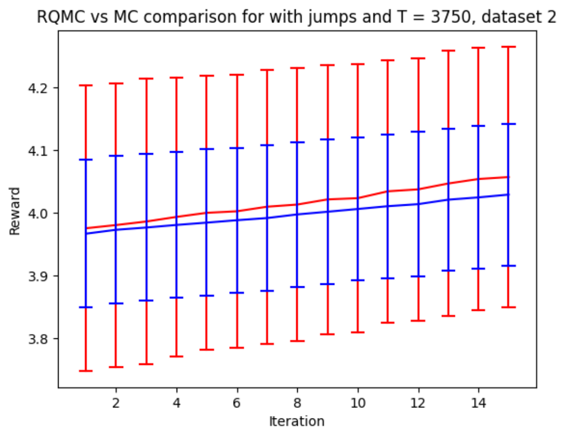


Figure 36: For these plots MC is graphed in red, RQMC is graphed in blue, and BEST TEN is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.





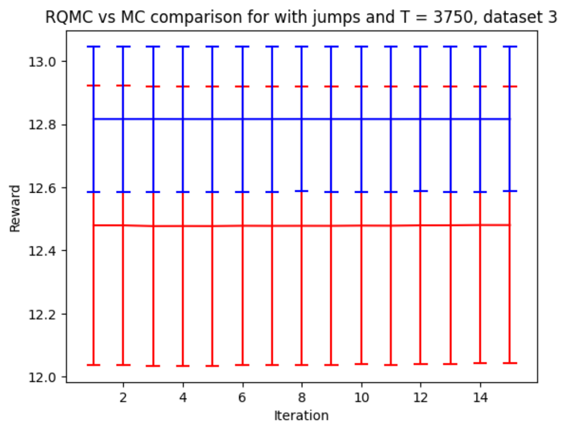
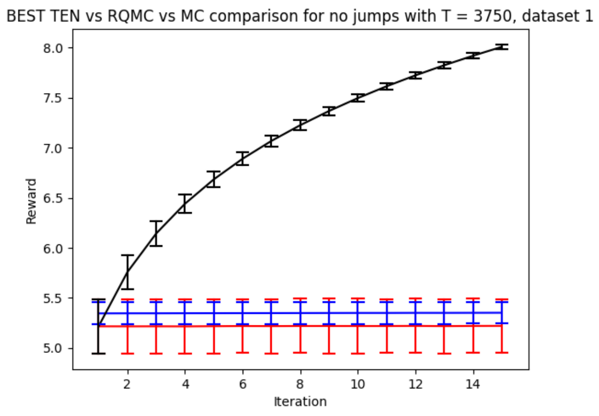
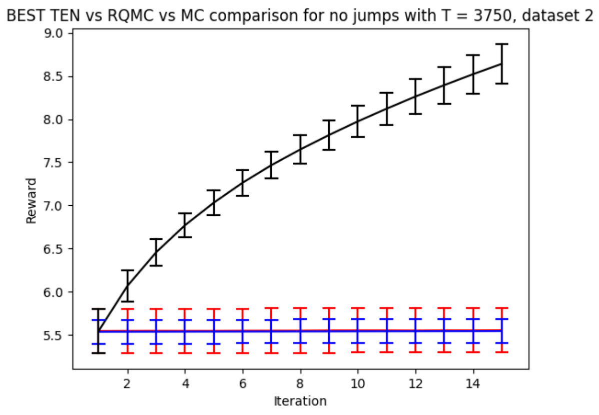


Figure 37: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.





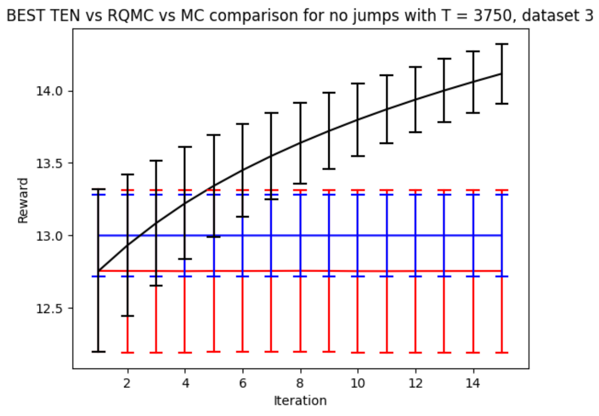


Figure 38: For these plots MC is graphed in red, RQMC is graphed in blue, and BEST TEN is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

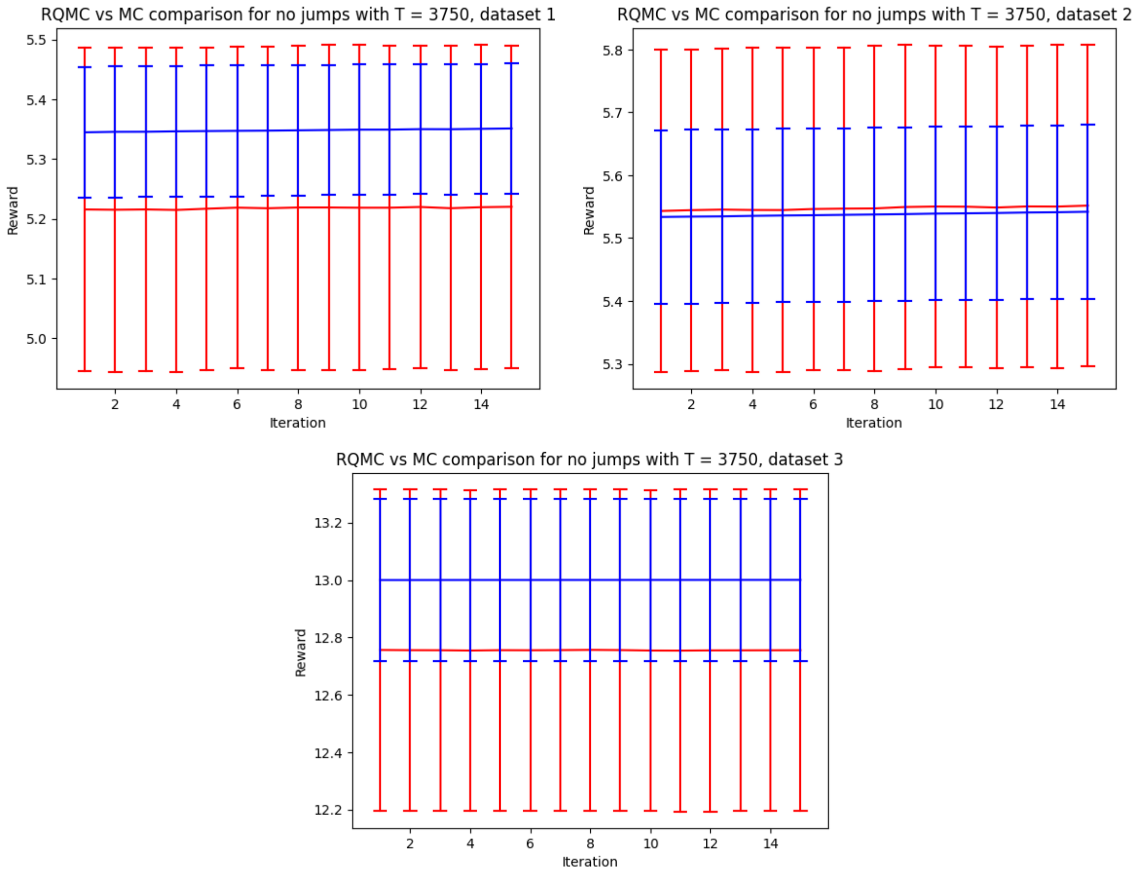
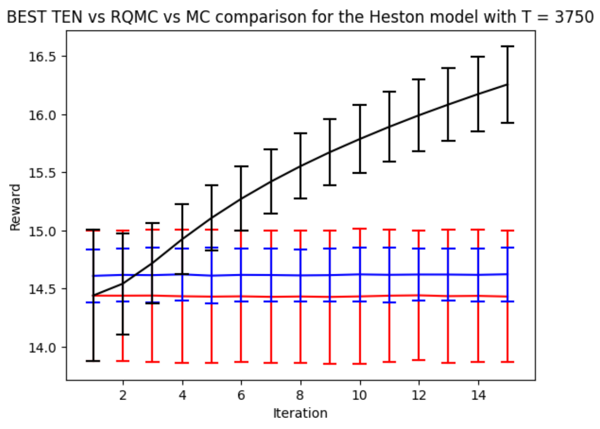


Figure 39: For these plots MC is graphed in red and RQMC is graphed in blue. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.



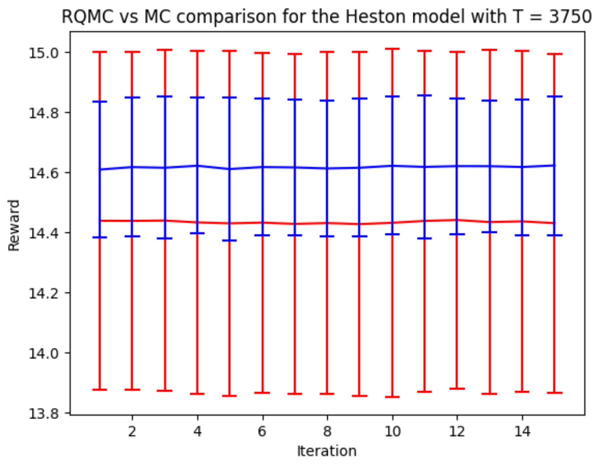


Figure 40: For these plots MC is graphed in red, RQMC is graphed in blue, and BEST TEN is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

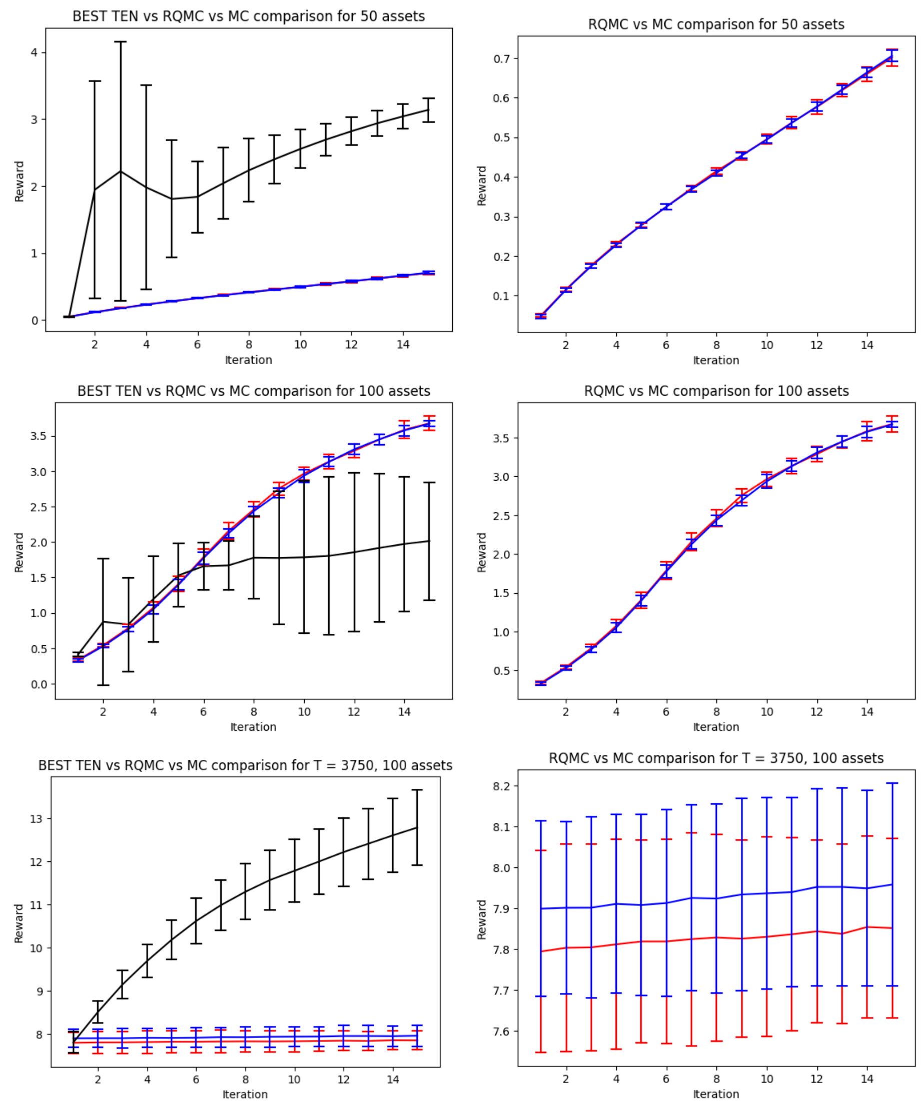
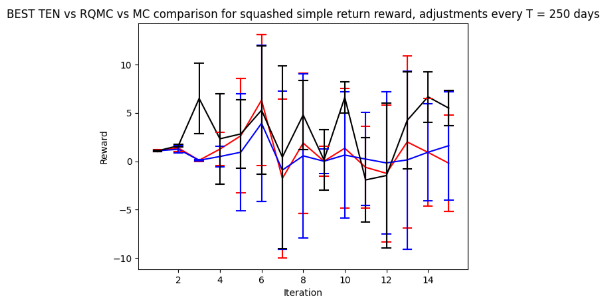
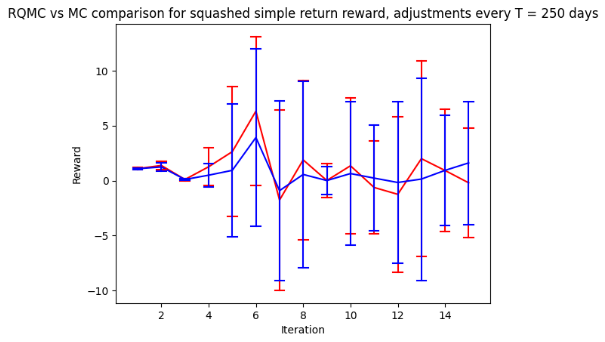
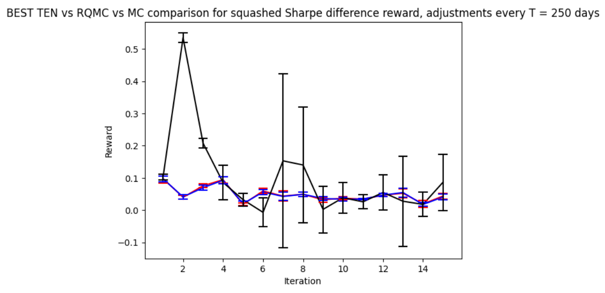


Figure 41: For these plots MC is graphed in red, RQMC is graphed in blue, and BEST TEN is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.







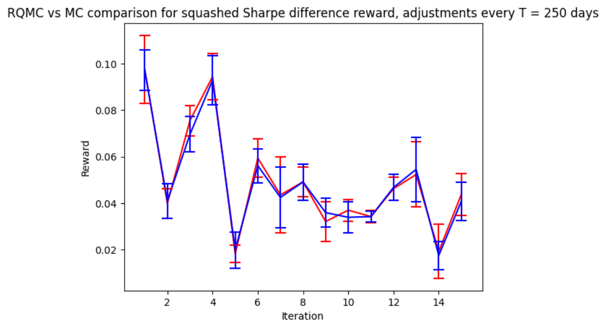
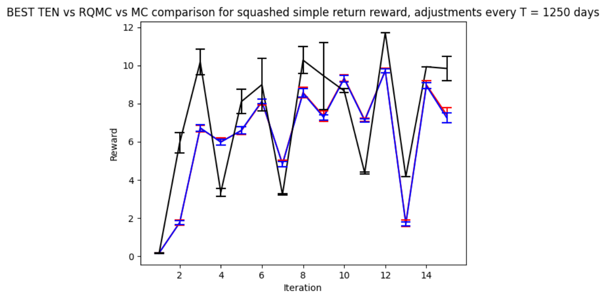
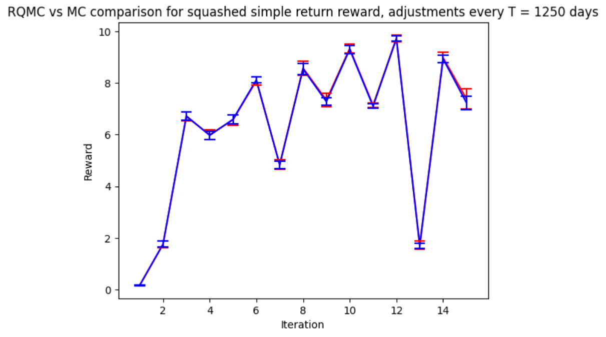
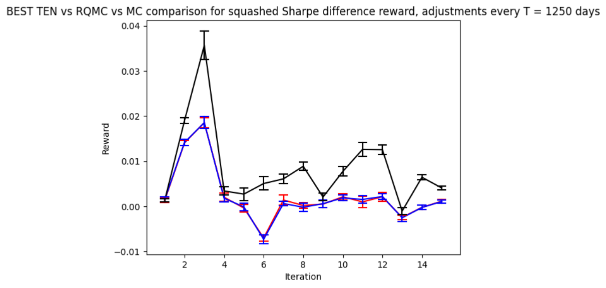


Figure 42: For these plots MC is graphed in red, RQMC is graphed in blue, and BEST TEN is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.







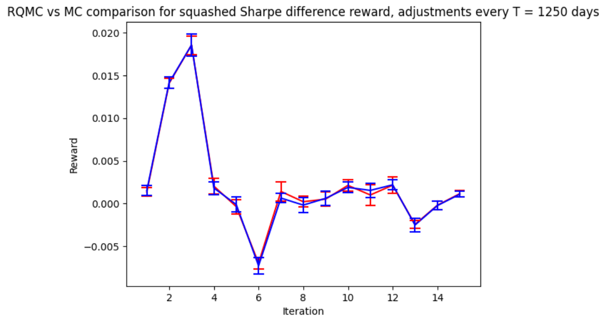


Figure 43: For these plots MC is graphed in red, RQMC is graphed in blue, and BEST TEN is graphed in black. The trend lines show the average total reward over the time period and the uncertainty bars show the standard deviation.

We make the following observations about the results from the graphs:

1. From Figures 21, 22, 23 and 24 we see that the Array-RQMC FIRST TEN algorithm with only the states permuted tends to outperform both RQMC and MC for the 2-asset and 5-asset cases, but it significantly underperforms RQMC and MC in the 10-asset and 25-asset cases as shown in Figures 25, 26, 27 and 28. This may be a consequence of the focus on the first few coordinates in the market, which may result in it seeing strong results in markets with fewer assets but it missing opportunities in markets with more assets. In addition, the ordering of assets in the market is arbitrary but the first few assets which are the focus of the sort with this Array-RQMC method are negative performers for markets with at least five risky assets, which can be shorted for some profit but may result in opportunities for high returns from investing in the assets with high Sharpe ratios being missed.

2. From Figure 34 we see that BEST TEN overwhelmingly outperforms both variants of the FIRST TEN algorithm in the 25-asset case where a significant dimensional reduction is required.

3. We see that RQMC and MC have negligible performance differences for most experiments, with the main difference being that MC showed a higher standrad deviation in its total rewards in many cases than RQMC, as shown in Figure 37.

4. We see in some cases like in Figure 21 that when applying Array-RQMC we accumulate a very high reward that decreases with more iterations, which is undesirable as further iterations should continue improving the policy.

5. From Figure 29 we see that with a high learning rate of 100 that the numerical methods do not change significantly after the first iteration. However for the lower learning rate of 0.1 we see a linear trend which does not show a diminishing effect with more iterations; this indicates that the gradient ascent algorithm has not effectively reached a high point of the reward function after many iterations with such a small step size.

6. From Figure 37 we see that with long time horizons both MC and RQMC show a significant increase in total reward for the first iteration and very small increases for the following itera-

tions; for longer time periods the returns over the rebalancing periods will be higher, so this will be similar to having a larger step size with high values of the learning rate.

7. From Figures 32 and 33 we see that a higher standard deviation allows for higher returns under otherwise identical conditions, and that the small choice of 5 for the standard deviation resulted in a linear trend with little learning similar to what we see with a learning rate that is too small.

8. From Figures 34 and 36 for example, we see that the choice of dataset can have a profound impact on the performance of the numerical methods under otherwise identical conditions. This especially affects the performance of BEST TEN, since it exploits the most relevant assets in the market which change from database to database.

9. From Figures 31, 36, 37, 38, 39 and 40 we see that the choice of correlations, the presence of jumps and the use of a Heston model do not have a significant impact on the performance of the numerical methods. The Heston model's fluctuation in the variance appears to cause the total rewards to fluctuate slightly more than in similar cases with the GBM.

10. We see in many cases that the BEST TEN method shows much higher uncertainty than MC and RQMC when operating on one dataset for every iteration, for example in the 50-asset case of Figure 41.

11. From Figure 41 we see that for the higher-dimensional cases with a time horizon of T = 250 days, the BEST TEN algorithm significantly outperforms both MC and RQMC in the 50-asset case but not in the 100-asset case. However for T = 3750 the 100-asset case outperforms MC and RQMC similarly to how it does in the long-term 10-asset cases.

12. From Figures 42 and 43 we see that for the cases where we generate new realizations of the price processes of the assets that make up the market, when using a squashed modified simple return the BEST TEN algorithm tends to fluctuate and does not convincingly outperform MC or RQMC, both when each iteration occurs over 250 and 1250 days. On the other hand, when the reward in the environment is the arctangent of the difference in the Sharpe ratio, we see that BEST TEN tends to show a noticeable improvement over the other two methods.

Remark 4.2.1: We note that the reward function is usually positively correlated with the overall portfolio return over the time period but this is not always the case. For example, for the 100-asset case with a period of 250 trading days we see positive total rewards but very negative returns. For the majority of cases the sum of the rewards is usually a good indicator of performance but when running the code both the sum of rewards and portfolio returns should be taken into account.

5 Conclusions

In summary, for the portfolio optimization problem in the financial market model simulation we investigated, we have found that RQMC decisively improves the policy evaluation performance compared to MC and while RQMC and MC have comparable policy iteration performance, a strategic reordering function choice with Array-RQMC can significantly improve its performance compared to the other numerical methods. Furthermore, we have found that these results apply under both long-term and short-term periods and different asset pricing models, among other market conditions. Finally, we have found that these results scale well to high-dimensional datasets and where the policies are iterated over evolving datasets, which is promising for applications in real-life datasets with many assets.

6 Future Work

Several extensions of this work may be useful to further test the usefulness of these numerical methods in the context of real-life financial problems:

1. The dimension of the state space may be further increased to investigate the practicality of the numerical methods when working with massive datasets. The notebooks provided with the paper support a 1000-asset implementation of the environment, but there is not the capacity to run it on the free version of Google Colab. With more advanced computing resources this may be investigated using MC policy evaluation. The maximum dimension that would be supported by these methods would be 21201, the maximum dimension supported by the scipy.stats.qmc module [14].
2. The state of the art soft actor-critic method should be explored in addition to the vanilla policy gradient method explored in this paper. While simple, the REINFORCE algorithm is noisy [35], so it is likely the performance may be improved using another RL method.
3. There should be further exploration of strategic choices of reordering functions for Array-RQMC to evaluate how this may further improve its performance. In particular, the BEST TEN strategy tends to result in undiversified portfolios because it exploits the assets that are currently the highest performers, so it would be desirable to derive a sorting function that can increase the robustness of the portfolio.
4. The methods should be tested on real-world data in addition to synthetic data.
5. Other continuous control problems in finance may be investigated where policy gradients may be applied, such as in option hedging [24].

References

[1] Mohamed, S., Rosca, M., Figurnov, M., & Mnih, A. (2020). Monte Carlo Gradient Estimation in Machine Learning. J. Mach. Learn. Res., 21(132), 1-62.

[2] L'Ecuyer, P., & Lemieux, C. (2002). Recent advances in randomized quasi-Monte Carlo methods. Modeling uncertainty, 419-474.

[3] Arnold, S. M., L'Ecuyer, P., Chen, L., Chen, Y. F., & Sha, F. (2022). Policy Learning and Evaluation with Randomized Quasi-Monte Carlo. arXiv preprint arXiv:2202.07808.

[4] Buchholz, A., Wenzel, F., & Mandt, S. (2018, July). Quasi-monte carlo variational inference. In International Conference on Machine Learning (pp. 668-677). PMLR.

[5] Owen, A. B. (2016, January). Quasi-Monte Carlo. MCMSki 5. Lenzerheide; Switzerland.

[6] Dodge, Y. (2008). The concise encyclopedia of statistics. Springer.

[7] L'Ecuyer, P. (2016, August). Randomized quasi-Monte Carlo: An introduction for practitioners. In International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing (pp. 29-52). Springer, Cham.

[8] Lemieux, C. (2018). Negative dependence, scrambled nets, and variance bounds. Mathematics of Operations Research, 43(1), 228–251.
https://doi.org/10.1287/moor.2017.0861

[9] Owen, A. B. (1997a). Monte Carlo variance of Scrambled Net Quadrature. SIAM Journal on Numerical Analysis, 34(5), 1884–1910. https://doi.org/10.1137/s0036142994277468

[10] Harase, S. (2019). Comparison of sobol' sequences in financial applications. Monte Carlo Methods and Applications, 25(1), 61–74. https://doi.org/10.1515/mcma-2019-2029

[11] Sun, X., Croke, B., Roberts, S., & Jakeman, A. (2021). Investigation of determinism-related issues in the Sobol' low-discrepancy sequence for producing sound global sensitivity analysis indices. ANZIAM Journal, 62. https://doi.org/10.21914/anziamj.v62.16094

[12] Lin, J., & Wang, X. (2008). New brownian bridge construction in quasi-monte carlo methods for computational finance. Journal of Complexity, 24(2), 109–133. https://doi.org/10.1016/j.jco.2007 .06.001

[13] Owen, A. B. (1995). Randomly permuted (T,M,s)-nets and (T, s)-sequences. Monte Carlo

and Quasi-Monte Carlo Methods in Scientific Computing, 299–317. https://doi.org/10.1007/978-1-4612-2552-2_19

[14] Quasi-monte Carlo Submodule (scipy.stats.qmc)#. Quasi-Monte Carlo submodule (scipy.stats.qmc) - SciPy v1.13.0 Manual. (n.d.). https://docs.scipy.org/doc/scipy/reference/stats.qmc.html

[15] L'Ecuyer, P., Lécot, C., & Tuffin, B. (2007). Randomized Quasi-Monte Carlo simulation of Markov chains with an ordered state space. Monte Carlo and Quasi-Monte Carlo Methods 2004, 331–342. https://doi.org/10.1007/3-540-31186-6_19

[16] Puchhammer, F., Ben Abdellah, A., & L'Ecuyer, P. (2021). Variance reduction with array-RQMC for tau-leaping simulation of stochastic biological and Chemical Reaction Networks. Bulletin of Mathematical Biology, 83(8). https://doi.org/10.1007/s11538-021-00920-5

[17] Abdellah, A. B., L'Ecuyer, P., & Puchhammer, F. (2019). Array-RQMC for option pricing under Stochastic Volatility models. 2019 Winter Simulation Conference (WSC). https://doi.org/10.1109/wsc40007.2019.9004819

[18] L'Ecuyer, P., Lécot, C., & L'Archevêque-Gaudet, A. (2009). On array-RQMC for Markov chains: Mapping alternatives and convergence rates. Monte Carlo and Quasi-Monte Carlo Methods 2008, 485–500. https://doi.org/10.1007/978-3-642-04107-5_31

[19] L'Ecuyer, P., Munger, D., Lécot, C., & Tuffin, B. (2018). Sorting methods and convergence rates for array-RQMC: Some empirical comparisons. Mathematics and Computers in Simulation, 143, 191–201. https://doi.org/10.1016/j.matcom.2016.07.010

[20] Bodie, Z., Kane, A., & Marcus, A. J. (2024). Investments. McGraw-Hill Education.

[21] Burgess, N. (2022, March). Correlated Monte Carlo Simulation using Cholesky Decomposition. SSRN. Oxford; England. Retrieved May 2, 2024, from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4066115.

[22] Healy, P. J. (2014, May). Lecture 11: An introduction to the multivariate normal distribution. Paul J Healy. https://docslib.org/doc/8390514/lecture-11-an-introduction-to-the-multivariate-normal-distribution

[23] Shehzad, H. T., Anwar, M. A., & Razzaq, M. (2023, Feb. 15). A Comparative Predicting Stock Prices Using Heston and Geometric Brownian Motion Models. https://doi.org/10.48550/arXiv.

2302.07796

[24] Rouwendaal, R. (2021). Option Hedging Under the Heston Model Using Deep Reinforcement Learning (thesis).

[25] Fornari, F., & Stracca, L. (2012). What does a financial shock do? First International Evidence. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.2224062

[26] Kou, S. G. (2002). A jump diffusion model for option pricing. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.242367

[27] Björk, T. (2020). Arbitrage theory in continuous time. Oxford University Press.

[28] Moody, J., Saffell, M., Liao, Y., & Wu, L. (1998). Reinforcement learning for trading systems and portfolios: Immediate vs future rewards. Decision Technologies for Computational Finance, 129–140. https://doi.org/10.1007/978-1-4615-5625-1_10

[29] Hambly, B., Xu, R., & Yang, H. (2023). Recent advances in reinforcement learning in Finance. Mathematical Finance, 33(3), 437–503. https://doi.org/10.1111/mafi.12382

[30] Grooten, B., Wemmenhove, J., Poot, M., & Portegies, J. (2022, March 22). Is Vanilla Policy Gradient Overlooked? Analyzing Deep Reinforcement Learning for Hanabi. https://doi.org/10.48550/arXiv.2203.11656

[31] Ratner, B. (2009). The correlation coefficient: Its values range between +1/-1, or do they? Journal of Targeting, Measurement and Analysis for Marketing, 17(2), 139–142. https://doi.org/10.1057/jt.2009.5

[32] Stone, P., & Hausknecht, M. (2016). Deep Reinforcement Learning in Parameterized Action Space. arXiv. https://doi.org/10.48550/arXiv.1511.04143

[33] Horrace, W. C. (2005). Some results on the multivariate truncated normal distribution. Journal of Multivariate Analysis, 94(1), 209–221. https://doi.org/10.1016/j.jmva.2004.10.007

[34] Gymnasium documentation. Basic Usage - Gymnasium Documentation. (2023). https://gymnasium.farama.org/content/basic_usage/

[35] Wu, C. (2023). Policy gradient. 6.7920: Reinforcement Learning: Foundations and Methods.

[36] Hu, H., Ghossoub, M., Schied, A., & Charpentier, A. (2023, August 4). Multiarmed Bandits Problem Under the Mean-Variance Setting. https://doi.org/10.48550/arXiv.2212.09192