Simulation Project

Cui Gehao, Du Haoqian, Han Yufeng, Lie Bob Jesse, Yuan Huan

July 15, 2024

Abstract

The project explores two distinct applications of Stochastic Approximation (SA) algorithms in optimization tasks. In **Project 1**, the focus was on optimizing investment strategies for an investor with capital distributed among three companies. The SA algorithm determined an allocation strategy yielding a maximum Sharpe ratio, suggesting approximately 29.17% of capital to company 1, 19.16% to company 2, and 51.67% to company 3. In **Project 2**, the objective was to minimize waiting times in a service station by optimizing the mean service time θ using SA. Results demonstrated convergence to an optimal θ value of approximately 3.4698, with a standard deviation of 0.0040. Additionally, an algorithm was developed to concurrently update θ and estimate q_θ within a fixed computational budget, showcasing efficient computational resource management.

Key words: Stochastic Approximation; Investment Optimization; Queuing System Optimization

Project 1

1 Introduction

The first project focuses on an investor with a total capital of 1, aiming to allocate this capital across three companies to maximize the risk-adjusted performance of the investment. The market value and returns of each company are influenced by independent uniformly distributed returns and various market factors, including common economic conditions and company-specific risks. The objective is to determine the optimal allocation vector that maximizes the Sharpe ratio, reflecting the trade-off between expected profit and risk. This is achieved by formulating and solving an optimization problem using an SA algorithm.

2 Description of Project 1

In this project, we aim to determine the optimal investment strategy for an investor with a total capital of 1 unit. The capital is to be distributed among n companies over a time period of t units. The market value of company i at time t is given by X_i. Let $x_i \in \mathbb{R}$, $i = 1, \ldots, n$ be given thresholds. Company i will not be able to generate any profit if $X_i < x_i$. If $X_i \geq x_i$, the return on the investment is given by Y_i. Investing a fraction p_i of the capital yields an expected return of

$$p_i \mathbb{E}[Y_i \mathbf{1}_{X_i \geq x_i}] \tag{1}$$

for company i, where

$$\sum_{i=1}^{n} p_i = 1 \quad \text{and} \quad 0 \leq p_i \leq 1. \tag{2}$$

Assume that $n = 3$ and Y_i are independent and uniformly distributed on $[0, X_i]$. We let

$$X_i = \frac{\rho V + \sqrt{1-\rho^2}\eta_i}{\max(W, 1)}, \quad 1 \leq i \leq n, \tag{3}$$

with η_i normally distributed with mean 0 and variance i modeling the company's idiosyncratic risk, V standard normally distributed modeling the common factor that affects the economy, and W exponentially distributed with rate 1/0.3 modeling common market shocks. The variables V, W, and η_i's are all independent. The weight factor is set to $\rho = 0.6$, and the thresholds are $x_1 = 2$, $x_2 = 3$, and $x_3 = 1$.

The investor seeks to find the optimal investment strategy by maximizing the risk-adjusted performance of the investment, also known as the Sharpe ratio, which leads to the objective function:

$$\max_{p_1, p_2, p_3 \geq 0, \sum p_i = 1} \mathbb{E}\left[\frac{\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}}{\text{std}\left(\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}\right)}\right], \tag{4}$$

where $\text{std}(X)$ denotes the standard deviation of the random variable X. To simplify the problem and solution process, we converted the objective function to:

$$\min_{p_1, p_2, p_3 \geq 0, \sum p_i = 1} -\mathbb{E}\left[\frac{\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}}{\text{std}\left(\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}\right)}\right], \tag{5}$$

In order to visualize the objective function for the optimization, we drew the 3D plot of the function (Figure 1 and Figure 2).

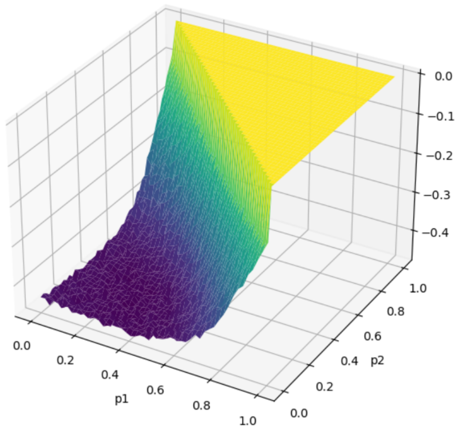

Figure 1: J function without random seed

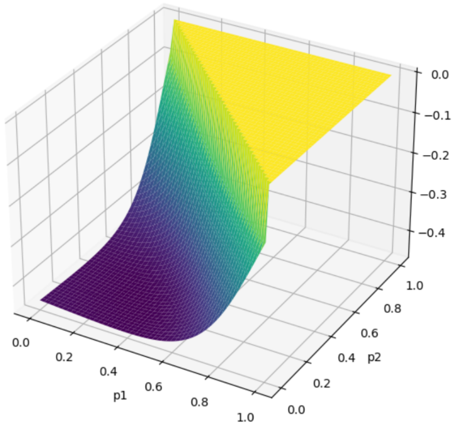

Figure 2: J function with random seed = 20

3 Motivation for the Project

In real-life scenarios, financial data is often noisy and uncertain. Therefore, it is crucial to use optimization methods that can handle such noise effectively. Stochastic Approximation (SA) algorithms are particularly suited for this purpose as they are designed to deal with noisy observations, making them ideal for financial applications.

4 Optimization Approach

To address the optimization problem, we employed a Stochastic Approximation (SA) algorithm, specifically used the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm to calculate the gradient. Here's a detailed explanation of the approach.

First of all, this optimization problem is well-posed because it has a clear objective function to maximize (the Sharpe ratio), subject to the constraints that the sum of the investment fractions p_i equals 1 and each p_i is between 0 and 1. The constraints ensure that the solution space is bounded and feasible.

What's more, the vector field in this context is the gradient of the objective function with respect to the investment fractions p_i. The coercivity of this vector field is ensured by the constraints and the nature of the Sharpe ratio, which promotes exploration within the feasible region.

Since the problem is well-posed, and the gradient is coercive for the problem. Therefore, the update rule: $\theta_{n+1} = \theta_n + \epsilon Y_n$ converges to a solution. This convergence is guaranteed by the Stochastic Approximation Theory, which states that under conditions of unbiased gradient estimates and a properly decreasing gain sequence, the iterative updates will converge to the true solution of the optimization problem. Therefore, we employ the Stochastic Approximation (SA) algorithm to conduct the optimization.

The Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm iteratively updates the investment fractions p_i using random perturbations and a decreasing gain size. The gain size decreases over iterations, which ensures convergence.

The whole algorithm can be described as follows:

1. Initialize p_1 and p_2 to 0, and compute $p_3 = 1 - p_1 - p_2$.
2. For each iteration i:

- Generate random perturbations r_1 and r_2.
  $P(r_1 = -1) = P(r_1 = 1) = \frac{1}{2}$, same as r_2.

- Calculate the step size as $\theta = \frac{1}{(i+10000)^{0.3}}$.

  Proving process: Because $\sum_{i=n}^{\infty} \epsilon = \infty$, $\sum_{i=n}^{\infty} \epsilon\theta^2 < \infty$ and $\sum_{i=n}^{\infty}(\frac{\epsilon^2}{\theta^2}) < \infty$, so $\epsilon = \frac{1}{\alpha}$ which α is linear polynomial about i and therefore $\theta = \frac{1}{k^c}$ which k is linear polynomial about i. Also, we have to ensure that $\frac{1}{i+100} < \frac{1}{\alpha} < \frac{1}{i+1}$ and $\frac{1}{(i+10^5)^c} < \frac{1}{k^c} < \frac{1}{(i+10)^c}$.

  Since $\frac{1}{i+100} < \frac{1}{\alpha} < \frac{1}{i+1}$, $\sum_{i=n}^{\infty} \frac{1}{i+100} < \Sigma_{i=n}^{\infty} \frac{1}{\alpha} = \infty$ and $\sum_{i=n}^{\infty} \epsilon\theta^2 < \sum_{i=n}^{\infty} \frac{1}{(i+1)^{2c+1}} < \infty$, so $2c + 1 > 1$, here's the formula 1. Besides that, we also can find that $\Sigma_{i=n}^{\infty} \frac{\epsilon^2}{\theta^2} < \Sigma_{i=n}^{\infty} \frac{(i+10^5)^{2c}}{(i+1)^2}$, we want to make it convergence. In order to do that, we assume $f(i) = \frac{(i+10^5)^{2c}}{(i+1)^2}$, when i $\to \infty$, $f(i)$ turns to $i^{\frac{1}{2-2c}}$. And because $f(i)$ should be convergence, $2 - 2c > 1$, which is the formula 2. Combine both formulas together, and we will find a range of c that $0 < c < 0.5$, and we choose c = 0.3 as our c in the θ formula.

  By referring to the "Taylor Expansion" and "Newton-Raphson Method" mentioned in the "Optimization and Learning via Stochastic Gradient Search", we determined the y1 and y2 with the formula list above.

- Estimate the gradients y_1 and y_2 using the objective function J:

$$y_1 = \frac{J(p_1 + \theta r_1, p_2 + \theta r_2) - J(p_1 - \theta r_1, p_2 - \theta r_2)}{2\theta r_1}, \tag{6}$$

$$y_2 = \frac{J(p_1 + \theta r_1, p_2 + \theta r_2) - J(p_1 - \theta r_1, p_2 - \theta r_2)}{2\theta r_2}. \tag{7}$$

- Update p_1 and p_2:

$$p_1 \leftarrow \max(p_1 - \text{step\_size} \cdot y_1, 0), \tag{8}$$

$$p_2 \leftarrow \max(p_2 - \text{step\_size} \cdot y_2, 0). \tag{9}$$

According to the requirements of the problem, $0 \leq p_1 + p_2 \leq 1$, and both p_1 and p_2 are greater than or equal to 0. When the generated points exceed the range, we need to project them back into the constraints. If the newly generated p_1 or p_2 is greater than 1, then assign p_1 and p_2 the value of 1. If p_1 or p_2 is less than 0, then assign p_1 or p_2 the value of 0. If $p_1 + p_2$ falls within the area where $x + y > 1$ and is bounded by $x = 1$ and $y = 1$ (denoted as area M), project the points onto the line $y + x = 1$.

For example, Figure 3 displays the projection process.

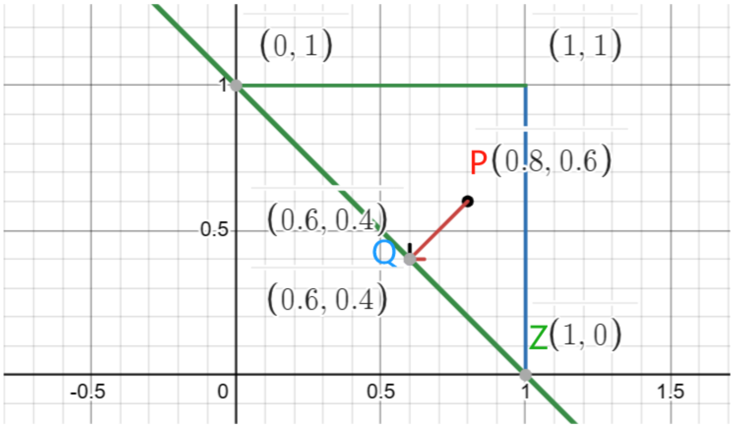

Figure 3: The Projection Process

If a point falls within area M, we will calculate its projection. Suppose the point on the line is Q, and arbitrarily take a point Z on the line $y + x = 1$ (here for the sake of calculation convenience, we take $(1, 0)$) and perform the following operation:

$$\overrightarrow{QP} = (p_1 - x_0, p_2 - y_0), \overrightarrow{QZ} = (1 - x_0, -y_0) \tag{10}$$

$$\because \overrightarrow{QP} \cdot \overrightarrow{QZ} = 0 \tag{11}$$

$$\Leftrightarrow \begin{cases} (p_1 - x_0)(1 - x_0) + (p_2 - y_0)(-y_0) = 0 \\ x_0 + y_0 = 1 \end{cases} \tag{12}$$

$$\Leftrightarrow (p_1 - 1 + y_0)y_0 = (p_2 - y_0)y_0 \tag{13}$$

$$\because y_0 \neq 0 \tag{14}$$

$$\therefore \begin{cases} x_0 = \frac{p_1 - p_2 + 1}{2} \\ y_0 = \frac{p_2 - p_1 + 1}{2} \end{cases} \tag{15}$$

- Ensure that the sum of p_1, p_2, and p_3 equals 1 by adjusting p_1 and p_2 if necessary.

The algorithm runs for a specified number of iterations, and the final values of p_1, p_2, and p_3 represent the optimal investment fractions.

5 Experimental Setting

Sample Size: we used a sample size of 10,000 for generating the random variables. This large sample size helps accurately estimate the expectations and standard deviations required for the Sharpe ratio calculation.

Confidence Bounds: to ensure the reliability of our results, we performed multiple experiments and recorded the means of the last 500 iterations for each experiment. This approach provides a measure of stability and confidence in the algorithm's convergence.

5.1 Algorithm Implementation

We implemented the SPSA algorithm with the following settings:

- **Initial Points:** p_1 and p_2 start at 0, with p_3 computed as $1 - p_1 - p_2$.
- **Iteration Count:** The algorithm runs for 3,000 iterations per experiment. The choice of 3,000 iterations is based on the total computational budget and time constraints for obtaining results. Also, 3,000 iterations are found to be sufficient for the algorithm to converge to a near-optimal solution (Figure 4), therefore it becomes a reasonable stopping criterion.

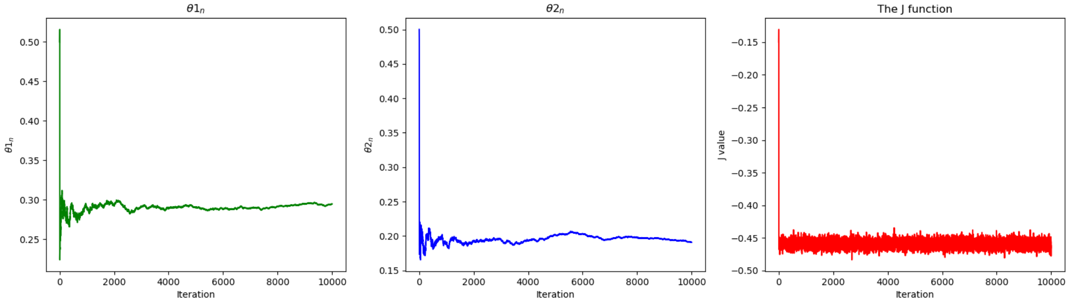

Figure 4: Result with 10, 000 Iterations

- **Step Size:** We tried fixed gain size 0.01 and decreasing gain size $\frac{80}{(i+100)}$ and $\frac{10}{(i+100)}$ to experiment with different strategies for adjusting the gain dynamically.
- **Gradient Calculation:** The gradient is estimated using random perturbations in both p_1 and p_2 directions.

6 Validation and Verification

Figure 4 presents the three-dimensional representation utilized in our prior analysis to elucidate the shape of the objective function. The annotated red circle in the figure indicates the position of the optimization result, thereby substantiating the validity of our experimental outcomes.

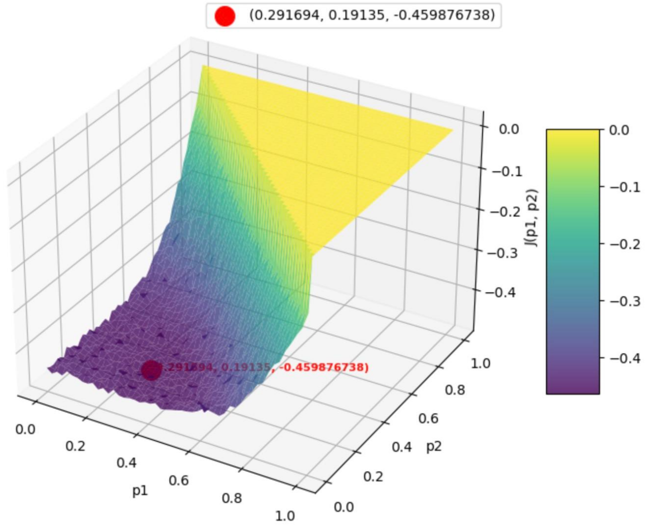

Figure 5: Result in 3D Plot of the Objective Function

7 Results and Analysis

The results are recorded in text files and visualized using plots. Each plot shows the evolution of p_1, p_2, and the objective function J over iterations. The mean values of p_1 and p_2 over the last 500 iterations are used to determine the optimal allocation strategy.

7.1 Gain Size Analysis

The figures show that optimization with a fixed gain size leads to weaker convergence, while a decreasing gain size behaves better and costs less. Specifically, we found that a gain size of $\frac{80}{(i+100)}$ converges slower than $\frac{10}{(i+100)}$. Therefore, the gain size is updated as $\frac{10}{(i+100)}$, leveraging the strengths of the SA algorithm.

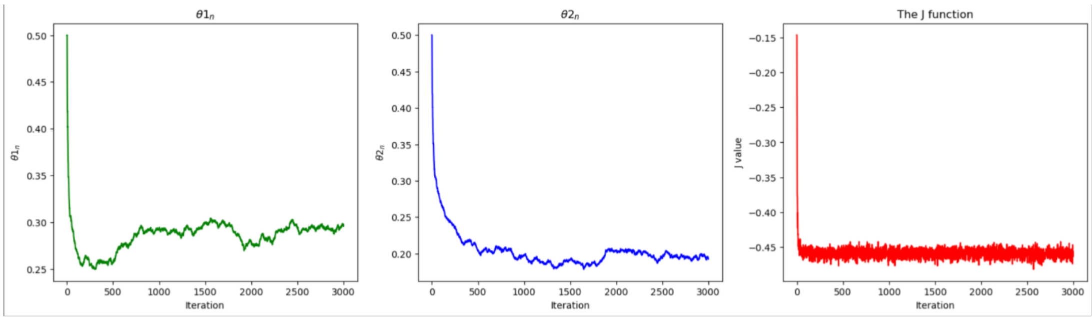

Figure 6: Fixed Gain Size 0.01

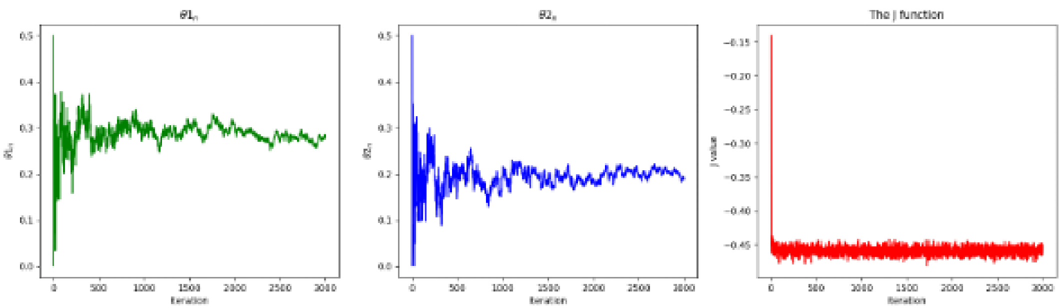

Figure 7: Decreased Gain Size $\frac{80}{(i+100)}$

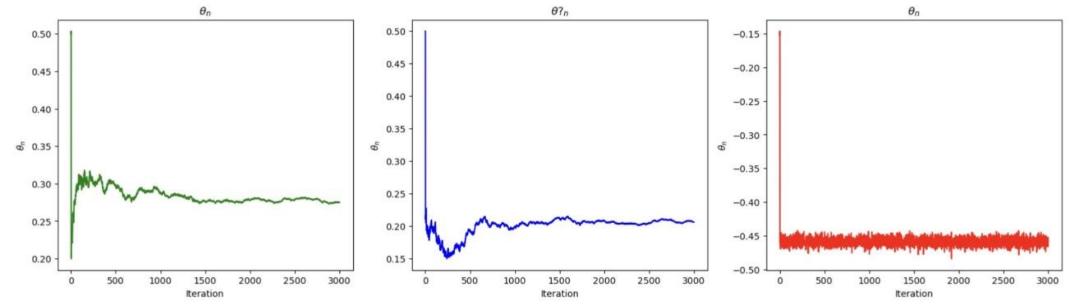

Figure 8: Decreased Gain Size $\frac{10}{(i+100)}$

7.2 Initial Point Analysis

We tried to choose 2 different initial points for our task. However, both can converge to a minimum value, as seen in the figures. Hence, there is no significant difference between the 2 initial points. Therefore, we picked one of them as our initial point which is (0.5, 0.5).

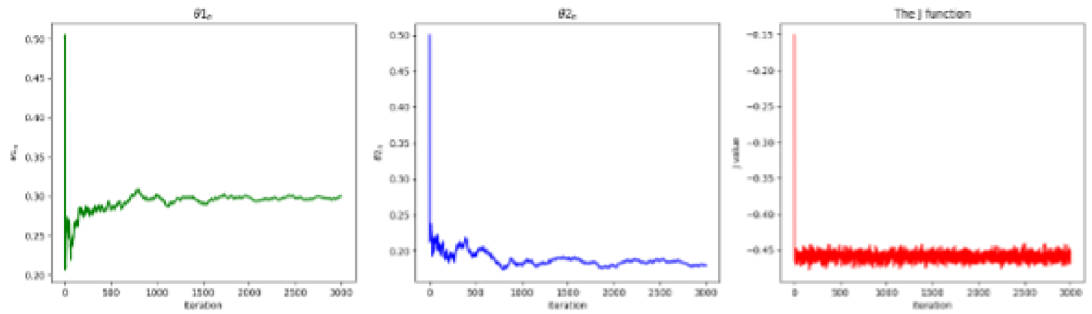

Figure 9: Initial Point (0.5, 0.5)

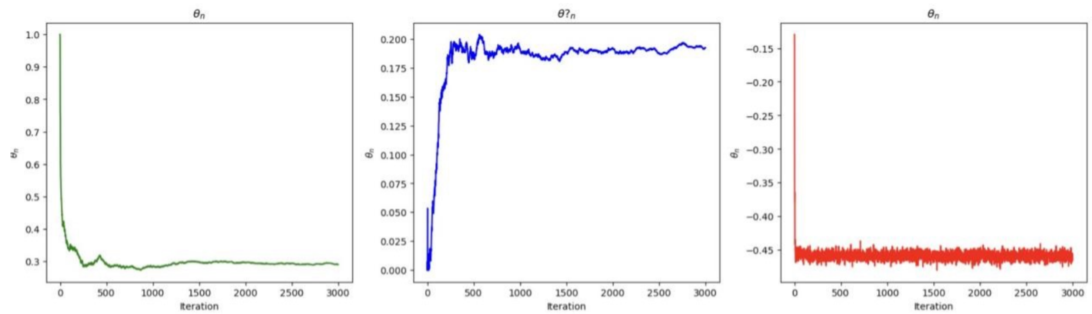

Figure 10: Initial Point (1, 0)

7.3 Output Analysis

Upon initial analysis, it was determined that $n = 216$ was insufficient to achieve normally distributed results for $\theta_n(p_1, p_2)$, as depicted in Figures 11 and 12. Subsequently, a larger sample size of 505 runs was utilized, resulting in approximately normally distributed outcomes, as illustrated in Figures 13 and 14. The estimated means $\mu_{p_1} = 0.291681$ and $\mu_{p_2} = 0.191590$ were derived from this sample size, indicating adequacy for assuming a normal distribution of $\widehat{p}$.

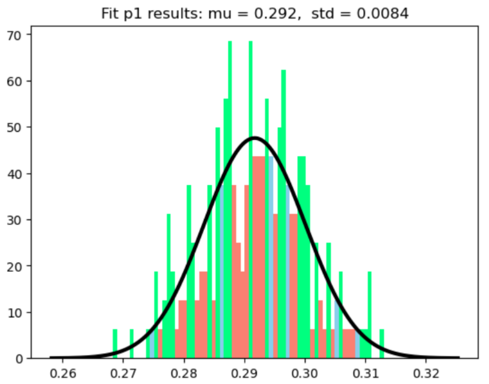

Figure 11: p1 Result Distribution of 216 Runs

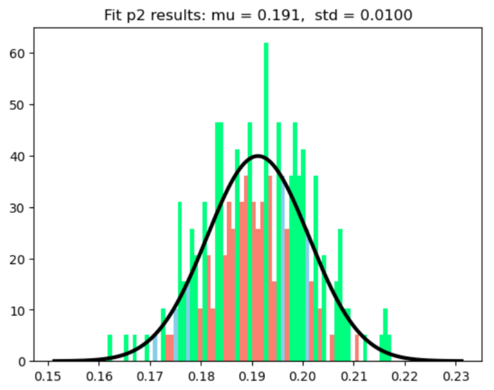

Figure 12: p2 Result Distribution of 216 Runs

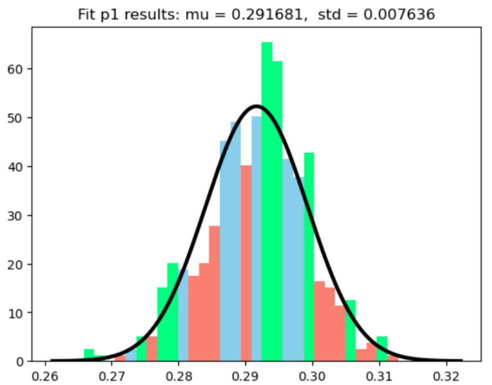

Figure 13: p1 Result Distribution of 505 Runs

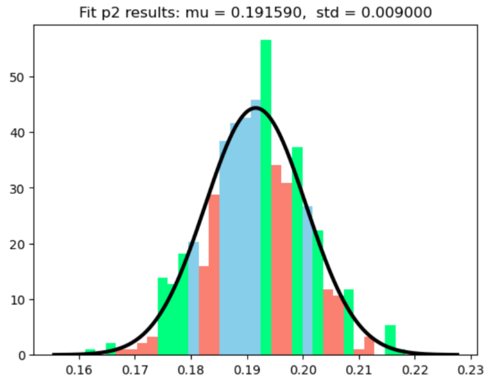

Figure 14: p2 Result Distribution of 505 Runs

For a sample size of 505, the 95% confidence interval for $\widehat{p_1}$ with distribution $N(0.29168, 0.00763)$ is $[0.27671, 0.29101]$. The 95% confidence interval for $\widehat{p_2}$ with distribution $N(0.19159, 0.00900)$ is $[0.17395, 0.20923]$.

8 Conclusion

In conclusion, the optimal investment strategy for the investor with a total capital of 1 unit, distributed among three companies, has been determined using the Stochastic Approximation (SA) algorithm. The results suggest that allocating approximately 29.17% of the capital to company 1, 19.16% to company 2, and the remaining 51.67% to company 3 yields the highest Sharpe ratio, maximizing the risk-adjusted performance of the investment.

The experimental results confirm the effectiveness of the SA algorithm in handling noisy observations and optimizing financial portfolios. The analysis also demonstrates the importance of selecting an appropriate step size and gain size for convergence. The confidence intervals and distribution analysis further validate the robustness of the obtained solutions.

Project 2 Question 1

1 Introduction

This project investigates optimising waiting times in a service station using a GI/GI/1 queueing model with a first-come, first-served (FCFS) discipline. Customers arrive according to a renewal point process with inter-arrival times $\{A_n\}$ that are independent and identically distributed (iid) with finite mean $E[A_n]$. The service times $\{S_n(\theta)\}$ are also iid and depend on a parameter θ, representing the average service time, which is our key variable for optimization. The primary objective is to minimize the average waiting time for customers while maintaining a cost-effective operation of the service station.

The main goal is to determine the optimal mean service time θ that minimizes the cost associated with the average waiting time $W_n(\theta)$ for the first N customers. This involves applying a Stochastic Approximation (SA) algorithm to find the optimal θ that satisfies the desired service level, specifically minimizing the 90% quantile of the average waiting time.

2 Methodology

2.1 Model Formulation

Define the waiting time $W_n(\theta)$ recursively:

$$W_{n+1}(\theta) = \max(0, W_n(\theta) + S_n(\theta) - A_{n+1}),$$

with initial condition $W_0(\theta) = S_0(\theta) = 0$.

Calculate the average waiting time over the first N customers:

$$W^N(\theta) = \frac{1}{N}\sum_{n=1}^{N} W_n(\theta).$$

2.2 Optimization Problem

Aim to minimize the expected squared deviation of the 90% quantile q_θ of $W^N(\theta)$ from a target value z:

$$\min_{0<\theta} E[(q_\theta - z)^2].$$

2.3 Stochastic Approximation Algorithm

Implement the SA algorithm to estimate the quantile and optimize θ. Use the estimator $F_\theta(w) = P(W^N(\theta) \leq w)$, which is monotone decreasing in θ for fixed w.

3 Experiment Setting

3.1 Parameter Selection

Inter-arrival times are exponentially distributed with mean value 5. Service times are exponentially distributed with mean value θ. Set $N = 10$ and $z = 8$.

3.2 Algorithm Implementation

We initially tested multiple different functions, including Simultaneous Perturbation Stochastic Approximation (SPSA) and Stochastic Approximation (SA), each with multiple different data. After evaluating their performance, we decided to use the SA algorithm. This decision was based on its superior performance in terms of stability and convergence towards the optimal mean service time θ.

For the question, it's an ill-posed problem, because $\theta > 0$. In order to make it a well-posed problem, we have to make the θ range bigger or equal to 0. Set

$$\min_{0\leq\theta} J(\theta) = \min_{0\leq\theta} E\left[(q_\theta - z)^2\right] \tag{16}$$

$$\nabla \hat{J}(\theta) = 2E[(q_\theta - z)] \tag{17}$$

$\nabla J(\theta) = 2E[(q_\theta - z)] \times \nabla(q_\theta - z)$, Because q_θ approximates the average service time for a $G/G/1$ queue with a mean service time of θ, the average waiting time for the top 90% of people. As θ increases, q_θ is a strictly monotonically increasing function, because as the average service time lengthens, the waiting time will definitely become longer. Therefore, the derivative of $(q_\theta - z)$ is positive. In this project we dropped it, which means setting this value to 1 to approximate the gradient.

4 Validation and Verification

The result we find from the normal distribution is exactly the same as the red spot we allocated in the figure below, which proves the validity of our experiment.

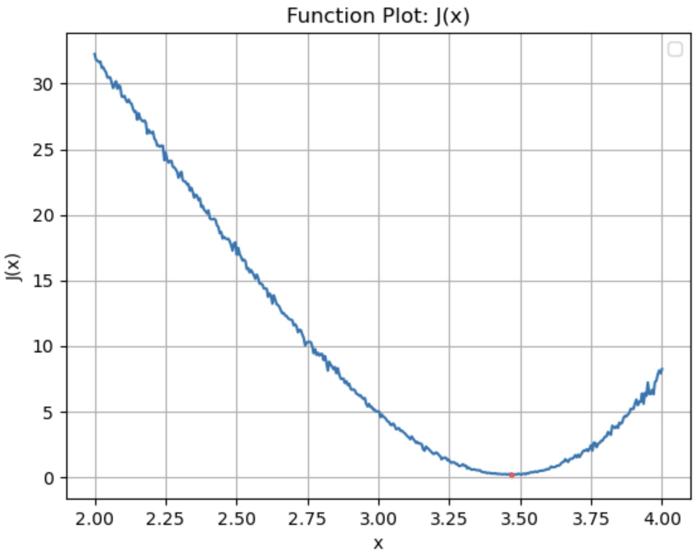

Figure 1: Result in the Objective Function

5 Results and Analysis

In this part, we will present the results of the SA algorithm, showing the convergence of θ towards the optimal value. The figure represented a normal curve, which is symmetric and centered about the mean value of 3.469775865. With a standard deviation of 0.0039682, the data points are tightly clustered around the mean, suggesting that the values are relatively close to each other.

The 95% confidence interval for $\widehat{\theta}$ with distribution $N(3.469775865, 0.0039682)$ is [3.4619980479, 3.47755368204], which means we can be 95% confident that the true population mean falls within the range of values provided by this confidence interval. The small breadth of the confidence interval indicates that the population mean was estimated with a high degree of precision. This suggests there is little variability between the observed data points and the underlying mean value.

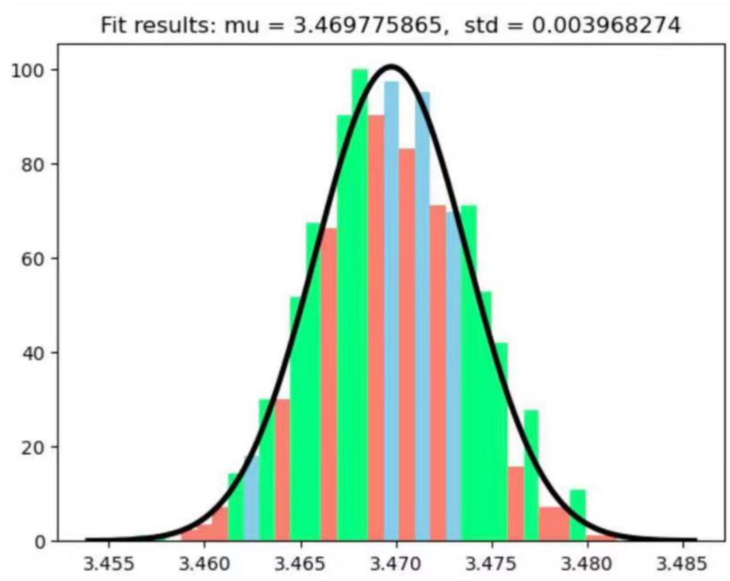

Figure 2: Result Distribution

5.1 Convergence of θ_n and $J(\theta)$

1. Left Diagrams:

- θ_n: Initial $\theta = 5$. Sharp decline initially, stabilizing at a lower value.

- $J(\theta)$: High initial θ value ($J(\theta) \approx 1200$). Steep initial decline, stabilizing at a lower value.

2. Middle Diagrams:

- θ_n: Initial $\theta \approx 3$. Significant fluctuations, indicating instability.

- $J(\theta)$: Mid-range initial θ value. Significant oscillations, indicating poor convergence.

3. Right Diagrams:

- θ_n: Initial $\theta \approx 3$. Rapid initial decline, then stabilization.

- $J(\theta)$: Lower initial θ value ($J(\theta) \approx 60$). Rapid initial decline, then stabilization.

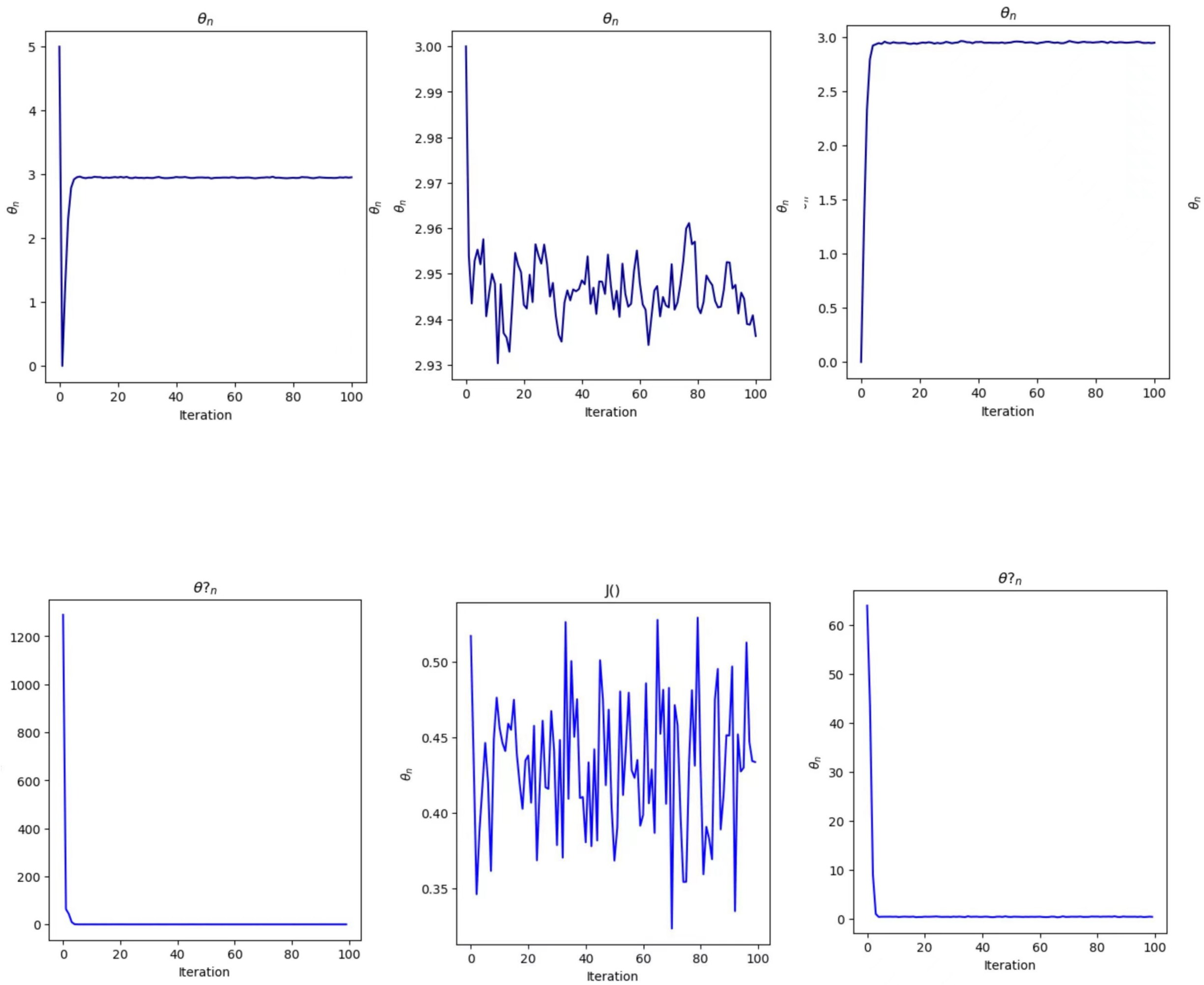

Figure 3: convergence diagrams

Project 2 Question 2

1 Introduction

In the given task, we have to perform the mk = C, optimizing (4) and make Wn(θ) approximate to w within a limited computational budget. To achieve an optimal result that converges well and correctly estimates the 90th percentile, we need to perform m iterations for the update of theta and k iterations to calculate Wn (θ). By optimizing the ratio of m to k, hence we can attain the desired outcome.

2 Implementation

The second task requires us to design an algorithm to simultaneously perform two types of updates: one for the standard Stochastic Approximation (SA) algorithm to find the optimal average service time θ, and another to estimate the quantile q_θ of the waiting time $W^N(\theta)$ under θ. We need to determine a good number of updates m and k, within a fixed budget mk, to improve computational efficiency.

We have to solve optimization problem below:

$$\min_{\theta > 0} \mathbb{E}[(q_\theta - z)^2],$$

where $F_\theta(w) := \mathbb{P}(W^N(\theta) \leq w)$ is the distribution function of $W^N(\theta)$, and F_θ is a monotonically decreasing function of θ.

2.1 Algorithm Design

We applied the following approach to design this algorithm:

1. Outer Loop: Update θ.

2. Inner Loop: For each θ, estimate the quantile q_θ of the waiting time $W^N(\theta)$.

Steps are as below.

1. Initialize θ_0 and $q_{0,0}$ (initial quantile estimate).

2. For each outer loop iteration i: a. Calculate the sequence of waiting times $\{W_n(\theta_i)\}$ under the current θ_i. b. Update $q_{i,j}$ k times using the inner loop to estimate the new quantile. c. Update θ based on the quantile estimate.

3. Return the final estimates of θ and q_θ.

The number of updates for the outer loop is m, and the number of updates for the inner loop is k, with a total budget of mk.

2.2 Choosing m and k

Within the fixed budget mk, we need to allocate m and k efficiently. Typically, the choice of m and k depends on the following factors:

1. Convergence Speed: The inner loop k should be large enough to ensure the convergence of the quantile estimate q_θ.

2. Update Frequency: The outer loop m should be large enough to ensure sufficient updates of the θ parameter.

We can choose m and k based on experience or experimentation. For example:

- Choose m and k such that each outer update has enough inner iterations to ensure the accuracy of q_θ. - Ensure that mk equals the fixed budget.

Assuming the total budget is B, we can set m and k as:

$$m = \sqrt{B}, \quad k = \sqrt{B}.$$

This ensures that $m \times k = B$ and that each update of θ has sufficient inner iterations to estimate the quantile.

3 Proof

To prove that the quantile estimate q_θ updated via the Stochastic Approximation (SA) algorithm will converge to the target quantile p after infinite iterations, we can leverage the convergence theory of the SA algorithm, specifically the Robbins-Monro algorithm's convergence conditions. The detailed information is in 3 between pages 137-139.

4 Result and Analysis

Here's the table of experiment data and final figure we choose.

Items	m	k	theta	q(theta)	q-8
1	1000	500	3. 474143025	8. 040816327	0. 040816327
2	707	707	3. 469593627	8. 055918367	0. 055918367
3	600	833	3. 47511119	8. 02755102	0. 02755102
4	500	1000	3. 475664233	7. 790380762	−0. 209619238
5	400	1250	3. 466433973	7. 72375	−0. 27625
6	300	1667	3. 469458013	7. 621333333	−0. 378666667
7	200	2500	3. 472145236	7. 4085	−0. 5915
8	150	3333	3. 486775509	8. 043877551	0. 043877551
9	100	5000	3. 468914156	8. 002564103	0. 002564103
10	75	6667	3. 467788228	7. 998489796	−0. 001510204
11	50	10000	3. 460175492	7. 603448276	−0. 396551724
12	30	16667	3. 425933218	7. 5	−0. 5

Figure 4: m=500, k=100000

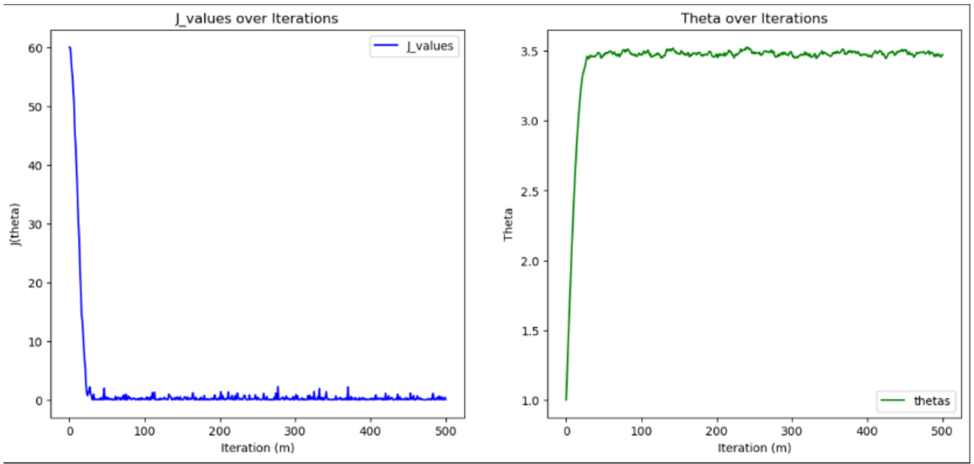

Figure 5: m=500, k=100000

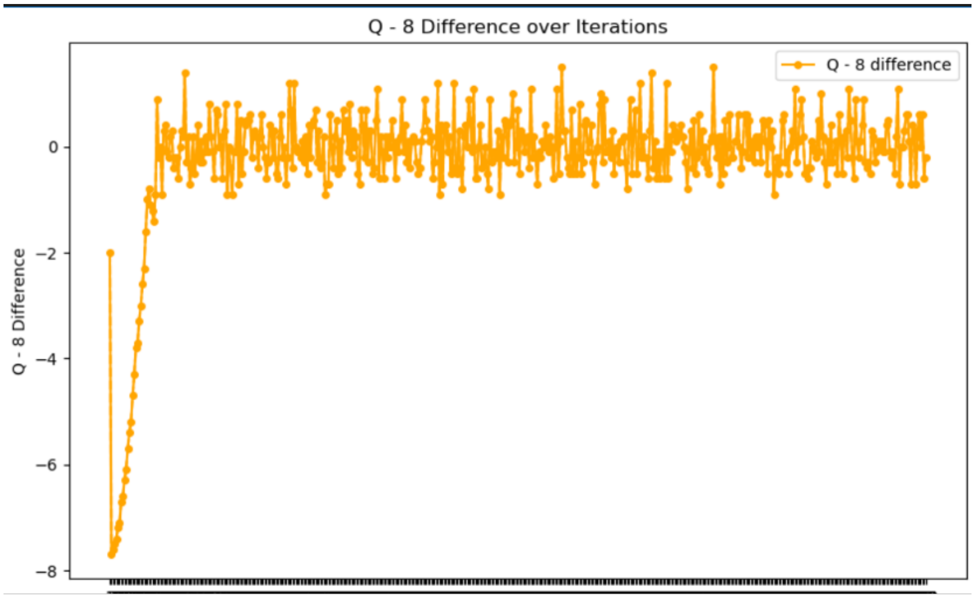

Figure 6: Q-8 Difference Over Iterations

And our final result is m=500, k=10000.

5 Conclusion

For the first task, our objective was to minimize the average waiting time for customers by finding the optimal mean service time θ through the application of a Stochastic Approximation (SA) algorithm. The results showed that the optimized θ converged to a mean value of approximately 3.4698 with a standard deviation of 0.0040.

For the second task, we used an algorithm to perform simultaneous updates for the average service time θ and the quantile estimate q_θ within a fixed computational budget. By optimizing the ratio of outer (m) and inner (k) loop iterations, we ensured efficient and accurate computation. Our experiments demonstrated that the chosen values of $m = 500$ and $k = 100000$ provided optimal results, with θ and $J(\theta)$ converging effectively.

Overall, this project successfully applied stochastic approximation techniques to optimize queueing performance, achieving precise and reliable results with efficient computational resource usage.

References

[1] Monetary Authority of Singapore. (2007). MAS' Framework for Impact and Risk Assessment of Financial Institutions. In https://www.mas.gov.sg. Retrieved July 13, 2024, from https://www.mas.gov.sg/-/media/mas/news-and-publications/monographs-and-information-papers/monograph–mas-framework-for-impact-and-risk-assessment.pdf.

[2] Tudoran, Razvan. (2020). On the Coercivity of Continuously Differentiable Vector Fields. Qualitative Theory of Dynamical Systems. 19. 10.1007/s12346-020-00394-1.

[3] V´azquez-Abad, F. J., Heidergott, B. (2024). Optimization and Learning via Stochastic Gradient Search. Retrieved July 13, 2024.

[4] Hanche-Olsen, H. (2007). Well-posedness for ODEs. In Well-posedness for ODEs. Retrieved July 13, 2024, from https://folk.ntnu.no/hanche/kurs/dynsys/2007v/wellposed.pdf.

[5] Gross, D., Harris, C. M. (2008). Fundamentals of Queueing Theory. Retrieved July 13, 2024.

Appendix A Code from Project 1

```
import numpy as np
import matplotlib.pyplot as plt

def J(p1,p2):# In order to make things easier we decrease the amount of the variable: 3->2
    p3=round(1-p1-p2,7) #Get the value of p3
    #Generating the random variables
    eta1 = np.random.normal(0, 1, 10000)
    eta2 = np.random.normal(0, np.sqrt(2), 10000)
    eta3 = np.random.normal(0, np.sqrt(3), 10000)
    w = np.random.exponential(0.3, 10000)
    V = np.random.normal(0, 1, 10000)
    #Calculate the Xi
    X1 = (0.6 * V + 0.8 * eta1) / np.maximum(w, 1)
    X2 = (0.6 * V + 0.8 * eta2) / np.maximum(w, 1)
    X3 = (0.6 * V + 0.8 * eta3) / np.maximum(w, 1)

    scale=[]#To calculate the sum of three companies
    for i in range(10000):
        scale1,scale2,scale3=0,0,0
        if X1[i]>=2: #To get Xi>=xi
            scale1=round(np.random.uniform(0,X1[i])*p1,7)
        if X2[i]>=3:
            scale2=round(np.random.uniform(0,X2[i])*p2,7)
        if X3[i]>=1:
            scale3=round(np.random.uniform(0,X3[i])*p3,7)
        scale.append(round(scale1+scale2+scale3,7))
    summary=sum(scale)/10000 #calculate the Sharpe ratio
    std=np.std(scale)
```

```python
    expect=round(summary/std,9)
    return -expect #To find the maxima of the J(theta) could be convert to find the minima of -J(tehta)
def J_graint(J,p1,p2,i): #Use the SPSA algorithm to calculate the gradient
    y1,y2=0,0 #gradient from the two directions
    r1=np.random.choice([1,-1])
    r2=np.random.choice([1,-1])
    theta=1/(i+10000)**0.3 #Using the decrasing theta
    y1=(J(p1+theta*r1,p2+theta*r2)-J(p1-theta*r1,p2-r2*theta))/(2*theta*r1)
    y2=(J(p1+theta*r1,p2+theta*r2)-J(p1-theta*r1,p2-r2*theta))/(2*theta*r2)
    return y1,y2
def SA(J_graint,J,theta_list1,theta_list2,number): # The stochastic approximation algorithm
    for i in range(number):
        step_size=10/(i+100) #Uing the decreasing learning rate
        Jvalue.append(J(theta_list1[i],theta_list2[i]))
        y1,y2=J_graint(J,theta_list1[i],theta_list2[i],i)
        suml=theta_list2[i]+theta_list1[i]
        if suml>1: #In this case we do the projection
            theta_list2[i]=theta_list2[i]-(suml-1)/2
            theta_list1[i]=theta_list1[i]-(suml-1)/2
        theta_list1.append(max(theta_list1[i]-step_size*y1,0))
        theta_list2.append(max(theta_list2[i]-step_size*y2,0))
    return theta_list1,theta_list2

p1_list=[] #It records the last 500 experiments' mean of p1
p2_list=[] #It records the last 500 experiments' mean of p2
```

```python
Experiment_number=4#The total experiment times
for num in range(Experiment_number):
    Jvalue=[] #To record the J(theta) values
    number=3000 #iteration times
    theta_list1=[0.5] #The starting point of p1
    theta_list2=[0.5] #The starting point of p2
    p1s,p2s=SA(J_graint,J,theta_list1,theta_list2,number)
    p1s=np.array(p1s[-501:-1]) #Get the final 500 points
    p2s=np.array(p2s[-501:-1])
    mean_p1 = np.mean(p1s) #Get the mean of the converge data
    mean_p2 = np.mean(p2s)
    p1_list.append(np.mean(p1s))
    p2_list.append(np.mean(p2s))
    file_path1 = r"point.txt"
    with open(file_path1, 'a') as f:
        f.write(f'Experiment {num}: Mean p1 = {mean_p1}, Mean p2 = {mean_p2}\n')#Write the p1 and p2 to a txt file
    fig, axs = plt.subplots(1,3, figsize=(20,5))
    axs[0].plot(theta_list1, color="green")
    axs[0].set_xlabel("Iteration")
    axs[0].set_ylabel(r"$\theta1_n$")
    axs[0].set_title(r"$\theta1_n$")
    axs[1].plot(theta_list2, color="blue")
    axs[1].set_xlabel("Iteration")
    axs[1].set_ylabel(r"$\theta2_n$")
    axs[1].set_title(r"$\theta2_n$")
    axs[2].plot(Jvalue, color="red")
    axs[2].set_xlabel("Iteration")
    axs[2].set_ylabel("J value")
    axs[2].set_title("The J function")
```

```
        save_path = f"{num}.png"
        fig.savefig(save_path) #Get the diagram
        plt.close(fig)
```

Appendix B Code from Project 2.1

```
import numpy as np
def J(theta,z):
    #np.random.seed(10)
    s=0
    d=0
    for k in range(100):
        W_mean=[]
        for j in range(700):
            A=np.random.exponential(5, 12)
            S=list(np.random.exponential(theta, 12))
            S.insert(0,0)
            W=[0]
            for i in range(11):
                W.append(max(0,W[i]+S[i]-A[i+1]))
            W_mean.append(np.mean(W[1:11]))
        W_mean.sort()
        l=W_mean[int(0.9*len(W_mean))]
        s+=(l-z)**2
        d+=l-z
    return s/100,d/100

theta=[]
def SA(J,theta_list,z,number):
    for i in range(number):
        theta=10/(i+100)
```

```python
        #theta_list.append(max(0,theta_list[i]-SPSA(J,theta_list[i],z)*theta))
        s,d=J(theta_list[i],z)
        theta_list.append(max(0,theta_list[i]-2*(d)*theta))
        Jvalue.append(s)
    return theta_list[-1]

for i in range(500):
    Jvalue=[]
    number=20
    z=8
    theta_list=[1]
    SA(J,theta_list,z,number)
    theta.append(np.mean(theta_list[15:-1]))

plt.figure(1)
mu = np.mean(theta)
std = np.std(theta)
number_of_bins=40
bins = np.linspace(mu-4*std, mu+4*std, number_of_bins)
n, bins, patches = plt.hist(theta, bins, density=True)

def true_value(i):
    return norm.pdf(((mu-4*std+i*2*4*std/(number_of_bins-1))+2*4*std/(2*(number_of_bins-1))), mu, std)

for i in range(number_of_bins-1):
    if n[i] < true_value(i):            # Give bar red color if lower than PDF
        patches[i].set_fc('salmon')
```

```
        if n[i] > true_value(i):            # Give bar green color if higher than PDF
                patches[i].set_fc('springgreen')
        if n[i] <= 0.05*n[i] + true_value(i) and n[i] >= -0.05*n[i] + true_value(i):
            patches[i].set_fc('skyblue')    # Give bar blue color if approx. equal to PDF

# Plotting the PDF of the normal distribution
x = np.linspace(mu-4*std, mu+4*std, 1000)
y = norm.pdf(x, mu, std)                # PDF: 1/(  (2  ))*exp(-(x-  )^2/2  ^2)

plt.plot(x, y, lw=3, color='black', label = 'PDF')
# print(" mean theta_n = ", mu, "standard deviation theta_n ", std)
title = "Fit results: mu = %.9f,  std = %.9f" % (mu, std)
plt.title(title)
#plt.title("SA experiment")
plt.show()

import matplotlib.pyplot as plt
fig, axs = plt.subplots(1,2, figsize=(10,5))
# Plot the iterate
axs[0].plot(theta_list, color="darkblue")
axs[0].set_xlabel("Iteration")
axs[0].set_ylabel(r"$\theta_n$")
axs[0].set_title(r"$\theta_n$")
axs[1].plot(Jvalue, color="blue")
axs[1].set_xlabel("Iteration")
axs[1].set_ylabel(r"$\theta_n$")
```

```
axs[1].set_title('J()')
```

Appendix C Code from Project 2.2

```
def J_function(theta0,q,k):
    #np.random.seed(20)
    s=0
    d=0
    z=8
    alpha=0.1
    w=0.9
    A=np.random.exponential(5,20002)
    S=np.random.exponential(theta0,20002)
    S=list(S)
    S.insert(0,0)
    W=[0]
    W_mean=[]
    for o in range(20001):
        W.append(max(0,W[o]+S[o]-A[o+1]))
    W_mean.append(np.mean(W[1:20001]))
    W_mean.sort()
    l=np.mean(W_mean[int(0.9*len(W_mean))-10:int(0.9*len(
        W_mean))+10])
    s+=(l-z)**2
    d+=l-z
    for j in range(k):#k times inner loop
        for ww in W_mean:
            if(ww<=q):
                q-=alpha*(1-w)
            else:
                q-=alpha*(0-w)
```

```
    return s,d,q


import matplotlib.pyplot as plt
import numpy as np
theta0=1
q=6
z=8
w=0.9
m=1000
k=1000
alpha=0.1
learning_rate=0.01
thetas=[theta0]
J_values=[J_function(theta0,z,k)[0]]
q_list=[q]
l_list=[]
for i in range(m):
    s,d,q=J_function(theta0,q,k)
    q_list.append(q)
    #theta0=theta0 -2*(10/(i+100))*d
    theta0=theta0 -2*0.01*d
    thetas.append(theta0)
    J_values.append(s)

plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.plot(J_values, label='J_values', color='blue')
plt.xlabel('Iteration (m)')
plt.ylabel('J(theta)')
plt.title('J_values over Iterations')
```

```
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(thetas, label='thetas', color='green')
plt.xlabel('Iteration (m)')
plt.ylabel('Theta')
plt.title('Theta over Iterations')
plt.legend()
differences = [q - 8 for q in q_list]
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(q_list) + 1), differences, label='Q - 8 difference', color='orange', marker='o', linestyle='-', markersize=4)
for i in range(1, len(q_list)):
    plt.plot([i, i+1], [differences[i-1], differences[i]], color='orange', linestyle='--')
plt.legend()
plt.title('Q - 8 Difference over Iterations')
plt.xticks(range(1, len(q_list) + 1))
plt.ylabel('Q - 8 Difference')
plt.show()
```