Simulation Project

Sun Xueli, Zhen Qi, Yang Ruizhen, Li Haoxin*

July 2024

*Group 18.

Question 1

1 Abstract

This research study aims to find the optimal investment strategy for an investor with a total capital of 1 to invest in three companies over a given time period, which maximizes the expected return on investment while accounting for the associated risks.

The gradient descent algorithm is used to recursively find the maximum value and relevant investing strategy. The Martingale Difference Noise Model is used to deal with stochastic factors in the problem. The SPSA algorithm is used to estimate the gradient. The projection algorithm is designed to ensure restrictions.

The result turns out that $(p_1, p_2, p_3)^T = (0.3741, 0.2070, 0.4189)^T$, and Sharpe Ratio reaches 0.5125.

The findings provide insights into the optimal investment strategies for investors with similar investment goals and constraints.

2 Content

3 Introduction

1. Problem Description

An investor has total capital of 1 that she wants to invest in n companies. The run time for the investment is t time units. The market value of company i at time t is given by X_i. Let $x_i \in R, i = 1, 2, 3$ be given thresholds. Company i will not be able to generate any profit if $X_i < x_i$. In case $X_i \leq x_i$, the return on the investment is given by Y_i. Investing a fraction pi of the capital yields expected return

$$p_i E[Y_i 1_{X_i \leq x_i}]$$

for company i, where

$$\sum_{i=1}^{n} p_i = 1,\ 0 \leq p_i \leq 1.$$

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

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

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

The investor wants to find the optimal investment strategy by maximizing the risk adjusted performance of the investment (also known as Sharpe ratio), which leads to

$$J(p) = E\left[\frac{\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}}{\text{std}\left(\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}\right)}\right]$$

Our goal is to find

$$\max_{p_1, p_2, p_3 \geq 0, \sum p_i = 1} J(p) \tag{1}$$

2. Problem Well-posedness

This section aims to prove that the problem, which is initialized in section 1.3.2, is well-posed.

The pre-requisite of a well-posed problem is being an NLP(non-linear problem). To declare the property, first, we take a deep look into the objective function. $J(p)$ is the expectation value of a relatively simple expression. Though containing random variables, the expression is no more than combination of elementary functions $w.r.t$ 3 independent variables(p_i).

Thus, we assume that the objective function and its first order derivative are continuous, which

could be inspected by the graph 1. Notice that the graph shows the objective value $w.r.t.$ p_1 and p_2, while $p_3 = p_1 + p_2$.

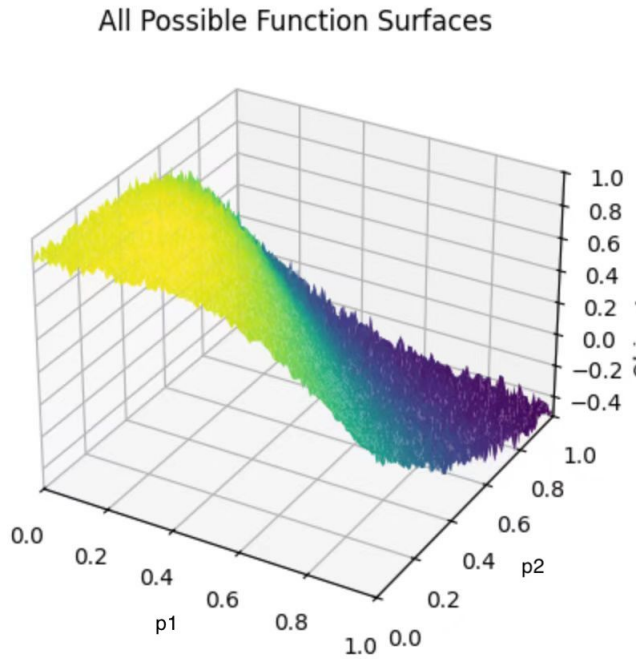

Figure 1: Graph for Objective Function

With $p_i \in [0, 1]$, we define $g_1(p)$ and $g_2(p)$ as,

$$g_1(p) = -p \qquad\qquad g_2(p) = p - 1$$

For $\forall x, y \in R^3$,

$$g_1(\alpha x + (1-\alpha)y) = \alpha g_1(x) + (1-\alpha)g_1(y)$$
$$g_2(\alpha x + (1-\alpha)y) = \alpha g_2(x) + (1-\alpha)g_2(y)$$

By definition, we can conclude that $g_i(\theta),\ i = 1, 2, 3$, is convex. As $h(\theta) = \theta_1 + \theta_2 + \theta_3 - 1$, $h(\theta)$is an affine function. Thus, $\max_{p\in\Theta} J(p), \Theta = \{p \in R^3 : g(p) \leq 0, h(p) = 0\}$ is an NLP.

Then we move on to prove that the problem is well-posed:

a. The solution set $\Theta = \{\{p \in R^3 : g(p) \leq 0, h(p) = 0\}$ is inevitable to be non-empty.

b. As J(p) is continuous and the boundary is compact, By Weierstrass J Theorem, there exists a minimum within constraint at the stationary point θ^*, which means the function contains only KKT points.

c. Since Θ is compact, the set of KKT points are confined to a bounded set.

The conditions suffices to prove that the problem is well-posed.

1.4 Description of the Optimization Approach

1. Introdunction to Martingale Difference Noise Model

We use the projection gradient descent algorithm, that is, set

$$p_n = \Pi_\Theta[p_{n-1} + \epsilon_n G(p_n)]$$

where Θ is the domain of p.

As the objective function is a stochastic process, we introduce the Martingale noise model. The information of the stochastic approximation is based on the natural filtration of the process, using the increasing sequence of σ -algebras:

$$\mathcal{F}_n := \sigma(p_0; \xi_0(p_0), ..., \xi_n(p_n)) \subset \mathcal{F}, n \leq 0$$

where we assume that $\mathcal{F}_0$ contains all P-null sets and $\xi_n(p_n)$ is the observation of $J(p_n)$ on a common measurable space $(\Omega, \mathcal{F})$. Additionally,

$$\begin{aligned} Y_n &= G(p_n) + \delta M_n + \beta_n = G(p_n) + \delta M_n \\ \delta M_n &:= Y_n - E[Y_n|\mathcal{F}_{n-1}] \\ V_n &:= E[\delta M_n^2] = E[(Y_n - E[Y_n|\mathcal{F}_{n-1}])^2] \end{aligned}$$

where Y_n is the proxy for $G(p_n)$, δM_n is noise for Y_n estimating $G(p_n)$, and β_n is arguably 0 using the SPSA algorithm, which will be proved in section 1.5.2. V_n denotes the variance for δM_n.

2. Introduction to the SPSA approach

The SPSA approach basically provides a practical way for estimating gradient at point p_n. The algorithm requires only 2 data points to estimate gradient at certain point, which effectively handle the high-dimensional optimization problem. The approach is unbiased in probability, which will be proved in detail in section 1.5.2.

3. Projection Method

We design a simple projection algorithm to cast points outside boundary back as well as remain the most of information. The algorithm is based on linear function on R^2. We will introduce it in section 1.6.5.

Overall, the algorithm could be written as,

$$p_n = \Pi_\Theta[p_{n-1} + \epsilon_n \nabla J^{SPSA}(p_n)]$$

In the next section, we will prove that the vector field estimated by the SPSA approach is coercive, and Martingale noise model does fit the stochastic problem.

5 Validation and Verification

1. Coercivity

a. To prove $G(p) = J^{SPSA}(p)$ is a coercive vector field, we first prove $\nabla J(p)$ is coercive, then move on to prove that $J^{SPSA}(p)$ is an unbiased proxy for $\nabla J(p)$.

From the previous assumption that $\nabla J(p)$ is a continuous function on Θ and its neighbor, since Θ is compact, by Weierstrass J Theorem

$$\exists\, L \in R, \text{ s.t. } \forall p \in \Theta, \|\nabla J(p)\| \leq L$$

which indicates the Lipschitz continuity of $J(p)$. Now prove that $\nabla J(p)$ is also locally Lipschitz continuous. For $\forall x, y \in R^3, \forall \alpha \in R$,

$$\nabla J(x + \alpha y) - \nabla J(x) = \int_0^\alpha \nabla^2 J(x + ty) y \, dt \tag{2}$$

$$\| \int_0^\alpha \nabla^2 J(x + ty) y \, dt \| = \|\nabla J(x + \alpha y) - \nabla J(x)\| \leq \alpha L \|y\| \tag{3}$$

Divide both side by $\alpha\|y\|$,

$$\| \frac{\int_0^\alpha \nabla^2 J(x + ty) \, dt}{\alpha} \| \leq L \tag{4}$$

Consider $\alpha \to 0$, by the Mean Value Theorem for Definite Integral,

$$\nabla^2 J(x) \leq L \tag{5}$$

Thus, we succeed to prove $J(p)$ is coercive vector field for the well-posed problem (1). We then move on to prove that $G^{SPSA}(p)$ is an unbiased proxy for $\nabla J(p)$.

b. Analysis of SPSA Algorithm

Let $\eta_n = \frac{1}{10+n^{\frac{1}{4}}}$, $\Delta_n \in R^3$ is a random vector s.t. $\{\Delta_n\}$ is a iid sequence, where $\Delta_n(i) \in \{-1, 1\}$, $P(\Delta_n(i) = -1) = \frac{1}{2} = P(\Delta_n(i) = 1)$. Let

$$\left(\nabla J^{\text{SPSA}}(p_n)\right)_i = \frac{\xi(p_n + \eta_n \Delta_n) - \xi(p_n - \eta_n \Delta_n)}{2\eta_n \Delta_n(i)}$$

We assume that $E[\xi(p_n)|\mathcal{F}_{n-1}] = J(p_n)$ and $E[\xi(p_n + \eta_n\Delta_n)|\mathcal{F}_{n-1}] = E[J(p_n + \eta_n\Delta_n)]$. Thus,

$$E\left[\left(\nabla J^{\text{SPSA}}(p_n)\right)_i |\mathcal{F}_{n-1}\right] = \frac{1}{2\eta_n \Delta_n(i)} E\left[J(p_n + \eta_n\Delta_n) - J(p_n - \eta_n\Delta_n)\right]$$

By Taylor Expansion,

$$
\begin{aligned}
&= \frac{1}{2\eta_n\Delta_n(i)} E\left[\left(J(p_n) + \nabla J(p_n)\eta_n\Delta_n + \frac{1}{2}\nabla^2 J(p_n)(\eta_n\Delta_n)^2 + \frac{1}{6}\nabla^3 J(x)(\eta_n\Delta_n)^3\right)\right. \\
&\quad \left. - \left(J(p_n) + \nabla J(p_n)(-\eta_n\Delta_n) + \frac{1}{2}\nabla^2 J(p_n)(-\eta_n\Delta_n)^2 + \frac{1}{6}\nabla^3 J(y)(-\eta_n\Delta_n)^3\right)\right] \qquad (6) \\
&= (\nabla J(p_n))_i + \sum_{k=1,k\neq i}^{3} [\nabla J(p_n)]_k E\left[\frac{\Delta_n(k)}{\Delta_n(i)}\right] + \frac{\eta_n^2}{6\Delta_n(i)}\Delta_n^3\left[\nabla^3 J(x) - \nabla^3 J(y)\right]
\end{aligned}
$$

where $x = p_n + t_1\Delta_n$ and $y = p_n - t_2\Delta_n$, $t_1, t_2 \in [0, \eta_n]$.

Because of the distribution of $\Delta_n(i)$,

$$E\left[\frac{\Delta_n(k)}{\Delta_n(i)}\right] = 0, k \neq i$$

which indicates that the only bias term is

$$\beta_n := \frac{\eta_n^2}{6\Delta_n(i)}\Delta_n^3\left[\nabla^3 J(x) - \nabla^3 J(y)\right]$$

Since $\nabla J(p)$ is locally Lipschitz continuous, it can be proved that $\nabla^2 J(p)$ is also locally Lipschitz continuous and $\nabla^3 J(p)$ is bounded(The proof would be similar to (2)-(5)). Thus,

$$\exists\, \delta \in R, s.t. \forall n,\ \|\beta_n\| \leq \eta_n^2\delta$$

Consider $G(p) = J^{SPSA}(p)$ in original gradient descent algorithm $p_n = p_{n-1} + \epsilon_n G(p_n)$. Then, $p_n = p_0 + \sum_{i=0}^{n}\epsilon_i G(p_i) = p_0 + \sum_{i=0}^{n}\epsilon_i J(p_i) + \sum_{i=0}^{n}\epsilon_i\beta_i$.

To analyze the bias term, we set $\epsilon_n = \frac{1}{n+1}$,

$$\sum_{i=0}^{n}\epsilon_i\beta_i \leq \sum_{i=0}^{n}\epsilon_i\eta_i^2\delta = \sum_{i=0}^{n}\delta\frac{1}{i+1}\frac{1}{i^{\frac{1}{2}}} < \infty$$

Thus, the bias term is asymptotically negligible.

Denote $v(p) = Var[\xi(p)]$, i.e., the variance of observation of the objective function given certain p. The variance of noise (defined in section 4.1)

$$V_n = \frac{v(p_n + \eta_n\Delta_n) - v(p_n - \eta_n\Delta_n)}{(2\eta_n\Delta)^2}$$

Assume that $v(p)$ is bounded in Θ (shown by graph), then $V_n = \mathcal{O}[(\eta_n\Delta_n)^{-2}] = \mathcal{O}(\eta_n^{-2})$. Thus,

$$\sum_{i=0}^{\infty}\epsilon_n^2 V_n = \sum_{i=0}^{\infty}\frac{1}{(i+1)^2}\mathcal{O}(\eta_i^2) = \sum_{i=0}^{\infty}\mathcal{O}\left[\frac{1}{(1+i)(10+i^{\frac{1}{4}})}\right] < \infty$$

We finally arrive at the conclusion that $J^{SPSA}(p)$ is a valid proxy for $\nabla J(p)$, and that $G(p) = J^{SPSA}(p)$ generates a coercive vector field for the well-posed problem (1).

2. Validation for Martingale Noise Model

This part aims to prove that we are able to find the solution p^* by the recursive algorithm $p_n = \Pi_\Theta[p_{n-1} + \epsilon_n \nabla J^{SPSA}(p_n)]$, where $\nabla J^{SPSA}(p_n)$ has been proved to be a valid proxy for $\nabla J(p_n)$. Again, we set $\epsilon_n = \frac{1}{n+1}$.

The algorithm satisfies four following conditions,

(a) $\sum_{i=0}^{\infty} \epsilon_i = \sum_{i=0}^{\infty} \frac{1}{i+1} = \infty$

(b) $\sum_{i=0}^{\infty} \epsilon_i \|\beta_i\| < \infty$, since $\beta_n = E[\nabla J^{SPSA}(p_n)] - \nabla J(p_n)$ vanishes asymptotically, as we prove in 1.5.1.

(c) $\sum_{i=0}^{\infty} \epsilon_n^2 V_n < \infty$, which is proved in section 5.1.

(d) $G(p_n)$ is Lipschitz continuous on Θ, and the trajectories of the projected ODE

$$\frac{dx(t)}{dt} = (\Pi_\Theta G)(x(t)),\ x(0) \in \Theta$$

has a unique asymptotically stable point p^*, since the problem is well-posed.

By theorem, we conclude that $\theta_n \to \theta^*$ when $n \to \infty$.

6 Program Description

1. Dimensionality Reduction:

Due to the complexity of projecting a vector in R^3, we project the three-dimensional problem into two-dimensional and simply substitute p_3 by $(1 - p_1 - p_2)$. The following is the proof of the validation of the method.

Assume the optimal solution obtained by directly solving the original problem is p_1, p_2, p_3 Since (p_1, p_2, p_3) satisfy the constraint condition

$$p_1 + p_2 + p_3 = 1 \Rightarrow p_3 = 1 - p_1 - p_2$$

Substituting p_3 in the original problem, we get:

$$\max J((p_1, p_2, 1 - p_1 - p_2)^T),\ p_1, p_2 \geq 0,\ p_1 + p_2 \leq 1$$

The solution to this optimization problem is $p^* = (p_1^*, p_2^*, 1 - p_1^* - p_2^*) = (p_1^*, p_2^*, p_3^*)^T$. Therefore, $(p_1, p_2, p_3)^T$ and $(p_1^*, p_2^*, p_3^*)^T$ are equivalent.

Note that in the following coding sections, due to programming indexing, p[0] and p[1] are respectively p_1 and p_2.

2. Generate the Objective Function

```
def objective_j(p,std_sample_time):

    threshold = [2,3,1]

    rate = 0.3

    def eta(i):
        return np.random.normal(0, np.sqrt(i+1), size = std_sample_time)

    def V():
        return np.random.normal(0, 1, std_sample_time)

    def W():
        return np.random.exponential(rate, std_sample_time)

    def Yi(xi):
        return np.random.uniform(0, xi, std_sample_time)

    Yi_values= np.zeros((3,std_sample_time))
    xi_values = np.zeros((3,std_sample_time))
    PiYi = np.zeros(std_sample_time)

    for i in range(3):

        xi_values[i] = (0.6 * V() + 0.8 * eta(i)) / (np.maximum(W(), 1))
        Yi_values[i] = np.where(xi_values[i] >= threshold[i], Yi(xi_values[i]), 0)

        if i != 2:
            PiYi += p[i] * Yi_values[i]
        else:
            PiYi += (1-p[0]-p[1]) * Yi_values[2]

    result_std = np.std(PiYi)

    return PiYi[0] / result_std if result_std!=0 else 0
```

The provided code takes in `p` (NumPy array $[p_1, p_2]$) and `std_sample_time` (the number of samples to estimate std $\left(\sum_{i=1}^{3} p_i Y_i \mathbf{1}_{X_i \geq x_i}\right)$). We generate $\eta \sim N(0, i+1), V \sim N(0,1), W \sim exponential(0.3)$ to simulate the stochastic factors. Note that using the NumPy library allows for much faster behavior in the program.

```
def exp_objective_j(p,exp_sample_time,std_sample_time):
    j_values = np.zeros(exp_sample_time)
    for i in range(exp_sample_time):
        j_values[i] = objective_j(p,std_sample_time)
    return j_values.mean()
```

This part takes in `p`, `exp_sample_time` (the number of samples to estimate the expection value, i.e., the value of objective function), and `std_sample_time`. The return value would be one observation of the objective function.

3. SPSA Algorithm

```
def estimate_gradient(p, i, rng):
    delta_i = rng.choice((-1, 1), size = 2)
    eta_i = 1/(10 + np.power(i,1/4))
        perturbation_high = p + eta_i * delta_i
    perturbation_low = p - eta_i * delta_i
    obj_perturbation_high = exp_objective_f(perturbation_high, SAMPLE_TIME)
    obj_perturbation_low = exp_objective_f(perturbation_low, SAMPLE_TIME)
    numerator = obj_perturbation_high - obj_perturbation_low
    denominator = 2*eta_i*delta_i
    gradient_estimate = np.divide(numerator, denominator)

    return gradient_estimate
```

The algorithm takes in `p`, `i` (the iteration index) and `rng` (random number generator). Notice that the algorithm only compute 2 values of the objective function, which could arguably shorten runtime.

4. Gradient Descent Algorithm

```
def SPSA(SEED, p_0, EPSILON_TYPE, EPSILON_VALUE, NR_ITERATIONS):
    rng = np.random.default_rng(SEED)
    rng_noise = np.random.default_rng(NOISE_SEED)
    p_set = np.zeros((NR_ITERATIONS+1, 2))  # 2 is the number of elements in each p vector
    gradients = np.zeros((NR_ITERATIONS, 2))
    objective_values = np.zeros(NR_ITERATIONS+1)
    p_set[0] = p_0
    objective_values[0] = exp_objective_f(p_set[0], SAMPLE_TIME)

    for i in range(NR_ITERATIONS):
        g = estimate_gradient(p_set[i], i, rng)
        gradients[i] = g
        if EPSILON_TYPE == 'fixed':
            p_set[i+1] = p_set[i] + EPSILON_VALUE * g
        if EPSILON_TYPE == 'decreasing':
            p_set[i+1] = p_set[i] + EPSILON_VALUE/(i+1) * g
        p_set[i+1] = project_onto_plane(p_set[i+1])
        objective_values[i+1] = exp_objective_f(p_set[i+1], SAMPLE_TIME)

    return return p_set, gradients, objective_values, rng_noise
```

The algorithm follows the core equation $p_n = \Pi_\Theta[p_{n-1} + \epsilon_n \nabla J^{SPSA}(p_n)]$, and iteratively finds the next `p` for `NR_ITERATIONS` times.

5. Projection

```
def project_onto_plane(p):

    if 1 >= p[0] >= 0 and 1 >= p[1] >= 0 and 0 <= p[0] + p[1] <= 1:
        return p
    else:
        if p[0] < 0:
            p[0] = 0
        if p[1] < 0:
            p[1] = 0
        if p[0] + p[1] > 1:
            if p[1] - p[0] > 1:
                return [0,1]
            elif p[1] - p[0] < -1:
```

```
                return [1,0]
            else:
                tmp0, tmp1 = p[0],p[1]
                p[0] = (tmp0 - tmp1 + 1) / 2
                p[1] = (tmp1 - tmp0 + 1) / 2
    return p
```

As shown in Figure 1, to Minimize information loss, the area in green is projected to point (0,1), the area in blue is projected onto the line AD, while the points in red are projected to (1,0).

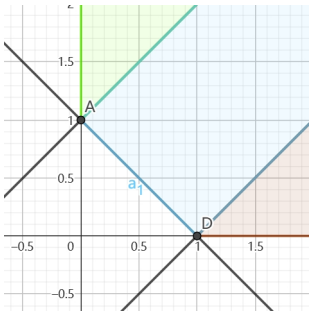

Figure 2: Projection Illustration

7 Results

1. Initialization

To save computing energy, we first initialize the program and find a reasonable `p`.

For the first run, we set:

```
p_0 = np.array([0.6, 0.4])
SEED = 1
EPSILON_VALUE = 0.001
EPSILON_TYPE = 'fixed'
NR_ITERATIONS = 1000
EXP_SAMPLE_TIME = 1000
STD_SAMPLE_TIME = 10
p_set,gradients,object_values = SPSA(SEED, p_0, EPSILON_TYPE, EPSILON_VALUE, NR_ITERATIONS)
```

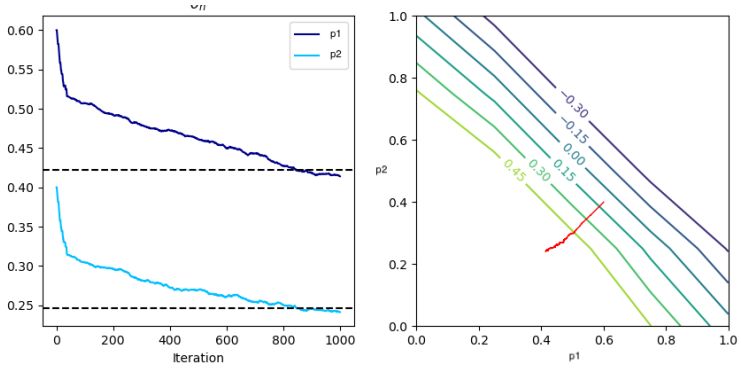

Figure 3: First Initialization

The result is around $p = (0.42, 0.23, 0.35)^T$, while there is still a clear downward trend for p_1. Thus, for the second run we set:

```
p_0 = np.array([0.42,0.23])
SEED = 1
EPSILON_VALUE = 0.001
EPSILON_TYPE = 'fixed'
NR_ITERATIONS = 1000
EXP_SAMPLE_TIME = 1000
STD_SAMPLE_TIME = 10
p_set,gradients,object_values = SPSA(SEED, p_0, EPSILON_TYPE, EPSILON_VALUE, NR_ITERATIONS)
```

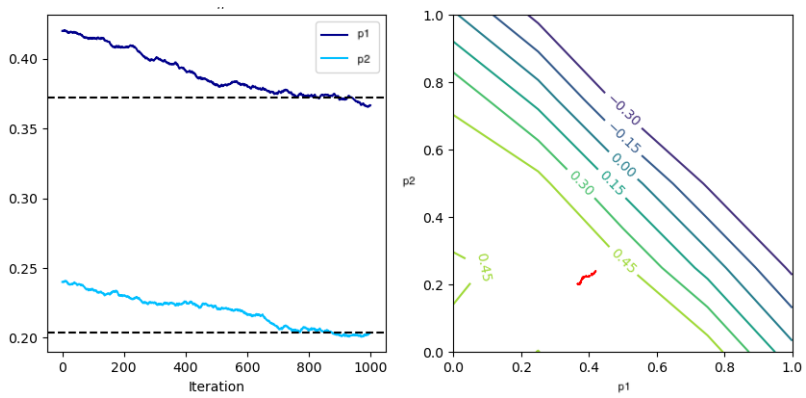

Figure 3: Second Initialization

From the result, we initialize the program with `p_0` = [0.37,0.20].

2. Output

For the choosing of fixed-epsilon and decreasing-epsilon, we compare the graph of these two methods. For fixed epsilon, we choose $\epsilon = 0.001$ and $\epsilon = 0.01$. For decreasing epsilon, we choose $\epsilon_n = \frac{1}{1+n}$ so that it satisfied all the conditions needed to perform the algorithm. The outputs and graphs are as below:

a. Fixed $\epsilon = 0.001$

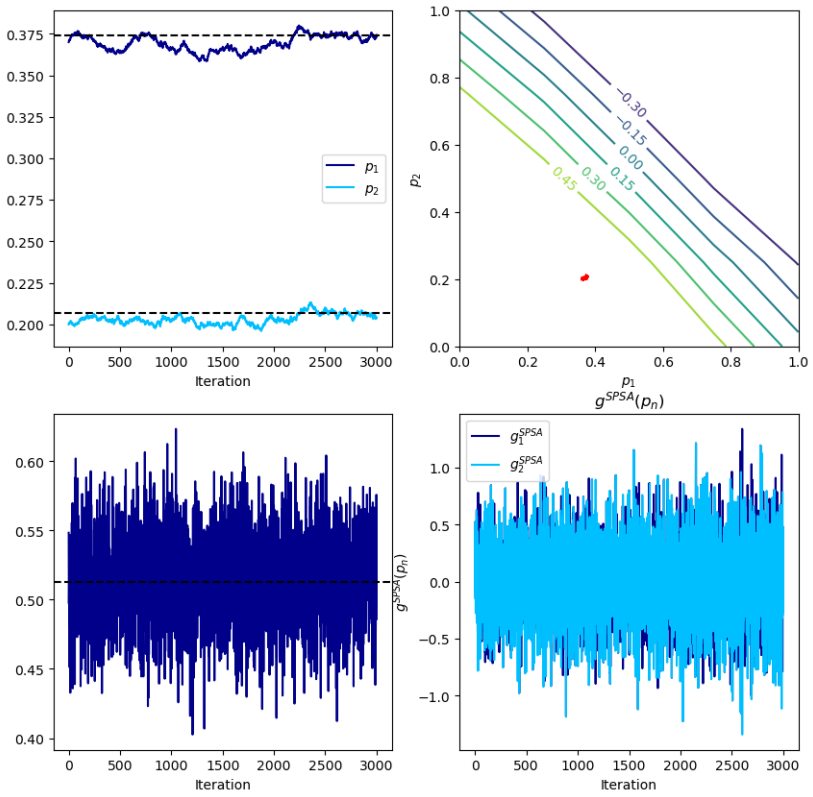

Figure 5: Output for fixed $\epsilon = 0.001$

```
p_0 = np.array([0.37, 0.20])
SEED = 1
EPSILON_TYPE = 'decreasing'
EPSILON_VALUE = 0.001
NR_ITERATIONS = 3000
EXP_SAMPLE_TIME = 1000
STD_SAMPLE_TIME = 10

# following are the output
p1: 0.3741045088018481
p2: 0.20706347666784272
Result mean of last 25% run: 0.5124946512012143
```

b. Fixed $\epsilon = 0.01$

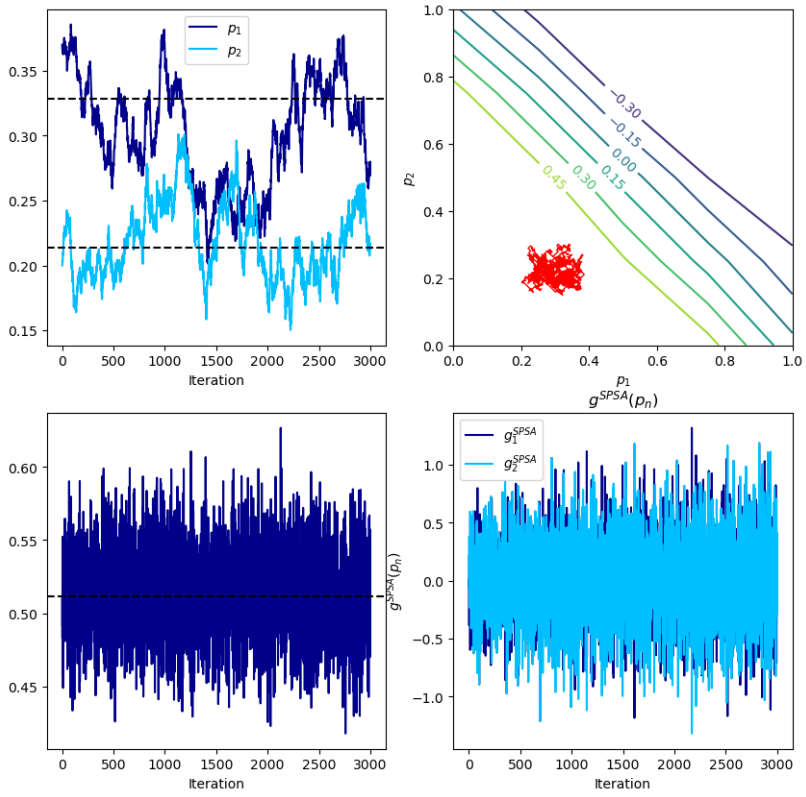

Figure 6: Output for fixed $\epsilon = 0.01$

```
p_0 = np.array([0.37, 0.20])
SEED = 1
EPSILON_TYPE = 'decreasing'
EPSILON_VALUE = 0.01
NR_ITERATIONS = 3000
EXP_SAMPLE_TIME = 1000
STD_SAMPLE_TIME = 10

# following are the output
p1: 0.3282773409301514
p2: 0.21339922003848874
Result mean of last 25% run: 0.5116953824230308
```

c. Decreasing $\epsilon_n = \frac{1}{1+n}$

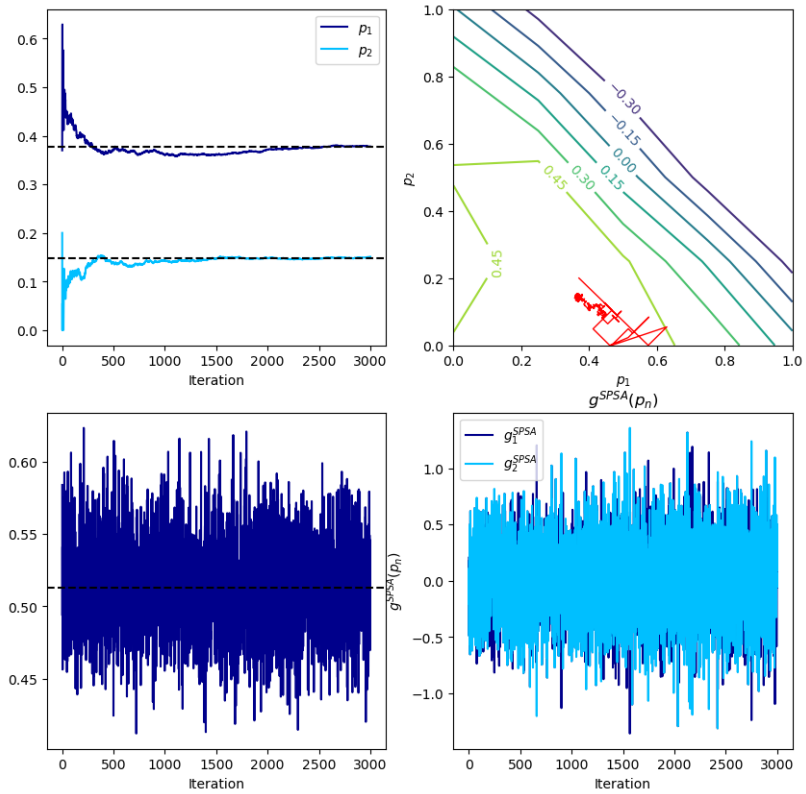

Figure 7: Output for decreasing $\epsilon_n = \frac{1}{1+n}$

```
p_0 = np.array([0.37, 0.20])
SEED = 1
EPSILON_TYPE = 'fixed'
EPSILON_VALUE = 1
NR_ITERATIONS = 3000
EXP_SAMPLE_TIME = 1000
STD_SAMPLE_TIME = 10

# following are the output
p1: 0.3773790986234757
p2: 0.14813186884048934
Result mean of last 25% run: 0.5126322688861615
```

After comparing the above outputs, we can conclude that $\epsilon = 0.001$ provides the best convergence to the optimal solution among the various choices.

3. SA Analysis

Assume the computing budget is N. $N = nk$, where n is the computing energy for each run and k is the total number of parallel runs. Specifically, n should be as small as possible and be able to let the output be stable around p^*.

However, since we seperate the calculation of standard deviation and expectation value in the objective function, the computing time grows significantly, consuming around 7 minutes for $n = 2000$. The relatively small k does not allow us to have enough samples for building confidence interval and SA analysis.

8 Conclusion

Our project aims to to find the optimal investment strategy for an investor to invest in companies over a given time period, which maximizes the expected return on investment while accounting for the associated risks (Sharpe Ratio).

First, we analyze the well-posedness of the problem. To optimize it, we use Martingale Noise Model to simulate the stochastic problem. Then, we analyze coercivity of $G(p)$. Practically, we use the SPSA algorithm and projection function to iteratively find the optimal p.

The algorithms work well to (weak/strong) converge to p^* and find a relatively high Sharpe Ratio. However, the algorithm is designed in a complex way, thus taking too much computing budget in single run, not allowing us to perform SA analysis.

For future work, this project's finding can be used in fields such as quantitative finance. But as the real-life financial data contain more noise, we need further study in how to model and optimize these problems to search for the best investment strategies.