SIAG/FME Code Quest 2023 Report
DeFi & RoboAdvising

Team: Finatics
Thu Nguyen Hai & Long Nguyen Ha Tuan

1 Introduction

The SIAG/FME Code Quest 2023 presents a platform for exploring the world of Decentralized Finance (DeFi) and RoboAdvising. Our team, Finatics, has embarked on this journey, focusing on the emulation of a Constant Product Market Maker (CPMM) using Python and developing strategies for effective liquidity supply in various pools.

The rest of the report is organized as follows. Section 2 discusses the method of stochastic gradient descent for optimization problem of CVaR. Section 3 showcases preliminary results from our numerical tests of our strategy. Finally, Section 4 concludes the report with some remarks.

2 Methodology

In the code quest, participants are presented with the challenge of optimizing the allocation of a predetermined initial wealth, denoted as x_0, across a specified number of investment pools n. This task is dictated by a set of parameters (k, p, σ, T) and additional details outlined in the accompanying file `params.py`. The objective is to strategically determine the proportion of x_0 to be invested in each pool to satisfy the performance criterion, which is intricately linked to the Conditional Value-at-Risk (CVaR) as a risk assessment metric. The performance measure, r_T, is expressed through a logarithmic function, quantifying the final wealth in relation to the initial investment.

Our approach to tackling this problem employs Stochastic Gradient Descent (SGD), adjusting investment allocations in accordance with the gradient of the loss function concerning CVaR, striking a pragmatic balance between risk management and return on investment. This strategic implementation is grounded in the methodologies advocated in [4]

2.1 Stochastic Gradient Descent for CVaR

We use gradient descent for minimizing CVaR with the following framework.

```
Algorithm: Stochastic Gradient Descent for CVaR
Require: Number of iterations $T$.
1. Initialize randomly $\theta_1 = (w_1, \dots, w_6) \in \mathcal{K}$
2. Set $\eta$ to a chosen learning rate value.
3. for $n = 1, ..., T$ do:
4.    Simulate 1000 returns $\{(r_T)_1, ..., (r_T)_{1000}\}$ based on $\theta_n$ and calculate CVaR
5.    Compute $g_n = \frac{\partial}{\partial \theta} \text{CVaR}(\theta_n)$.
6.    Update $\theta_{n+1} = \text{proj}_K(\theta_n - \eta g_n)$.
7. return $\theta_T$
8. Select $\theta$ to minimize CVaR that meets probability constraint
```

The algorithm iteratively optimizes the allocation of initial wealth across investment pools using Stochastic Gradient Descent (SGD) with respect to Conditional Value-at-Risk (CVaR).

Number of iterations T Here, T denotes the number of iterations the algorithm will run, serving as the temporal horizon over which the optimization is conducted. When suitable η is selected, we set 30 iterations to see the function converge.

Wealth distribution $\boldsymbol{\theta_n}$ The vector $\boldsymbol{\theta}_n$ represents the distribution of the initial wealth across six different pools. The feasible set $\mathcal{K}$ for $\boldsymbol{\theta}$ is defined by the following two conditions: the sum of w_i over d dimensions equals 1, i.e., $\sum_{i=1}^{6} w_i = 1$, and each w_i is non-negative, i.e., $w_i \geq 0$ for all i. We start with $\boldsymbol{\theta}_1$ by randomly allocating wealth to the pools following the uniform distribution. After updating $\boldsymbol{\theta_n}$ in step 6 of the algorithm, it is possible that the updated $\boldsymbol{\theta_n}$ may not lie within the feasible set $\mathcal{K}$. To address this, a projection method in [3] and [2] is implemented to project the updated $\boldsymbol{\theta_n}$ back onto the simplex defined by $\mathcal{K}$, thereby ensuring that the constraints are satisfied.

Learning Rate $\boldsymbol{\eta}$ The learning rate η is crucial in optimization algorithms like SGD, as it controls the size of the updates to the model's parameters during training. To empirically determine an appropriate bound for η, we start with values of 1, 10, 20. With $T = 10$ iterations, we assess the improvement in CVaR with each updated θ. Initial results guide a narrowed search for the optimal η within the interval of 1 to 10, optimizing the convergence towards the best θ and minimizing CVaR. We continue the process till suitable value of η is found. Here, we choose η is 3 after empirical test.

Gradient descent $\boldsymbol{g_n}$ Due to the fact that our CVaR is calculated based on simulations and it is impossible to take the derivatives of θ, we resort to using a numerical approximation method for the gradient descent process. This allows us to estimate the gradient of CVaR with respect to our decision variables. The numerical approximation for the gradient descent is given by the following formula:

$$g(\theta) = \nabla f(w_1, \dots, w_6) = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3}, \frac{\partial f}{\partial w_4}, \frac{\partial f}{\partial w_5}, \frac{\partial f}{\partial w_6} \right)$$

where

- $f(\cdot)$ is the objective function which in our case is the CVaR,
- $\frac{\partial f}{\partial w_i} \approx \frac{f(\theta+\epsilon_i) - f(\theta-\epsilon_i)}{2\epsilon_i}$,

- and ϵ_i is a small perturbation applied to w_i while keeping to the constraints $\sum_{i=1}^{6} w_i = 1$ and $w_i \geq 0$ for all i.

2.2 Generalization on Different Seeds

Building upon the framework established in the previous section, our focus shifts to addressing the stochastic nature of CPMM liquidity minting. The robustness of our model against variations in the simulation environment is of significant importance, especially considering the potential impact of overfitting to specific simulation seeds. To this end, let our loss function be defined as $\text{CVaR}(\theta; z) : \mathbb{R}^6 \to [0, 1]$, with **the additional parameter $\boldsymbol{z}$** referring to the trading events $(v_t, \mathit{event\_type}_t, \mathit{event\_direction}_t)$, simulated under the parameters $(\rho, \kappa, \sigma, T)$. For our experiments, $(\rho, \kappa, \sigma, T)$ are held constant at default values, with generalization achieved by varying z.

To ensure that our model's performance is not overly fitted to any particular seed, we employed a minibatch approach in our algorithm. Specifically, k distinct random seeds were utilized to generate a diverse set of samples $S = \{z_1, \ldots, z_k\}$. At each step of the gradient descent, a subset of b out of the k samples is randomly selected to estimate the gradient. This selection process effectively reduces the influence of any single seed on the overall estimation of θ, enhancing the robustness of the model. The primary objective remains the identification of initial weights θ_1 based on i.i.d samples from $\mathcal{K} \subseteq \mathbb{R}^6$, aimed at minimizing the expected CVaR, i.e., $E_z[\text{CVaR}(\theta_1; z)]$.

The Stochastic Gradient Descent algorithm is modified to be as follows:

```
Algorithm Stochastic Gradient Descent for CVaR with Minibatch
Require: Samples $S = \{z_1, ..., z_k\}$ based on the $k$ seeds, $z_0$ based on the default seed,
number of iterations $T$, and minibatch size $b$.
1. Initialize randomly $\theta_1 = (w_1, \ldots, w_6) \in \mathcal{K}$
2. Set $\eta$ to a chosen learning rate value.
3. for $n = 1, ..., T$ do
4.    Simulate 1000 returns $\{(r_T)_1, ..., (r_T)_{1000}\}$ based on $\boldsymbol{z_0}$ and $\theta_n$ and calculate CVaR
5.    for $i$ in $b$ random samples from $S$ do
6.       Compute $g_{n,i} = \frac{\partial}{\partial \theta} \text{CVaR}(\theta_n; z_i)$.
7.    return $g_n = \frac{1}{b} \sum_{i=1}^{b} g_{n,i}$
8.    Update $\theta_{n+1} = \text{proj}_K(\theta_n - \eta g_n)$.
9. return $\theta_T$
10. Repeat steps 1-9 to determine the optimal value of $\theta_1$
```

In this iteration of our analysis, we adhered to a structured approach consistent with the methodologies outlined previously. Specifically, we fixed the total number of iterations at 300 and the number of simulated returns in each iteration at 1000. All procedures including the estimation of the gradient and the projection of θ onto the simplex $\mathcal{K}$ also remain the same. By doing so we aim to facilitate comparative analysis while ensuring comprehensive exploration of the parameter space.

We employed a minibatch size of $b = 5$ and sampled z for the total of $k = 500$ seeds. This selection was determined on an empirical basis to strike a balance between minimizing computational costs and maintaining the algorithm's robustness against stochastic fluctuations.

While an ideal analysis would entail simulating 1000 returns across all 500 pools to accurately compute the Conditional Value at Risk (CVaR), because of limited computational resources we only tracked the CVaR for the default seed. We also made another simplification of ignoring the effect of θ_1 on trading actions z. Given the marginal impact of initial wealth allocations on the marginal prices (due to the relative insignificance of θ_1 to the initial reserves R^X, R^Y), we opted to pre-simulate and store the z values (independent of θ adjustments) to reduce having to re-simulate all 500 $z's$ for each new θ.

3 Results

We have identified the optimal allocations from Sections 2.1 and 2.2. We have named these allocations as follows:

- The optimal allocation from Section 2.1 is referred to as **SGD**.
- The optimal allocation from Section 2.2 is referred to as **SGD_Gen**.

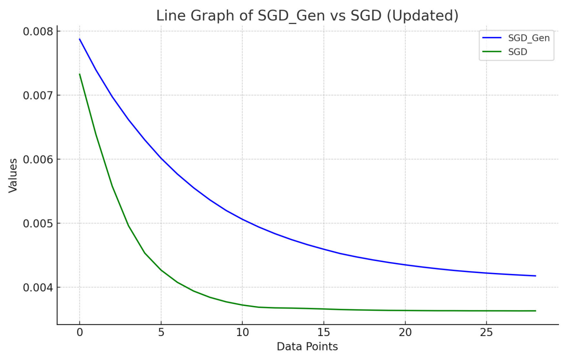
The specific optimal allocations for each strategy are as follows:

For **SGD**, the optimal allocation is:

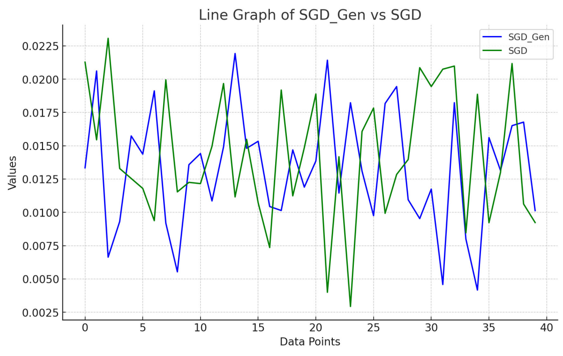
$$[0.125663, 0.295787, 0.190112, 0.158619, 0.229809, 0.000010] \quad (1)$$

For **SGD_Gen**, the optimal allocation is:

$$[0.151229, 0.217142, 0.194189, 0.219471, 0.191215, 0.026755] \quad (2)$$



(a) CVaR with 30 iterations on fixed seed



(b) Final CVaR on 300 seeds

Figure 1: Optimal allocations generated with SGD and SGD Generalization and tested with additional random seeds

For fixed seed in the competition, both optimal allocations satisfy the constraint of probability of log return greater than 0.05 must greater than 0.8, and **SGD** yields better result for minimizing CVaR of 0.0036. We also test both allocations to additional 300 seeds and the results are quite similar mean CVaR of 0.014 with **SGD** has lower volatility.

4 Remarks

In our strategy, the optimal theta displays a significantly low allocation of initial wealth to the final pool. This is primarily attributed to its notably higher volatility relative to other pools. In [1], it is mentioned that there is a direct correlation between the volatility of the rate Z and the PL impacts on a LP's position. They suggest that an effective liquidity provision strategy should dynamically adjust its spread based on the observed volatility. This insight substantiates our observation that the final pool receives almost negligible distribution due to its high volatility.

Regarding runtime issues, we advise against using the free version of Google Colab. Our experiments indicate that the runtime on Google Colab is approximately 1.5 times longer than when executed on local with VSCode, making it less efficient for our purposes.

References

[1] Álvaro Cartea, Fayçal Drissi, and Marcello Monga. "Predictable Losses of Liquidity Provision in Constant Function Markets and Concentrated Liquidity Markets". In: *Applied Mathematical Finance* 30.2 (2023), pp. 69–93. DOI: `10.1080/1350486X.2023.2277957`.

[2] Yunmei Chen and Xiaojing Ye. *Projection Onto A Simplex*. `https://arxiv.org/abs/1101.6081`. arXiv preprint arXiv:1101.6081. 2011.

[3] John Duchi et al. "Efficient projections onto the l1-ball for learning in high dimensions". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 272–279. DOI: `10.1145/1390156.1390191`.

[4] Takanori Soma and Yuichi Yoshida. *Statistical Learning with Conditional Value at Risk*. `https://arxiv.org/abs/2002.05826`. arXiv preprint arXiv:2002.05826. 2020.