SIAG/FME Code Quest 2023

Daniele Maria Di Nosse[1] and Federico Gatta[1]

[1]Scuola Normale Superiore, Pisa, Italy

January 31, 2024

1 Introduction

Nowadays, Decentralized Finance and Crypto markets are hot topics in the financial world. Every day, billions of dollars are traded in decentralized exchanges [DefiLlama, 2023] whose behaviour is driven by special programs called smart contracts. A smart contract is a set of immutable instructions that dictate how crypto assets are handled in the exchange and how the price is formed. This dynamic totally excludes the need for an intermediary and a market maker, resulting in the so-called Automated Market Markers (AMM). AMMs are made up of several pools that act in a decentralized manner. Each pool is characterized by two or more coins. The relative amounts of these coins determine the price of them. The two main actors are Liquidity Providers (LPs) and Liquidity Takers (LTs). The former provides liquidity to the pool by adding coins and the latter takes the liquidity by swapping the coins. The LPs are paid for their service with a share of the pool represented by Liquidity Providing coins (LP coins), while LTs pay the service with a fee.

The challenge is divided into two different tasks: completing the provided amm class and optimizing a liquidity provision strategy over a certain number of two-coin pools. Specifically, by assuming the point of view of an LP operating in these crypto pools, the second point of the challenge aims to design a liquidity provision strategy that minimizes the investment risk represented by the Conditional Value at Risk (CVaR) while keeping most of the return distribution mass above a certain threshold. Despite this goal can be naively achieved with time-consuming methodologies such as grid search or randomized search approach, we propose a two-step strategy that both dramatically reduces the computational demand and provides accurate solutions. That is, we first approximate the target function with a Kernel Ridge Regression (KRR), and then the minimum of the KRR is used as the starting point for the Sequential Least Squares Programming (SLSQP) minimization.

The following of this report is organized as follows. Section 2 formalizes the problem. Section 3 describes our approach and shows its advantages. Section 4 draws the conclusions.

2 The Problem

The challenge's task is the minimization of the CVaR_α for a certain random variable at a fixed level α. Let r_T be the realization of the random variable representing the investment (log) return at time T. Thus, $-r_T$ embodies the loss. So far, the most used risk measure has been the Value

at Risk (VaR), which is defined as the loss distribution quantile. That is, given a confidence level $\alpha \in (0, 1)$ (usually $\alpha \approx 0.9$):

$$\mathrm{VaR}_\alpha(r_T) = -\inf\left[z \in \mathbb{R} | \mathbb{P}(r_T \leq z) \leq 1 - \alpha\right]. \tag{1}$$

Nowadays, the attention is shifting from VaR to the CVaR, which is defined as the mean loss beyond the α-quantile of the distribution, that is:

$$\mathrm{CVaR}_\alpha(r_T) = \frac{1}{1-\alpha}\int_\alpha^1 \mathrm{VaR}_s(r_T)ds \approx \mathbb{E}[-r_T | - r_T \geq VaR_\alpha] \tag{2}$$

In our context, the trading environment is characterized by the tuple (k, p, σ, T) that describes the frequency of arrival, the type probability, and the volume volatility of the orders, as well as the time horizon. Let's consider the total initial wealth of the LP as x_0, denominated in coin-X. The LP puts a fraction of his wealth on each of the n pools. The portfolio weights are in the admissible set

$$\mathcal{S} := \left\{ \boldsymbol{\theta} = (\theta_1 \cdots \theta_n) \in [0, 1]^n \ s.t. \ \sum_{j=1}^{n} \theta_j = 1 \right\}$$

The choice of the portfolio weights $\boldsymbol{\theta}$ affects both the number of LP coins generated, the pool state vector and, as a consequence, the final return of the strategy r together with the CVaR_α. In order to stress the dependence of the objective function 2 from the weight vector $\boldsymbol{\theta}$ through the performance criterion r_T, we will write $\mathrm{CVaR}_\alpha(\boldsymbol{\theta})$ instead of $\mathrm{CVaR}_\alpha(r_T)$. Thus, for the challenge to be completed, we have to find the optimal weights vector $\hat{\boldsymbol{\theta}}$ such that:

$$\min_{\boldsymbol{\theta} \in \mathcal{S}} \mathrm{CVaR}_\alpha(\boldsymbol{\theta}) = \hat{\boldsymbol{\theta}} \tag{3}$$

subject to the following constraint:

$$\mathbb{P}\left[r_T > \xi\right] > q \tag{4}$$

The values of α, q, ξ, as for the trading environment (k, p, σ, T), can be found in the parmas.py file provided by the challenge organizers. To summarize, we face up a constrained minimization problem, with both equality and inequality constraints, linear and non-linear. However, we have found that the non-linear constraint 4 is automatically satisfied if we approximate the argmin for CVaR_α. Thus, in the following, we look at it only in the evaluation of the results and not even during the computation.

3 Our Approach

Estimation of the CVaR_α function is extremely high-demanding. The heaviest part is in the simulation of the paths in the pools. To mitigate this issue, we propose to introduce the use of Cython in the optimization procedure. Cython [Behnel et al., 2010] is a Python library that allows developers to write Python code that can be compiled into native C extensions. It combines the flexibility of the former with the efficiency of the latter. Cython is also compatible with most Python libraries and frameworks, making it easy to integrate with existing Python projects. Nevertheless, its drawback is the different management of the numpy internal random numbers generator, which can bias the project evaluation due to the different evolution scenarios. To solve this issue and to further reduce the simulations' complexity, we generate all the random numbers at once before

using Cython. That is, we generate and store the arrays used to determine the orders arrivals, types, directions and volumes via the method **save_random_numbers** of the amm class. Then, when using Cython, we just load these arrays. The modified simulation engine is coded in the **simulate** module in the amm_cython.pyx file.
Despite there is the expression 2 for the target function $\text{CVaR}_\alpha(\boldsymbol{\theta})$, its direct optimization would be extremely expensive, as every iteration is highly demanding. In this situation, our idea is to divide the procedure into two steps: firstly we minimize an approximation $f(\boldsymbol{\theta})$ of the target function. In this way, we aim to approach the minimum $\hat{\boldsymbol{\theta}}$ to obtain a good starting point for the second step in which we use the SLSQP routine on expression 2. The first task is usually achieved with a neural network, such as in [Büchel et al., 2022]. However, the critical point is that generating the dataset for training the network could be as expensive as directly optimising $\text{CVaR}_\alpha(\boldsymbol{\theta})$, leading to no significant advantages in terms of computational time. So, we have used linear models, as, if well-specified, they require a small-size dataset to obtain a good fit.

3.1 Kernel Ridge Regression

Kernel Ridge Regression (KRR) [Vovk, 2013] is a linear regression approach, where the input variables are handled by a kernel K. Given a dataset $\{\left(\boldsymbol{\theta}^{(i)}, CVaR_\alpha^{(i)}\right)\}_{i=1}^N$, where each $\left(\boldsymbol{\theta}^{(i)}, CVaR_\alpha^{(i)}\right)$ is a randomly generated pair of portfolio weights and CVaR_α values, the approximation function is:

$$f(\boldsymbol{\theta}) = \sum_{i=1}^{N} \alpha_i K(\boldsymbol{\theta}, \boldsymbol{\theta}^{(i)}) \qquad K(\boldsymbol{\theta}, \boldsymbol{\theta}^{(i)}) = -\sum_{j=1}^{n} \frac{\left(\theta_j - \theta_j^{(i)}\right)^2}{\theta_j + \theta_j^{(i)}}$$

where we have used the adjusted χ^2 kernel. The loss function is the same as in the ridge regression, that is mean square error with L_2 penalization on the coefficients. There is a closed form for fitting the model, so it requires a small computational time. We exploit the scikit learn implementation: **sklearn.kernel_ridge.KernelRidge**.
In this step, the key point is to determine the dataset size N, as its generation can be extremely hard. However, we have found that even a small N leads to accurate results, with a very high R^2 determination coefficient. In particular, $N = 10$ has been used. The resulting model has $R^2 \approx 0.995$. Furthermore, it is interesting to observe that the minimum CVaR_α in the KRR training set is -0.46%, while the KRR minimization output is -0.37%. This shows that KRR is really able to learn useful patterns and to efficiently approximate the CVaR_α function also in unseen points, and it is not merely copying the input data. Finally, it is noteworthy to observe that the dataset creation can be easily parallelized to fully exploit the computational power. We accomplished this by using the built-in Python package **multiprocessing** and dividing the task over 2 processes (maximum number provided by Google Colab's free plan).

3.2 Sequential Least Square Programming

The Sequential Least Squares Programming (SLSQP) algorithm is widely used for solving nonlinear optimization problems. It has been chosen due to its flexibility, as it can handle both equality and inequality constraints, as well as non-convex objective functions. SLSQP is a type of Sequential Quadratic Programming (SQP) method, where the minimum is found by iterations [Wright, 2006]. At each step, the original problem is approximated by a quadratic programming one, which is solved via a quasi-Newton method. So, the formulation at step k is:

$$\min_d \; CVaR_\alpha(\boldsymbol{\theta}^{(k)}) + \nabla CVaR_\alpha(\boldsymbol{\theta}^{(k)})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2_{\theta\theta} \mathcal{L}\left(\boldsymbol{\theta}^{(k)}, \boldsymbol{\lambda}^{(k)}, \sigma^{(k)}\right) \boldsymbol{d}$$

where $\mathcal{L}$ is the Lagrangian function associated to the problem and λ and σ are the Lagrange multipliers:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma) = CVaR_\alpha(\boldsymbol{\theta}) - \boldsymbol{\lambda}^T\boldsymbol{\theta} - \sigma\left(1 - \mathbf{1}^T\boldsymbol{\theta}\right)$$

and the constraints can be reformulated as:

$$\boldsymbol{\theta}^{(k)} + I_n\boldsymbol{d} \geq \mathbf{0} \quad \& \quad \left(1 - \mathbf{1}^T\boldsymbol{\theta}^{(k)}\right) - \mathbf{1}^T\boldsymbol{d} = 0$$

where I_n is the identity matrix in $\mathbb{R}^{n\times n}$, and $\mathbf{0}, \mathbf{1}$ are the zero and one vectors in $\mathbb{R}^n$.

3.3 Results

To assess the goodness of our proposal, we have compared it with two naive minimization approaches, namely the randomized Grid Search (carried out by iterating over 10000 random seeds) and the SLSQP (without the KRR step, that is with $\boldsymbol{\theta}^{(0)}$ initialized to the equal-weighted portfolio). Furthermore, also the partial results obtained by only using the KRR are shown. The comparison is summarized in Table 1.

	Grid Search	**KRR**	**SLSQP**	**KRR+SLSQP**
$\mathbb{P}[r_T > \xi]$	84.50%	84.20%	84.50%	**84.60%**
$\mathbb{E}[r_T]$	16.0985%	**16.1744%**	16.1306%	16.1250%
VaR_α	3.1832%	3.2228%	3.2196%	**3.2237%**
CVaR_α	-0.3693%	-0.3731%	-0.3632%	**-0.3627%**
$\hat{\boldsymbol{\theta}}$	[0.11057, 0.34389, 0.17021, 0.13990, 0.22973, 0.00567]	[0.14408, 0.29763, 0.17649, 0.16160, 0.19700, 0.02320]	[0.12360, 0.29529, 0.19724, 0.15761, 0.2262, 0.00001]	[**0.12848, 0.29556, 0.18971, 0.15680, 0.22943, 0.00001**]

Table 1: Comparison of results obtained with the different approaches.

As shown, the optimal vectors $\hat{\boldsymbol{\theta}}$ obtained by all the methods are very similar to each other. However, our strategy slightly outperforms the competitors in terms of the performance metric. In particular, the KRR performs worse, and the optimal $CVaR_\alpha$ it finds is 2.7% lower than the KRR+SLSQP. This shows the utility of the SLSQP step in our procedure: without it, the computational times would be lower, but also the solution accuracy would be damaged. Finally, the histogram of the returns obtained with the optimal portfolio of our strategy is shown in Figure 1.

3.4 Computational Times

In this subsection, we provide a deeper insight into our approach by discussing the computational times. All the tests are run on Google Colab (free plan). As shown in the previous subsection, the ultimate result found by KRR + SLSQP outperforms the Grid Search, the SLSQP and the KRR. Firstly, we provide an insight into our procedure in Table 2. We show the mean times over 10 iterations. Each step of the procedure is individually analyzed.

Compile Cython	**Random Numbers**	**KRR Dataset and Fit**	**KRR Min**	**SLSQP Min**
7.571	4.898	37.475	0.060	171.178

Table 2: Insight on the computational times of our approach. The times shown are the mean over 10 iterations expressed in seconds.

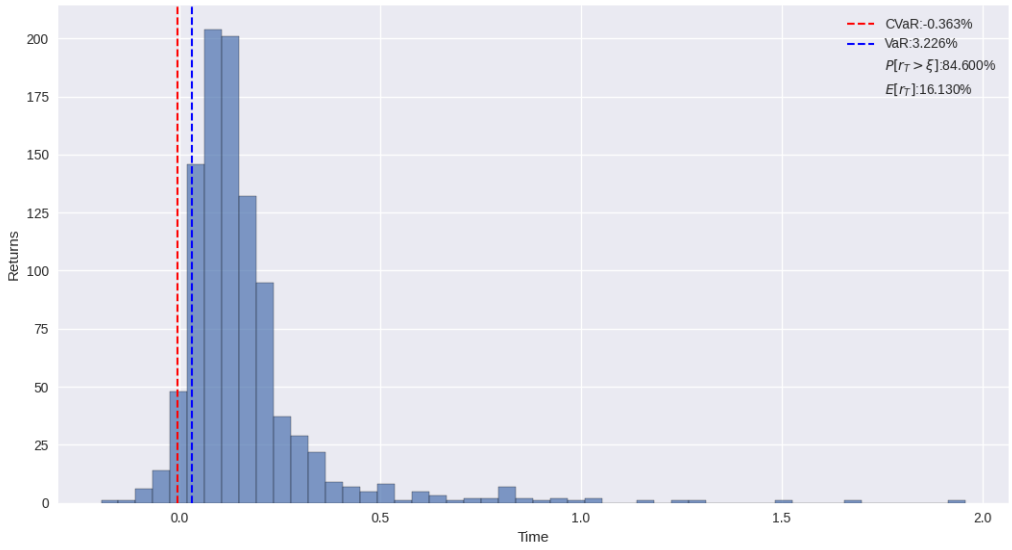

Figure 1: Distribution of r_T under the optimal value $\hat{\boldsymbol{\theta}}$ of the weight vector obtained with the KRR+SLSQP approach. The vertical red line corresponds to $\text{CVaR}_\alpha(\hat{\boldsymbol{\theta}})$, while the blue one to $\text{VaR}_\alpha(\hat{\boldsymbol{\theta}})$.

As shown, despite the KRR providing a starting point close to the ultimate minimum, most of the effort is still put into direct CVaR_α minimization via SLSQP. Furthermore, even if the last step's computational time is smaller than the SLSQP with random initialization, the total time of the two steps is bigger. Table 3 summarizes the computational times of the different approaches.

Grid Search	**KRR**	**SLSQP**	**KRR+SLSQP**
37345	50	200	221

Table 3: Computational times comparison. Each time is obtained as the mean over 10 iterations.

It is noteworthy to observe that the price paid in terms of computational time is remunerated in terms of a lower CVaR_α.
Lastly, we discuss the importance of the Cython implementation. As shown in Table 2, it requires about 12 seconds to compile and generate random numbers. However, this time is well-spent. Indeed, as shown in Table 4, there is a noteworthy advantage both in using Cython and in simulating all the random numbers at once.

	Cython	Standard	Standard*
Time (sec)	207	260	526

Table 4: Average computational time (10 runs) required for the Cython and standard implementations. The * refers to the original simulation method with repetitive sampling.

3.5 Other Attempts

Despite the main approach with KRR and SLSQP, we have also tried other ways to minimize the CVaR_α function. In particular, inspired by [Krokhmal et al., 2003], we attempt to rephrase the

problem by directly minimizing losses while constraining the CVaR_α. Theorem 3 in [Krokhmal et al., 2003] demonstrates that swapping the CVaR_α with the expected loss function and changing the constraint on expected reward with a constraint on the CVaR_α yields the same efficient frontier as long as the CVaR_α, the expected loss function, and the set of all possible weights are convex. Although we did not discover a straightforward analytical way to verify these conditions, we proceeded with the alternative formulation. However, the results were slightly worse, so we switched back to the original setting.

4 Conclusion

Our approach to the challenge is based on two steps. The first one mitigates the evaluation complexity by approximating the target function with KRR. The second exploits SLSQP to refine the approximated solution. The result obtained outperforms the Grid Search and SLSQP without the initial guess provided by KRR. Furthermore, also the computational time is reasonable. A possible improvement of our approach could be using only the KRR. In this way, by reducing the accuracy of the solution by less than 3%, the times would decrease by 75%. However, we have kept the SLSQP step as it provides a lower CVaR_α value, which is the ultimate task of this challenge.

References

[Behnel et al., 2010] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2010). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39.

[Büchel et al., 2022] Büchel, P., Kratochwil, M., Nagl, M., and Rösch, D. (2022). Deep calibration of financial models: turning theory into practice. *Review of Derivatives Research*, pages 1–28.

[DefiLlama, 2023] DefiLlama (2023). https://defillama.com.

[Krokhmal et al., 2003] Krokhmal, P., Palmquist, J., and Uryasev, S. (2003). Portfolio optimization with conditional value-at-risk objective and constraints. *Journal of Risk*, 4.

[Vovk, 2013] Vovk, V. (2013). Kernel ridge regression. In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 105–116. Springer.

[Wright, 2006] Wright, S. J. (2006). *Numerical optimization*.