Solutions to the C11 Labs Research Lead Assignments

Luca Nicoli

November 24, 2024

Challenge 2: Long Tail Alpha

The contract at the address provided is the *Autonomous Converter Contract* (ACC), which is an integral part of the Metronome 1.0 ecosystem. ACC provides the mechanism for converting between the Metronome token (MET) and Ether (ETH) at market-driven rates. This contract works as a Liquidity Pool (LP) maintaining liquidity and price stability by adjusting exchange rates based on the system's relative supply of MET and ETH.

Overview of Metronome 1.0

Metronome (MET) is a cryptocurrency designed for longevity, self-governance, and cross-blockchain portability. It operates via autonomous smart contracts that handle all token issuance, governance, and transactions. Its token supply begins with an initial auction distributing 10 million MET, of which 20% is retained by founders under a gradual release schedule, and 80% is sold through a Descending Price Auction. Subsequent Daily Price Auctions (DPA) mint new tokens at the greater of 2,880 MET per day or 2% of the existing supply annually. Proceeds from all auctions are directed to a Proceeds Contract (ACC), which funds an Autonomous Converter Contract enabling MET-ETH conversions at market-driven rates, mitigating slippage via a dynamic supply balance. Portability between blockchains is enabled through export-import mechanisms that burn tokens on the source chain and mint them on the target, with validators ensuring supply integrity and security. This structure ensures decentralization, predictable token economics, and resilience against governance disputes, aiming to create a stable, enduring currency ecosystem.

Flow of the Metronome Protocol

Metronome operates through four autonomous smart contracts, each with a distinct role in ensuring the system's decentralization, predictability, and functionality:

1. Token Contract

Purpose: Implements the MET cryptocurrency as an ERC-20 token with additional functionalities for enhanced security and cross-blockchain portability.
Functions:

- Handles token balances, transfers, and approvals.
- Supports batch transfers (`multiTransfer`) and enables custom porters for blockchain migration.
- Manages minting and burning of tokens during auctions and blockchain export/import processes.

2. Auctions Contract

Purpose: Manages token distribution through descending price auctions (DPAs).
Functions:

- Executes the *Initial Supply Auction*, distributing 80% of the initial MET supply.
- Conducts *Daily Supply Lots*, minting new tokens at a predictable rate (2,880 MET/day or 2% of the existing supply annually, whichever is greater).
- Sets auction parameters such as start time, initial price, price decrement rate, and floor price.
- Sends 100% of auction proceeds to the Proceeds Contract.

3. Proceeds Contract

Purpose: Stores auction proceeds and ensures liquidity for MET-ETH conversions.
Functions:

- Retains ETH collected from auctions, ensuring all proceeds remain decentralized and on-chain.
- Transfers 0.25% of its total balance daily to the Autonomous Converter Contract.
- Provides a buffer to smooth out fluctuations in daily auction proceeds.

4. Autonomous Converter Contract

Purpose: Facilitates MET-to-ETH and ETH-to-MET conversions at market-driven rates using dynamic supply balancing.
Functions:

- Maintains MET and ETH reserves to ensure continuous liquidity.
- Adjusts prices automatically based on reserve balances.
- Serves as a fallback marketplace for MET outside of auctions.
- Enables arbitrage opportunities to stabilize MET prices relative to ETH.

Arbitrage Opportunity

Arbitrage in Metronome arises from the price differential between the daily descending price auction (DPA) and the Autonomous Converter Contract (ACC). Specifically:

- If the price of MET in the DPA is lower than the equivalent price in the ACC (as determined by the MET/ETH ratio in the ACC), an arbitrage opportunity exists.
- You can buy MET in the DPA at a lower cost, then sell it in the ACC at a higher price to obtain ETH, profiting from the price difference.

This mechanism relies on the predictable and transparent pricing structure of the auction and the dynamic market-driven rates in the ACC. Arbitrageurs play a crucial role in stabilizing prices across these platforms, as their activity balances supply and demand discrepancies. To understand how to catch the arbitrage opportunity, we first need to define what we mean by arbitrage. Let us assume that the initial amount of ETH is e_0 and MET m_0 and that after certain operations, we end up with our final amounts e_1 and m_1. For simplicity, we assume $m_0 = m_1 = 0$. Then we want:

$$e_1 > e_0 \tag{1}$$

Meaning that under the condition of zero MET asset variation, we obtained a net profit $e_1 - e_0 > 0$. If we define the *Price function* $\mathbb{P}^x$ the amount of token y received for a unit of token x, then assuming that we are selling ETH (e_d) for MET (m_d) from the DPA at price $\mathbb{P}_D^{met}$ and selling all these METs to the ACC for ETh e_a at price $\mathbb{P}_A^{eth}$, ignoring all transaction fees we have that:

$$m_d = \mathbb{P}_D^{met} e_d \tag{2}$$

$$e_a = \mathbb{P}_A^{eth} m_d \tag{3}$$

Working the equations out, we have that:

$$\frac{e_a}{e_d} = \mathbb{P}_A^{eth} \mathbb{P}_D^{met} \tag{4}$$

And since we require $\frac{e_a}{e_d} > 1$, then we need that

$$\mathbb{P}_A^{eth} > \frac{1}{\mathbb{P}_D^{met}} = \mathbb{P}_D^{eth} \tag{5}$$

When this condition is met, then there is an arbitrage opportunity.
Let us now get a closer look at the price functions (See Appendix for the derivation of the price function of the ACC):

$$\mathbb{P}_D^{eth}(t) = \mathbb{P}_{D,0}^{eth} \cdot (0.99)^{\frac{t}{60}} \tag{6}$$

$$\mathbb{P}_A^{eth}(m_d) = \frac{E_{A,0}}{M_{A,0} + 2m_d} \tag{7}$$

Where $\mathbb{P}_{D,0}^{eth}$ is the initial ETH price in the Daily auction, $M_{A,0}$, and $E_{A,0}$ are the initial MET and ETH amounts in the ACC respectively, and m_d is the amount of MET sold to the ACC. The initial ETH price of the daily auction is twice the previous auction closing price. If zero MET is sold in the previous DPA, the price of the following day's DPA will begin at 1/100th of the last price at which MET was purchased.In Fig. 1 we report the two price functions (Eq. 6 – left, and Eq. 7 – right).

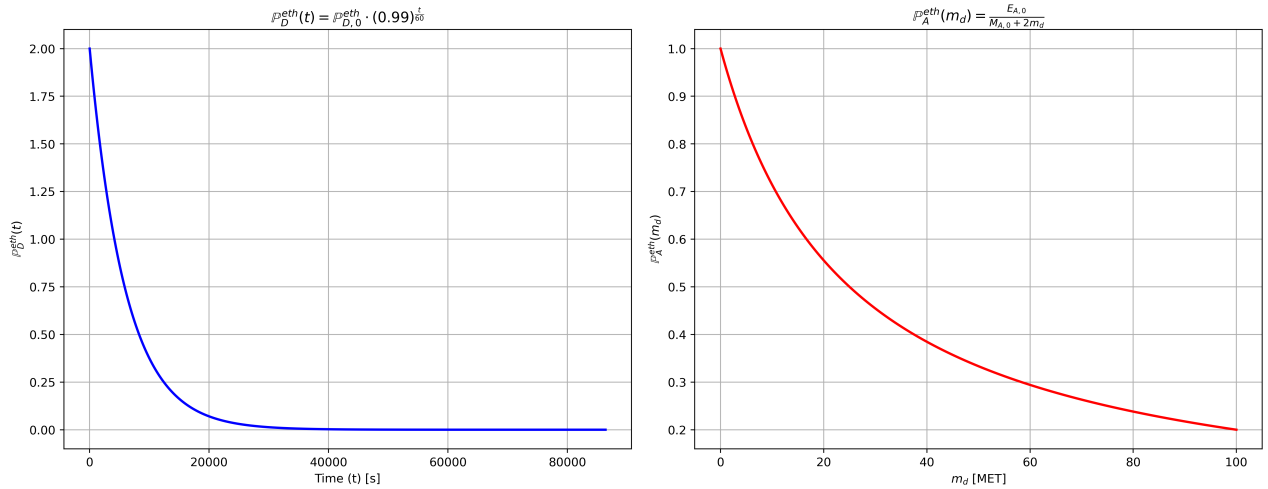

Figure 1: (left) DPA $\mathbb{P}_D^{eth}(t)$ price as a function of time throughout the day. (right) ACC price $\mathbb{P}_A^{eth}(m_d)$ as a function of the MET bought at the DAC and sold to the pool m_d.

To leverage the arbitrage, we can substitute $\mathbb{P}_D^{eth}$, and $\mathbb{P}_A^{eth}$ into Eq. 5:

$$\frac{E_{A,0}}{M_{A,0} + 2m_d} > \mathbb{P}_{D,0}^{eth} \cdot (0.99)^{\frac{t}{60}} \tag{8}$$

This leads to the following arbitrage condition for m_d:

$$\frac{1}{2}\left[\frac{E_{A,0}}{\mathbb{P}_{D,0}^{eth}}(0.99)^{\frac{-t}{60}} - M_{A,0} - \epsilon\right] \geq m_d \geq \epsilon \tag{9}$$

Where $\epsilon = 10^{-18}$ is the minimum possible amount of MET considerable. While $M_{A,0} + \epsilon$ is a constant, the other term on the left-hand side grows in time. Thus, there might exist a minimum time for which the condition expressed by Eq. 9 holds, and thus, we can make arbitrage.
More formally:

$$\begin{aligned} &\text{if } \exists\ t^* \in [0, T] \text{ s.t. } \frac{E_{A,0}}{\mathbb{P}_{D,0}^{eth}}(0.99)^{\frac{-t^*}{60}} = M_{A,0} + \epsilon, \text{ then} \\ &\forall t \in [t^*, T]\ \exists\ m_d^* \geq \epsilon, \text{ s.t. } \forall m_d \in [\epsilon, m_d^*], \text{ then } \frac{e_a}{e_d} > 1 \text{ holds} \end{aligned} \tag{10}$$

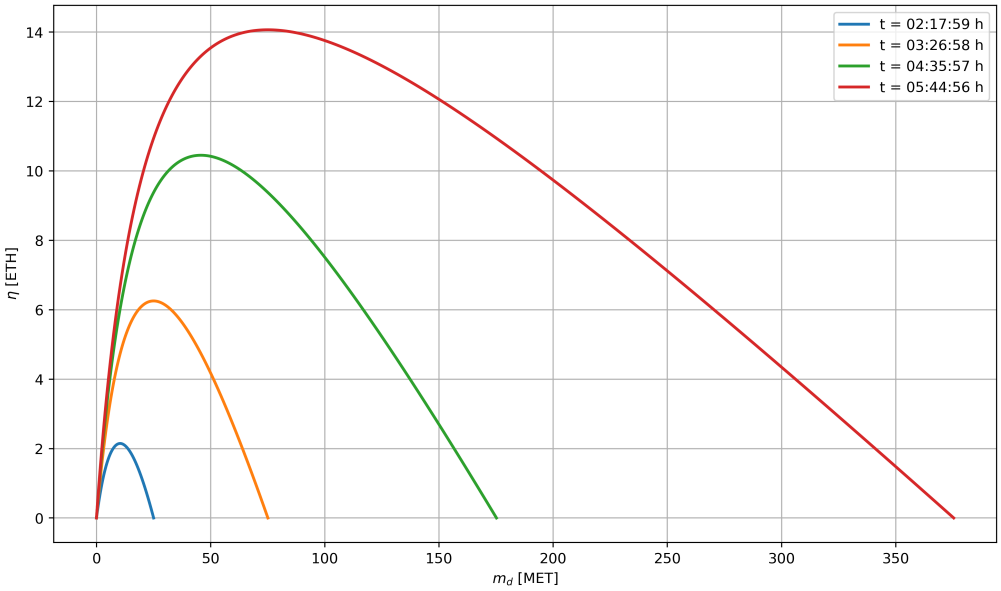

Figure 2: Profit η as a function of the amount of the amount MET (m_d) bought during the DPA for different times $t \in [t^*, T]$. For this example, we employed $M_{A,0} = 50$, $D_{A,0} = 50$, $P^{eth}_{D,0} = 2$, and therefore the computed $t^* \approx$ 01:08:59 hours for which conditions in Eq. 10 hold.

Where T is the DPA duration, i.e. 24 hours. The condition expressed in the first line of Eq. 10 can be easily checked at the beginning of each day, i.e. when we know the values of:

1. $M_{A,0}$, and $E_{A,0}$. The latter will increase at the beginning of each day due to the injection in the ACC of 0.25% of the total accumulated balance of the Proceeds Contract.

2. $P^{eth}_{D,0}$ the DPA's starting price.

If the condition is met at time t^* then for each subsequent time we can perform arbitrage. Nevertheless, we are still facing another problem. Indeed, when we make arbitrage we'd like to maximize our profit ($\eta = [e_a - e_d]$).

$$\max_{m_d \in [\epsilon, m_d^*]} \eta = \max_{m_d \in [\epsilon, m_d^*]} \left[\mathbb{P}^{eth}_A m_d - \mathbb{P}^{eth}_D m_d \right] = \max_{m_d \in [\epsilon, m_d^*]} \left[\frac{E_{A,0} m_d}{M_{A,0} + 2m_d} - m_d P_{D,0} (0.99)^{\frac{t}{60}} \right] \tag{11}$$

In Fig. 2 we report the shape of the profit η as a function of the amount of MET m_d bought during the DPA for different times $t \in [t^*, T]$. As it can be noticed, the profit function η has a maximum in the window $m_d \in [\epsilon, m_d^*]$, and this maximum grows in time. In Fig. 3 is depicted the maximum of the profit ($\max \eta$) and its derivative as a function of the time $t \in [t^*, T]$.
It is interesting to notice that the maximum profit converges to a maximum value by the end of the day. However, such a high-profit condition is unlikely to happen since we are in a strongly adversarial environment, and plenty of competitors will try to leverage the arbitrage opportunity as we do. Most probably, the arbitrage opportunity is attacked before the maximum profit has the inflection (the maximum of its derivative) since, after that point, waiting time will reduce the increase in profit per unit time whilst the probability of an adversarial trader leveraging the arbitrage opportunity increases. In addition to this, we also need to consider that the reservoirs in the ACC might change during the day due to others trading (and thus, we need to keep our conditions updated), but also we need to consider that to leverage higher profits we need to buy more MET (as it can be seen from Fig. 2). However, only a limited amount of MET can be acquired during the day. In particular, the amount of tokens in each DPA is the greatest between (i) 2,880 MET per day or (ii) an annual rate

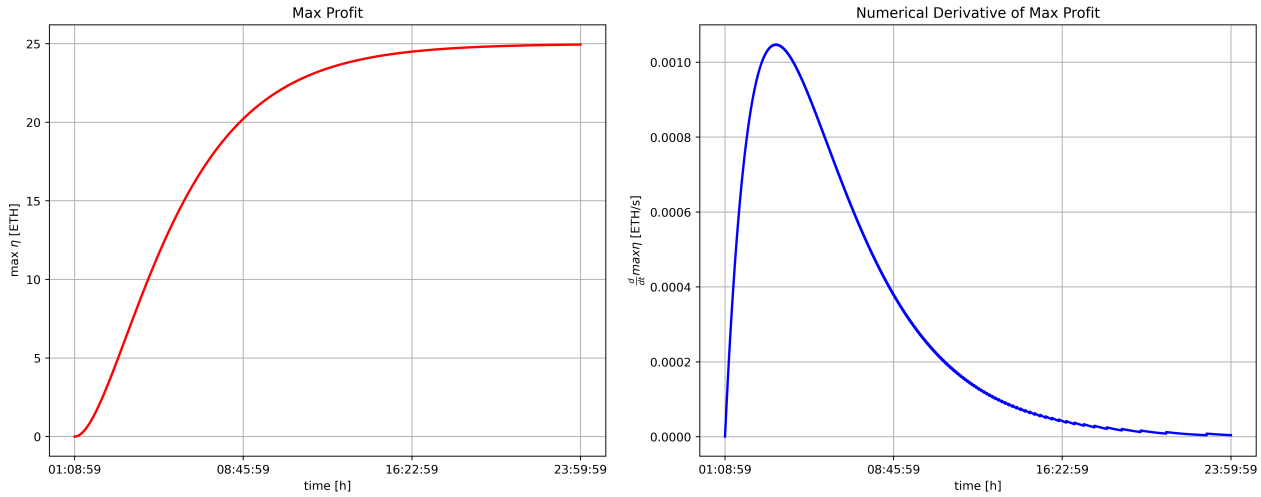

Figure 3: Maximum of the profit function (left – $\max_{m_d} \eta$) and its time derivative (right – $\frac{d}{dt} \max_{m_d} \eta$) of as a function of the time $t \in [t^*, T]$. For this example, we employed $M_{A,0} = 50$, $D_{A,0} = 50$, $P^{eth}_{D,0} = 2$, and therefore the computed $t^* \approx 01:08:59$ hours for which conditions in Eq. 10 hold.

equal to 2.0000% of the then-outstanding supply per year.
Therefore, under the conditions that the ACC liquidity pool has not changed its reservoirs and that there are still enough MET to be bought in the DPA such that we can leverage our maximum profit, the more we wait after the time t^*, the higher our profit but the lower our chance to catch it. Techniques like real-time data monitoring, game-theoretic models, and adaptive risk management could be employed to optimize the timing and execution of arbitrage trades. To develop an effective quantitative strategy, we should (i) model the likelihood of other users purchasing MET during the DPA and (ii) model the probability that these users will attempt to leverage the arbitrage opportunity. With these quantities or reasonable approximations thereof, we can identify the "sweet spot" where we can maximize our profit while also increasing the likelihood of successfully executing the arbitrage trade before competitors, ensuring that there is still enough MET available in the DPA to make this move. Achieving this balance is a complex task, as it requires continuously adapting to dynamic market conditions and the actions of other traders.
However, I feel that all these considerations point towards the idea that *"the quicker the better"* to maximize our chance of effectively leveraging this arbitrage opportunity. One could keep on making small trades right after the time $t*$. These trades should be packed in an atomic bundle, so only if all the trades can be executed can we effectively perform the arbitrage. We should keep an eye on the mempool to see if other MEV searchers are trying to leverage the arbitrage opportunity, although, with a high probability, they might also use private atomic bundles, sending them directly to the block-builders and thus skipping the public mempool. One should:

1. Given the initial reserves in the ACC compute t^* and the corresponding t_{max} such that the profit $\max_{m_d} \eta(t^*) > c \cdot (1 + f)$, where $c > 0$ is a variable that accounts for the amount of profit more than the transaction fee (f) we are willing to get;

2. At the time t_{max} we perform the transactions exchanging the computed m_d, thus perturbing the reserves in the ACC.

3. Iterate with the new reserves in the ACC

In Fig. 4, I am plotting the results of such a simulated strategy over a time window of 1 min after the first t^*. I have set $c = 50\%$, and 50 ETH and 40 MET as the initial reserves in the ACC. The timestep to find the optimal t_{max} is set to 0.25 s, although it's not optimal (probably if it would

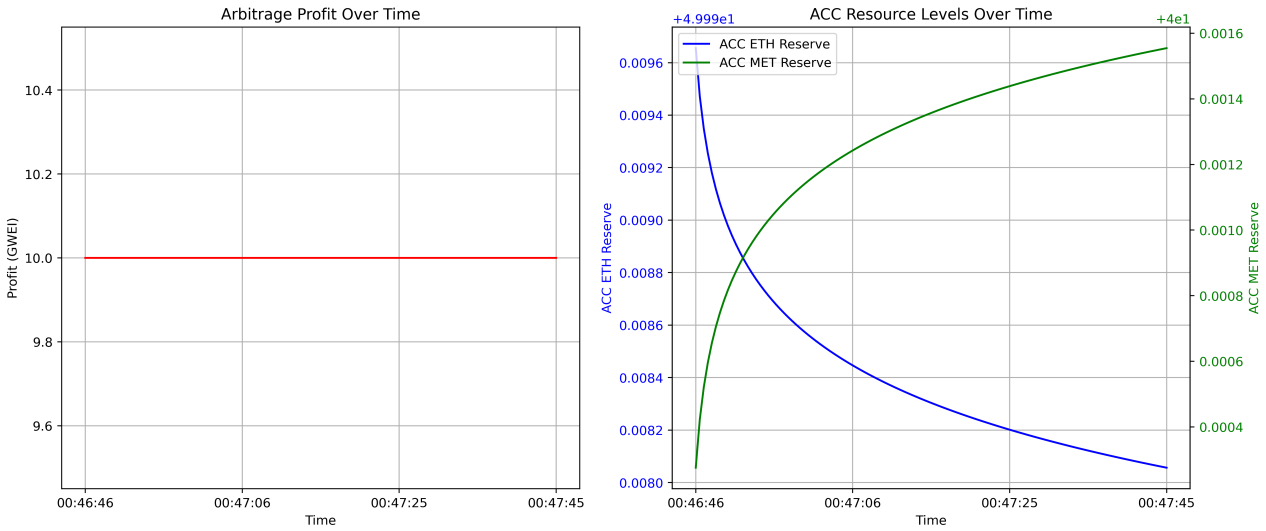

Figure 4: (left) Net profit of the strategy in GWEI as a function of time. (right) ACC reserves in time.

be possible, it would be better to perform trades at higher frequency). The fee is set to $f = 20$ GWEI. Nevertheless, we can still see that the net profit, i.e., $\eta - f$ [GWEI], is constant. The code implemented to simulate this strategy is simple and should be optimized and numerically stabilized. In particular, the time at which we perform the trades is set to be precisely the time for which the net profit is $\max_{m_d} \eta(t^*) = c \cdot (1 + f)$. This might be challenging due to numerical precision, and thus, in order to avoid weird fluctuations in the profit, I had to hardcode its value and smooth the curve out. You can find more details about this in appendix. However, being this a proof of concept, I believe this can already give a hint of the results of the strategy.

Simultaneously, we might also pursue a more aggressive approach. We should keep on performing our atomic trades each time while looking to the mempool for possible trades with the ACC. These trades can be sandwiched to reach two different goals: (i) earn ETH from the sandwich and (ii) apply a veto to large interactions with the ACC. Indeed, if we are capable of spotting a transaction with the ACC that would not perturb the ETH/MET distribution too much, i.e. we can keep on with our DPA-ACC arbitrage, then we could still profit from this transaction by sandwiching it with small transactions. Most users might have set a slippage tolerance limit on their transaction Tx. Thus, if we sandwich Tx with small transactions, then there is a high chance of Tx landing and us profiting. If on the other hand, we spot transactions with the ACC that aim at balancing out the ETH/MET spread in the ACC and thus reducing our time-window for the arbitrage $[t^*, T]$, then it might be profitable to perform a sandwich attack with large transactions, thus inducing a large slippage and most likely resulting in the abortion of the user's transaction. Of course, in this case, we need to be very careful since the cost might outweigh the benefits. All these considerations are valid, considering the price we would need to pay to create a private atomic transaction bundle and the price needed to have our bundle accepted by block-builders.

Challenge 3: DeFi Math

We are given a DeFi pool with two assets: USDC and ETH. The liquidity level of the pool is governed by the invariant function:

$$L(x, y) = x^2y + y^2x \tag{12}$$

where:

- x represents the USDC reserves,
- y represents the ETH reserves,
- $L(x, y)$ is the liquidity level of the pool.

Task A: Determining Reserves

We are provided with the following data:

- The initial liquidity level $L_0 = 10$,
- The USDC reserves $x_0 = 42$.

Our goal is to determine the ETH reserves y_0, given the values of L_0 and x_0.
Let's rearrange the equation:

$$x_0^2 + x_0y_0 - L_0/x_0 = 0 \tag{13}$$

We can solve this quadratic equation using the quadratic formula:

$$y_0 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{14}$$

Thus, the ETH reserves y_0 are approximately 0.005668169284575697. The other solution is discarded since it is negative.

Task B: Swap Scenario - Exchanging ETH for USDC

We will assume $dx, dy > 0$ always. Thus, Eq. 12 becomes:

$$L_0 = (x_0 - dx)^2(y_0 + dy) + (y_0 + y)^2(x - dx) \tag{15}$$

We can work it out:

$$\begin{aligned}
\frac{L_0}{(y_0 + dy)} &= (x_0 - dx)^2 + (y_0 + dy)(x_0 - dx) \\
\frac{L_0}{(y_0 + dy)} &= (x_0^2 - 2x_0dx + dx^2) + (y_0x_0 + dyx_0 - y_0dx - dxdy) \\
\frac{L_0}{(y_0 + dy)} - x_0(x_0 + y_0 + dy) &= -dx(2x_0 + y_0 + dy) + dx^2
\end{aligned} \tag{16}$$

This means that this equation can be rewritten as

$$dx^2 + bdx + c = 0 \tag{17}$$

Where

$$b = -(2x_0 + y_0 + dy) \tag{18}$$

$$c = \frac{-L_0}{(y_0 + dy)} + x_0(x_0 + y_0 + dy) \tag{19}$$

Thus:

$$dx_{1,2} = \frac{(2x_0 + y_0 + dy) \mp \sqrt{(2x_0 + y_0 + dy)^2 - 4\left(\frac{-L_0}{(y_0+dy)} + x_0(x_0 + y_0 + dy)\right)}}{2} \tag{20}$$

Now, it is interesting to notice that the solution associated to

$$dx_2 = \frac{(2x_0 + y_0 + dy) + \sqrt{(2x_0 + y_0 + dy)^2 - 4\left(\frac{-L_0}{(y_0+dy)} + x_0(x_0 + y_0 + dy)\right)}}{2} \tag{21}$$

Is not acceptable because $dx_2 > x_0 \ \forall dy$, meaning we would be removing more liquidity from the LP than is actually available. Indeed:

$$dx_2 = x_0 + A \tag{22}$$

Where

$$A = \frac{y_0 + dy + \sqrt{(2x_0 + y_0 + dy)^2 - 4\left(\frac{-L_0}{(y_0+dy)} + x_0(x_0 + y_0 + dy)\right)}}{2} > 0 \ , \forall dy \tag{23}$$

Thus the only solution acceptable is:

$$dx_1 = \frac{(2x_0 + y_0 + dy) - \sqrt{(2x_0 + y_0 + dy)^2 - 4\left(\frac{-L_0}{(y_0+dy)} + x_0(x_0 + y_0 + dy)\right)}}{2} \tag{24}$$

The solution dx as a function of dy is reported in Fig. 5, and for $dy = 5$ ETH sold to the pool leads to an acquired amount of USDC of $dx = 41.628480$.

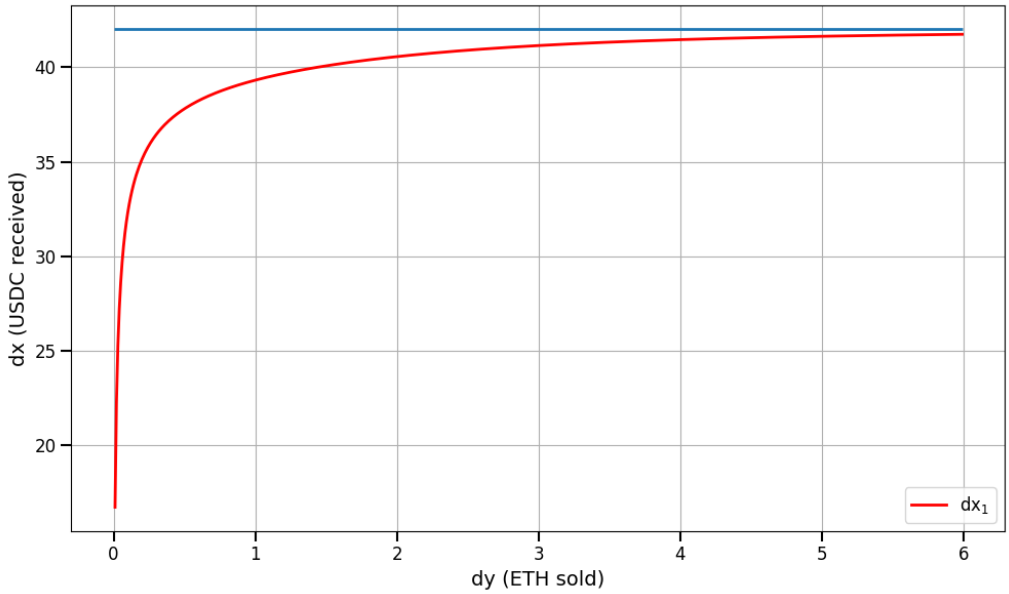

Figure 5: Received USDC as a function of the sold ETH. The blue horizontal line represents the initial x_0 reservoir of the pool.

Task C: Arbitrage

We are happy of having a rich degen friend, since we have a nice arbitrage opportunity.
The situation we have in front is equivalent of having two LPs, one with fixed USDC/ETH price $\mathcal{P}_{friend} = 10$, and one with a price function (see Fig. 6) that can be calculated as:

$$\mathcal{P}_{pool}(dy) = \frac{dx_1(dy)}{dy} \tag{25}$$

The price that the pool offers is such that:

$$\mathcal{P}_{pool}(dy) > \mathcal{P}_{friend} \text{ , for } dy < y_{lim} = 4.15 \tag{26}$$

Thus, if we would buy $y_1 < y_{lim}$ ETH from our friend selling him $\mathcal{P}_{friend} \cdot y_1$ USDC, and then sell these ETH y_1 to the pool receiving back x_1 USDC, we will have that $x_1 > \mathcal{P}_{friend} \cdot y_1$.

We are almost happy. Indeed, we would like to find out which is the amount y_1 we should buy to our friend to maximize our profit (see Fig. 7):

$$\Gamma = (dx^* - dy^* \cdot \mathcal{P}_{friend}) \tag{27}$$

Which has a maximum profit $\Gamma = 33.43107093$ USDC for $y^* = 0.3$ ETH. Thus by selling 30 USDC for 0.3 ETH to our friend and then exit the position with 63.43107093 USDC by selling those ETH to the LP.

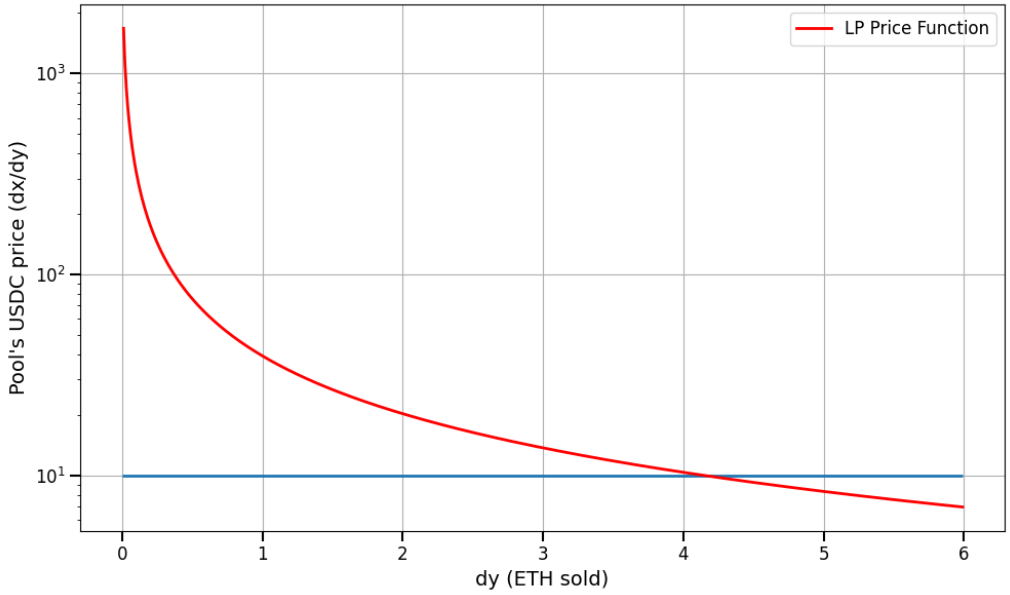

Figure 6: Computed $\mathcal{P}_{pool}(dy)$ USDC price as a function of the dy ETH sold to the LP and reported in log scale for clarity. The blue line indicates the fixed $\mathcal{P}_{friend}$.

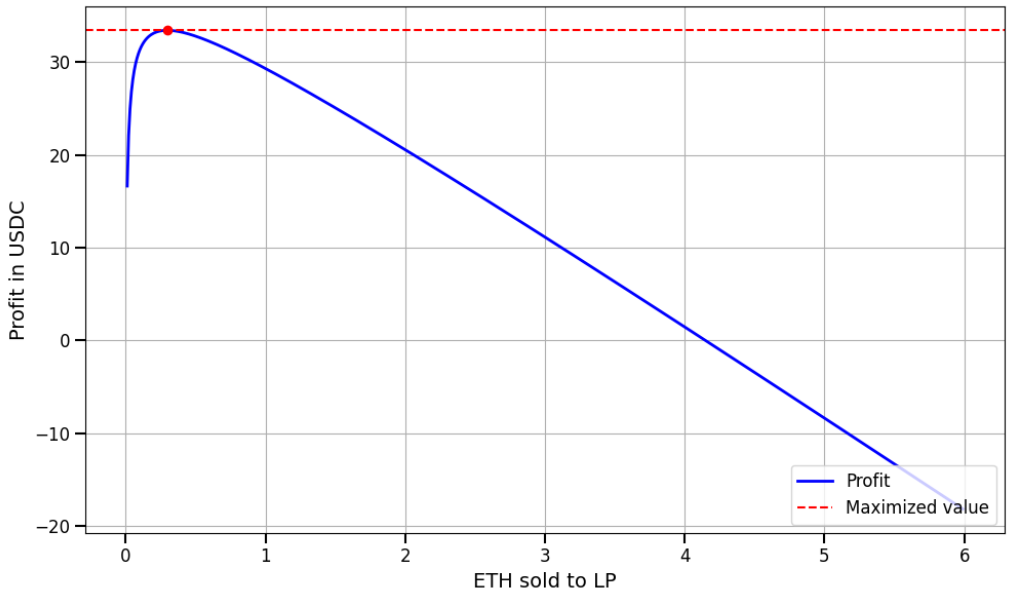

Figure 7: Computed profit obtained by selling all ETH to the LP. The ETH are bought from our friend.

Challenge 5: Market Making

A Market Maker (MM) is an entity that provides liquidity to the market by quoting buy and sell prices for an asset (in this case, a token) and profiting from the difference between the bid price (buy) and the ask price (sell). The market maker must balance:

1. Providing liquidity: Posting buy and sell orders at various price levels.
2. Risk management: Avoid situations where orders are not filled, leading to potential losses.
3. Profit from the spread: Making money from the difference between the price at which they buy (bid) and sell (ask).

In this task, the goal is to design an MM that provides quotes around a predictive fair value for an asset. In the model we will use, the MM places *Limit Orders* (LOs) on the *Limit Order Book (LOB)* at strategic distances from the current midprice to maximize profits while managing the likelihood of their orders being executed.
To develop the algorithm, we will first need to derive a *predictive fair value* for the asset, which serves as the baseline for placing our quotes. The model we employ assumes that the market maker is a *risk-neutral trader* with *costless inventory management* and infinite patience and that all relevant information is public. The MM will observe the current LOB and output quotes (i.e., buy and sell limit orders) based on its computed *predictive fair value*. The approach assumes that the predictive fair value will guide the optimal placement of these LOs, ensuring the MM remains competitive.
Let's split the problem into two separate subtasks: evaluating the fair price and formulating the optimal quotes.

Fair price prediction

Several methods can be used to estimate a fair price in real-time trading. In the following, we will introduce the formula of two simple *fair price* estimators.

Mid-Price of the Best Bid and Ask

The mid-price is the average of the best bid and ask prices in the order book:

$$S^{mid} = \frac{\text{Best Bid} + \text{Best Ask}}{2} \tag{28}$$

Each of these quantities is evaluated at a specific time T. The mid-price reflects an actionable price by capturing the center of market activity. However, in thin or volatile markets, wide spreads can make the mid-price less reliable. Thus, it is best suited for liquid markets with tight spreads.

Weighted Mid-Price (Depth-Weighted)

This method takes into account both the best bid and ask prices, weighting them by the available depth (size) at each level:

$$S^{wmid} = \frac{(\text{Bid Price} \times \text{Bid Size}) + (\text{Ask Price} \times \text{Ask Size})}{\text{Bid Size} + \text{Ask Size}} \tag{29}$$

Each of these quantities is evaluated at a specific time T. The weighted mid-price adjusts for order book imbalances, providing a fairer estimate in uneven markets.
However, it can be affected by large orders or price manipulation.
Thus, it is useful for markets with frequent imbalances on one side of the order book.

For liquid markets, these two approaches are usually sufficient. In the following, we will employ the Weighted Mid-Price approach since if the market were perfectly balanced, it would converge to the mid-price estimate

Optimal quotes estimate

The problem of estimating the optimal quotes is of vital relevance for the MM. Indeed, Garman (1976) demonstrated that market makers require a bid-ask spread to remain viable by managing inherent imbalances between buy and sell orders.[1] These imbalances arise from varying trader needs and preferences, such as risk tolerance and market access, which create temporary disparities in order flow.

The need of MM's quotes spread

In Garman's model, the **market maker** acts as a monopolist, setting **bid and ask prices** to receive and process orders while aiming to maximize profit and avoid running out of cash or inventory—a condition that would cause them to fail. This model uses the **Gambler's Ruin Problem** to quantify the likelihood of a market maker going bankrupt:[2]

$$\Pr_{failure} = \left(\frac{\Pr_{loss} \cdot \text{Loss}}{\Pr_{gain} \cdot \text{Gain}} \right)^{\text{Initial Welth}} \tag{30}$$

Here, the market maker risks failure if their **probability of losing inventory or cash** exceeds their **probability of gaining** it, which depends on the relative probabilities of buy and sell order arrivals, denoted as λ_a for buys and λ_b for sells. Calling p_a , and p_b the ask and bid prices respectively, the Gambler's Ruin Problem can be expressed by the following equations:

$$\begin{aligned} \lim_{t\to\infty} \Pr_{failure}(t) &\approx \mathcal{K} \in (0,1) \text{ , if } \lambda_b > \lambda_a \\ &= 1 \text{ otherwise} \end{aligned}$$

$$\begin{aligned} \lim_{t\to\infty} \Pr_{failure}(t) &\approx \mathcal{J} \in (0,1) \text{ , if } \lambda_a p_a > \lambda_b pb \\ &= 1 \text{ otherwise} \end{aligned}$$

In order for a market maker to remain viable, the conditions from these equations must both be satisfied simultaneously. This requires that the following inequalities hold at the same time:

$$\lambda_b > \lambda_a \quad \text{and} \quad \lambda_a p_a > \lambda_b p_b$$

For these inequalities to be true together, it must always hold that $p_a > p_b$, which establishes the bid-ask spread. Thus, bid-ask spreads are vital conditions for an MM.
However, the problem now is to define the optimal prices the MM should employ to maximize its terminal wealth.

Formal Derivation

Given that market participants operate in a competitive, efficient environment, our MM algorithm will need to continually adjust her quotes based on incoming market data and her estimation of the predictive fair value. This ensures the MM remains competitive in the market while minimizing the risk of unfilled orders, leading to *optimal profitability*. Let's assume that the MM seeks a strategy $(\delta_t^{\pm})_{0\leq t\leq T}$ that maximizes cash at the terminal time T without penalizing uncertainty in her sales directly. At this terminal time, the MM will liquidate her remaining inventory Q_T using a *market order* (MO) at the midprice. Additionally, the MM restricts her inventory within specified bounds

$q^{\pm}_{lim}$, ensuring that it does not exceed an upper limit $q^{+}_{lim} > 0$ or fall below a lower limit $q^{-}_{lim} < 0$, where both limits are finite. Penalties on inventory levels are also applied at terminal time through a coefficient $\alpha \geq 0$, representing the liquidity-taking fees and the impact of the MO as it moves through the LOB, and $\phi \geq 0$, the parameter penalizing running inventory levels. With this preamble, the MM seeks to solve this stochastic optimization problem to maximize its cash at terminal date T:[3]

$$H(t,x,S,q) = \sup_{\delta^{\pm}} \mathbb{E}\left[X_T + Q^{\delta}_T(S^{\delta}_T - \alpha Q^{\delta}_T) - \phi \int_t^T (Q_u)^2 du\right] \tag{31}$$

Where, for each time $t \in [0,T]$ we define:

1. S_t is the midprice which is a stochastic variable defined as

   $$S_t = S_0 + \sigma W_t$$

   Where $\sigma > 0$, and W_t is a Brownian motion connected to price fluctuations;

2. $\delta^{\pm}_t$ are the depths at which the MM posts LOs (i.e. the spread of its quotes), and are the variables against which the MM needs to optimize;

3. X_t is the MM's cash process satisfying

   $$dX^{\delta}_t = (S_t + \delta^{+})dN^{\delta,+}_t - (S_t - \delta^{-})dN^{\delta,-}_t$$

   Where $N^{\delta,\pm}_t$ denote the counting process for the MM's filled sell (+) and buy (-) LOs;

4. The MM's inventory satisfies

   $$Q^{\delta}_t = N^{\delta,+}_t - N^{\delta,-}_t$$

We further assume that other participants buy (+) and sell (-) MOs arrive at Poisson times with rates $\lambda^{\pm}$. Equation 31 is a stochastic control problem and can be solved by employing the dynamic programming principle (DPP) and the related non-linear partial differential equation (PDE) known as the Hamilton-Jacobi-Bellman (HJB) equation also called the dynamic programming equation (DPE). The DPE for this class of problems read:[3]

$$\begin{aligned} 0 =& \partial_t H + \frac{1}{2}\sigma^2 \partial_{SS} H - \phi q^2 \\ &+ \lambda^{+} \sup_{\delta^{+}} \left\{ e^{-\kappa^{+}\delta^{+}} \left(H\left(t, x + \left(S + \delta^{+}\right), q - 1, S\right) - H \right) \right\} \mathbb{I}_{q > \underline{q}} \\ &+ \lambda^{-} \sup_{\delta^{-}} \left\{ e^{-\kappa^{-}\delta^{-}} \left(H\left(t, x - \left(S - \delta^{-}\right), q + 1, S\right) - H \right) \right\} \mathbb{I}_{q < \bar{q}}, \end{aligned} \tag{32}$$

Where $\mathbb{I}$ is the indicator function. Equation 32 it is representing a stochastic diffusion problem with control. Indeed, the first two terms $\partial_t H + \frac{1}{2}\sigma^2 \partial_{SS} H$ are typical of diffusion processes. Additionally, the term ϕq^2 captures the effect of penalizing deviations of inventories from zero along the entire path of the strategy, which represents the inventory cost due to misalignment with the target inventory. In the second line of the DPE, the supremum over δ^{+} contains the terms corresponding to the arrival of a market buy order, which is filled by a limit sell order. Similarly, in the last line of the DPE, the supremum over δ^{-} contains analogous terms for the market sell orders that are filled by limit buy orders. In addition, $\kappa^{\pm}$ is a parameter that adjusts the likelihood of a LO being filled when a MO arrives, according to:

$$\Lambda^{\pm}_t = \lambda^{\pm} e^{-\kappa^{\pm}\delta^{\pm}_t} \tag{33}$$

Given the task of the exercise and my choice of developing an MM that provides quotes around a predictive fair value, we will make the following assumptions for the MM:

1. It does not impose a penalty on running inventories;
2. It does not incur a terminal inventory penalty (i.e., $\varphi = \alpha = 0$)
3. There are no constraints on the amount of inventory the strategy may accumulate (i.e., $|q_{lim}| \to \infty$);
4. At terminal time T, the inventory is unwound at the midprice.

Given these assumptions, the optimal MM's strategy, i.e. the solution to Eq. 31 becomes: [3]

$$\delta^{+,*}(t,q) = \frac{1}{\kappa^+} \quad \text{, and} \quad \delta^{-,*}(t,q) = \frac{1}{\kappa^-} \tag{34}$$

The strategy in Eq. 34 translates into the MM posting in the LOB, maximizing the probability of the LOs being filled.
However, $\kappa^{\pm}$, which in our case defines the whole MM's strategy, needs to be estimated. Even in this simple case, $\kappa^{\pm}$ is not easy to estimate, and the whole optimal strategy might depend upon several factors, such as the latest trading prices, market volatility, and order imbalances. [4]

MM implementation: *MarketMaker* class

The *MarketMaker* class simulates a market maker's process of generating bid and ask quotes. It takes order book data (bids and asks) and calculates a fair price for the asset using the volume-weighted midpoint of the best bid and ask. Based on this fair value, the class generates bids and asks for quotes by applying a spread, determining the prices at which the market maker is willing to buy and sell. The spread from the *fair price* is defined as

$$\delta^{\pm} = \pm c \cdot \Delta_t^{\pm}, \text{ where } c \in [0, 1] \tag{35}$$

Where $\Delta_t^{\pm}$ are the observed best ask/bid prices from the orderbook at time t. Although I know I shouldn't simulate the strategy, I found a kind of appealing having only a couple of constants to define. Thus, although it is a very rough estimation, and there is no risk management, with this strategy, we are always sure that we are offering competitive quotes.

market_maker.py

```python
import pandas as pd
from typing import Optional

class MarketMaker:


    def calculate_fair_value(self, current_orderbook: pd.Series) -> float:
        """
        Calculate the fair value based on the selected method.

        Args:
            current_orderbook (pd.Series): A row from the orderbook containing bid and ask
                                           data.

        Returns:
            float: The predicted fair value.
        """
        if self.fair_price_method == 'wmid':
            # Weighted mid price (wmid)
            best_bid_price, best_bid_quantity = current_orderbook['bids'][0]
            best_ask_price, best_ask_quantity = current_orderbook['asks'][0]
            fair_value = (
                (best_bid_price * best_bid_quantity) + (best_ask_price * best_ask_quantity)
                ) / (best_bid_quantity + best_ask_quantity) if (best_bid_quantity +
                best_ask_quantity) > 0 else 0
        else:
            raise ValueError(f"Unknown fair price method: {self.fair_price_method}")
        return fair_value

    def generate_quotes(self, fair_value: float) -> dict:
        """
        Generate quotes for the market maker (bid and ask prices) around the fair value.

        Args:
            fair_value (float): The predicted fair value.

        Returns:
            dict: A dictionary with bid and ask quotes.
        """
        # Extract the best bid and ask prices from the orderbook
        best_bid_price = current_orderbook['bids'][0][0]  # The best bid price
        best_ask_price = current_orderbook['asks'][0][0]  # The best ask price
        # Calculate the spreads
        bid_spread = fair_value - best_bid_price
        ask_spread = best_ask_price - fair_value
        # Calculate quotes
        bid_price = fair_value  - (self.spread_pct/100) * bid_spread
        ask_price = fair_value  + (self.spread_pct/100) * ask_spread

        return {'bid': bid_price, 'ask': ask_price}
```

Figure 8: Shortened code snippet of the *MarketMaker* class.

Risks and limitations

While this model provides a structured approach to market-making, there are several risks and limitations:

- **Unrealistic Assumptions**: The model assumes *infinite patience*, *no trading costs*, and *risk neutrality*, which may not reflect real-world conditions where transaction costs, inventory management, and risk considerations are present.
- **Market Conditions**: The model assumes that other market makers' actions are independent of the MM's decisions and that market orders follow a simple, predictable distribution. In practice, market dynamics can be more complex, with *information asymmetry*, sudden market shocks, and other factors that influence trading decisions.
- **Predictive Fair Value Accuracy**: The quality of the quotes depends on the accuracy of the *predictive fair value*. If the predictive model is inaccurate, the MM's quotes may be suboptimal, leading to either missed opportunities or excessive risk exposure.

Thus, my implementation of this model into the *MarketMaker* class described before has several limitations and risks. It assumes that the best bid and ask prices ($\Delta_t^{\pm}$) and their midpoint sufficiently represent the asset's fair value, which may not hold in volatile or illiquid markets where price movements are rapid or order book imbalances occur. The spread calculation, $\delta^{\pm} = \pm c \cdot \Delta_t^{\pm}$, uses a constant c and does not dynamically adapt to changes in volatility, liquidity, or market conditions, potentially resulting in inefficient pricing. Moreover, the model lacks inventory risk management, exposing the market maker to significant risks of accumulating unbalanced positions and adverse price movements. Additionally, the model relies only on the best bid and ask, ignoring order book depth, and does not account for the market maker's impact on prices, which can lead to signaling risks or inefficiencies. Furthermore, external factors such as news, macroeconomic data, or correlated asset prices are not incorporated, potentially undermining the accuracy of the fair value calculation. Dependency on accurate order book data exposes the model to risks from spoofing or manipulation. Key risks include inventory risk, where unbalanced positions due to imbalanced order flows can result in significant losses, and adverse selection risk, where informed traders exploit mispriced quotes, leaving the market maker at a disadvantage. Other risks include liquidity risk in shallow markets, volatility exposure due to rapid price changes, operational risks such as system errors or downtime, and regulatory risks associated with improper market-making behavior. To address these limitations, the model can incorporate dynamic spread adjustments based on market conditions, implement inventory risk management mechanisms, use deeper order book analysis, refine fair value calculations with external data.

Results

We first analyze the behavior of the fair price against the actual trade price (See Fig. 9) throughout the period under analysis.

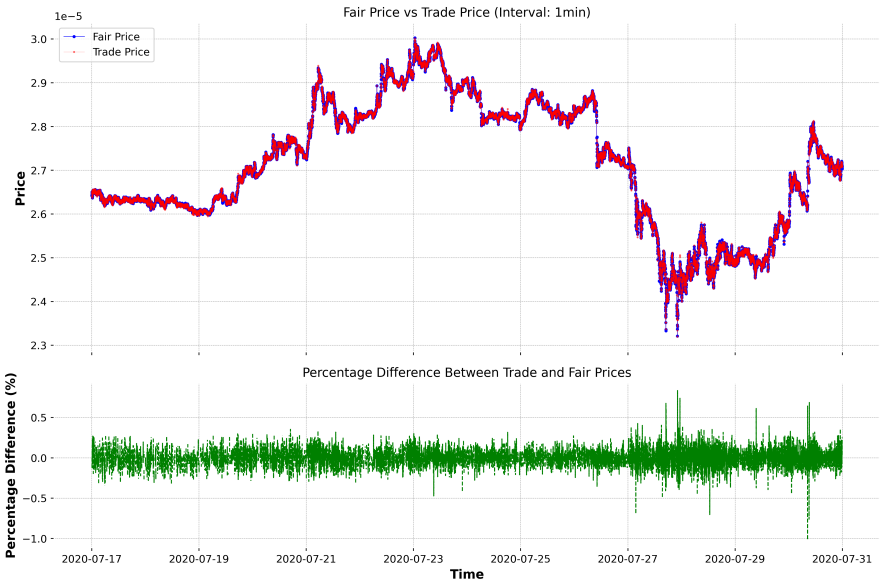

Figure 9: Fair price (blue) and actual trade price (red) as a function of time. In the bottom panel, we report the difference in percentage between the two prices.

As can be noticed, the difference between the prices is bound below 1.02% for all market events. Thus, our fair price is a good approximation of the true trade price.

Let's now analyze the market trend using a candlestick chart (See Fig. 10). After an initial slight downtrend and low-volatility market behavior (small candles and wicks), we observe a clear uptrend from July 19 to 23. This period is characterized by consecutive green candles, signaling strong buying pressure. On July 19, a bullish engulfing pattern initiates this uptrend, as a larger green candle completely engulfs the preceding red candle. The buy/sell volume ratio reaches 1.145 on July 19, suggesting strong buyer dominance. This is further supported by increased bid depth, indicating ample liquidity to sustain the upward movement (See Fig. 11).
A reversal pattern forms around July 23, with the market exhibiting more erratic price action due to low liquidity (evidenced by the long wicks in the candlesticks and the lowest cumulative depths in both the bid and ask orders, see Fig. 11). This lack of depth leads to large price fluctuations, and on July 27, a sharp red candle with long wicks indicates selling pressure. At this point, the buy/sell ratio drops to 0.864, confirming the dominance of sellers. The sudden price drop may be due to external factors such as news or market sentiment shifts, leading to a spike in volatility.
From July 28-29, green candles appear, signaling potential recovery.

An ideal MM should continuously adapt its strategy to the prevailing market conditions. In a low-volatility, down-trending market with small candles and wicks, the MM should focus on providing liquidity with tight bid-ask spreads while minimizing inventory risk by adjusting bids and asks frequently. During a bullish trend characterized by consecutive green candles and strong buying pressure, the MM should narrow the spread to capture increased trading volume while carefully accumulating inventory on the buy side. It is crucial to monitor the buy/sell volume ratio to gauge the strength of the trend and adjust the strategy accordingly. When a reversal occurs, and the market enters a downtrend, the MM should widen the spread to mitigate risk, reduce exposure to falling prices by unwinding positions, and employ hedging strategies to protect against downside risk. Finally, during

periods of recovery, the MM should adjust the spread to reflect the renewed buying pressure but remain flexible, watching for signs of continuation or further reversal. By monitoring market indicators like buy/sell ratios, order book depth, and liquidity, the MM can continuously adjust its behavior, offering liquidity when possible while managing risk through dynamic spread adjustments, inventory management, and hedging. In this way, the MM maximizes profitability while minimizing risk across varying market conditions.

Our MM strategy is not ideal. In particular, the spread is set as a fixed proportion of the observed bid/ask spread in the order book ($\delta^{\pm} = \pm c \cdot \Delta_t^{\pm}$, with $c = 50\%$).

In a low-volatility phase, this approach is likely to provide liquidity efficiently, as the MM quotes close to the market spread. However, in a bullish market, the MM may find itself less competitive, as it may not capture trades at higher volumes if the spread narrows quickly. During a downtrend or reversal, the strategy could be less effective, as the MM might not widen its spread fast enough to account for increased volatility, leading to potential exposure to larger price movements. Finally, in a recovery phase, the strategy could perform better as the market stabilizes, but the MM must remain cautious of sudden reversals. Overall, while this strategy can provide liquidity in calm markets, it might not be robust enough in volatile or fast-moving conditions, potentially exposing the MM to risk or missed opportunities if it does not adjust dynamically to market changes.

This is consistent with the observed behavior of the MM in practice (See dotted lines in Fig. 10), where the quotes are consistently tight and close to the asset's closing price, reflecting the rigid spread strategy. However, as noted, this approach does not account for high-volatility periods, during which the MM would need to widen the spread to manage risks more effectively.

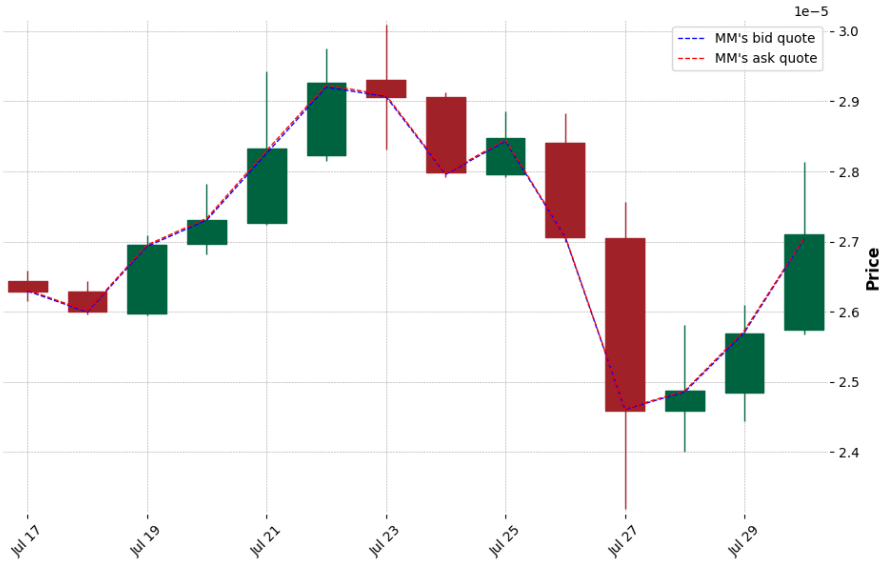

Figure 10: Daily candle stick graph of the price of the asset. The blue and red dotted lines represent the MM's bid and ask quote respectively.

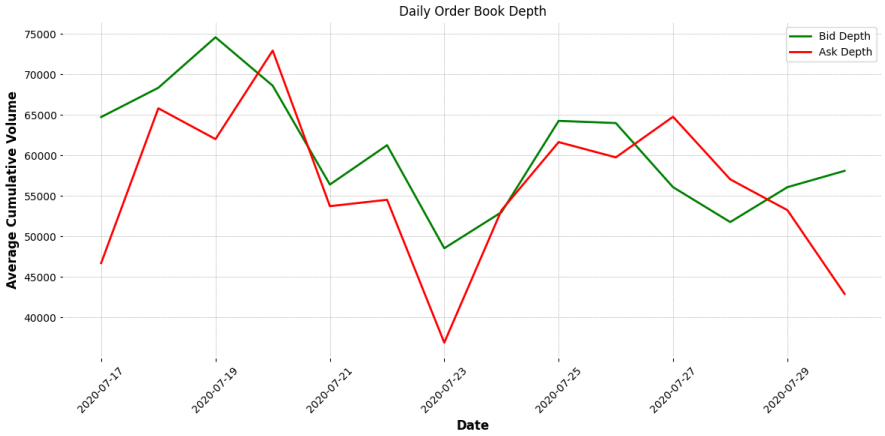

Figure 11: Daily Orderbook depth.

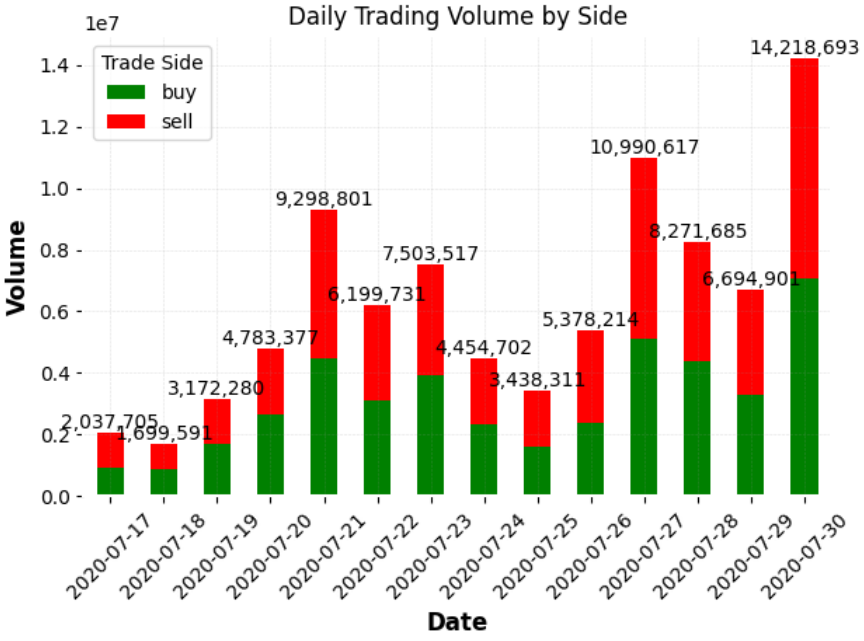

Figure 12: Daily Volume analysis.

Reflection on Machine Learning Models for Strategy Optimization

Although simulating the strategy was not part of the task, I got excited by the potential of using machine learning (ML) techniques to train a market-making strategy. Specifically, one could design a system involving two Long Short-Term Memory (LSTM) models trained on three-quarters of the provided dataset. The first LSTM model aims to simulate the dynamics of the order book, while the second model simulates trade execution within the order book. One could test and evaluate these models on the remaining quarter of the dataset. These models can be used as surrogate oracles to train a reinforcement learning (RL) strategy. In this setup, the MM is treated as a RL agent interacting with the oracle's order book, posting quotes that could potentially be filled by the trades generated by the trade-oracle. The RL agent learns an optimal strategy by minimizing a loss function that is properly defined. This approach enables the MM to adapt to the simulated market environment, refining its strategy over time by learning from trial and error.

Although exciting, at least for me, this methodology does have some limitations. One significant challenge is the complexity of accurately simulating the market dynamics, as real-world order books and trading patterns are influenced by a wide range of factors that might not be captured by the LSTM models. Additionally, the agent's performance may be highly dependent on the quality and representativeness of the training data, which can impact how well the strategy generalizes to unseen market conditions. Furthermore, while the RL agent can adapt to simulated environments, translating this to real-world market conditions might prove difficult, especially if the market experiences drastic changes or unforeseen events that are not part of the training data.

Nevertheless, it was fun thinking about it.

Appendix I: Task 2 – Derivations of the ACC price formula

As discussed in the Metronome's *Owner's Manual*[5] we have that:

$$t = s_0 \left[\sqrt{1 + \frac{e}{e_0 + e}} - 1 \right] \tag{36}$$

$$r = r_0 \left[1 - \left(1 - \frac{t}{s_0 + t} \right)^2 \right] \tag{37}$$

Where s_0 is the Smart Token Supply, e is the reserve tokens received in exchange for smart tokens t, e_o r_0 are the initial reserve token supplies.

$$\begin{aligned}
y(x) &= y_0 \left[1 - \left(1 - \frac{s_0 \left(\sqrt{1 + \frac{x}{x_0 + x}} - 1 \right)}{s_0 + s_0 \left(\sqrt{1 + \frac{x}{x_0 + x}} - 1 \right)} \right)^2 \right] \\
&= y_0 \left[1 - \left(1 - \frac{\left(\sqrt{1 + \frac{x}{x_0 + x}} - 1 \right)}{\sqrt{1 + \frac{x}{x_0 + x}}} \right)^2 \right] \\
&= y_0 \left[1 - \left(\frac{1}{\sqrt{1 + \frac{x}{x_0 + x}}} \right)^2 \right] \\
&= y_0 \left[1 - \frac{1}{1 + \frac{x}{x_0 + x}} \right] \\
&= y_0 \left(\frac{\frac{x}{x_0 + x}}{1 + \frac{x}{x_0 + x}} \right) \\
&= y_0 \left(\frac{x}{x_0 + 2x} \right)
\end{aligned} \tag{38}$$

Appendix II: Task 2 – Numerical issues with numerical simulation of the arbitrage strategy

As stated in the main text, I am trying to perform this strategy to overcome the other adversarial MEV players.

1. Given the initial reserves in the ACC compute t^* and the corresponding t_{lim} such that the profit $\max_{m_d} \eta(t_{lim}) > c \cdot (1 + f)$, where $c > 0$ is a variable that accounts for the amount of profit more than the transaction fee (f) we are willing to get;

2. At the time t_{lim} we perform the transactions exchanging the computed m_{lim}, thus perturbing the reserves in the ACC.

3. Iterate with the new reserves in the ACC

A pseudo-snippet of the code implemented to do this is in Fig. 13, where it is visible the original function used to compute the profit at the trade time t_{lim} associated with a purchased m_{lim} METs from the DAC. This t_{lim} was chose to be such that $\max_{m_d} \eta(t_{lim}) = c \cdot (1 + f)$. However, due to numerical accuracy, we often couldn't match the equality conditions, finding $\max_{m_d} \eta(t_{lim}) > c \cdot (1 + f)$, and resulting in fluctuations in the computed net profit (see Fig. 14). To solve this problem I hardcoded the expected value of the profit (the commented line in the snippet).

strategy.py

```
t_star, met_max = find_t_star(time)
t_lim, m_lim = find_t_max_profit(t_star, met_max, PROFIT_PCT_STRATEGY, time_step)

# Initial resources
DAP_m0, m0, e0 = MET_LIMIT, M_0, E_0

# List to store the results
results = []
time = np.arange(t_star, t_star + 60, time_step)

print(" ")
print("Simulating the arbitrage strategy")
# Loop over each time step
for t in time:
    if t >= t_lim and DAP_m0 > 0:  # Execute arbitrage if time exceeds limit
        if m_lim > DAP_m0: # We cannot profit enough from this trade, just exit
            return results

        # Calculate the ETH obtained from the ACC with m_d = m_lim
        e_a = P_ACC(m_lim, e0, m0)*m_lim
        e0 -= e_a  # Update ACC ETH reservoir
        m0 += m_lim  # Update ACC MET reservoir
        DAP_m0 -= m_lim
        profit = get_profit_value(m_lim, t_lim)
        #profit = (1 + PROFIT_PCT_STRATEGY) * FEE
```

Figure 13: Pseudo-snippet of the code implementing my strategy

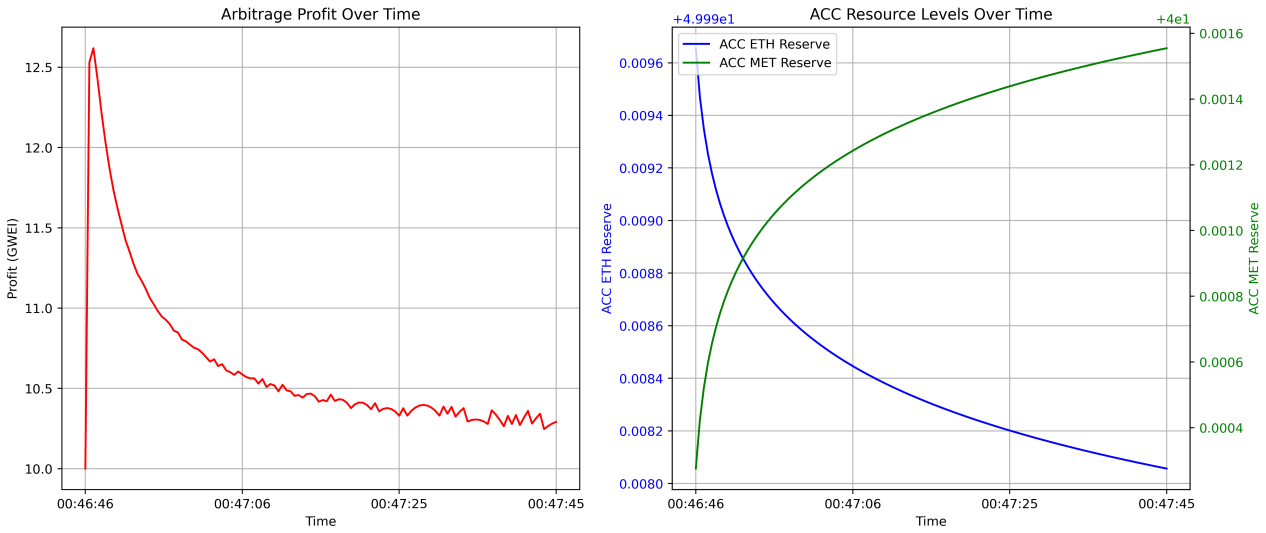

Figure 14: (left) Net profit of the strategy in GWEI as a function of time. (right) ACC reserves in time.

References

[1] Garman, M. B. *J. financ. econ.*, 3(3):257–275, **1976**.

[2] Aldridge, I. *High-frequency trading: a practical guide to algorithmic strategies and trading systems*, volume 604. John Wiley & Sons, **2013**.

[3] Cartea, Á., Jaimungal, S. and Penalva, J. *Algorithmic and high-frequency trading*. Cambridge University Press, **2015**.

[4] Xiong, Y., Yamada, T. and Terano, T. In *2015 Winter Simulation Conference (WSC)*, pages 324–335. IEEE, **2015**.

[5] Software, A. Owner's Manual, **2024**. Accessed: 2024-11-24.