# The Dynamic AMM: An Updating Linear Pricing Rule (V2)

Thrackle Research

February 19, 2025

## Contents




**Abstract**

We provide an overview of a novel DeFi market maker called the Dynamic AMM. The Dynamic AMM is a pricing mechanism that automatically updates its price curve in response to arriving trades. These updates cause the amount of collateral in the market to increase (via what we term "smart fees"), mitigate the market's vulnerability to sandwich attacks, and have many other valuable properties.

In sections 1-4, we orient the reader in a general way by illustrating how this liquidity solution differs from other market makers. In section 5, we compare the behavior of the Dynamic AMM with existing market makers, using real historical trade data. In the Appendix, we delve more deeply into the same material, giving more mathematical detail and some formal proofs.




## 1 Related Work and Preliminary Notes

The DeFi liquidity landscape has evolved dramatically over the last decade. Today, there exists a broad and diverse array of liquidity protocols, each with different strengths and intended uses. A comprehensive survey of contemporary liquidity protocols is beyond the scope of this whitepaper, but we will consider a few relevant techniques here.[1]

### 1.1 AMMs and TBCs

Let us begin by reviewing some terminology and common DeFi market making techniques, and discussing how the Dynamic AMM differs from them. In the section below, we discuss the difference between an AMM and a TBC, and show that the Dynamic AMM differs from both in that its pricing curve can evolve – that is, its pricing curve changes over time in response to the trades executed on it.

The most common way to provide decentralized liquidity is with an "automated market maker" (AMM). Less common are "token bonding curves" (TBC). The constant product approach shown in figure 1 is probably the most common technique in the DeFi sector. A generalization of CPMMs are Geometric Mean Market Makers.[1]

AMMs and TBCs are usually treated separately, with the latter being used to sell tokens that are not yet circulating widely (during a so-called "bootstrapping" phase), or for issuing NFTs one at a time.

However, these two mechanisms are in some cases mathematically equivalent. Specifically, for a finite trading range (for example, a single Uniswap V3 position) the pricing rule can be transformed into a single, finite TBC. Therefore, we can think of static AMMs and TBCs as different views of the same basic mathematical structure, although the two are used in different settings and feel different psychologically to the user. See figure 2 for a rendering of a simple TBC.

[1] Such a survey is given in [2].

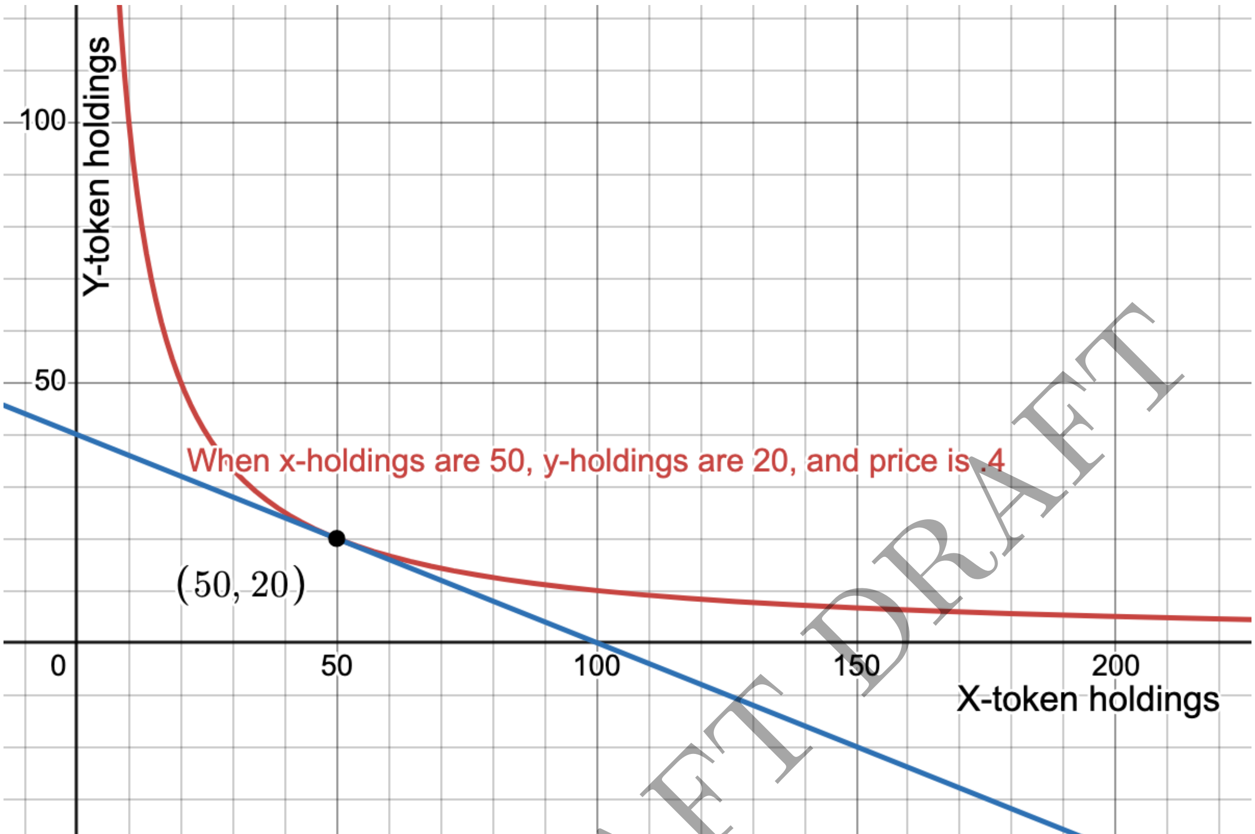



Figure 1: In an AMM, both axes represent token amounts held in the pool. Spot price is given by the negative slope of the tangent line.



### 1.2 Updating Price Rules

Let us now distinguish between between AMMs and TBCs – both of which rely on invariant bonding curves – and *updating* price rules like the Dynamic AMM, whose logic can change over time.

The Dynamic AMM, composed of a line whose parameters constantly adapt in response to the arrivals of trades, is an *updating* TBC. In this paper, we depict the price function in the usual TBC-space ($x$-axis is amount minted, $y$-axis is price), but it must be remembered that any such picture shows a snapshot in time, since the true price function is constantly changing.[2]

The Dynamic AMM is not the first updating price rule. Several projects, for example, have used price oracles to update their pricing curves, often with the goal of mitigating losses to arbitrageurs. Arrakis's HOT AMM, the Lifinity protocol, swaap.finance's Matrix-MM, and DODO's "proactive market maker" all rely on price oracles to keep quotes from going stale and reduce a pool's vulnerability to informed traders.

These projects, and others like them, are updating price rules. That is to say, their price rules can vary over time (unlike in Uniswap V2 or a standard TBC). However, it is important to distinguish between their approach, which relies on oracles and where updates can happen independently of trades, and the one in question here. In the Dynamic AMM, curve updates occur in response to trades, and they are not primarily intended to reduce the profits of arbitrage traders. Our mechanism is thus more fully decentralized, in addition to its other advantages discussed below.

Curve Finance has released two projects that are not oracle-based, and whose curve updates, like ours, occur in response to arriving trades.[3] These projects achieve a high degree of control over price impact; although they are updating price curves that, like the Dynamic AMM, react to arriving trades rather than oracles, they solve very different problems and have very different use cases.

[2] We term such snapshots the "intermediate TBC."

[3] Respectively, StableSwap (for trading stablecoins) and CryptoSwap (a more complex mechanism for all purpose cryptocurrency trading). See [3] and [4].

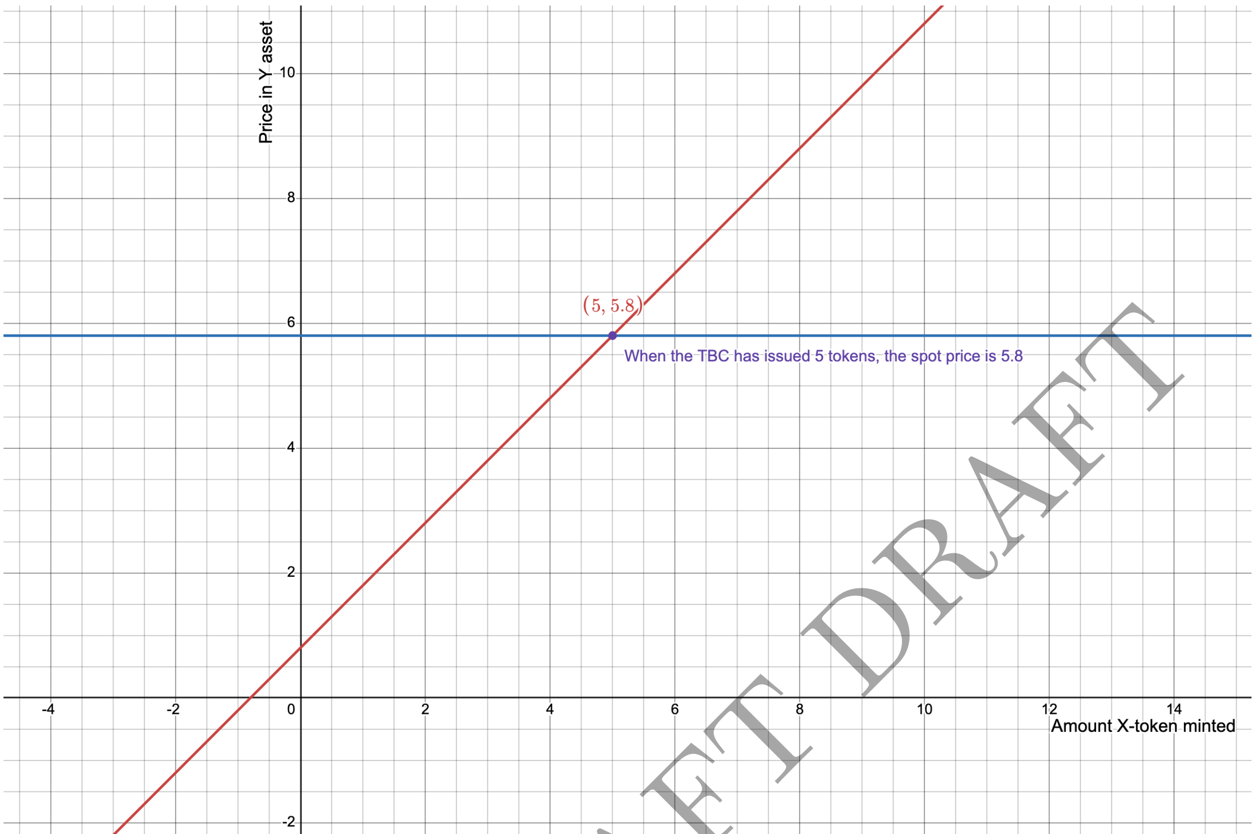



Figure 2: A TBC maps amount minted (x-axis) to price (y-axis).



### 1.3 Use Cases for the Dynamic AMM

This mechanism can be used to bootstrap liquidity without providing initial collateral as is required with a conventional AMM. For this reason, it is ideal for new projects where tokens have not had time to circulate widely.

It also has a unique way of accruing capital without relying on separate buy and sell functions, and even before standard percentage-based fees are imposed.[4] This makes the Dynamic AMM a good fit for community-oriented tokens where the market maker wishes to incentivize behaviors that enrich the community.

Finally, the Dynamic AMM will produce less and less volatility as more $x$-token circulates. This makes it a good home for projects that wish to promote thriving, stable economies rather than economies fueled by short-term speculation.

### 1.4 Operating in a closed system

Let us further note that, in the discussion to follow, it should be understood that the Dynamic AMM is the only market for $x$-token. Thus all considerations that depend upon external markets are not relevant here.[5]

## 2 Dynamic AMM Design Overview

### 2.1 General Approach

The Dynamic AMM is composed of a line whose slope and intercept adapt to the arrival of trades. Thus, the basic price function is a simple line:

$$f_n(x) = b_n x + c_n$$

[4] We term these capital accrual properties "smart fees."

[5] Loss-versus-holding, for example, depends on having another place to liquidate portfolio holdings. If we consider the market owner the sole liquidity provider, this metric cannot apply.

Why is the price function $f$ subscripted? Because for any given moment in the curve's lifetime, a different line will apply, depending on the history of updates. The logic governing these updates is detailed in the Appendix, but note that these adjustments ensure three key things:

1. **price** increases with outstanding supply,
2. **price impact** decreases with increased outstanding supply, and
3. for a given amount of x-token in outstanding supply, **collateral**[6] **continually accrues** to the market owner with every transaction. See Section 4.3.

## 2.2 Dynamic AMM Variables (constant)

In this section, we discuss the quantities that the Dynamic AMM uses. Some of these are constants set at instantiation, whereas some are updated, as specified in section 3. Here, we discuss these variables one at a time and provide some helpful illustrations.

We begin with variables that are set once upon instantiation and never altered again.

| Description | Variable | Definition or Bounds |
|---|---|---|
| Initial tokens deposited into TBC | $x_{\text{add}}$ | $x_{\text{add}} > 0$ |
| $y$-intercept upon TBC initialization | $p_{\text{lower}}$ | $p_{\text{lower}} \geq 0$ |
| Vector Field Parameter | $V$ | $V > 0$ |
| Initial Concentration Parameter | $C_0$ | $C_0 > 0$ |
| Initial minimum value of $x$ | $x_{\text{min}}^{(0)}$ | $x_{\text{min}}^{(0)} > 0$ |
| Initial virtual collateral in pool | $D_V^{(0)}$ | $D_V^{(0)} = \frac{V}{2 \cdot C_0} \cdot \left(x_{\text{min}}^{(0)}\right)^2 + p_{\text{lower}} \cdot x_{\text{min}}^{(0)} = \mathbf{D}\left(x_{\text{min}}^{(0)}, \mathbf{b}(0, C_0), p_{\text{lower}}\right)$ |
| NFT amount field value at initialization | $W_0$ | $W_0 > 0$ |

The $x_{\text{add}}$ value represents the maximum number of x-tokens the Dynamic AMM can sell.[7] $x_{\text{min}}$ is the lowest possible $x$-value the price curve can achieve. Transactions that would push the $x$-value outside this realizable trading region fail.

### Concentration Parameter C

The concentration parameter C is discussed in detail below in 2.5. Let us note here that C is a number that informs the slope update's sensitivity to order size, that it is set once and never changed again,[8] and that it is fully determined by the user-supplied variables discussed above.

### Vector Field Parameter $V$

The parameter $V$ controls the sensitivity of the slope field to trade size. A higher $V$ value will, in general, produce more price volatility. This can be seen by examining the equation for the slope field (the equation that $x$-value to the slope of the pricing line):

$$s(x) = \frac{V}{x + C}$$

Holding $x$ and $C$ constant, we can easily see that slope is increasing in $V$. Thus, $V$ is a tuning parameter for the Dynamic AMM; higher values will lead to more price fluctuation.

### $D_{\mathbf{V}}$

The constant $D$v is equal to the integral of the initial price function on the interval (0, $x_{\text{min}}$). Since that interval is not in the range of allowable trading ($x_{\text{min}}$,$x_{\text{add}} + x_{\text{min}}$), this collateral will never be accessible to traders. Thus, we can think of it as "virtual" collateral. This quantity will be important when deploying the Dynamic AMM's revenue function. See figure 3.

[6] Here, we define collateral as the value in $y$-token of all the $x$-token that traders have purchased from the market.

[7] Note the difference between this value and the maximum $x$-value achievable in the price curve (they are different when $k$ is greater than 0 – see discussion below).

[8] $C$ and $x_{\text{add}}$ can change with liquidity adds and removals. See below at Section 2.8

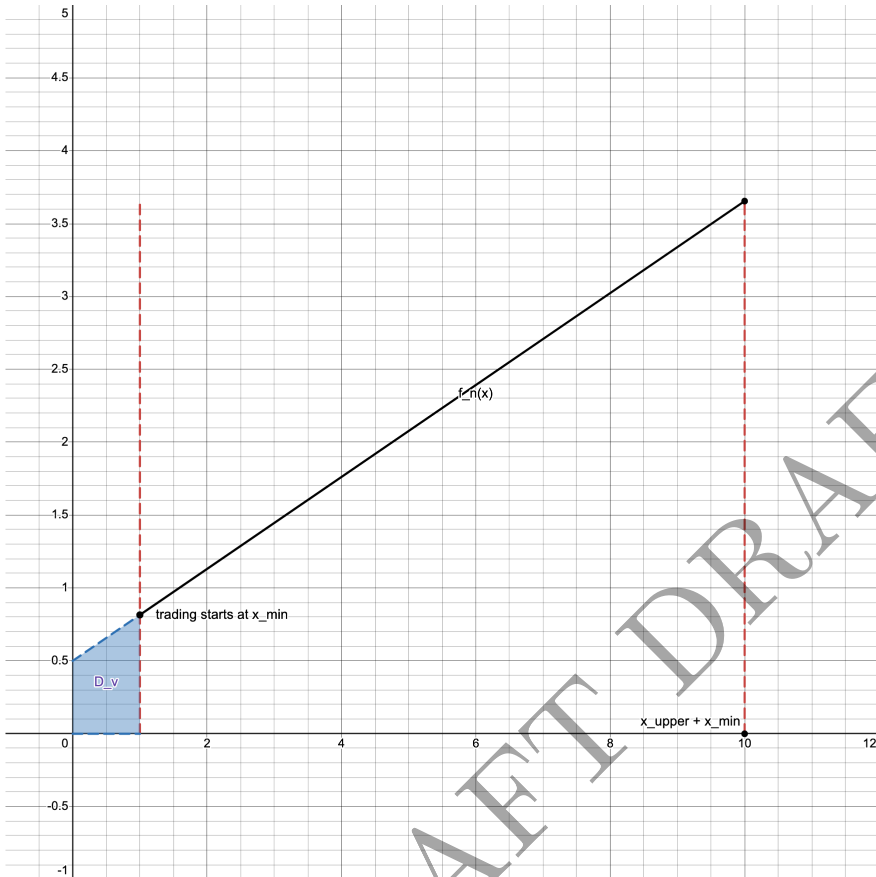



Figure 3: Dynamic AMM, showing the presence of virtual collateral $D_{\mathrm{V}}$



## 2.3 Dynamic AMM Variables (not constant)

We now discuss the quantities that the Dynamic AMM maintains over time, updating with each transaction. See table 1.

#### Amount in: A

Within any given transaction, we designate the amount of $x$-token deposited into or taken out of the Dynamic AMM with the variable $A$. A positive value means that a trader is buying the $x$ token. A negative value means that a trader is selling the $x$ token. This is shown in figure 4. Note that even in the case of a buy or sell order of $y$-token, we need an $A$ value. This value can be calculated using the inverse cost functions discussed in Section 2.4.

## 2.4 Cost Functions

In order to actually execute a transaction, we need more than just the spot price: we need functions that take in a transaction amount, and output the correct cost for that order size and trade direction. To evaluate the cost of a buy or sell order of $x$-token, we must:

1. Calculate the value (in y-token) of current outstanding supply.
2. Calculate the value (in y-token) of (current outstanding supply + transaction amount).
3. Take the difference between the two. This gives us the cost (in y-token) of transacting the proposed amount.

To evaluate the cost of a buy or sell order of $y$-token, we must:

1. Calculate the value (in x-token) of current collateral.
2. Calculate the value (in x-token) of (current collateral + transaction amount).

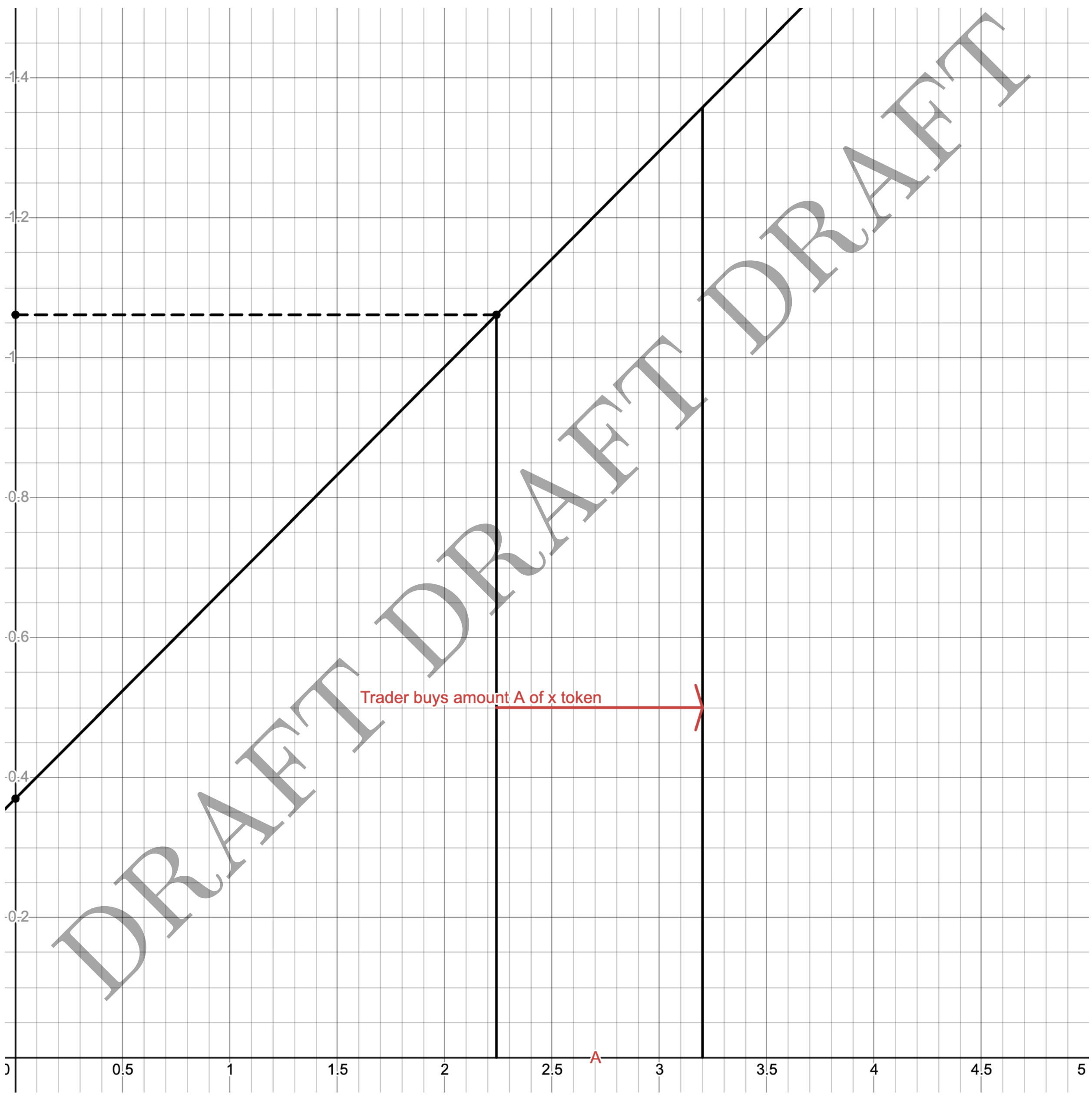




Figure 4: A buy order.

Table 1: TBC-maintained variables

| Description | Variable | Initial Value |
|---|---|---|
| Value of $x$ before the $n$–th transaction | $x_n$ | $x_{\min}^{(0)}$ |
| Area under the TBC curve over $[0, x_n]$ before the $n$–th transaction | $D_n$ | $D_V^{(0)}$ |
| Parameter $b$ (slope of intermediate TBC) before the $n$–th transaction | $b_n$ | $\frac{V}{x_{\min}^{(0)}+C_0} = b(x_{\min}^{(0)}, C_0)$ |
| Parameter $c$ ($y$-intercept of intermediate TBC) before the $n$–th transaction | $c_n$ | $\frac{V}{2C_0(x_{\min}^{(0)}+C_0)} \cdot (x_{\min}^{(0)})^2 + p_{\text{lower}}$ |
| ..equivalent parameter $c$ equation | $c_n$ | $\mathbf{c}(x_{\min}^{(0)}, \mathbf{b}(x_{\min}^{(0)}, C_0), \mathbf{b}(0, C_0), p_{\text{lower}})$ |
| Spot price before the $n$–th transaction | $p_n$ | $\frac{V}{x_{\min}^{(0)}+C_0} \cdot x_{\min}^{(0)} + c_n$ |
| ...equivalent spot price equation | $p_n$ | $\mathbf{p}(x_{\min}^{(0)}, b_0, c_0)$ |
| Concentration parameter | $C_n$ | $C_0$ |
| Minimum value of $x$ | $x_{\min}^{(n)}$ | $x_{\min}^{(0)}$ |
| Maximum value of $x$ | $x_{\max}^{(n)}$ | $x_{\min}^{(0)} + x_{\text{add}}$ |
| Virtual collateral in the pool | $D_V^{(n)}$ | $D_V^{(0)}$ |
| Sum of NFT amount fields minted by the pool | $W_n$ | $W_0$ |

3. Take the difference between the two. This gives us the cost (in x-token) of transacting the proposed amount.

These functions are listed in Table 2.

| Variable | Description | Input | Output |
|---|---|---|---|
| $F_{\text{n}}$ | Cost Function | $x$-token transaction amount | value in $y$-token |
| $F_{\text{n}}^{-1}$ | Inverse Cost Function | $y$-token transaction amount | value in $x$-token |

Table 2: Cost Functions

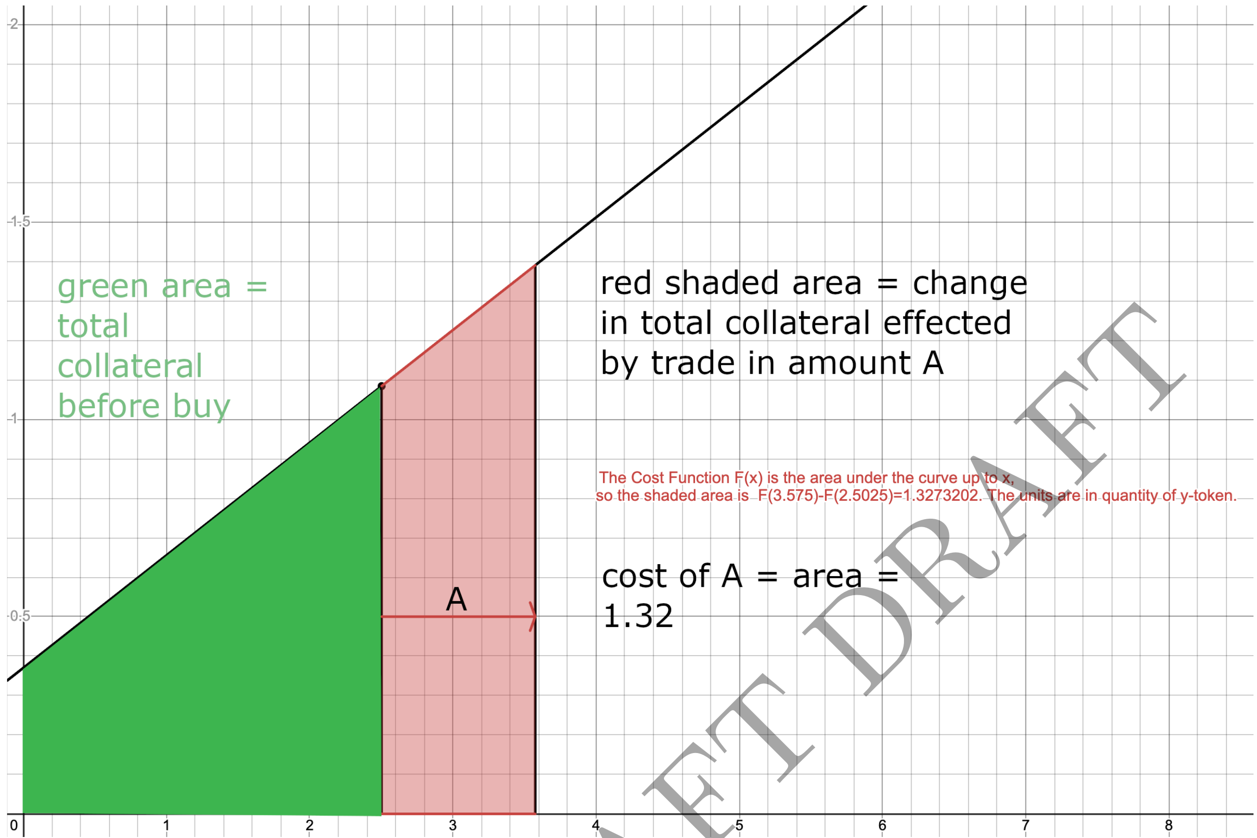



Figure 5: The cost function used to evaluate the cost of a buy order.



### 2.5 Level Curves, the Concentration Parameter C, and Generalizing the TBC

Let us now build some intuition for how the line adapts. In the Dynamic AMM, every $x$ value is mapped to a unique slope value. This is achieved with a slope field, which is defined as the following family of level curves:

$$g(x) = \ln(x + C) + K$$

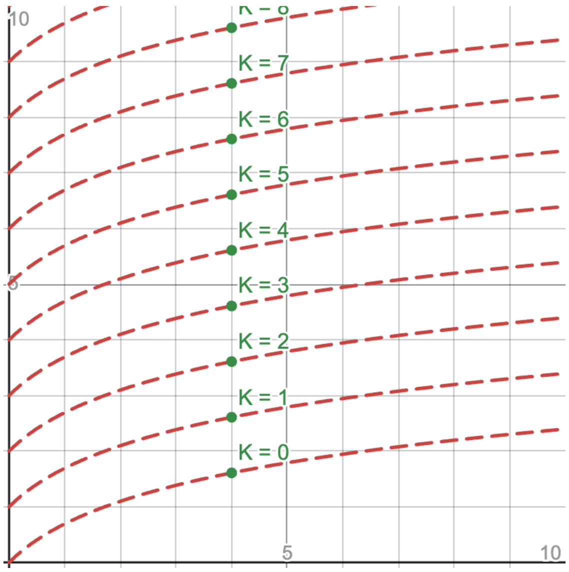
$K$ simply shifts the log curves up or down. It appears here to help illustrate the family of level curves, but is not a genuine Dynamic AMM variable.

**C parameter**

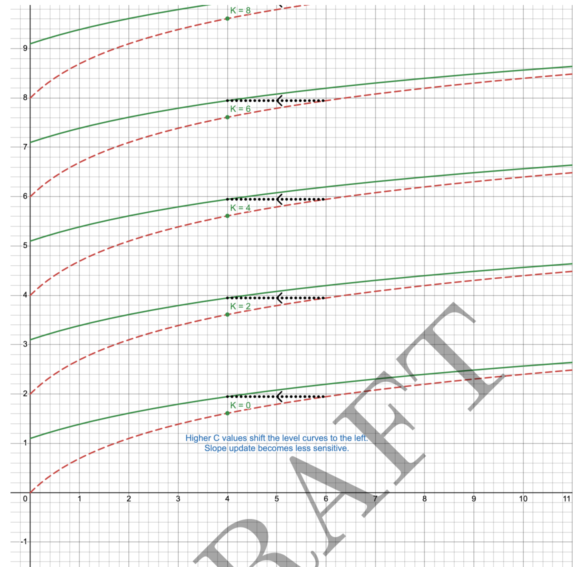
We term $C$ the "concentration parameter" because it controls the degree of concentration of price in a given supply range. C achieves this by shifting the level curves left and right, effectively controlling the slope update's sensitivity to changes in supply. Geometrically, higher C values shift the log curves to the left. This means that the level curves are closer to flat at lower values of supply, which produces greater price stability or, equivalently, less sensitivity in slope updates. See figure 6.

| Description | Variable | Bounds |
|---|---|---|
| Concentration Parameter | $C$ | $C = \frac{x_{\text{add}}}{p_{\text{add}} - p_{\text{lower}}}$ |

Table 3: C: The Concentration Parameter




(a) Level curves (or "generating curves") with C = 1




(b) Level curves shifted (C = 3). Shifting the log curves to the left means that they are closer to level at lower $x$-values.

Figure 6: Comparison of level curves for different values of C

**Formal Definition of C**

C is determined upon instantiation by $x_{\text{add}}$, $p_{\text{add}}$, and $p_{\text{lower}}$. We set the value of $C$ in terms of these variables to be

$$C = \frac{x_{\text{add}}}{p_{\text{add}} - p_{\text{lower}}} = \frac{1}{\frac{p_{\text{add}} - p_{\text{lower}}}{x_{\text{add}} - 0}} = \frac{1}{g'(0)}.$$

Note that the derivative of (any) $g$ (as defined in section 2.5) is

$$g'(x) = \frac{1}{x + C}.$$

When you plug in $x = 0$ to $g'(x)$, you get the slope of the tangent line to (any) $g$ at $x = 0$. Thus C is always equal to the reciprocal of the slope of the tangent line to $g$ when $x = 0$.

| Description | Variable | Bounds |
|---|---|---|
| Concentration Parameter | $C$ | $C = \frac{x_{\text{add}}}{p_{\text{add}} - p_{\text{lower}}}$ |

Table 4: C: The Concentration Parameter

**Slope is a deterministic function of supply, generalizing the TBC**

The line $f_n(x)$ will be tangent to the level curve which passes through the point $(x_n, p_n)$,[9] where $x_n$ is the current $x$-value and $p_n$ is the current spot price. Note that this means that while a given $x$-value does not pick out a unique price (because there are many lines with the correct slope), it *does* pick out a unique slope $b_n$. We term the level curve associated with a given $x$-value and price the *generating curve*. We term the property of a unique slope for every supply value the "path independence of slope" (this is an important property of the Dynamic AMM; see section 2.6).

Note that the path independence of slope property means that slope, not price, is a deterministic function of supply. Another way to say this is that **price impact** rather than **price** is a function of supply. In this respect, we consider the Dynamic AMM to be a generalization of the idea of a TBC.

[9]The point of tangency will be the point $(x_n, p_n)$.

## 2.6 Path Dependence and Path Independence in the Dynamic AMM

As alluded to above, when it comes to path independence in the Dynamic AMM, we focus on **price** and **slope.** It is important to understand that the Dynamic AMM differs from conventional TBCs with respect to these two properties:

- The **asset price in the Dynamic AMM is path dependent**. More specifically, one can show that if $x_n$ is equal to $x_{n+\Delta n}$ then $f_n(x_n)$ is strictly less than $f_{n+\Delta n}(x_{n+\Delta n})$.
- The **slope** $b$ **of the Dynamic AMM is path independent**. That is to say, if $x_n$ is equal to $x_{n+\Delta n}$ then $b_n$ will always be equal to $b_{n+\Delta n}$.

In other words, if the Dynamic AMM starts at some x-value and then returns to that x-value after some path of transactions, the price will have risen since the first visit (path dependent). However, the slope will be the same as the first visit (path independent).

## 2.7 Limiting TBCs vs Intermediate TBCs

Let us introduce two concepts:

- An **intermediate TBC**, which is simply a way of referring to an individual $f_{\mathrm{n}}(x)$ pricing function.
- A **limiting TBC**, which is a curve representing optimal obtainable prices for the trader. These prices are obtainable if we grant the trader the ability to split their orders into infinitely many smaller sub-orders; it is not, in other words, generally going to resemble the actual price curve yielded by the Dynamic AMM. Nevertheless, the concept gets to the heart of the Dynamic AMM and is important to understand. See section 4.5.

The limiting TBC at any point is given by the equation

$$f_{\lim}(x) = p_n + \frac{1}{2}\left(\frac{x}{x+C} - \frac{x_n}{x_n+C}\right) + \frac{1}{2}\ln\left(\frac{x+C}{x_n+C}\right)$$

where $x_n$ is the $x$ value at time $n$, $p_n$ is the current price, and $C$ is the concentration constant. An example of an intermediate TBC, and its accompanying limiting TBC, is shown in figure 7.

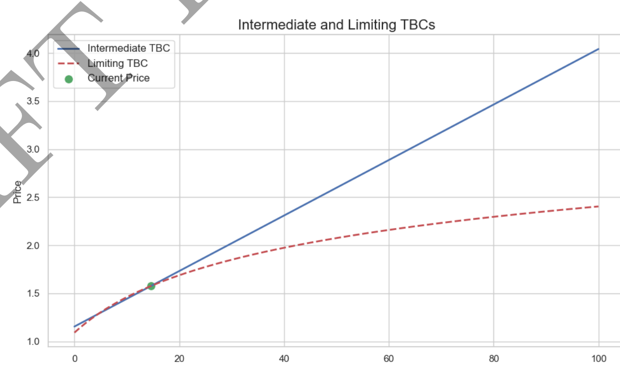



Figure 7: An example intermediate TBC and limiting TBC. The limiting TBC in red shows the prices a trader would obtain if infinitesimal order splitting were possible.

## 2.8 Liquidity Adds and Withdrawals

It may seem counterintuitive to speak of adding liquidity to a TBC. Nevertheless, as previously observed, a TBC is mathematically equivalent to an AMM defined for a finite number of tokens. Thus it is possible to add liquidity to a TBC, and it is therefore possible to deploy the Dynamic AMM as a general purpose pricing mechanism.

In order to add liquidity to the Dynamic AMM, it is necessary to consider the ratio of assets being put up as liquidity. A given Dynamic AMM state implies a ratio of one asset to the other.[10] So, for example, if

[10] In a sense, that's what a price is.

the price of one $x$-token is 10 $y$-token, a liquidity add in the amounts of 10 $y$-token and 1 $x$-token would be exactly proportional.

What if some party wishes to provide liquidity in amounts other than the current proportion? In that case, the Dynamic AMM follows convention in the DeFi industry by simulating the swap that would render the token amounts to be in correct proportion, and then performing the resulting (correctly proportioned) liquidity add. See the Appendix for a detailed mathematical explanation of how liquidity adds and proportion adjustments are achieved.[11]

#### Tracking Liquidity Positions and Pool Ownership

The introduction of liquidity positions requires a mechanism for tracking pool ownership. This is achieved by issuing NFTs that track the value of a given liquidity position (as a fraction of the total amount of liquidity in the pool) and the cumulative amount of value so far redeemed by the NFT owner in question.

### 2.8.1 Example Liquidity Add Flow

| **Liquidity Add Flow** | |
|---|---|
| **Step** | **Description** |
| 1 | Liquidity Provider submits a call to the Dynamic AMM to add liquidity amounts in quantities A and B. |
| 2 | Simulated swap adjusts A and B to be in the correct proportion (A*, B*). If adjustments were needed, Dynamic AMM state changes: $x_{\text{min}}$, $x_{\text{max}}$, $x_{\text{current}}$, $D_V$, $b$, $c$, and $C$ are all updated. |
| 3 | A* and B* are added to the pool. |
| 4 | An NFT is minted to the Liquidity Provider indicating the value of the created position. |

Table 5: Liquidity Add Flow

Table 5 diagrams the process of adding liquidity. It is important to note that **a liquidity add can affect the spot price**.[12] It also affects the other state variables: slope ($b$), the $x_{\text{max}}$ value, the $D$ value, the amount of virtual collateral $D_V$, and the concentration parameter $C$.

## 2.9 Pool Ownership and Revenue Extraction

Note that the existence of these LP tokens implies that ownership of the pool is shared among liquidity providers (the amount of value stored in the NFT represents the portion of the entire pool that belongs to the NFT owner). Holders of these NFTs, therefore, can use them in two different scenarios:

1. When withdrawing liquidity from the pool.
2. When distributing accumulated excess capital from the pool to liquidity providers.

The liquidity removal is straightforward; the LP can pull out the amount of liquidity originally added.[13]

#### Extracting Excess Capital

The second use for the LP NFT – regarding excess capital – is less straightforward. What is this excess capital? Recall the remark above, that for a given $x$-token value, collateral continually accrues to the market owner with each transaction.[14] Most of that collateral will supply liquidity for trades in the reachable domain of the Dynamic AMM – that is, $x_{\text{min}} < x < x_{\text{max}}$. But some of that increased collateral will **not** be accessible to traders. That is, even if all available $x$-token were sold back to traders, some collateral would be left over.

This is illustrated in figure 8. In that illustration, we see an initial price function $f_{\text{n}}(x)$. The area under that curve on $(0,\, x_{\text{min}})$ is our virtual collateral, $D_{\text{V}}$. Suppose we begin at $x = x_{\text{min}}$, some trading takes place, and then there is a large selloff so that we return to $x = x_{\text{min}}$. Our new price curve will be higher:

[11] Note that the "swap" here is a simulated swap, so the gas fees associated with such a transaction are not incurred. But the Dynamic AMM state changes just as if the swap went through.

[12] Liquidity changes, in fact, will in general do this, except in the rare case where A and B are exactly in proportion.

[13] Note that this withdrawal function will pay the LP with assets in the proportion implied by current AMM state.

[14] We term this property the monotonicity of spot price for given supply value; see Appendix.

the Dynamic AMM will look like the green green line $f_i(x)$. Much of the accumulated collateral will have been issued back to traders during the selloff, but an amount will remain even after the last token has been sold back to the market. This amount is highlighted in green, and it will become the property of the market owner(s).

It is this quantity that must be shared by market owners (i.e., liqudity providers), in amounts corresponding to the size of their positions. Thus, the LP NFTs must also track the cumulative amount of revenue so far paid out to LP position holders, so that they can redeem up to (but not more than) the amounts they own. Table 6 shows the basic information that an LP NFT must store.

| Field | Meaning |
|---|---|
| `token_id` | Unique identifier for the LP NFT. |
| `amount` | Represents the portion of liquidity ownership associated with this NFT. |
| `revenue_parameter` | Ensures the correct amount of excess capital is paid out to NFT holder. |

Table 6: LP NFT Fields

**Withdrawing Liquidity**

When a liquidity provider wishes to withdraw a portion of the liquidity stored in an LP NFT, three things must happen (see Appendix for a more detailed explanation):

1. An amount of $x$-token and $y$-token corresponding to the value of the NFT is issued to the LP.
2. An amount of excess capital belonging to the LP is issued to the LP.[15].
3. The Dynamic AMM state is updated to reflect the resulting change.

As mentioned above, LPs can use their LP NFTs to redeem value in two ways: they can withdraw the liquidity they originally added, or they can redeem the amount of excess collateral to which their LP position entitles them.

These two functions are separate, but related. An LP can request a revenue payout without withdrawing any liqudity, but a liquidity pull automatically triggers a proportional revenue payout.[16]

In either case, whenever revenue is paid out, the LP NFT in question must be updated (in the **revenue_parameter** field) so that the LP can never redeem more revenue than is owed.

## 2.10 Note on Interpreting the Pricing Curve

It may be useful to think about how to interpret the variables stored in the Dynamic AMM.

Let us begin with $x_n$. Typically, in TBC space, we think of the $x$-axis as representing the total number of tokens minted by the TBC. That is the case for the Dynamic AMM – until someone chooses to add liquidity to the pool.

In a TBC that accepts external liquidity changes, $x_n$ does **not** represent total outstanding supply. Instead, the proper way to think of $x_n$ is the following:

> In a TBC that accepts liquidity additions and withdrawals, $x_n$ is the **amount of $x$-token required to drain the pool of all its collateral.**

Now, let us consider the user-supplied variable $x_{\text{add}}$. This represents the initial maximum number of tokens for sale by the Dynamic AMM. However, it is **not** the highest value that $x_n$ can achieve. Remember that the reachable trading region is offset by the presence of $x_{\text{min}}$. Thus the highest reachable $x$-value is actually $x_{\text{max}}$

Finally, let us reiterate that the value of $x_{\text{max}}$ is subject to change with the addition of new liquidity. We can interpret this as meaning that, when an LP adds liquidity to the pool, more $x$-token is now for sale (and, correspondingly, more $y$-token is held as collateral.)

[15]This amount must take into account the cumulative amount of revenue so far paid out to the LP in question

[16]The reason for this is subtle: if the liquidity were paid out without the corresponding revenue payout, it would become impossible to recover the revenue in question.

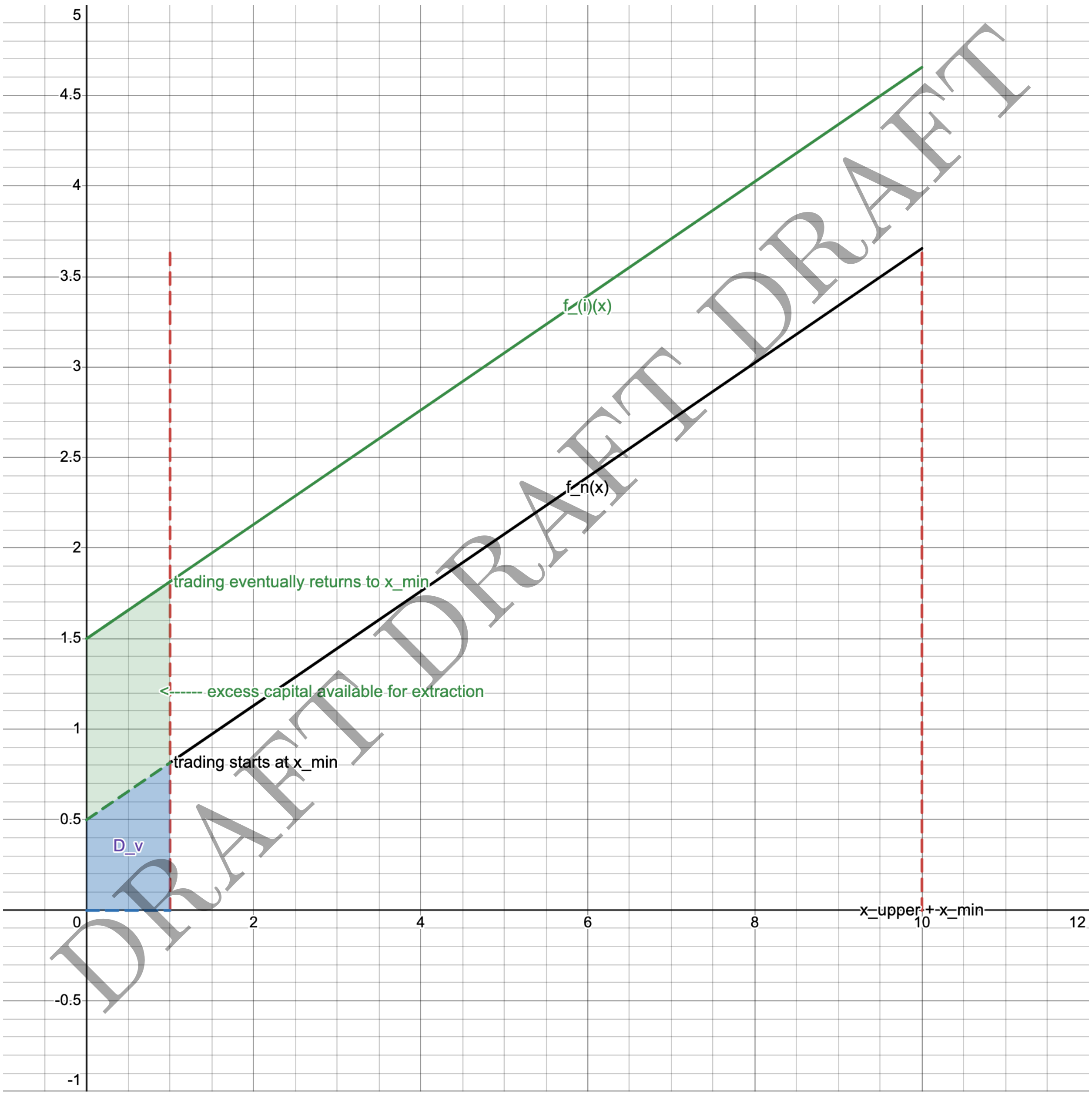



Figure 8: Excess capital accrues at x-values below $x_{\min}$. This excess value is available to market owners via the revenue function.

## 2.11 Setting initial Dynamic AMM parameters

Above, we have discussed the various parameters used to initialize the Dynamic AMM, and which parameters it maintains over time. Using those variables, however, it is not immediately obvious how to set initial values. In order to facilitate the selection of reasonable initial values, we have developed logic that requires the user to specify four easily interpretable values:

1. `l`: the specified multiple of initial price amount.
2. `Q`: the cost to the exploiter[17] of pushing the initial price up by a factor of $l$.
3. `T_target`: a certain number of tokens to sell, such that item (4) is true.
4. `p_target`: the minimum price of the asset after `T_target` tokens have been purchased.

The reasoning behind these variables, and exactly how they are used to compute the internal parameters of the Dynamic AMM, is explained in the Appendix.

# 3 Transaction Flow

## 3.1 8-step Transaction Flow

During each transaction, various calculations are made and variables are updated. These happen in a specific order, which is shown in table 7.[18]

Table 7: 8-step transaction flow

| Step | Variable to Update | Explanation |
|---|---|---|
| 1 | $A$ | $A_n$ is computed in four ways given below in 1a, 1b, 1c, and 1d. |
| 1a | The user submits an order to buy a quantity $q$ of $x$-token. | $A_n = +q$ |
| 1b | The user submits an order to sell a quantity $q$ of $x$-token. | $A_n = -q$ |
| 1c | The user submits an order to sell a quantity $q$ of $y$-token. | $A_n = F_n^{-1}(F_n(x_n) + q) - x_n$ |
| 1d | The user submits an order to buy a quantity $q$ of $y$-token. | $A_n = F_n^{-1}(F_n(x_n) - q) - x_n$ |
| 2 | IF $x_{\min} \leq x_n + A_n \leq x_{\max}$, proceed | Swap must remain within reachable trading bounds. |
| 3 | $x$ | $x_{n+1} = x_n + A_n$ |
| 4 | $D$ | A new area under the TBC curve in range $[0,\ x_{n+1}]$ is evaluated: $D_{n+1} = \frac{1}{2} b_n x_{n+1}^2 + c_n x_{n+1}$ |
| 5 | $s$ (this is equal to slope of price function line, see step 7) | New value for $s$ is evaluated: $s_{n+1} = \frac{V}{x_{n+1}+C}$ |
| 6 | $c$ | $c_{n+1} = \frac{D_{n+1} - \frac{1}{2} s_{n+1} x_{n+1}^2}{x_{n+1}}$ |
| 7 | $b$ | $b_{n+1} = s_{n+1}$ |
| 8 | $p$ | A new price is evaluated: $p_{n+1} = f_{n+1}(x_{n+1})$ |

## 3.2 Illustration of Transaction Flow

Let us focus on the steps in table 7 a few at a time and illustrate them. First, we give ourselves a starting point. In figure 9, we see the Dynamic AMM awaiting a trade.

[17] See section 4.6

[18] Note that the following illustration proceeds with a $k$ value of 0 for simplicity (i.e. $x_{\min} = 0$).

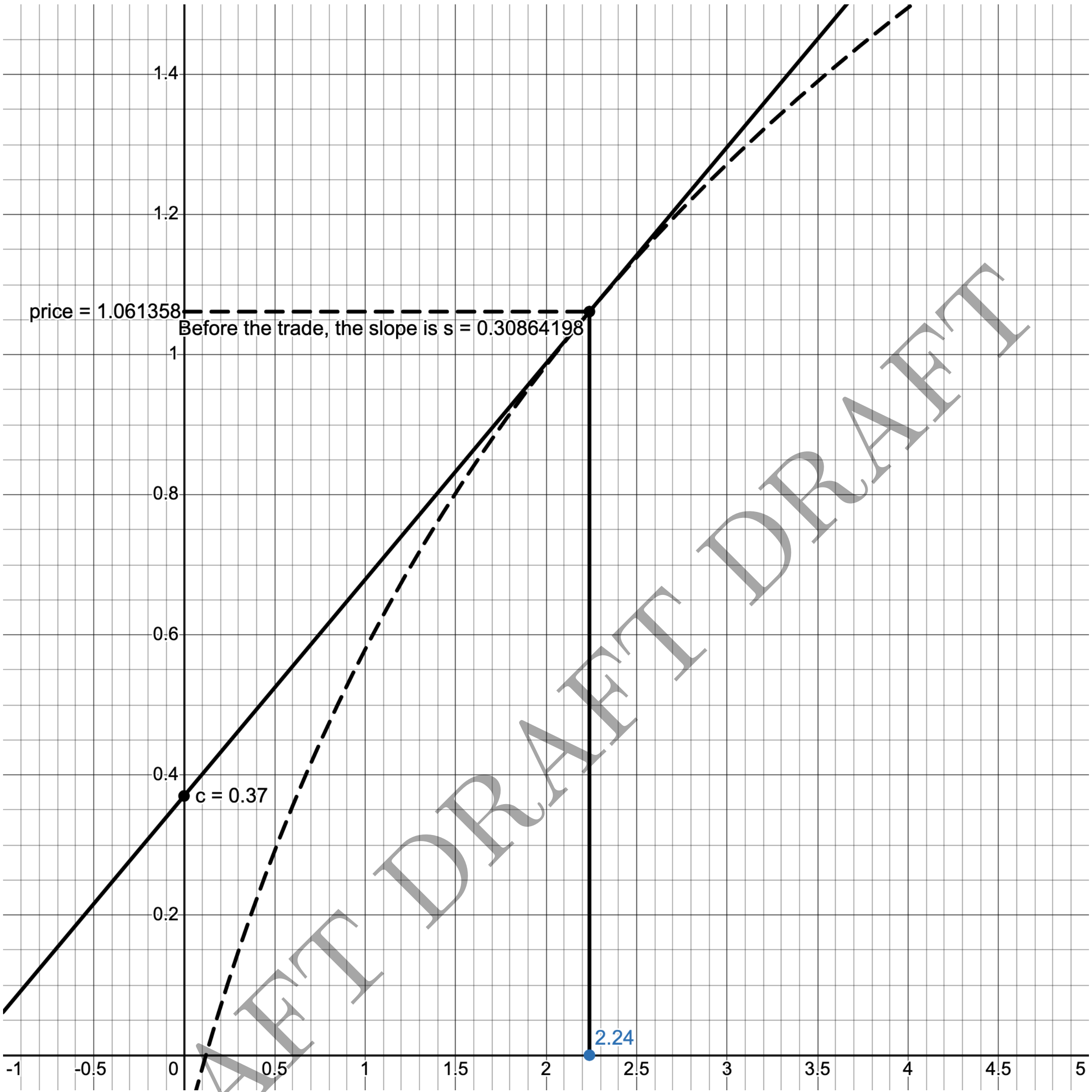
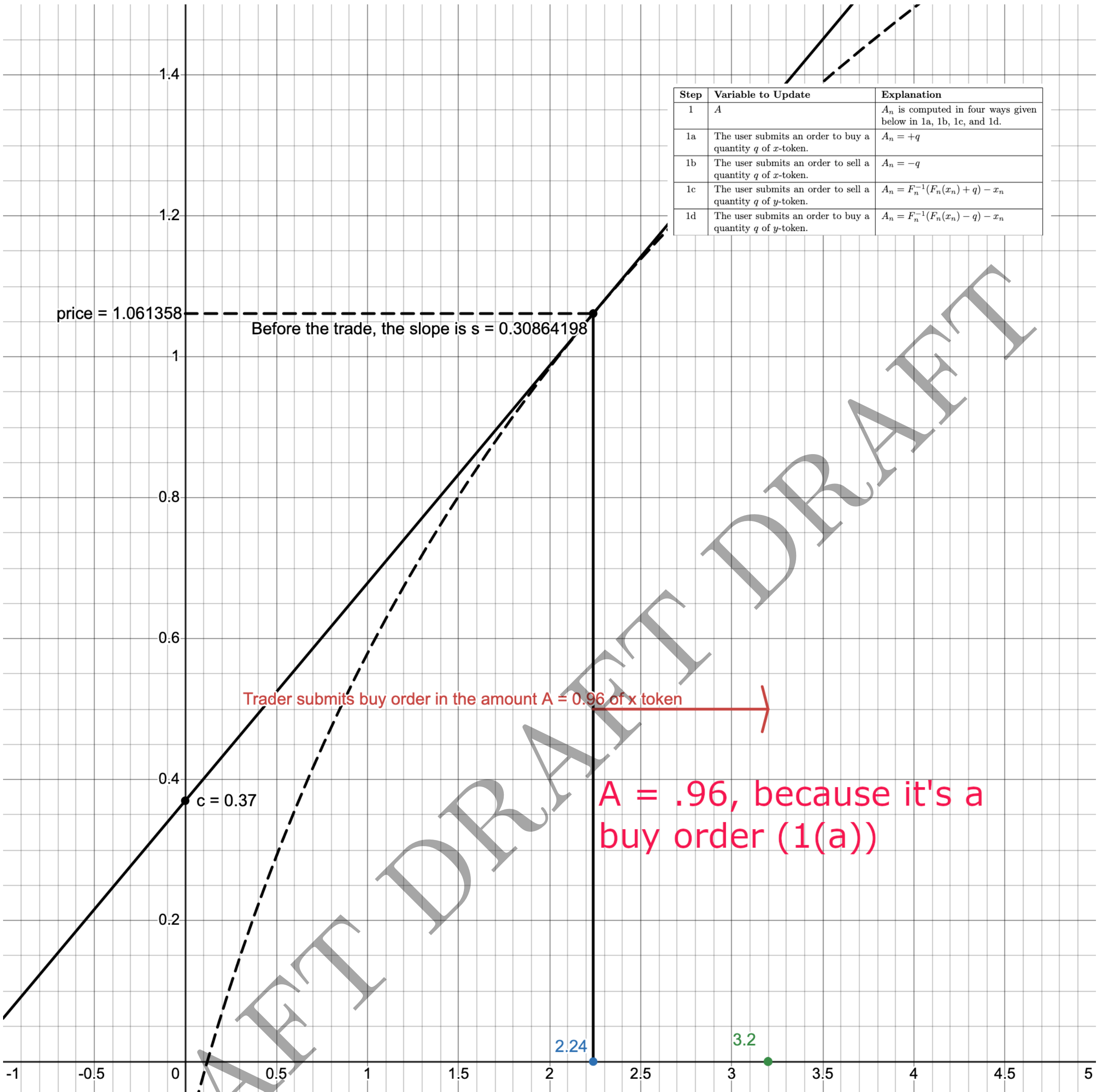



Figure 9: The ALTBC, before a trade. The generating level curve is dotted, while the pricing function is a solid line.



Suppose a trader submits an order. The first thing the ALTBC must do is set the value of A, which is the amount of $x$-token in or out. If the order is a buy or sell of x-token, A is simply set to that value. If it's an order for a buy or sell of y-token, however, we must use the inverse cost function (see 2.4) to evaluate the corresponding amount of $x$-token (see steps 1(c) and 1(d) in table 7). In figure 10, we see the value A being set, in the simple case of a buy of $x$-token.




| Step | Variable to Update | Explanation |
|---|---|---|
| 1 | $A$ | $A_n$ is computed in four ways given below in 1a, 1b, 1c, and 1d. |
| 1a | The user submits an order to buy a quantity $q$ of $x$-token. | $A_n = +q$ |
| 1b | The user submits an order to sell a quantity $q$ of $x$-token. | $A_n = -q$ |
| 1c | The user submits an order to sell a quantity $q$ of $y$-token. | $A_n = F_n^{-1}(F_n(x_n) + q) - x_n$ |
| 1d | The user submits an order to buy a quantity $q$ of $y$-token. | $A_n = F_n^{-1}(F_n(x_n) - q) - x_n$ |



Figure 10: A trade arrives, and A is set accordingly.

We come to step 2 in table 7. The ALTBC must verify that A does not violate the maximum supply constraint set upon instantiation. In this case, the order size A = .96 keeps supply within bounds, so it is allowed to execute.

Now the ALTBC evaluates the cost of this order (see section 2.4). In this case, this is the integral under the current price function line between current supply (= 2.24) and current supply + A (= 3.2). See figure 11.

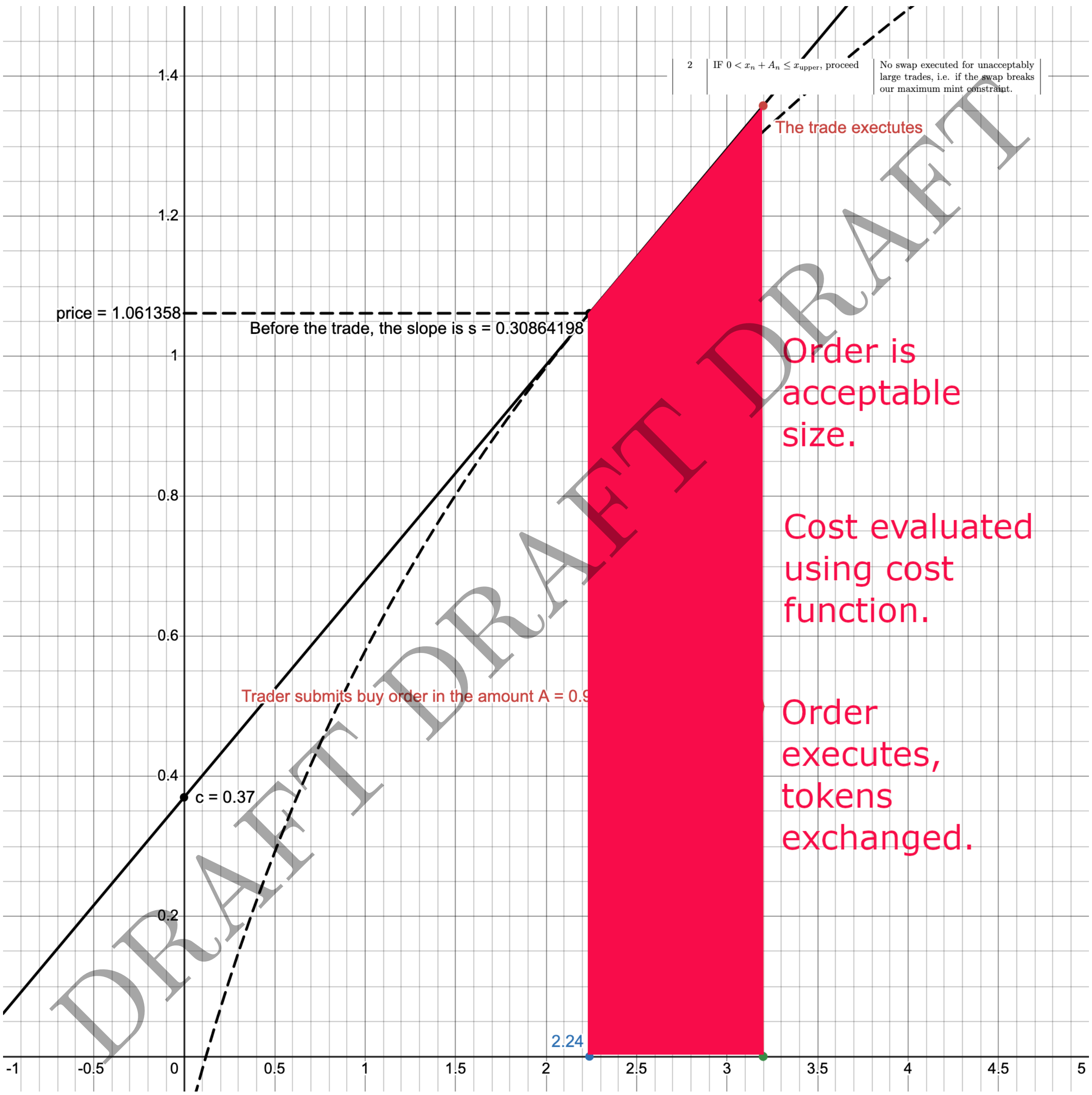
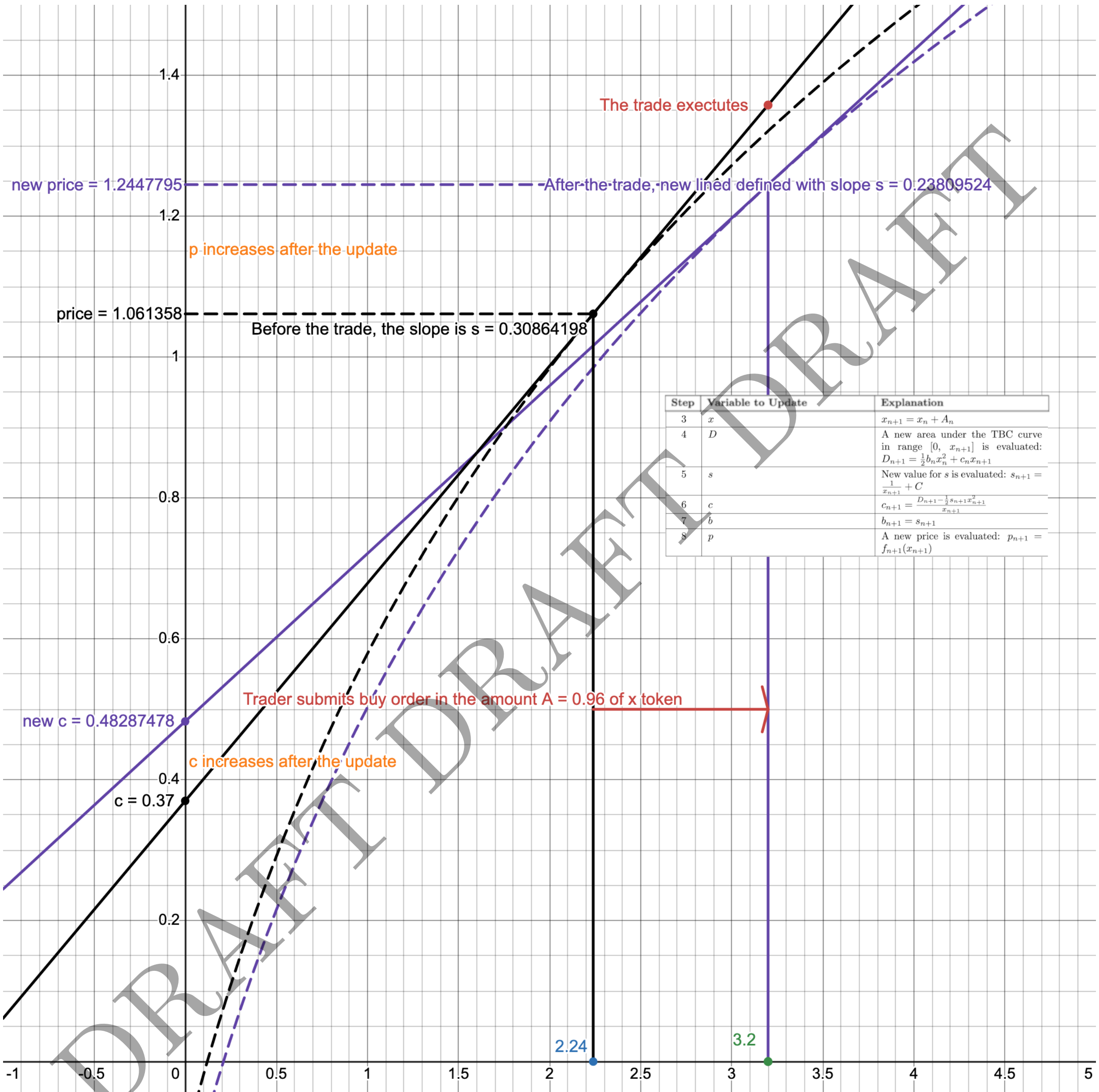




Figure 11: The trade size is checked, the cost is evaluated, and the order executes.

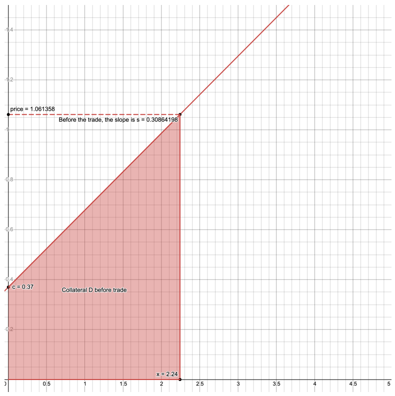
We come to steps 3-8 in table 7: updates to the variables in the pricing function. This is depicted in figure 12. Note the various movements in ALTBC variables, tracking table 7: $x$ goes up, slope goes down, price goes up, the $y$-intercept c goes up. Also depicted is the definition of a new level curve adapted to this supply and price. The collateral update ($D$) (step 4 in table 7) is depicted in figure 13.[19]



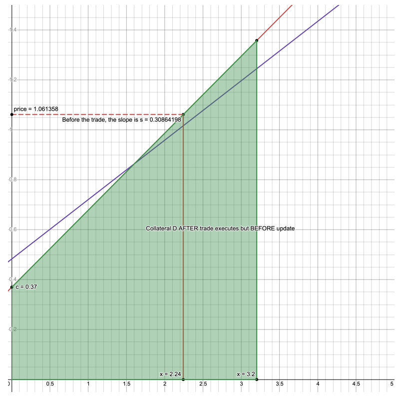

| Step | Variable to Update | Explanation |
|---|---|---|
| 3 | $x$ | $x_{n+1} = x_n + A_n$ |
| 4 | $D$ | A new area under the TBC curve in range $[0,\ x_{n+1}]$ is evaluated: $D_{n+1} = \frac{1}{2}b_n x_n^2 + c_n x_{n+1}$ |
| 5 | $s$ | New value for $s$ is evaluated: $s_{n+1} = \frac{1}{x_{n+1}} + C$ |
| 6 | $c$ | $c_{n+1} = \frac{D_{n+1} - \frac{1}{2}s_{n+1}x_{n+1}^2}{x_{n+1}}$ |
| 7 | $b$ | $b_{n+1} = s_{n+1}$ |
| 8 | $p$ | A new price is evaluated: $p_{n+1} = f_{n+1}(x_{n+1})$ |

Figure 12: The pricing rule updates, yielding a new price, slope and c value.

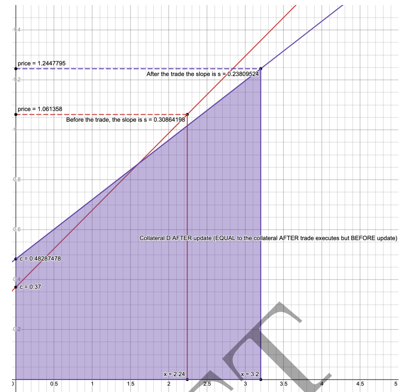
[19]Note here the collateral preserving design of the ALTBC: the area under the first price line (pre-update) on the interval (0,3.2) is equal to the area under the second price line (post-update) on that same interval.




(a) Collateral (D) before buy of size A.




(b) Collateral (D) after buy order of size A




(c) Collateral (D) after buy order of size A, after line update.

Figure 13: Step 4: collateral (D) update with buy order of size A. Collateral preservation: the area in purple equals the area in green.

In figure 14 we remove some of the extraneous information to show the difference between the original price function line and the new (current) price function line.


Figure 14: Comparison of just the first and second pricing function lines.

# 4 Notable Properties

In this section, we focus on several of the advantages and notable properties of the Dynamic AMM.

## 4.1 Directional consistency of spot price.

The price rule changes as traders buy and sell over time. Specifically, the slope of the TBC line decreases as outstanding supply increases. In spite of this decrease, the Dynamic AMM will always obey the law of supply and demand; namely, the price will always increase with buy trades, and will always decrease with a sell trades.

## 4.2 As $x$ increases, volatility decreases.

As $x$ values go up, the slope of the line will decrease. This means that the price impact of a given trade will be smaller and smaller the more $x$-token is minted. Note that this contour (volatility decreasing as more $x$-token is in supply) is the opposite of a Uniswap pool. See section 5.

## 4.3 Price and collateral are monotonic at any given $x$-value.

Suppose the TBC has minted 100 tokens at transaction #10. The current price f(x) = 200. Then, 3 transactions arrive: a buy of 10, a sale of 15, and a buy of 5. The $x$-value traces the path 100, 110, 95, 100 – a round trip returning to 100. What has happened to the collateral and the price? This property says that both are always higher at transaction #13 than at transaction #10.

## 4.4 Innate MEV sandwich attack mitigation

A sandwich attack operates by "sandwiching" a given trade by two trades designed by the exploiter. The first trade pushes the price up. The "sandwiched" trade settles at a less favorable price than anticipated. Then, the attacker sells at a slightly better price than they purchased at, pocketing the difference at the sandwiched trader's expense.

No AMM can completely remove the profitability of these kinds of attack.[20] The Dynamic AMM, however, establishes an upper bound on their profitability. The Dynamic AMM's stability, along with its collateral preservation property, make it so that only a finite range of sandwich attack sizes are profitable for a given target trade. The section below illustrates this property.

### 4.4.1 Illustration of innate MEV sandwich attack mitigation.

Let us fix the target trade at 1 $x$-token coins. Let us also fix the various parameters internal to the Dynamic AMM (the concentration parameter C, for example) and designate an initial $x$-value for the Dynamic AMM of 2.

The attacker now has just one decision to make: the magnitude of the sandwich attack itself. Let us designate this magnitude as $S$. The attacker wishes to extract as much profit as possible, i.e. the difference between the initial purchase price for $S$ tokens and the subsequent sale revenue for $S$ tokens should be maximized (with the sale price being greater). In the Dynamic AMM, some $S$ values will yield profits (see figure 16). However, if the sandwich size is too large, the sandwich attack will produce a loss (see figure 17).

This offers a crucial advantage over conventional liquidity solutions. Note that in the case of Uniswap V2, MEV is mathematically unbounded.[21] In the Dynamic AMM, however, there is an upper limit on how much a sandwich attacker can earn, even before fees and slippage tolerances enter the picture. Figure 15 shows this bound clearly by plotting the attacker's profit and loss as a function of sandwich attack size. See Appendix A.5 for the derivation of MEV as a function of attack size.

## 4.5 Bounded incentive for trade-splitting

Recall that the main feature of the Dynamic AMM is that the slope of the line decreases as outstanding supply increases. A natural, but not immediately obvious, consequence of this property is that for any given trade, it will be in the trader's interest to split that order up into many smaller orders. We can think of this feature as the Dynamic AMM making a compromise. It sacrifices a small amount of "size consistency"[5], in order to make significant gains in flexibility, price stability, security against MEV attacks, and profitability.

[20] Except one that provides a constant price. See [5]

[21] In practice, user-supplied slippage tolerances are required to prevent this fact from causing problems.


Figure 15: MEV as a function of sandwich attack size, fixing the victim trade size and other parameters. An attack size 1.003 is maximally profitable, whereas an attack size 3 yields a loss.

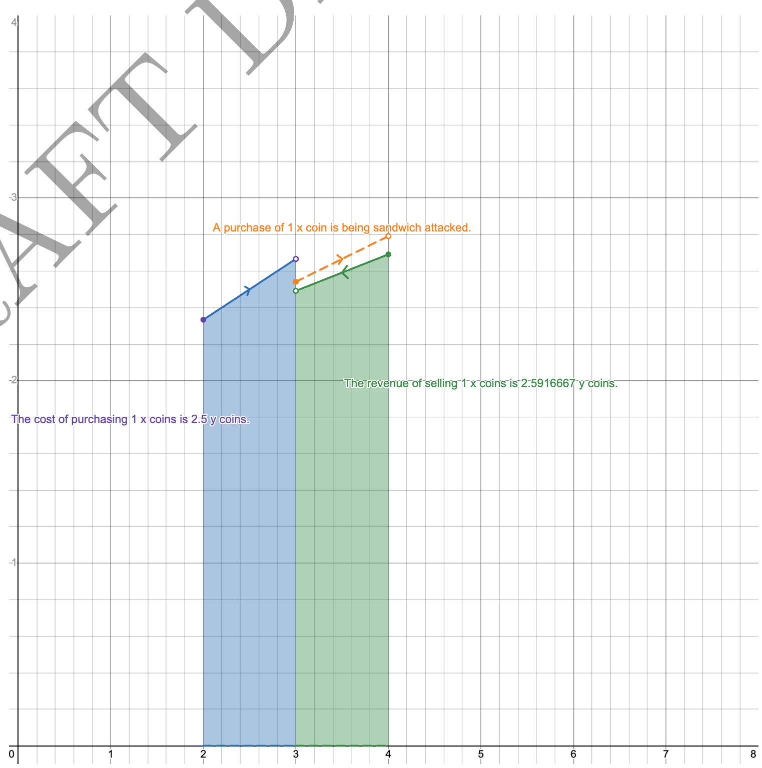



Figure 16: Fixing victim trade size, starting supply, and other parameters, a sandwich attack of size 1 is profitable.

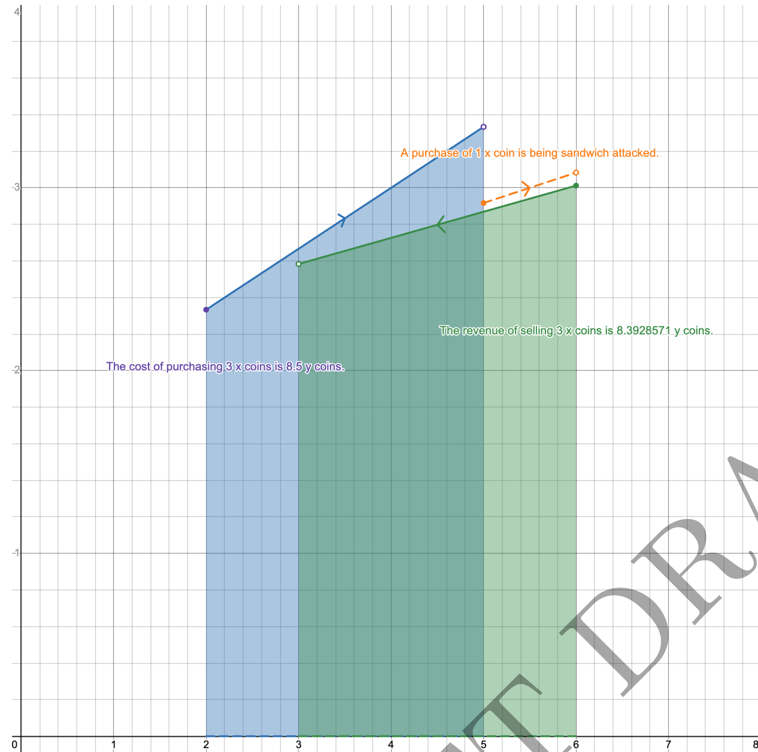



Figure 17: Fixing victim trade size, starting supply, and other parameters, a sandwich attack of size 3 produces a loss for the attacker.

All liquidity solutions make tradeoffs with these properties. In the section below, we explain how the Dynamic AMM engenders this incentive, and why we think it does not represent a meaningful downside to the design.

### 4.5.1 Illustration of Bounded Incentive to Split

Consider an order of some fixed size, split up into $n$ separate orders. As $n$ grows, the number of times the slope of the price curve decreases also grows. Every price update represents an improvement, from the perspective of the trader, over a smaller number of order partitions. Thus it is optimal for the trader to split orders into infinitely many sub-orders.[22] This infinite splitting yields an optimal curve (representing the best achievable price for a given order size, from a given supply starting point). As noted above, these best-achievable prices constitute the *limiting TBC*.

#### Note on the Limiting TBC

The optimal curve (or, alternatively, the Limiting TBC), representing the best achievable price for a given trade, assuming infinitesimal trader splitting, has a closed-form expression, already given above:[23]

$$p_0 + \frac{1}{2}\left(\frac{x}{x+C} - \frac{x_0}{x_0+C}\right) + \frac{1}{2}\ln\left(\frac{x+C}{x_0+C}\right)$$

where $p_0$ is the spot price, $x_0$ is the current outstanding supply, and $x$ is the *next* outstanding supply value (i.e., the one resulting from the buy or sell of $|x - x_0|$ $x$-token.)

Note that this optimal curve, plotted as the dashed line below, is **not** the Dynamic AMM. It represents an ideal (but fundamentally not achievable) price for a given order size, from the reference point of a single supply position. In practice, every trade moves the Dynamic AMM up or down a given $f_n(x)$ price function, which will yield something very different from the optimal curves shown in the plots below.

[22] Optimal in the specific sense that the resultant spot price is lowest.

[23] If the reader is interested, we supply a derivation for this formula, which is a consequence of Riemann sums, in Appendix A.4.

**Analysis of Incentive to Split**

How does the incentive to split work in practice? Two factors matter most here: the number $n$ of partitions, and the starting supply position $x$ from which the partitioned trade is initiated. Obviously, as $n$ grows, the difference between the optimal price and the achieved price decreases (that is the definition of the incentive to split). However, note that this difference also decreases as $x$ grows.

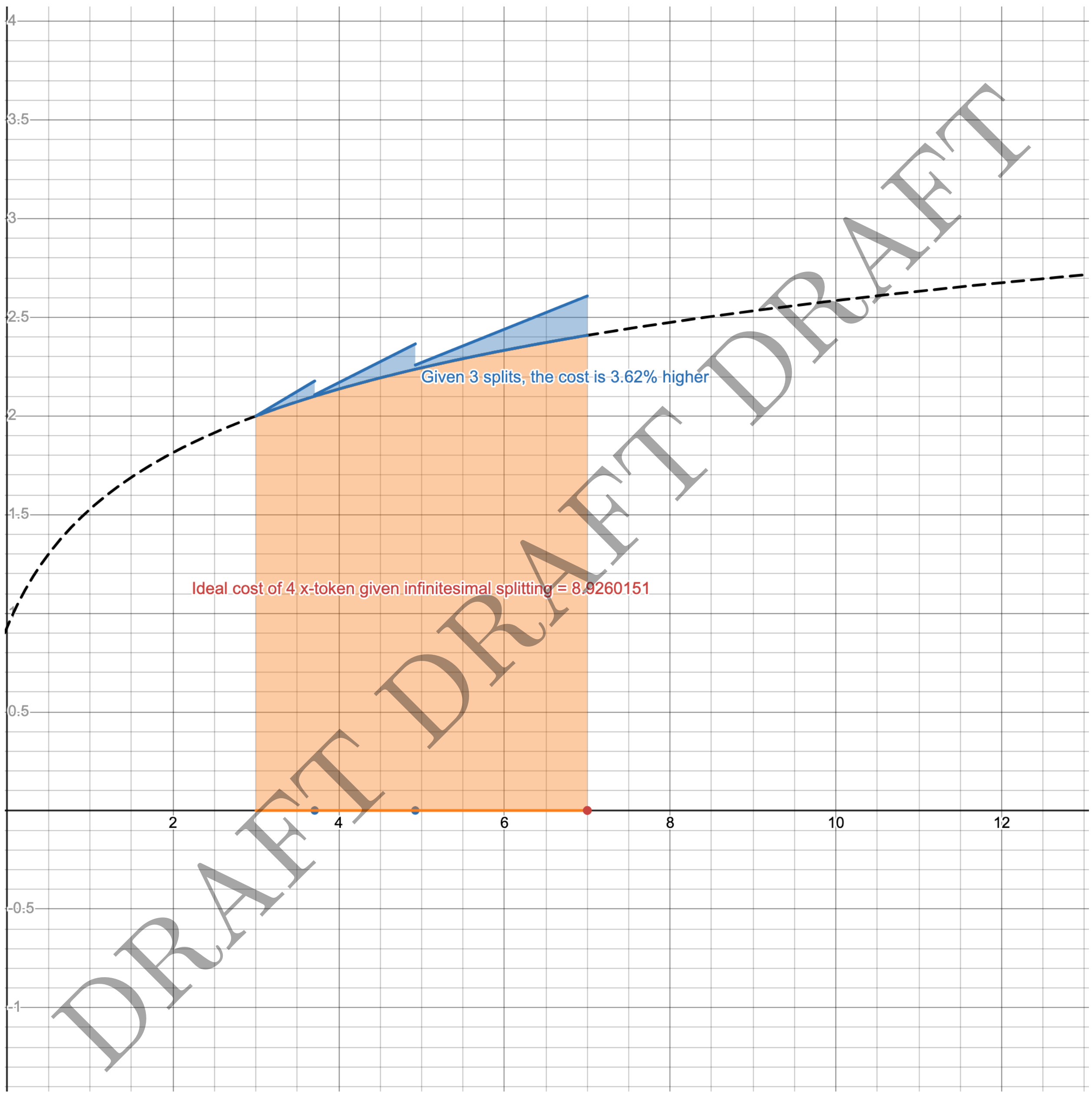





Figure 18: In a low supply setting, 3 partitions lead to a price that is 3.62% higher than optimal.

Figures 18 and 19 show this effect. Let us fix the partition size at 3. In figure 18, we see this partition strategy at work in a relatively low supply situation (initial $x = 3$). These three partitions yield a realized price that is 3.62% higher than the ideal price. Traders might well, in this scenario, feel strongly about how many partitions they are allowed.

In figure 19, on the other hand, we are in a higher supply situation (initial x = 30). Here, the same partition strategy achieves a much closer-to-optimal price outcome. What this means is that, as supply increases, traders care less and less about trying to optimize their partitions, and the incentive will be weaker.

Thus, the incentive to split only really applies in relatively low-supply settings. Even in those settings, however, traders are unlikely to actually deploy complex partition schemes. This is because such schemes take time and deliberation, and because at lower supply, the volatility of the asset is always highest. Given

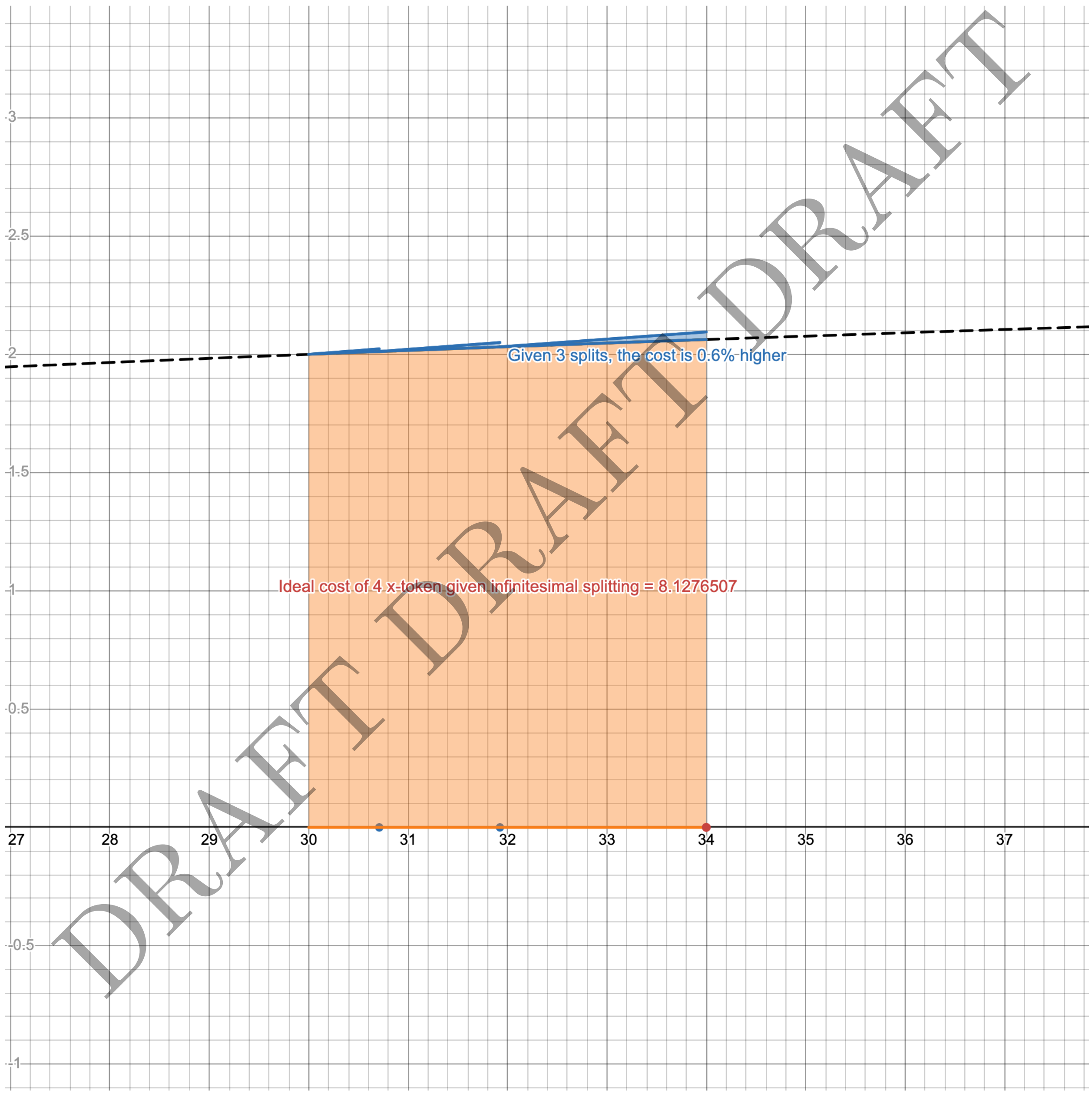



Figure 19: In a higher supply setting, the price after three partitions is already close to the optimal price.

the higher price volatility and greater competition in the low supply setting, the risk of destructive levels of partition optimization is relatively low.

#### Optimal Order Partitions are Not Uniform

A note on the optimal orders above: while the optimal curve is achievable with uniform infinitesimal splitting, for any given number of partitions the order sizes are non-uniform; hence the uneven size of the partitioned orders in figures 18 and 19. Attackers wishing to obtain optimal order splits face a non-trivial analysis problem, which they would have to execute quickly in the context of the high pressure that often attends assets traded in low-supply circumstances. We note this here because this mathematical obstacle is one of the things that makes implementations of optimal splitting less likely in real life.[24] This is especially true in low-supply settings (the only settings where such schemes really matter), when traders are jockeying for $x$-token.

#### Fees and Rules

Note that the above analysis makes no assumptions on ecosystem rules or protocol fees. The incentive to split will be significantly mitigated by gas fees alone. Furthermore, simple rules imposing caps on order frequency per address, or a minimum allowable trade size, could also remove this incentive altogether.

### 4.6 Notes on $x_{\min}$

The presence of a minimum $x$-value greater than 0 ensures that prices cannot be costlessly manipulated by exploiters. That is to say, if traders **could** push outstanding supply to zero, there might arise situations where a high volume of round trips to and from $x = 0$ would push the price up and up, without costing the trader anything (other than transaction fees).[25]

$x_{\min}$ imposes a cost on these round trips, so that a malicious actor seeking to inflate prices early on in the Dynamic AMM's lifecycle will have to pay to do so.

How much will this price inflation cost the trader? That depends on the value of $x_{\min}$.

#### The Constant $k$

We will think of $x_{\min}$ as a percentage of $x_{\text{add}}$, denoted with $k$. It is useful, then, to know something about how the Dynamic AMM's behavior will change with different values of $k$.

$k$ presents a trade-off. On one hand, higher values of $k$ impose higher costs to the malicious trader wishing to push prices up. Figure 20 shows the cost to traders of pushing up price by a factor of 11 (arbitrarily selected) as a function of $k$ size.[26]

On the other hand, higher values of $k$ reduce the market's flexibility. A higher $k$ value means an increase in the minimum achievable price. It also means that the initial slope value at the beginning of the Dynamic AMM's lifespan is lower (meaning prices will move less, earlier on). Finally, it means that, fixing a given trade size, the size of the update to the Dynamic AMM will be smaller. These three effects are shown in figures 21 through 23. Note the three measures of the Dynamic AMM's flexibility in green: initial price, initial slope, and update size (as a percent change in slope from one line to the other). Initial price goes up, while initial slope and update size go down with increases to $k$. Note also that the realizable trading region is shifted to the right, but its size stays the same (a high $k$ value does **not** mean a decrease in the total number of tokens for sale, a value that is stored as $x_{\text{add}}$).



[24] Sandwich attackers can still make gains with less-than-optimal splits; nevertheless the central point is that systematically splitting orders seems like it might be more trouble than it is worth, especially in the early stages of an asset's lifecycle.

[25] Of course, these trades would also not profit the exploiter - the sole gain would be to sabotage the price mechanism by pushing the price up artificially.

[26] We measure the cost of this price increase as a portion of market capitalization, which is defined here as the lowest price on the starting Dynamic AMM multiplied by $x_{\text{add}}$.

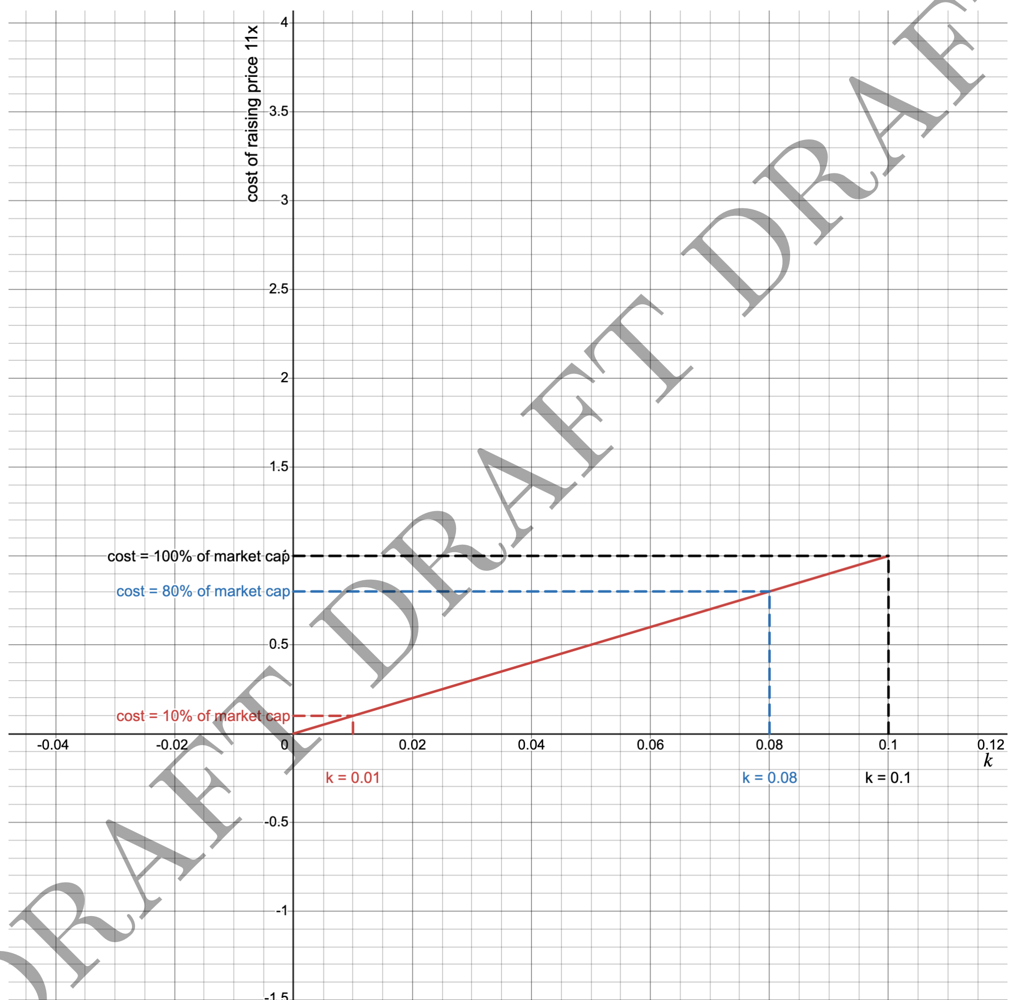





Figure 20: Cost of raising price 11x (measured as a percentage of total market capitalization) as a function of $k$. To push the price up to 11x its initial value, with $k = .1$, it would cost the trader 100% of market capitalization.

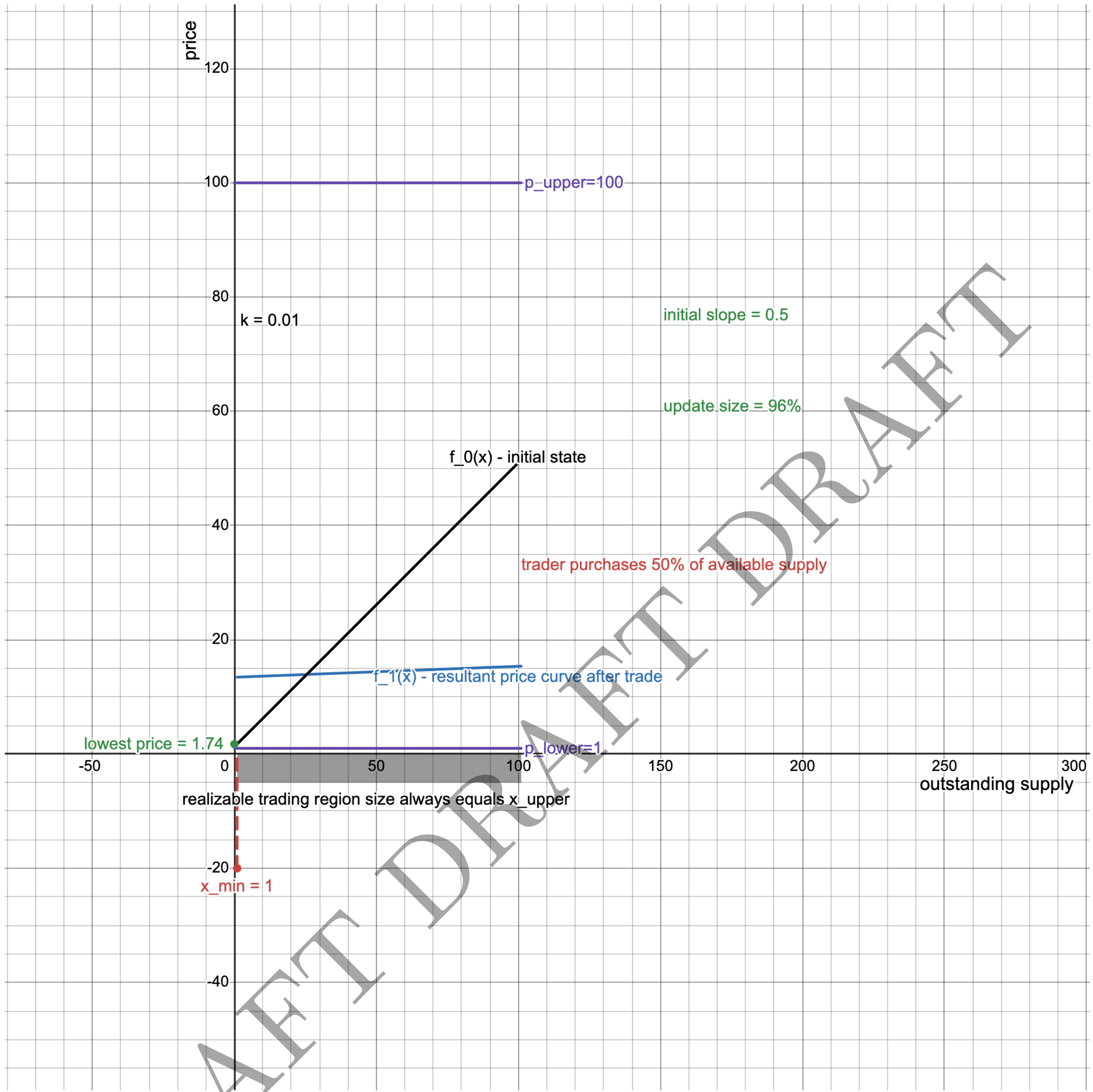



Figure 21: Buying 50% of available supply, $k = .01$. A low $k$ value means a lower initial spot price, a higher initial slope and a more meaningful update to the price curve.

# 5 Experiments

In this section, we present some experimental results, using real-world trade data to compare the Dynamic AMM with Uniswap pools. In figure 24, we compare the two pools using data from the DOGE-USDT pair. In figure 25, we use TRAX-USDT, and in figure 26, we use ADA-USDT.

## 5.1 Caveats

There are some caveats we must make before turning to these results:

1. The raw data represent actual buys and sells executed on Uniswap pools (both V2 and V3 are used). These trades took place in the context of actual historical market prices, which differ from the price sequences generated by the Dynamic AMM. Since prices affect demand, however, it might be reasonable to expect different trade sequences to occur in a world where the Dynamic AMM alone was setting prices. In our experiments, we ignore this consideration and hold demand constant regardless of price.

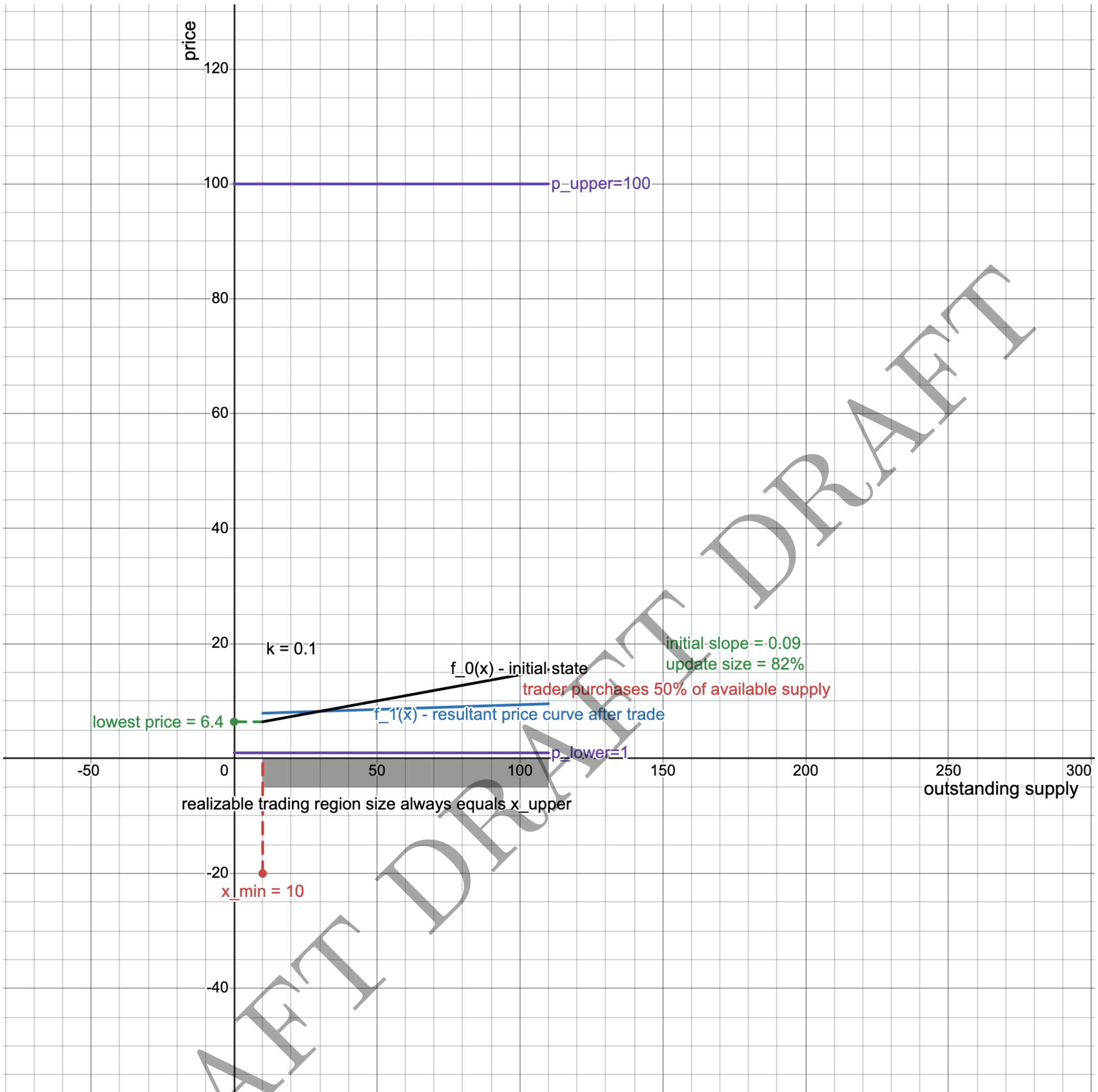



Figure 22: Buying 50% of available supply, $k = .1$. A higher $k$ value means a higher initial spot price, a lower initial slope and a less meaningful update to the price curve after the same trade.



2. The Dynamic AMM can be parameterized in many different ways, which lead to significantly different price behaviors. We have chosen certain parameters that lead to instructive results. The parameters used below are explained in section 2.11 and in the Appendix.

3. In order to accommodate the sequences of trades in the real-world data, we initialize the Dynamic AMM and then run a large initial purchase to prevent the $x$ value from hitting zero and blocking some of the trades in our datasets.

## 5.2 Discussion

These plots show four things:

1. Whereas Uniswap V2 pools require an initial deposit of collateral, the Dynamic AMM achieves similar price movement without requiring any initial collateral allotment.

2. Whereas Uniswap V3 pools require active management, the Dynamic AMM achieves similar price movements without any active management.

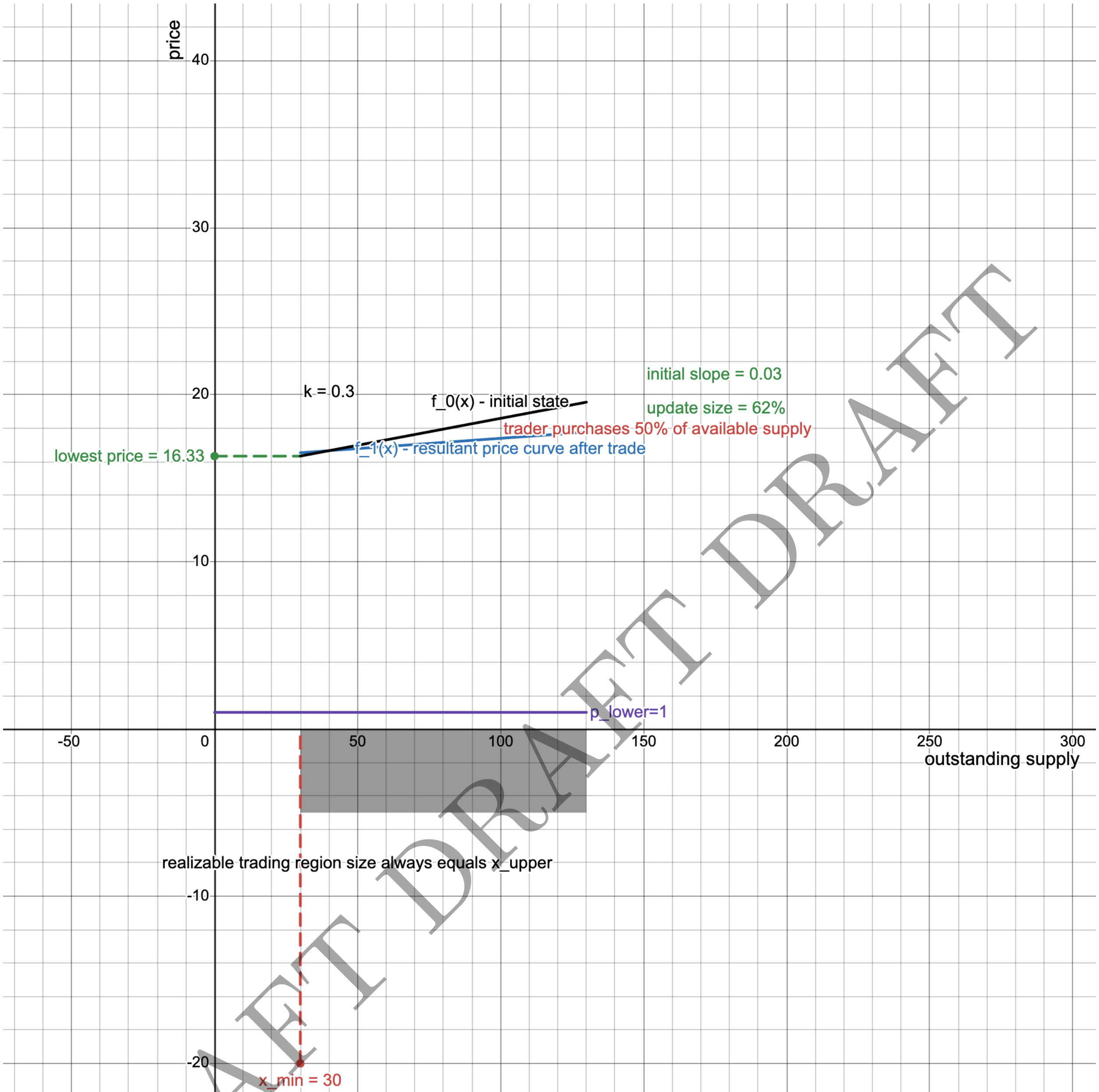




Figure 23: Buying 50% of available supply, $k = .3$. A high $k$ value means a high initial spot price, a low initial slope and a less meaningful update to the price curve after the same trade.

3. The Dynamic AMM has inherent price stability, compared with Uniswap.
4. The `p_mid` value affects this stability, with higher values leading to greater volatility.
5. When volume is high but buys and sells roughly cancel each other in the aggregate, the price and collateral will steadily drift upward.
6. For tokens that trade in the $1 range, the Dynamic AMM is a good combination of bootstrapping utility and price discovery.

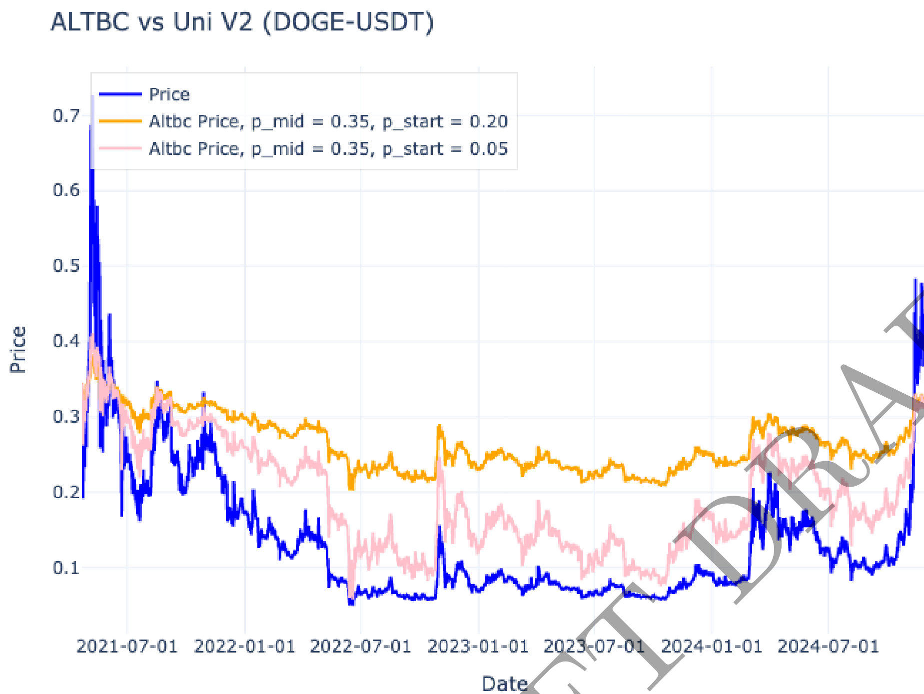



Figure 24: Comparison of Dynamic AMM and UniV2 on the DOGE-USDT trading pair.

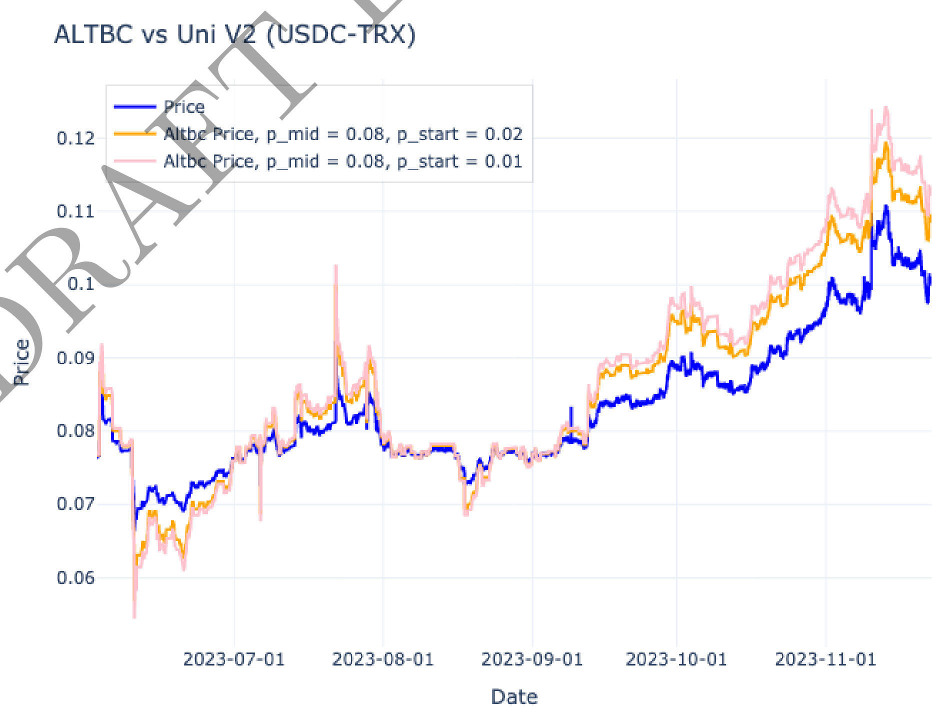



Figure 25: Comparison of Dynamic AMM and UniV2 on the TRX-USDC trading pair.

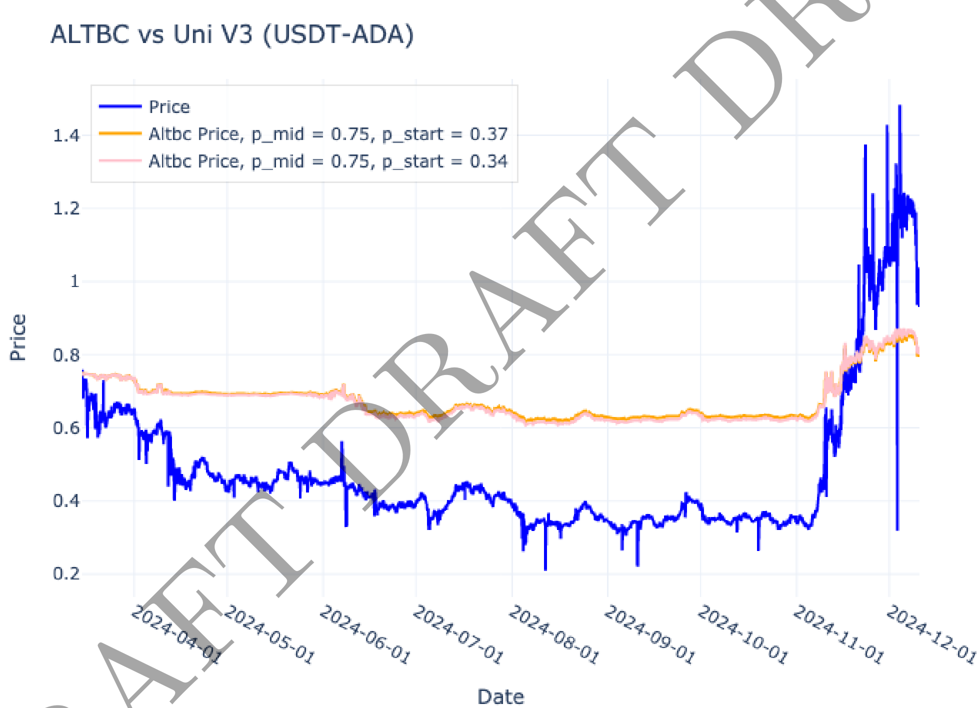



Figure 26: Comparison of Dynamic AMM and UniV3 on the ADA-USDT trading pair.

# A Proofs and Derivations

## A.1 Formal Description of ALTBC Trading and Price Impact

**Initial state.** Let $x_0$ be the amount of game tokens sold by the protocol. We consider a linear token bonding curve defined by

$$f(x) = b_0 x + c_0 \ ,$$

where $b_0, c_0 > 0$. Let

$$s(x) = \frac{V}{x + \Gamma_0} \ .$$

We define the *cost function* by

$$F(x) = \frac{1}{2} b_0 x^2 + c_0 x \ .$$

The amount of collateral in the pool is $F(x_0)$.

**Trade.** Suppose that a trader buys or sells an amount $a$ of game tokens. If $a > 0$ we consider that the trader buys $a$ game tokens, and if $a < 0$ we consider that the trader sells $-a$ game tokens. In any case, the amount of game tokens sold by the protocol after the trade is

$$x_1 = x_0 + a \ .$$

The amount of collateral either paid or received by the trader is given by $\Delta = F(x_1) - F(x_0)$. If $\Delta > 0$, then the trader pays that amount and the collateral in the pool increases ($F(x_1) > F(x_0)$), and if this amount is negative, then the trader receives an amount $-\Delta$ of collateral and the amount of collateral in the pool decreases ($F(x_1) < F(x_0)$).

Note that, since $F$ is a strictly increasing function, we obtain that

$$\Delta > 0 \Leftrightarrow F(x_1) > F(x_0) \Leftrightarrow x_1 > x_0 \Leftrightarrow a > 0 \ .$$

**State after the trade.** After the trade, the amount of game tokens sold by the protocol is

$$x_1 = x_0 + a \ .$$

The parameters $b$ and $c$ of the token bonding curve are updated as follows:

$$b_1 = \frac{V}{x_1 + \Gamma_0}$$

and

$$c_1 = \frac{F(x_1) - \frac{1}{2} b_1 x_1^2}{x_1} = \frac{\frac{1}{2} b_0 x_1^2 + c_0 x_1 - \frac{1}{2} b_1 x_1^2}{x_1} = \frac{1}{2}(b_0 - b_1) x_1 + c_0 \ .$$

Thus, the token bonding curve will be updated as

$$f_1(x) = b_1 x + c_1 \ .$$

The amount of collateral in the pool after the trade is

$$F(x_1) = \frac{1}{2} b_0 x_1^2 + c_0 x_1 \ .$$

The cost function will be also updated as

$$F_1(x) = \frac{1}{2} b_1 x^2 + c_1 x \ .$$

If we use the updated cost function to compute the amount of collateral in the pool after the trade we obtain that this amount should be $F_1(x_1)$. With our definition of the updated parameters we get

$$F_1(x_1) = \frac{1}{2} b_1 x_1^2 + c_1 x_1 = \frac{1}{2} b_1 x_1^2 + \left( \frac{1}{2}(b_0 - b_1) x_1 + c_0 \right) x_1 = \frac{1}{2} b_1 x_1^2 + \frac{1}{2} b_0 x_1^2 - \frac{1}{2} b_1 x_1^2 + c_0 x_1 =$$
$$= \frac{1}{2} b_0 x_1^2 + c_0 x_1 = F(x_1) \ ,$$

as desired.

**Price impact of the trade.** The price impact of the trade is given by

$$\frac{|\text{ effective price of the trade} - \text{spot price before the trade }|}{\text{spot price before the trade}}\,.$$

Thus, we obtain that

$$\begin{aligned}
\text{Price Impact} &= \left|\frac{\frac{F(x_1)-F(x_0)}{a} - (b_0x_0+c_0)}{b_0x_0+c_0}\right| = \left|\frac{F(x_1)-F(x_0)}{a(b_0x_0+c_0)} - 1\right| = \\
&= \left|\frac{\frac{1}{2}b_0x_1^2 + c_0x_1 - (\frac{1}{2}b_0x_0^2 + c_0x_0)}{a(b_0x_0+c_0)} - 1\right| = \left|\frac{\frac{1}{2}b_0(x_1^2-x_0^2) + c_0(x_1-x_0)}{a(b_0x_0+c_0)} - 1\right| = \\
&= \left|\frac{(\frac{1}{2}b_0(x_1+x_0)+c_0)(x_1-x_0)}{a(b_0x_0+c_0)} - 1\right| = \left|\frac{(\frac{1}{2}b_0(x_1+x_0)+c_0)a}{a(b_0x_0+c_0)} - 1\right| = \\
&= \left|\frac{\frac{1}{2}b_0(x_1+x_0)+c_0}{b_0x_0+c_0} - 1\right| = \left|\frac{\frac{1}{2}b_0x_1 + \frac{1}{2}b_0x_0 + c_0 - (b_0x_0+c_0)}{b_0x_0+c_0}\right| = \\
&= \left|\frac{\frac{1}{2}b_0x_1 - \frac{1}{2}b_0x_0}{b_0x_0+c_0}\right| = \left|\frac{\frac{1}{2}b_0(x_1-x_0)}{b_0x_0+c_0}\right| = \left|\frac{\frac{1}{2}b_0a}{b_0x_0+c_0}\right|\,.
\end{aligned}$$

Therefore, if $p_0$ is the spot price before the trade we obtain that

$$\text{Price Impact} = \frac{|a|b_0}{2p_0}\,.$$

## A.2 Directional Consistency of Spot Price

Suppose that $x_0$ is the current outstanding supply and $p_0$ is the current spot price. Let $x_1$ and $p_1$ be the corresponding quantities after the proposed update. Then,

$$\operatorname{sgn}(x_1 - x_0) = \operatorname{sgn}(p_1 - p_0).$$

**Proof:**

$$\begin{aligned}
p_1 &= p_0 - \frac{x_0}{x_0+C} + \frac{1}{2}\frac{x_1}{x_0+C} + \frac{1}{2}\frac{x_1}{x_1+C} \\
&= p_0 + \frac{1}{2}\frac{x_1-x_0}{x_0+C} + \frac{1}{2}\left(\frac{x_1}{x_1+C} - \frac{x_0}{x_0+C}\right) \\
&= p_0 + \frac{1}{2}\frac{x_1-x_0}{x_0+C} + \frac{1}{2}(h(x_1) - h(x_0)),
\end{aligned}$$

where

$$h(x) = \frac{x}{x+C}$$

is monotonically increasing. As such, if $x_1 > x_0$ (respectively $x_1 < x_0$) in the right hand side of the expression

$$p_1 - p_0 = \frac{1}{2}\frac{x_1-x_0}{x_0+C} + \frac{1}{2}(h(x_1) - h(x_0)),$$

it follows that the RHS will be positive (respectively negative) and so $p_1 > p_0$ (respectively $p_1 < p_0$).

## A.3 Monotonicity of Spot Price in time for fixed $x$

Suppose that $x_0$ is the current outstanding supply and $p_0$ is the current spot price. The next time that $x_0$ is visited (if at all), the spot price $p$ at this time will be greater than $p_0$.

**Proof:** Let us first establish some useful notation. Let

$$x_0, x_1, \ldots, x_n = x_0$$

denote the evolution of the outstanding supply as it departs and returns to $x_0$ for the first time after $n$ steps. Let

$$p_0, p_1, \ldots, p_n = p$$

be the corresponding spot prices. We will need two lemmas.

**Lemma 1:** If $n = 2$, monotonicity of spot price in time holds.

**Proof:**

$$\begin{aligned}
p_2 &= p_0 - \frac{x_0}{x_0 + C} + \frac{1}{2}\frac{x_1}{x_0 + C} + \frac{1}{2}\frac{x_1}{x_1 + C} - \frac{x_1}{x_1 + C} + \frac{1}{2}\frac{x_2}{x_1 + C} + \frac{1}{2}\frac{x_2}{x_2 + C} \\
&= p_0 + \frac{1}{2}\frac{x_1 - x_0}{x_0 + C} + \frac{1}{2}\frac{x_2 - x_1}{x_1 + C} + \frac{1}{2}\left(\frac{x_2}{x_2 + C} - \frac{x_0}{x_0 + C}\right) \\
&= p_0 + \frac{1}{2}\frac{x_1 - x_0}{x_0 + C} + \frac{1}{2}\frac{x_0 - x_1}{x_1 + C} \\
&= p_0 + \frac{1}{2}\frac{x_1^2 - 2x_0x_1 + x_0^2}{(x_0 + C)(x_1 + C)} = p_0 + \frac{1}{2}\frac{(x_1 - x_0)^2}{(x_0 + C)(x_1 + C)} > p_0.
\end{aligned}$$

**Lemma 2:** For a given $i$, if $x_i < x_{i+1} < x_{i+2}$ (or respectively $x_i > x_{i+1} > x_{i+2}$), omitting the visit to $x_{i+1}$ leaves the AMM with a strictly greater spot price when arriving at $x_{i+2}$.

**Proof:** For a given $i$, assume that $x_i < x_{i+1} < x_{i+2}$ (or respectively $x_i > x_{i+1} > x_{i+2}$). From the proof for lemma 1, we see that

$$\begin{aligned}
p_{i+2} &= p_i + \frac{1}{2}\frac{x_{i+1} - x_i}{x_i + C} + \frac{1}{2}\frac{x_{i+2} - x_{i+1}}{x_{i+1} + C} + \frac{1}{2}\left(\frac{x_{i+2}}{x_{i+2} + C} - \frac{x_i}{x_i + C}\right) \\
&< p_i + \frac{1}{2}\frac{x_{i+1} - x_i}{x_i + C} + \frac{1}{2}\frac{x_{i+2} - x_{i+1}}{x_i + C} + \frac{1}{2}\left(\frac{x_{i+2}}{x_{i+2} + C} - \frac{x_i}{x_i + C}\right) \\
&= p_i + \frac{1}{2}\frac{x_{i+2} - x_i}{x_i + C} + \frac{1}{2}\left(\frac{x_{i+2}}{x_{i+2} + C} - \frac{x_i}{x_i + C}\right),
\end{aligned}$$

where the final expression is the spot price in the AMM had $x_{i+1}$ been omitted from the path. Note that the inequality holds for both cases.

**Completion of the proof of the main result:** Here it might be useful to include an example graph of the $x_i$ sequence with the abscissa being the index. Consider a global extremum of this graph that occurs at $(i, x_i)$. From lemma 2, we can assume WLOG (by adding an $x$ value as necessary) that $x_{i-1} = x_{i+1}$. In other words, this will only make the trade path worse for the AMM (with a lower final spot price). Then, we can remove $x_i$ and $x_{i+1}$ from the trade path by using Lemma 1. Again, this will only leave the AMM with a lower final spot price. Repeating this process will leave the path only $x_0$ at the end. As such, we have shown that the final spot price $p$ associated with a given trade path is strictly greater than $p_0$.

**General Lemma:** From the above proofs, we note that the spot price associated with an arbitrary sequence $x_0, x_1, \ldots, x_n$ is given by

$$p_n = p_0 + \frac{1}{2}\sum_{1}^{n}\frac{x_i - x_{i-1}}{x_{i-1} + C} + \frac{1}{2}\left(\frac{x_n}{x_n + C} - \frac{x_0}{x_0 + C}\right)$$

### A.4 Splitting a trade into $N$ steps

Let $x_0$ be the initial amount of game tokens. We consider an updating linear TBC defined by

$$f(z) = b(x)z + c(x)\ ,$$

where $b(x)$ and $c(x)$ are functions that depend on the amount of game tokens $x$ before the trade is performed.

Suppose that we want to buy an amount $A$ of game tokens in $N$ steps, where the amount of tokens bought at each step depends on a chosen function $g$ as follows. Let $g\colon [0,1] \to \mathbb{R}$ be a continuous and monotonically increasing function such that $g(0) = x_0$ and $g(1) = x_0 + A$. For each $j \in \{1, 2, \ldots, N\}$ we define

$$x_j = g(\tfrac{j}{N}) \ .$$

Note that $x_N = x_0 + A$. For each $j \in \{1, 2, \ldots, N\}$, the amount of game token to be bought at the step $j$ is $x_j - x_{j-1}$. Thus, for all $j \in \{1, 2, \ldots, N\}$, the amount of game tokens sold by the protocol after step $j$ is $x_j$.

For each $n \in \{0, 1, \ldots, N\}$, let $p_n$ be the spot price of the TBC after step $n$. Similarly, for each $n \in \{1, 2, \ldots, N\}$, let $b_n$ and $c_n$ be the values of the parameters $b$ and $c$ of the TBC after step $n$. Note that $b_n = b(x_n)$ for all $n \in \{1, 2, \ldots, N\}$.

Applying the formulas given in the first section, we obtain that, for all $n \in \{1, 2, \ldots, N\}$,

$$\begin{aligned}
p_n &= b_n x_n + c_n = b_n x_n + \tfrac{1}{2}(b_{n-1} - b_n)x_n + c_{n-1} = \\
&= b_n x_n + \tfrac{1}{2}(b_{n-1} - b_n)x_n + p_{n-1} - b_{n-1}x_{n-1} = \\
&= p_{n-1} + b_n x_n + \tfrac{1}{2}b_{n-1}x_n - \tfrac{1}{2}b_n x_n - b_{n-1}x_{n-1} = \\
&= p_{n-1} + \tfrac{1}{2}b_{n-1}x_n + \tfrac{1}{2}b_n x_n - b_{n-1}x_{n-1} = \\
&= p_{n-1} + \tfrac{1}{2}b_{n-1}x_n + \tfrac{1}{2}(b_n x_n - b_{n-1}x_{n-1}) - \tfrac{1}{2}b_{n-1}x_{n-1} = \\
&= p_{n-1} + \tfrac{1}{2}b_{n-1}(x_n - x_{n-1}) + \tfrac{1}{2}(b_n x_n - b_{n-1}x_{n-1}).
\end{aligned}$$

With an inductive argument it is not difficult to prove that

$$p_N = p_0 + \sum_{j=1}^{N} \left(\tfrac{1}{2}b_{j-1}(x_j - x_{j-1}) + \tfrac{1}{2}(b_j x_j - b_{j-1}x_{j-1})\right) \ .$$

Thus,

$$\begin{aligned}
p_N &= p_0 + \frac{1}{2}\sum_{j=1}^{N} b_{j-1}(x_j - x_{j-1}) + \frac{1}{2}\sum_{j=1}^{N}(b_j x_j - b_{j-1}x_{j-1}) = \\
&= p_0 + \frac{1}{2}\sum_{j=1}^{N} b(x_{j-1})(x_j - x_{j-1}) + \frac{1}{2}(b_N x_N - b_0 x_0) = \\
&= p_0 + \tfrac{1}{2}(b(x_N)x_N - b(x_0)x_0) + \frac{1}{2}\sum_{j=1}^{N} b(g(\tfrac{j-1}{N}))(g(\tfrac{j}{N}) - g(\tfrac{j-1}{N})).
\end{aligned}$$

Now, the sum

$$\sum_{j=1}^{N} b(g(\tfrac{j-1}{N}))(g(\tfrac{j}{N}) - g(\tfrac{j-1}{N}))$$

is a Riemann-Stieltjes sum of the integral

$$\int_0^1 b(g(x))\,\mathrm{d}g(x) \ .$$

If $g$ is continuously differentiable then

$$\int_0^1 b(g(x))\,\mathrm{d}g(x) = \int_0^1 b(g(x))g'(x)\,\mathrm{d}x = B(g(1)) - B(g(0)) = B(x_N) - B(x_0) \ ,$$

where the function $B$ is a primitive of the function $b$.

Therefore,

$$\begin{aligned}
\lim_{N\to+\infty} p_N &= \lim_{N\to+\infty}\left(p_0 + \frac{1}{2}(b(x_N)x_N - b(x_0)x_0) + \frac{1}{2}\sum_{j=1}^{N} b(g(\tfrac{j-1}{N}))(g(\tfrac{j}{N}) - g(\tfrac{j-1}{N}))\right) = \\
&= p_0 + \frac{1}{2}(b(x_N)x_N - b(x_0)x_0) + \frac{1}{2}\int_0^1 b(g(x))\,\mathrm{d}g(x) = \\
&= p_0 + \frac{1}{2}(b(x_N)x_N - b(x_0)x_0) + \frac{1}{2}(B(x_N) - B(x_0)).
\end{aligned}$$

It is interesting to observe that the value of this limit is independent of the chosen function $g$.

Now, if the function $b$ is monotonically decreasing, it follows that

$$\sum_{j=1}^{N} b(g(\tfrac{j-1}{N}))(g(\tfrac{j}{N}) - g(\tfrac{j-1}{N})) \geq \int_0^1 b(g(x))\,\mathrm{d}g(x) \ .$$

Thus,

$$p_N \geq p_0 + \frac{1}{2}(b(x_N)x_N - b(x_0)x_0) + \frac{1}{2}\int_0^1 b(g(x))\,\mathrm{d}g(x) \ .$$

Therefore,

$$\inf_{n\in\mathbb{N}} p_N = p_0 + \frac{1}{2}(b(x_N)x_N - b(x_0)x_0) + \frac{1}{2}\int_0^1 b(g(x))\,\mathrm{d}g(x) \ .$$

Now, we want to compute the amount of collateral the trader has to pay in order to buy the amount $A$ of game tokens. For each $j \in \{1, 2, \ldots, N\}$, let $F_{j-1}(x)$ be the cost function before step $j$ of the trade. Then, for all $j \in \{1, 2, \ldots, N\}$,

$$F_{j-1}(x) = \frac{1}{2}b_{j-1}x^2 + c_{j-1}x \ ,$$

and the amount of collateral that the trader has to pay in step $j$ is $F_{j-1}(x_j) - F_{j-1}(x_{j-1})$.

Let $F_N$ be defined by

$$F_N(x) = \frac{1}{2}b_N x^2 + c_N x \ .$$

Note that the total amount $Y_N$ of collateral that the trader has to pay is given by

$$\begin{aligned}
Y_N &= \sum_{j=1}^{N}(F_{j-1}(x_j) - F_{j-1}(x_{j-1})) = \sum_{j=1}^{N} F_{j-1}(x_j) - \sum_{j=1}^{N} F_{j-1}(x_{j-1}) = \\
&= \sum_{j=1}^{N} F_j(x_j) - \sum_{j=1}^{N} F_{j-1}(x_{j-1}) = F_N(x_N) - F_0(x_0) = \frac{1}{2}b_N x_N^2 + c_N x_N - \frac{1}{2}b_0 x_0^2 - c_0 x_0 = \\
&= (p_N - b_N x_N)x_N + \frac{1}{2}b_N x_N^2 - \frac{1}{2}b_0 x_0^2 - c_0 x_0 = p_N x_N - \frac{1}{2}b_N x_N^2 - \frac{1}{2}b_0 x_0^2 - c_0 x_0 \ .
\end{aligned}$$

Since $x_0$, $x_N$, $b_0$, $c_0$ and $b_N$ are fixed and $x_N > 0$, in order to minimize $Y_N$ it suffices to minimize $p_N$, which was done previously.

## A.5 MEV Formula

The measure of MEV used here is given by the idea of a sandwich attack. If a non-malicious trader submits a transaction of quantity $Q$ then an attacker could "sandwich" either side of this transaction with a buy/sell transaction pair each of size $S$. If the current outstanding supply of token is $x_n$ at the time the attack is initiated then the profit the attacker attains is given by the following equation for $MEV(S, Q)$:

$$MEV(S,Q) = \frac{SV}{2} \cdot \left(\frac{x_n + Q}{C + x_n + Q + S} + \frac{Q}{C + x_n + S} - \frac{x_n}{C + x_n}\right)$$

There are a few important takeaways to consider here:

- Given a fixed non-malicious transaction quantity $Q$, the size of $MEV(S, Q)$ for the ALTBC is bounded. This is in sharp contrast to other markets (like Uniswap V2) where $MEV(S, Q)$ goes to infinity as $S$ goes to infinity
- For each $Q$ there is a "critical" value of $S$ above which the MEV will be negative. One show that this value $S_{crit}$ will satisfy the following equation:

$$\frac{x_n + Q}{(C + x_n + Q) + S_{crit}} + \frac{Q}{(C + x_n) + S_{crit}} = \frac{x_n}{C + x_n}$$

# References


[1] Guillermo Angeris, Akshay Agrawal, Alex Evans, Tarun Chitra, and Stephen Boyd. Constant function market makers: Multi-asset trades via convex optimization. In *Handbook on Blockchain*, pages 415–444. Springer, 2022.

[2] Mohammad Ali Asef and Seyed Mojtaba Hosseini Bamakan. From x* y= k to uniswap hooks: A comparative review of decentralized exchanges (dex). *arXiv preprint arXiv:2410.10162*, 2024.

[3] Michael Egorov. Stableswap-efficient mechanism for stablecoin liquidity. *Retrieved Feb*, 24:2021, 2019.

[4] Curve Finance. Understanding crypto pools, 2024. Accessed: 2024-12-10.

[5] Andreas Park. The conceptual flaws of decentralized automated market making. *Management Science*, 69(11):6731–6751, 2023.