# Arbitrage opportunities in Constant Product Market Maker networks

Alan King, Bohang Wei, Chris Williams, Keith Houck, Ryo Kawahara, Mikio Takeuchi, Aliza Heching, and Nitin Gaur

**Abstract** Prices in markets reach equilibrium through arbitrage processes. This paper presents a theoretical result, the threshold inequality, that holds whenever there is a potential for arbitrage. The effectiveness of this signal is discussed in the context of a canonical type of arbtrage called the carry trade.

## 1 Introduction

Prices of digital tokens are determined by traders. Opportunities for arbitrage exist whenever a given trade can be routed through a different sequence of transactions. This paper presents basic results for the existence of arbitrage opportunities in networks of constant product market makers.

## 2 Constant product market makers

Constant product market makers are variety of automated market maker (AMM), which are platforms for the automated exchange of digital assets, also called tokens. The widely understood properties of AMMs are:

- Orders are priced in real-time using a simple algorithm
- Orders are automatically cleared and settled within a single transaction

Aliza Heching · Alan King · Keith Houck · Ryo Kawahara · Mikio Takeuchi
IBM Research

Bohang Wei
Ernst and Young

Nitin Gaur · Chris Williams
State Street

- Full transparency into the history and current state of the exchange

Uniswap is a constant product market maker that is one of the most active AMM platforms. Quoting from the Uniswap Whitepaper [1]

> Uniswap smart contracts hold liquidity reserves of various tokens, and trades are executed directly against these reserves. Prices are set automatically using the constant product (x*y=k) market maker mechanism, which keeps overall reserves in relative equilibrium. Reserves are pooled between a network of liquidity providers who supply the system with tokens in exchange for a proportional share of transaction fees.

Uniswap exchanges apply the constant product market maker (CP-MM) mechanism whereby every transaction preserves the product of the number of reserve tokens. The Uniswap v3 protocol is presented in the whitepaper [2]. This is a relatively mature CP-MM protocol with features such as liquidity pool "ticks", which function as liquidity provider limit orders, and exchange fees that can vary by tick.

## 2.1 Marginal price

The ratio of token reserve quantities $(q_a, q_b)$ in a constant product exchange can be viewed as a price, in which a quantity $\delta_a$ of $a$-tokens has the $b$-token value

$$\delta_b = \frac{q_b}{q_a} \cdot \delta_a \tag{1}$$

Constant product means that the product of post-transaction reserves $(q_a + \delta_a)$ and $(q_b - \delta_b)$ equals the product of pre-transaction reserves:

$$q_a \cdot q_b = (q_a + \delta_a) \cdot (q_b - \delta_b) \tag{2}$$

For infinitely small trades the term $\delta_a \cdot \delta_b$ can be neglected, then solving for $\delta_b$ gives the relation (1). The reserves ratio $q_b/q_a$ is a *marginal* price, which holds only for infinitesimal trades.

## 2.2 Swap trades

Actual trades are not infinitesimal, and exchanges will charge fees. Uniswap exchanges apply the constant product rule *after* applying a haircut $(1 - f) \cdot \delta_a$ to the incoming leg, so $\delta_b$ is

$$\delta_b = \frac{q_b \cdot (1 - f) \cdot \delta_a}{q_a + (1 - f) \cdot \delta_a} \tag{3}$$

This swap is completely characterized by the state of the exchange: the reserves $(q_a, q_b)$ and the fee $f$.

### 2.2.1 Aside: Uniswap V2 and V3

Let $k = q_a \cdot q_b$, then

$$(q_a + \delta_a) \cdot (q_b - \delta_b) = k + f \cdot \delta_a \cdot (q_b - \delta_b)$$

In Uniswap V2, fees remain in the pool so the value $k$ always increases. Uniswap V3 introduced many innovations for liquidity providers, the details of which are beyond the scope of this paper. As shown in formula (6.12) in [2], the V3 output formula is equivalent to (3), but $k$ is constant.

## 2.3 Reserve fractions

The marginal price and exchange fee are in formula (3), but there is also a non-linear term:

$$\delta_b = \left(\frac{q_b}{q_a}\right) \cdot (1-f) \cdot \delta_a \cdot \left(\frac{1}{1 + (1-f) \cdot \delta_a / q_a}\right)$$

The first three factors apply the marginal price to the discounted incoming leg, but the final factor is the non-linear effect of the constant product rule.

$$\begin{aligned} \textit{marginal price} \quad &: \quad \frac{q_b}{q_a} \\ \textit{haircut} \quad &: \quad (1-f) \\ \textit{incoming leg} \quad &: \quad \delta_a \\ \textit{non-linear term} \quad &: \quad \frac{1}{1 + (1-f) \cdot \delta_a / q_a} \end{aligned}$$

The non-linear impact of the CP rule is a function of $\delta_a / q_a$. Taking this hint, redefine swaps as *reserve-fraction trades*:

$$\begin{aligned} X_a &:= \delta_a / q_a \\ X_b &:= \delta_b / q_b \end{aligned}$$

The CP rule now becomes

$$1 = (1 - X_b) \cdot (1 + (1-f) \cdot X_a)$$

Solve for $X_b$

$$X_b = \frac{(1-f) \cdot X_a}{1 + (1-f) \cdot X_a} \tag{4}$$

and solve for $X_a$:

$$X_a = (1 - f)^{-1} \cdot \left(\frac{X_b}{1 - X_b}\right) \tag{5}$$

In this formula $X_a$ is the reserve fraction required to produce exactly $X_b$.

## 2.4 Routes

A trader wishing to swap $a$-tokens for $b$-tokens faces many choices. Each such choice is called a *route*. Routes are composed of two basic elements: hops and splits.

### 2.4.1 Hops

A hop is a swap transaction. For example, consider a two-hop route $a \longrightarrow c \longrightarrow b$. The route starts with a swap $X_a^0$ on exchange $\mathrm{CP}_0(a, c)$:

$$X^0c = \frac{(1 - f_0) \cdot X_a^0}{1 + (1 - f_0) \cdot X_a^0} \tag{6}$$

and ends with a swap on exchange $\mathrm{CP}_1(b, c)$:

$$X_b^1 = \frac{(1 - f_1) \cdot X_c^1}{1 + (1 - f_1) \cdot X_c^1} \tag{7}$$

### 2.4.2 Splits

A split divides an incoming swap leg $X_a$ into multiple swap legs. For example, consider a two-way split on exchanges $\mathrm{CP}_0(a, b)$ and $\mathrm{CP}_1(a, b)$. Splits satisfy the conservation condition:

$$X_a^0 + X_a^1 = 1$$

## 2.5 Network model for routes

Routes can be modeled as directed networks The arcs of the network are exchanges, and the nodes are virtual accounts that hold tokens. For example, an $a \longrightarrow b$ route starts from an account with $a$-tokens, splits and hops along exchange-arcs, and collects the $b$-tokens produced at the terminal end.

Trades along cycles correspond to flows along the arcs, and routes can be optimized using the max-flow algorithm, for example. One can verify that flow variables are the split fraction choices. Importantly, the data needed to compute the swaps can be collected automatically and executed within *single transaction*.

### 2.6 Cycles

A cycle is a route that begins and ends in the same token type. Cycles create arbitrage opportunities. To illustrate the basic idea, suppose the marginal price of a single-hop route $a \longrightarrow b$ is greater than a double-hop route $a \longrightarrow c \longrightarrow b$:

$$\left(\frac{q_b}{q_a}\right) > \left(\frac{q_b}{q_c}\right) \cdot \left(\frac{q_c}{q_a}\right)$$

Consider the cycle route $c \longrightarrow a \longrightarrow b \longrightarrow c$. Borrow $\delta_c$, and create a transaction with the swaps:

$$\begin{aligned} Swap &: \left(\frac{q_a}{q_c}\right) \cdot \delta_c \longrightarrow \delta_a \\ Swap &: \left(\frac{q_b}{q_a}\right) \cdot \delta_a \longrightarrow \delta_b \\ Swap &: \left(\frac{q_c}{q_b}\right) \cdot \delta_b \longrightarrow \Delta_c \end{aligned}$$

After paying back the loan, this transaction earns $\Delta_c - \delta_c$, which is positive because:

$$\begin{aligned} \Delta_c &= \left(\frac{q_c}{q_b}\right) \cdot \left(\frac{q_b}{q_a}\right) \cdot \delta_a \\ &> \left(\frac{q_c}{q_b}\right) \cdot \left(\frac{q_b}{q_c} \cdot \frac{q_c}{q_a}\right) \cdot \delta_a \end{aligned}$$

and

$$\delta_a = \left(\frac{q_a}{q_c}\right) \cdot \delta_c$$

This trade is called a *carry trade arbitrage*. The trade uses differences in prices between two routes. The critical step in any arbitrage opportunity is to identify the "unit of account" in which profit or loss is measured. The opportunity in this section is generated by a price difference between single-hop $a \longrightarrow b$ and a double-hop $a \longrightarrow c \longrightarrow b$. The natural unit of account is determined by the cycle, in this case it is $c$-tokens.

## 3 Carry trade arbitrage opportunity

Different routes for the same trade lead to carry trade opportunities. This section addresses two basic questions:

- What conditions characterize arbitrage opportunities?
- What is the maximum possible arbitrage profit?

### 3.1 Threshold inequality

Characterization of arbitrage ought to be viewed in terms of readily available information, such as reserve quantities and marginal prices. This identifies an arbitrage characteristic in terms of *reserve valuations* in a common unit of account.

The analysis in this section considers only that there is a common unit of account in which one can observe marginal prices $p_a$ and $p_b$ for $a$-tokens and $b$-tokens. Consider the reserves $(q_a, q_b)$ in the CP$(a,\ b)$ exchange and introduce the reserve valuations:

$$\begin{aligned} V_a &= p_a \cdot q_a \quad &:& \quad \text{value of reserve } q_a \\ V_b &= p_b \cdot q_b \quad &:& \quad \text{value of reserve } q_b \end{aligned}$$

The profit in terms of fractional trades is

$$\textit{profit} \quad : \quad V_b \cdot X_b \ - \ V_a \cdot X_a \tag{8}$$

$$\textit{return} \quad : \quad \frac{V_b \cdot X_b}{V_a \cdot X_a} \ - \ 1 \tag{9}$$

Apply the formula (4) for $X_b$ in terms of $X_a$ to describe return as a function of the trade $X_a$.

$$\left(\frac{V_b}{V_a}\right) \cdot \left(\frac{1-f}{1+(1-f)\cdot X_a}\right) \ - \ 1 \tag{10}$$

This is a most convenient formula for return. It will be used several times. The first application is the derivation of the threshold inequality.

**Theorem 1** *(Threshold Inequality)*

*Suppose there exists an $a \longrightarrow b$ arbitrage in a common unit of account. Then the threshold inequality must hold:*

$$\left(\frac{V_b}{V_a}\right) \cdot (1-f) \ > \ 1 \tag{11}$$

*where $V_a$ and $V_b$ are the marginal valuations of the* CP$(a,\ b)$ *reserves.*

***Proof*** Let $X_a > 0$ be a trade that generates positive return. The formula (10) for return must be strictly positive. Multiply both sides by the denominator $(1 + (1 -$

$f) \cdot X_a)$. Eliminating the positive term $(1 - f) \cdot X_a$ from the right side completes the proof. □

**Corollary 1 (Arbitrage interval)**

*Every profitable $a \longrightarrow b$ arbitrage trade lies in the arbitrage interval:*

$$\left(0,\ \frac{V_b}{V_a} - (1 - f)^{-1}\right) \tag{12}$$

*Moreover, the profit is exactly zero at the bounds.*

***Proof*** A zero investment in the $a \longrightarrow b$ trade clearly has zero profit. The threshold inequality (11) implies that the upper bound of the interval is the upper limit of profitable $a \longrightarrow b$ trades. The final conclusion follows from the continuity of the profit as a function of $X_a$ at the endpoints of the interval. □

**Corollary 2 (No-arbitrage interval)**

*When the pool value ratio lies in the no-arbitrage interval:*

$$\left[(1 - f),\ (1 - f)^{-1}\right] \tag{13}$$

*there can be no arbitrage trades (in either direction).*

***Proof*** A profitable trade in the $a \longrightarrow b$ implies the threshold inequality. Now switch $a$ and $b$, observe that all quantities are positive, and deduce that a profitable trade in the reverse $b \longrightarrow a$ direction implies:

$$\frac{V_b}{V_a} < (1 - f)$$

These two conclusions show that no arbitrage can occur for value ratios inside the no-arbitrage interval. □

## 3.2 Optimal trade

Within the framework developed for the threshold inequality it is also possible to determine the optimal trade. First, calculate the necessary condition:

$$V_b \cdot \frac{\partial X_b}{\partial X_a} - V_a = 0$$

Compute the derivative of $X_b$ as a function of $X_a$:

$$\begin{aligned}\frac{\partial X_b}{\partial X_a} &= \frac{\partial}{\partial X_a}\left(\frac{(1-f)\cdot X_a}{1+(1-f)\cdot X_a}\right)\\ &= \frac{(1-f)}{1+(1-f)\cdot X_a} - \frac{(1-f)^2 X_a}{(1+(1-f)X_a)^2}\\ &= \frac{(1-f)}{(1+(1-f)\cdot X_a)^2}\end{aligned}$$

Clear fractions and rearrange to obtain the expression:

$$(1+(1-f)\cdot X_a)^2 = \frac{V_b\cdot(1-f)}{V_a} \tag{14}$$

Only the positive root leads to an $a \longrightarrow b$ trade. Solving for the positive root yields the trade that satisfies the first-order necessary condition:

$$X_a = (1-f)^{-1}\cdot\left[\left(\frac{V_b\cdot(1-f)}{V_a}\right)^{1/2} - 1\right] \tag{15}$$

Next step is to prove that this is a maximum. The second derivative of the profit expression is strictly negative for any profitable $a \longrightarrow b$ trade, so the necessary condition (15) does identify a local maximum. According to Theorem 1, profit is zero at the endpoints of the arbitrage interval, so this maximum must lie in the interior of this interval. Hence the equation (15) identifies the trade with maximum profit.

## 4 Two-hop carry trade example

The framework used to prove the threshold inequality does not specify how the marginal prices are derived. When the arbitrage is between a single-hop and a double-hop, as in Section 2.6, the marginal prices come from actual swaps. This enables a deeper look at the conditions for arbitrage.

### 4.1 Model the cycle as a route

Consider the route that implements the cycle $c \longrightarrow a \longrightarrow b \longrightarrow c$. Construct the route representation of the cycle using the method of Section 2.5:

| | | |
|---|---|---|
| $N_{0,c}$ | : | node containing $c$-tokens |
| $N_a$ | : | node containing $a$-tokens |
| $N_b$ | : | node containing $b$-tokens |
| $N_{1,c}$ | : | node containing $c$-tokens |
| $\mathrm{CP}_c$ | : | directed arc from $N_{0,c}$ to $N_a$ via exchange $\mathrm{CP}_0(c, a)$ |
| $\mathrm{CP}_a$ | : | directed arc from $N_a$ to $N_b$ via exchange $\mathrm{CP}(a, b)$ |
| $\mathrm{CP}_b$ | : | directed arc from $N_b$ to $N_{1,c}$ via exchange $\mathrm{CP}_1(b, c)$ |

The set of nodes is $\mathcal{N} = \{N_{0,c}, N_a, N_b, N_{1,c}\}$ and the set of arcs is $\mathcal{A} = \{\mathrm{CP}_c, \mathrm{CP}_a, \mathrm{CP}_b\}$. Now consider a flow through the cycle:

$$
\begin{aligned}
\delta_c =& \text{outflow from node } N_{0,c} \\
& \text{transformed on arc } \mathrm{CP}_0 \text{ to } \delta_a \\
\delta_a =& \text{inflow to node } N_a \\
& \text{outflow from node } N_a \\
& \text{transformed on arc } \mathrm{CP}_a \text{ to } \delta_b \\
\delta_b =& \text{inflow to node } N_b \\
& \text{outflow from node } N_b \\
& \text{transformed on arc } \mathrm{CP}_b \text{ to } \Delta_c \\
\Delta_c =& \text{inflow to node } N_{1,c}
\end{aligned}
$$

The changes (inflow minus outflow) in the quantity of tokens at each node is:

$$
\begin{aligned}
D(N_{0,c}) &= 0 - \delta_c \\
D(N_a) &= \delta_a - \delta_a \\
D(N_b) &= \delta_b - \delta_b \\
D(N_{1,c}) &= \Delta_c - 0
\end{aligned}
$$

The total change from this flow is the sum of the changes on the set of nodes

$$
\sum_{N \in \mathcal{N}} D(N) = \Delta_c - \delta_c \tag{16}
$$

This, of course, is the profit (or loss) in the flow. To maximize the profit is a very complex undertaking. To see this, recall that

$$
\begin{aligned}
\Delta_c &= \frac{q_{1,c} \cdot (1 - f_1) \cdot \delta_b}{q_{1,b} + (1 - f_1) \cdot \delta_b} \\
\delta_b &= \frac{q_b \cdot (1 - f) \cdot \delta_a}{q_a + (1 - f) \cdot \delta_a} \delta_a \qquad = \frac{q_{0,a} \cdot (1 - f_0)\delta_c}{q_{0,c} + (1 - f_0) \cdot \delta_c}
\end{aligned}
$$

Progress can be made by applying approximations to the marginal prices used to generate in the values

$$V_a = p_a \cdot q_a$$
$$V_b = p_b \cdot q_b$$

where $p_a$ is the buying price for $a$-tokens and $p_b$ is the selling price for $b$-tokens in the unit of account, which is $c$-tokens.

#### 4.1.1 Price for buying *a*-tokens

The optimal arbitrage begins with buying $a$-tokens in the amount $\delta_a$, where $\delta_a$ is to be determined by the maximization. These tokens are obtained from the exchange $CP_0(a,\ c)$ with reserves $(q_{0,a},\ q_{0,c})$. To get the appropriate quantity $\delta_c$, apply the formula (5) with $X_a^0 := \delta_a / q_{0,a}$. Then

$$\begin{aligned}\delta_c &= q_{0,c} \cdot (1-f_0)^{-1} \cdot \left(\frac{X_a^0}{1-X_a^0}\right)\\ &= \left(\frac{q_{0,c}}{q_{0,a}}\right) \cdot (1-f_0)^{-1} \cdot \left(\frac{1}{1-X_a^0}\right) \cdot \delta_a\end{aligned}$$

The approximate buying price of $a$-tokens for small trades is

$$p_a = \left(\frac{q_{0,c}}{q_{0,a} \cdot (1-f_0)}\right) \tag{17}$$

#### 4.1.2 Price from selling *b*-tokens

The arbitrage completes by selling $\delta_b$ tokens on the exchange $\mathrm{CP}_1(b,\ c)$. To calculate the number $\Delta_c$ of $c$-tokens returned, apply the formula (4) with $X_b^1 := \delta_b / q_{1,b}$. Then

$$\begin{aligned}\Delta_c &= q_{1,c} \cdot (1-f_1) \cdot \frac{X_b^1}{1+(1-f_1) \cdot X_b^1}\\ &= \left(\frac{q_{1,c}}{q_{1,b}}\right) \cdot (1-f_1) \cdot \frac{1}{1+(1-f_1) \cdot X_b^1} \cdot \delta_b\end{aligned}$$

The approximate selling price of $b$-tokens for small trades is

$$p_b^1 = \left(\frac{q_{1,c} \cdot (1-f_1)}{q_{1,b}}\right) \tag{18}$$

### 4.1.3 Approximate optimization

The maximization can now be performed using the small trade approximate prices (17) and (18)

$$\begin{aligned} V_a &= p_a^0 \cdot q_a \\ &= \left( \frac{q_{0,c}}{q_{0,a} \cdot (1 - f_0)} \right) \cdot q_a \end{aligned}$$

$$\begin{aligned} V_b &= p_b^1 \cdot q_b \\ &= \left( \frac{q_{1,c} \cdot (1 - f_1)}{q_{1,b}} \right) \cdot q_b \end{aligned}$$

and applying the formula for the optimal solution (15).

## 4.2 Numerical examples

This section presents numerical examples to understand whether the approximate optimal solution from the section above will be ruled out by the exact formula (16).

| $q_b$ | $V$ | $X_a$ | $X_0 c$ | $X_b$ | $X_1 c$ | est. retn | act. retn |
|---|---|---|---|---|---|---|---|
| 103.63 | 1.0300 | 0.0133 | 0.0136 | 0.0131 | 0.0134 | 1.35 | -1.34 |
| 106.73 | 1.0610 | 0.0284 | 0.0293 | 0.0275 | 0.0285 | 2.86 | -2.90 |
| 109.94 | 1.0928 | 0.0437 | 0.0458 | 0.0417 | 0.0437 | 4.41 | -4.52 |
| 113.23 | 1.1256 | 0.0592 | 0.0631 | 0.0557 | 0.0592 | 5.97 | -6.20 |
| 116.63 | 1.1593 | 0.0749 | 0.0812 | 0.0695 | 0.0747 | 7.56 | -7.94 |
| 120.13 | 1.1941 | 0.0908 | 0.1002 | 0.0830 | 0.0905 | 9.17 | -9.73 |
| 123.73 | 1.2299 | 0.1070 | 0.1202 | 0.0964 | 0.1063 | 10.80 | -11.58 |
| 127.45 | 1.2668 | 0.1235 | 0.1413 | 0.1096 | 0.1222 | 12.46 | -13.48 |
| 131.27 | 1.3048 | 0.1402 | 0.1635 | 0.1226 | 0.1383 | 14.14 | -15.43 |
| 135.21 | 1.3440 | 0.1571 | 0.1869 | 0.1354 | 0.1544 | 15.85 | -17.42 |
| 139.26 | 1.3843 | 0.1743 | 0.2117 | 0.1480 | 0.1705 | 17.58 | -19.46 |
| 143.44 | 1.4258 | 0.1917 | 0.2379 | 0.1605 | 0.1867 | 19.34 | -21.54 |
| 147.75 | 1.4686 | 0.2094 | 0.2657 | 0.1727 | 0.2028 | 21.13 | -23.66 |
| 152.18 | 1.5127 | 0.2274 | 0.2952 | 0.1848 | 0.2190 | 22.94 | -25.81 |
| 156.74 | 1.5580 | 0.2456 | 0.3265 | 0.1967 | 0.2351 | 24.78 | -28.00 |
| 161.45 | 1.6048 | 0.2641 | 0.3600 | 0.2084 | 0.2512 | 26.65 | -30.21 |
| 166.29 | 1.6529 | 0.2829 | 0.3957 | 0.2200 | 0.2672 | 28.54 | -32.45 |
| 171.28 | 1.7025 | 0.3019 | 0.4338 | 0.2314 | 0.2832 | 30.47 | -34.72 |
| 176.42 | 1.7536 | 0.3213 | 0.4748 | 0.2426 | 0.2991 | 32.42 | -37.01 |
| 181.71 | 1.8062 | 0.3409 | 0.5188 | 0.2537 | 0.3149 | 34.40 | -39.31 |

**Table 1** Post-swap reserve balances, marginal price, and $b$- and $a$-token valuations.

## References

[1] Hayden Adams. *Uniswap Whitepaper*. (accessed) 18-March-2024). Feb. 2020. URL: `https://hackmd.io/@HaydenAdams/HJ9jLsfTz`.

[2] Hayden Adams et al. *Uniswap v3 Core*. (accessed: 25-October-2023). Mar. 2021. URL: `https://uniswap.org/whitepaper-v3.pdf`.

## 5 Appendix: Python code

```
import numpy as np
# Swap exact input - equation (4)
def swap_fwd(x0, f):
    assert(x0 >= 0)
    assert(x0 <= 1)
    x1 = (1 - f) * x0 / (1 + (1 - f) * x0)
    return x1

# Swap exact output - equation (5)
def swap_bwd(x1, f):
    assert(x1 >= 0)
    assert(x1 <= 1)
    x0 = x1/ ( (1 - f) * (1 - x1) )
    return x0

# Optimal arbitrage - equation (15)
def opt_arbitrage(v, f):
    leg0 = (1 - f) * (np.sqrt((1 - f) * v) - 1)
    leg1 = swap_fwd(leg0, f)
    ret = v * leg1 / leg0 - 1
    return leg0, leg1, ret
```

**Fig. 1** Swap formula and optimal trade

```
for n in range(N):
    q_b = q_b * (1 + 10*FEE)
    V = q_b * P_B / (Q * P_A)
    # print(V, FEE)
    X_a, X_b, R_est = opt_arbitrage(V, FEE)
    X_0c = swap_bwd(X_a, FEE)
    X_1b  = X_b * q_b / Q             # convert to fraction of q_1c
    X_1c = swap_fwd(X_1b, FEE)
    R_exact = X_1c / X_0c - 1
```

**Fig. 2** Code to generate Table 4.2