

Automated Exchange Economies

Bryan Routledge
Carnegie Mellon
University

Yikang Shen
Carnegie Mellon
University

Ariel Zetlin-Jones
Carnegie Mellon
University

May 31, 2024

Making Markets via Smart Contracts

- How to design a “centralized” exchange on a distributed ledger?
 - Key friction: verifiable communications are (typically) costly
 - Suggests limit order books may be impractical
- Existing solution: ad hoc pricing functions called automated market makers
- Our research: establish a framework to evaluate how AMMs support liquidity provision and exchange

Making Markets via Smart Contracts

- An Automated Market Maker is a Smart Contract
 - Smart contract \Leftarrow deterministic, verifiable script on a blockchain
- AMM Smart Contract has two key functions:
 1. Liquidity Provision Rules
 - LPs deposit or withdraw a portfolio of tokens:
 - Deposit (Mint): $(+e_a, +e_b)$ or Withdraw (Burn): $(-e_a, -e_b)$
 2. Liquidity Taking Rules:
 - LTs swap tokens at some pre-specified schedule
 - e.g. Swap a for b : $(+q_a, -q_b)$

Making Markets via Smart Contracts

The screenshot shows the Etherscan interface for a smart contract. At the top, the Etherscan logo is on the left, and navigation links for Home, Blockchain, Tokens, NFTs, Resources, Developers, and More are on the right. Below the logo, the contract address is displayed as 0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852. There are buttons for Buy, Exchange, Play, and Gaming. A feature tip suggests adding a private address tag. The main navigation bar includes Transactions, Internal Transactions, Token Transfers (ERC-20), NFT Transfers, Contract (highlighted), Events, Analytics, Multichain Portfolio, and Info. Below this, there are buttons for Code, Read Contract, and Write Contract, along with a search bar for source code. A warning section titled "Similar Match Source Code" indicates that the contract matches the deployed bytecode of another contract (0xB4e16d01...1Ec28C9Dc) but that the constructor portion might be different. The contract details section shows the name "UniswapV2Pair", optimization enabled "Yes with 999999 runs", compiler version "v0.5.16+commit.9c3226ce", and other settings "default evmVersion, GNU GPLv3 license". At the bottom, there is a section for "Contract Source Code (Solidity)" with buttons for IDE, Outline, and More Options, and icons for copy, share, and refresh.

Etherscan

Home Blockchain Tokens NFTs Resources Developers More | Sign In

Contract 0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852 Buy Exchange Play Gaming

Feature Tip: Add private address tag to any address under My Name Tag !

Transactions Internal Transactions Token Transfers (ERC-20) NFT Transfers **Contract** Events Analytics Multichain Portfolio Info Advanced Filter

Code Read Contract Write Contract Search Source Code

Similar Match Source Code

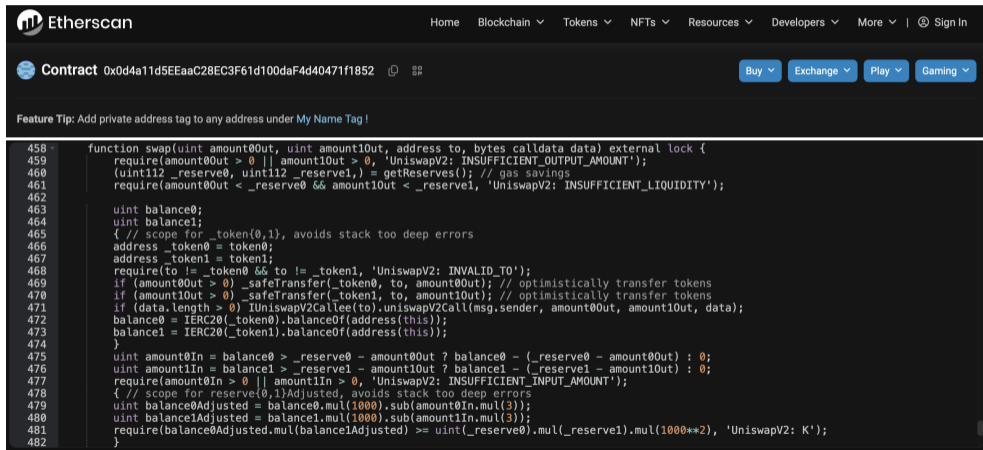
- This contract matches the deployed Bytecode of the Source Code for Contract 0xB4e16d01...1Ec28C9Dc
- The constructor portion of the code might be different and could alter the actual behaviour of the contract

Contract Name: **UniswapV2Pair** Optimization Enabled: **Yes with 999999 runs**

Compiler Version: **v0.5.16+commit.9c3226ce** Other Settings: **default evmVersion, GNU GPLv3 license**

Contract Source Code (Solidity) IDE Outline More Options

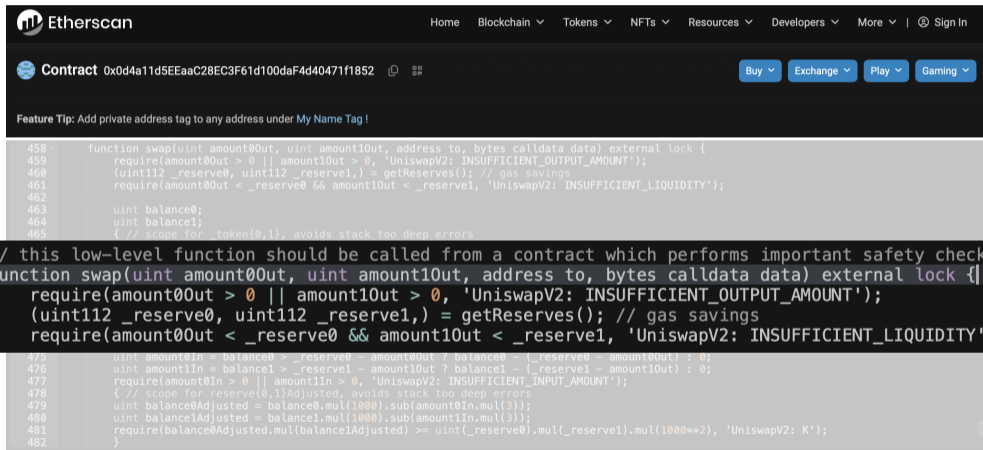
Making Markets via Smart Contracts



The screenshot shows the Etherscan website interface. At the top, there is a navigation bar with links for Home, Blockchain, Tokens, NFTs, Resources, Developers, and More. A 'Sign In' button is also present. Below the navigation bar, the page title is 'Contract 0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852'. There are buttons for 'Buy', 'Exchange', 'Play', and 'Gaming'. A 'Feature Tip' message says 'Add private address tag to any address under My Name Tag!'. The main content area displays the source code for a swap function in Solidity, starting at line 458 and ending at line 482. The code includes checks for sufficient output and liquidity, and calculates adjusted balances for two tokens.

```
458 function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
459     require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
460     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
461     require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');
462
463     uint balance0;
464     uint balance1;
465     { // scope for _token{0,1}, avoids stack too deep errors
466         address _token0 = token0;
467         address _token1 = token1;
468         require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
469         if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
470         if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
471         if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
472         balance0 = IERC20(_token0).balanceOf(address(this));
473         balance1 = IERC20(_token1).balanceOf(address(this));
474     }
475     uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
476     uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
477     require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
478     { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
479         uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
480         uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
481         require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
482     }
```

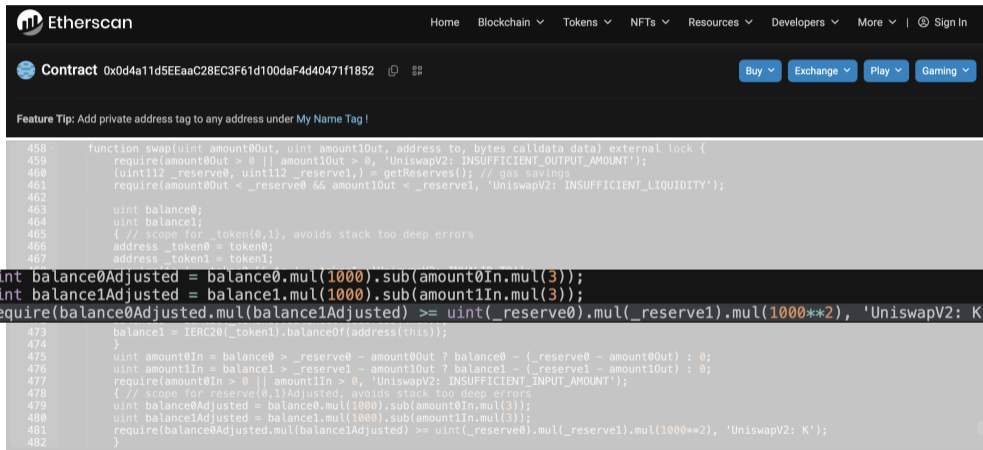
Making Markets via Smart Contracts



The screenshot shows the Etherscan interface for a smart contract. The contract address is 0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852. The code snippet is as follows:

```
458     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
459         require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
460         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
461         require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');
462
463         uint balance0;
464         uint balance1;
465         { // scope for token{0,1}, avoids stack too deep errors
// this low-level function should be called from a contract which performs important safety checks
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
    require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');
475     uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
476     uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
477     require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
478     { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
479         uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
480         uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
481         require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
482     }
```

Making Markets via Smart Contracts



The screenshot shows the Etherscan interface for a smart contract. The contract address is 0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852. The source code is displayed in a dark-themed editor. A specific section of the code is highlighted in a dark box, showing the calculation of adjusted balances and a liquidity requirement check.

```
458     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
459         require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
460         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
461         require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');
462
463         uint balance0;
464         uint balance1;
465         { // scope for _token{0,1}, avoids stack too deep errors
466             address _token0 = token0;
467             address _token1 = token1;
468
469             uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
470             uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
471             require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
472
473             balance1 = IERC20(_token1).balanceOf(address(this));
474         }
475         uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
476         uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
477         require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
478         { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
479             uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
480             uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
481             require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
482         }
```

Making Markets via Smart Contracts

- Liquidity Taking Rules:

- Swap a for b : $(+q_a, -q_b)$

- Rule implemented as function embedded in smart contract

- Price schedule defined by “Constant Product Rule”:

$$(e_a + q_a)(e_b - q_b) = e_a e_b$$

- Slope of schedule defines implicit relative price of token b for a

Making Markets via Smart Contracts

- Questions
 - How should LPs choose deposits on AMMs?
 - How does design of the price schedule impact gains to trade between LPs and LTs?
- This paper:
 - Develop simple, tractable economic framework to answer these questions
 - Findings:
 - Adverse selection distorts intermediation *quantities* rather than *prices*
 - Typically suboptimal for LPs to deposit tokens in equal values as conventionally suggested
 - Efficiency of price function: trade-off between volume and adverse selection

Related Literature

- AMM Price “discovery”
 - **Do AMM prices reflect “true” prices?**
 - Angeris and Chitra (2020), Angeris et al (2021), Aoyagi (2022)

- AMM Liquidity
 - **What are the costs of creating AMM liquidity?**
 - Capponi and Jia (2021), Milionis et al (2022), Hasbrouck, Rivera, and Saleh (2022), Lehar and Parlour (2023), Fabi and Prat (2023)

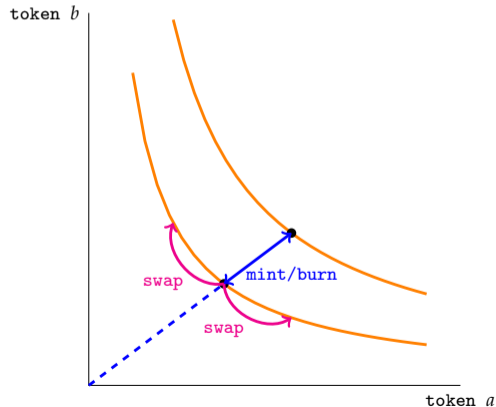
- AMM Design
 - **What is the optimal price function?**
 - Park (2022), Bergault et al (2023), Goyal et al (2023), Milionis, Moallemi, and Roughgarden (2023)

Active LIQUIDITY MANAGEMENT

Liquidity Providers

- Industry/Literature defines liquidity providers as *passive*
 1. Interact with contract infrequently
 2. Only use Deposit/Withdraw functions

- What does the data say?



Liquidity Providers are Infrequent but “Active”

Uniswap Transaction Counts

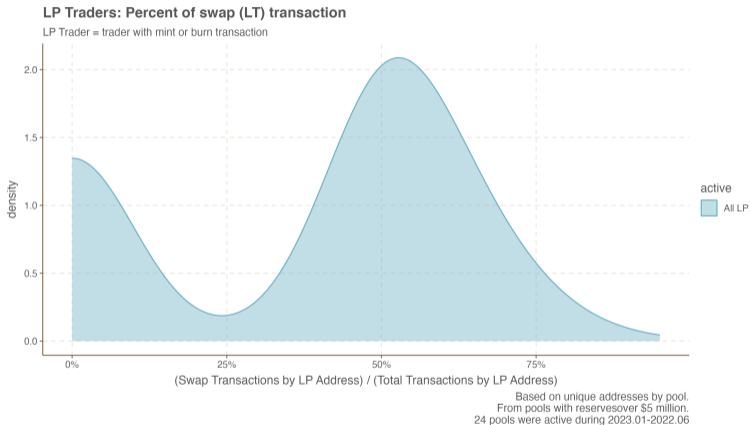
2023-01-01 – 2023-06-30

trader	burns	mints	swaps	total
LP	5,375	24,838	5,693	35,906
LT	0	0	1,252,596	1,252,596
Total	5,375	24,838	1,258,289	1,288,502

From pools with reserves over \$5 million.
24 pools were active during this time period

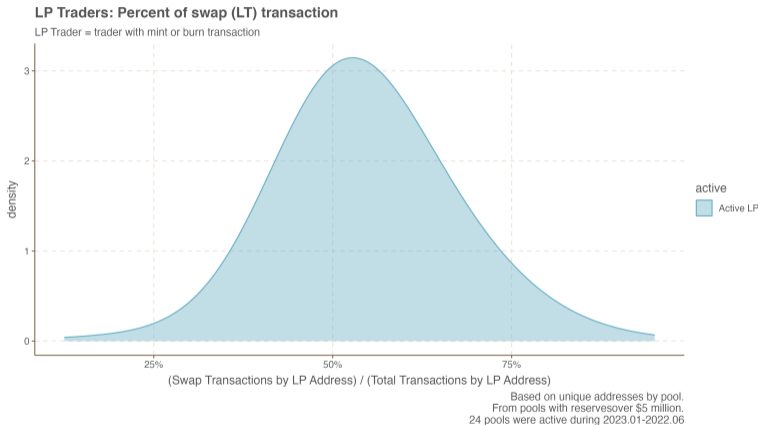
- LPs have few interactions with contract relative to non-LPs
- LPs do use both functionalities
 - 👉 LP actions impact exchange prices

Liquidity Providers are Infrequent but “Active”



- By address: some LPs use Swap Functions, some do not

Liquidity Providers are Infrequent but “Active”



- By address: some LPs use Swap Functions, some do not
- Among active LPs, swaps make up large portion of activity

Liquidity Providers are Heterogeneous

Uniswap **TRADER** Counts

2023-01-01 – 2023-06-30

	Unique Traders	Total Transactions	Liquidity Provisions	Liquidity Takings
LP active	1,854	10,069	43.5%	56.5%
LP passive	940	25,813	100.0%	0.0%

Based on unique addresses by pool From pools with reserves over \$5 million. 24 pools were active during this time period

- Some LP addresses are passive and some are active
- Our paper addresses behavior of active LPs

ENVIRONMENT

Environment

- 2-by-2 economy (2 agents, 2 assets) in finite time
- Two risk-neutral agents:
 - Alice (LP) owns endowments (E_a, E_b) of a pair of tokens a and b
 - Bob (LT) may trade using the AMM (large number of “Bob”s)
- Timing in each period
 1. LP deposits tokens with exchange
 2. Public information about assets realized
 3. LT trades at exchange

Assets and Information

- Tokens $i \in \{a, b\}$ yield terminal value $\exp(d_{i,T})$ where

$$d_{i,T} = \sum_{t=0}^T y_{i,t} + \epsilon_i$$

- Interpret $\exp(d_{i,T})$ as future “price” or service flow from the token
- Residual independent uncertainty realized at T : $\mathbb{E}[\exp(\epsilon_i)] = 1$
- Public information $y_{i,t}$ arrives each period:
 - $y_{i,t} = 0$ with prob $\hat{\pi}$, $y_{i,t} = -\Delta_l$ or $+\Delta_h$ with prob $(1 - \hat{\pi})/2$
 - Beginning of period beliefs

$$\mu_{i,t} = \mathbb{E}[\exp(d_{i,T}) | y_0, \dots, y_{t-1}] = \mathbb{E}_t[\exp(d_{i,T})]$$

Information, Assets, and Preferences

- LP makes deposits with expected valuation $\mu_{i,t} = \mathbb{E}_t[\exp(d_{i,T})]$
- LT trades with expected valuation $\hat{\mu}_{i,t} = \mathbb{E}_{t+1}[\exp(d_{i,T})] \exp(\eta_i)$
(η_i is a preference shock)

Information, Assets, and Preferences

- LP makes deposits with expected valuation $\mu_{i,t} = \mathbb{E}_t[\exp(d_{i,T})]$
- LT trades with expected valuation $\hat{\mu}_{i,t} = \mathbb{E}_{t+1}[\exp(d_{i,T})] \exp(\eta_i)$
(η_i is a preference shock)

☞ Expositional assumption

- If $y_{i,t} \in \{-\Delta_l, \Delta_h\}$ (for some i) then $\eta_a = \eta_b = 0$

- Information event ($y_{i,t} \in \{-\Delta_l, \Delta_h\}$ some i) \Rightarrow **pure informed trading** event
- No information ($y_{a,t} = y_{b,t} = 0$) \Rightarrow **pure taste/noise trading** event
- LP trades-off losses from informed trading with gains from noise trading

LT's Problem

- Bob/LT faces a price schedule and maximizes expected dividends:

$$\max_{q_a, q_b} -\hat{\mu}_{a,t}q_a + \hat{\mu}_{b,t}q_b$$

subject to

$$(e_{a,t} + q_a)(e_{b,t} - q_b) = e_{a,t}e_{b,t}$$

LT's Problem

- Bob/LT faces a price schedule and maximizes expected dividends:

$$\max_{q_a, q_b} -\hat{\mu}_{a,t}q_a + \hat{\mu}_{b,t}q_b$$

subject to

$$(e_{a,t} + q_a)(e_{b,t} - q_b) = e_{a,t}e_{b,t}$$

- Optimality implies

$$\frac{\hat{\mu}_{b,t}}{\hat{\mu}_{a,t}} = \frac{e_{a,t} + q_a}{e_{b,t} - q_b} \equiv \frac{x_{a,t}}{x_{b,t}}$$

LT's Problem

- Bob/LT faces a price schedule and maximizes expected dividends:

$$\max_{q_a, q_b} -\hat{\mu}_{a,t}q_a + \hat{\mu}_{b,t}q_b$$

subject to

$$(e_{a,t} + q_a)(e_{b,t} - q_b) = e_{a,t}e_{b,t}$$

- Optimality implies

$$\frac{\hat{\mu}_{b,t}}{\hat{\mu}_{a,t}} = \frac{e_{a,t} + q_a}{e_{b,t} - q_b} \equiv \frac{x_{a,t}}{x_{b,t}}$$

- Impose this behavior and examine Alice/LP's optimal choice of deposits

☞ Alice/LP's ex-post allocation satisfies:

$$x_{a,t}x_{b,t} = e_{a,t}e_{b,t}, \quad \hat{\mu}_{a,t}x_{a,t} = \hat{\mu}_{b,t}x_{b,t}$$

LP's Dynamic Problem

- Assume probability of pure noise trade event is π and pure informed trade is $1 - \pi$

$$V_T(E_a, E_b, \vec{\mu}_T) = \mu_{a,T}E_a + \mu_{b,T}E_b$$

$$V_t(E_a, E_b, \vec{\mu}_t) = \max_{e_a, e_b} \pi \mathbb{E}V_{t+1}(E'_a, E'_b, \vec{\mu}_{t+1}) + (1 - \pi) \mathbb{E}V_{t+1}(E'_a, E'_b, \vec{\mu}_{t+1})$$

$$\text{with } \begin{aligned} E'_a &= E_a - e_a + x_a && \text{Accounting} \\ E'_b &= E_b - e_b + x_b \end{aligned}$$

$$\begin{aligned} \mu_{t+1} &= \mu_t && \text{if } y_t = 0 && \text{Beliefs} \\ \mu_{t+1} &= \hat{\mu}_t && \text{if } y_t \neq 0 \end{aligned}$$

$$\begin{aligned} e_a e_b &= x_a x_b && \text{Constant Product} \\ \hat{\mu}_{a,t} x_a &= \hat{\mu}_{b,t} x_b && \text{Bob's optimality} \end{aligned}$$

- Rest of talk focus on one-shot game (drop t subscripts)

OPTIMAL LIQUIDITY PROVISION

LP's Problem

- LP's one-shot problem

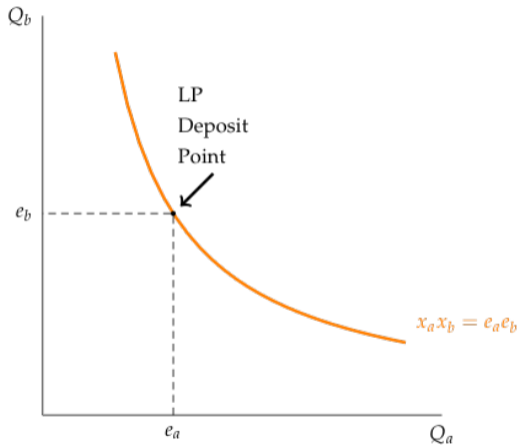
$$\max_{e_a, e_b} \pi \sum_i \mu_i \mathbb{E}[x_i - e_i] + (1 - \pi) \sum_i \mathbb{E}[\hat{\mu}_i(x_i - e_i)]$$

subject to

$$x_a x_b = e_a e_b, \quad \hat{\mu}_a x_a = \hat{\mu}_b x_b, \quad 0 \leq e_j \leq E_j$$

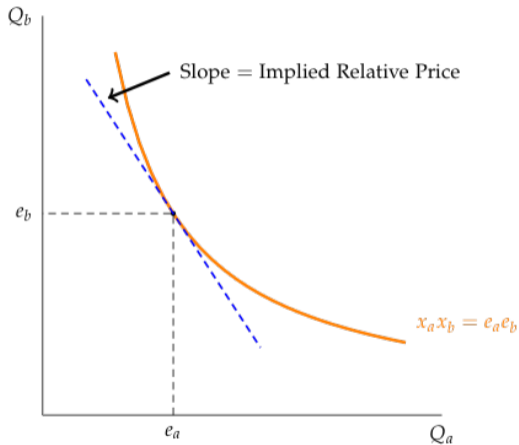
- LPs deposit choice influences shape and position of pricing curve

AMM Economics in a Graph



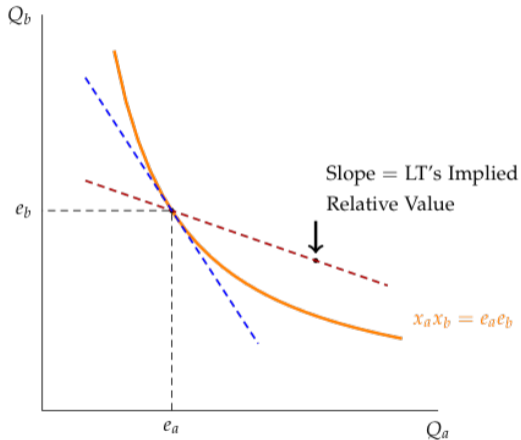
- CPMM implicitly defines relative price of tokens for LTs

AMM Economics in a Graph



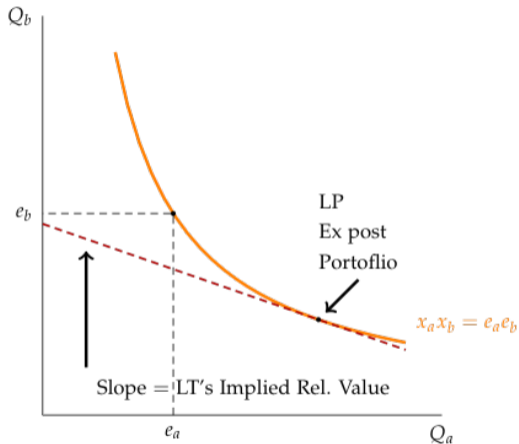
- CPMM implicitly defines relative price of tokens for LTs

AMM Economics in a Graph



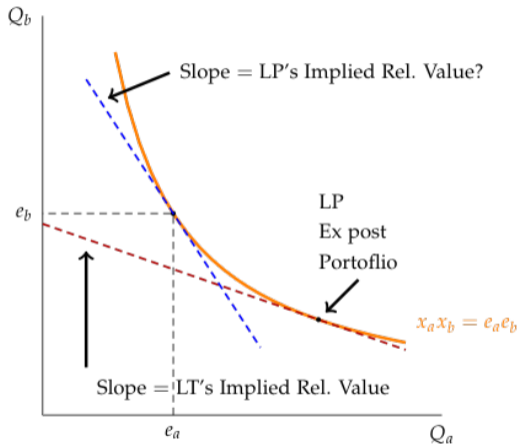
- Bob (LT) trades if relative valuation is different from CPMM implicit relative price

AMM Economics in a Graph



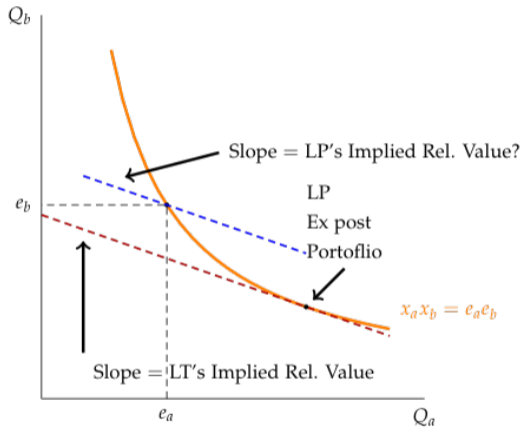
- Bob (LT) trades if relative valuation is different from CPMM implicit relative price

AMM Economics in a Graph



- Alice (LP) gains if relative valuation close to initial CPMM implicit relative price

AMM Economics in a Graph



- Alice (LP) loses if (ex post) relative valuation is similar to that of Bob (LT)

LP's Problem

- Re-write LP's problem

$$\max_{e_a, e_b} [\pi\gamma_U + (1 - \pi)\gamma_I] \sqrt{\mu_a e_a} \sqrt{\mu_b e_b} - (\sqrt{\mu_a e_a} - \sqrt{\mu_b e_b})^2$$

where

- γ_U, γ_I functions of distributions of belief dispersion $H(\mu_i / \hat{\mu}_i)$
- $\gamma_U > 0$ and $\gamma_I < 0$

LP's Problem

- Re-write LP's problem

$$\max_{e_a, e_b} [\pi\gamma_U + (1 - \pi)\gamma_I] \sqrt{\mu_a e_a} \sqrt{\mu_b e_b} - (\sqrt{\mu_a e_a} - \sqrt{\mu_b e_b})^2$$

where

- γ_U, γ_I functions of distributions of belief dispersion $H(\mu_i/\hat{\mu}_i)$
- $\gamma_U > 0$ and $\gamma_I < 0$
- Gains to LP only when π is large enough

LP's Problem

- Re-write LP's problem

$$\max_{e_a, e_b} [\pi\gamma_U + (1 - \pi)\gamma_I] \sqrt{\mu_a e_a} \sqrt{\mu_b e_b} - (\sqrt{\mu_a e_a} - \sqrt{\mu_b e_b})^2$$

where

- γ_U, γ_I functions of distributions of belief dispersion $H(\mu_i / \hat{\mu}_i)$
- $\gamma_U > 0$ and $\gamma_I < 0$
- Gains to LP only when π is large enough
- When gains to LP, deviation from equal-value deposit yields first order gains and second order losses

LP's Problem

- Re-write LP's problem

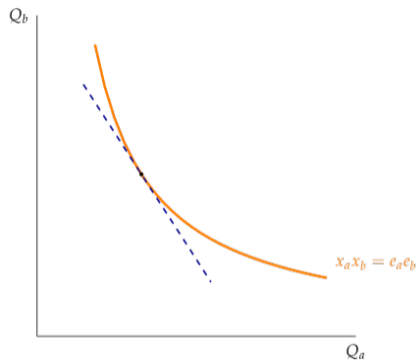
$$\max_{e_a, e_b} [\pi\gamma_U + (1 - \pi)\gamma_I] \sqrt{\mu_a e_a} \sqrt{\mu_b e_b} - (\sqrt{\mu_a e_a} - \sqrt{\mu_b e_b})^2$$

where

- γ_U, γ_I functions of distributions of belief dispersion $H(\mu_i/\hat{\mu}_i)$
- $\gamma_U > 0$ and $\gamma_I < 0$
- Gains to LP only when π is large enough
- When gains to LP, deviation from equal-value deposit yields first order gains and second order losses
- Revision to conventional wisdom:
 - ☞ “LPs should deposit in equal values only if no gains to trade in market”

LP's Problem: A Simple Rule

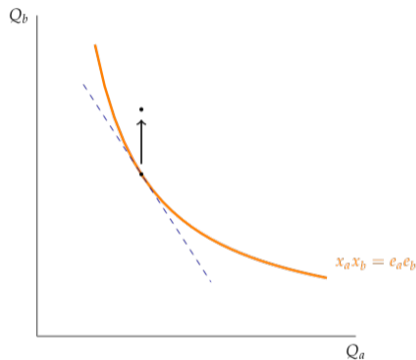
- With only uninformed trading, easy for LP to guarantee no losses



- Tangency and constant product implies $\mu_a e_a = \mu_b e_b$

LP's Problem: A Simple Rule

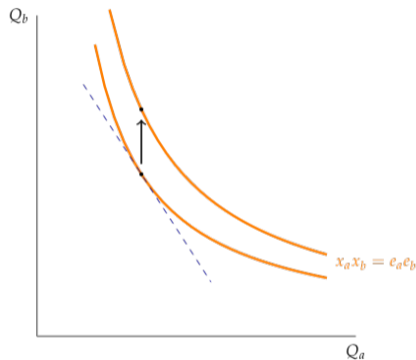
- With only uninformed trading, easy for LP to guarantee no losses



- Tangency and constant product implies $\mu_a e_a = \mu_b e_b$

LP's Problem: A Simple Rule

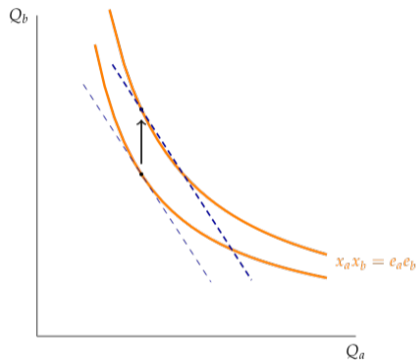
- With only uninformed trading, easy for LP to guarantee no losses



- Tangency and constant product implies $\mu_a e_a = \mu_b e_b$

LP's Problem: A Simple Rule

- With only uninformed trading, easy for LP to guarantee no losses



- Tangency and constant product implies $\mu_a e_a = \mu_b e_b$
- Small deviations yield second order losses around the deposit point but first order gains for larger trades

Proposition (*Optimal Liquidity*)

The optimal liquidity deposit with π proportion of uninformed trading and $1 - \pi$ proportion of informed trading satisfies

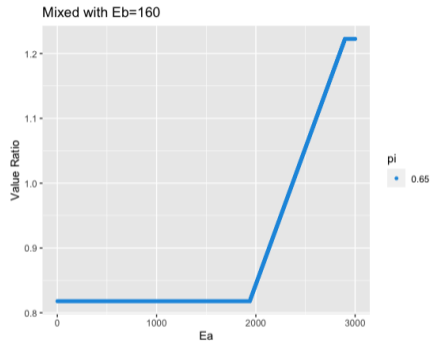
$$e_a^* = E_a, e_b^* = \min \left\{ \left(\frac{\pi}{2} \left(\mathbb{E}_U[\omega] + \mathbb{E}_U \left[\frac{1}{\omega} \right] \right) + (1 - \pi) \mathbb{E}_I[\psi] \right)^2 \frac{\mu_a}{\mu_b} E_a, E_b \right\}, \text{ if } \mu_a E_a \leq \mu_b E_b$$

and

$$e_a^* = \min \left\{ \left(\frac{\pi}{2} \left(\mathbb{E}_U[\omega] + \mathbb{E}_U \left[\frac{1}{\omega} \right] \right) + (1 - \pi) \mathbb{E}_I[\psi] \right)^2 \frac{\mu_b}{\mu_a} E_b, E_a \right\}, e_b^* = E_b, \text{ if } \mu_a E_a > \mu_b E_b$$

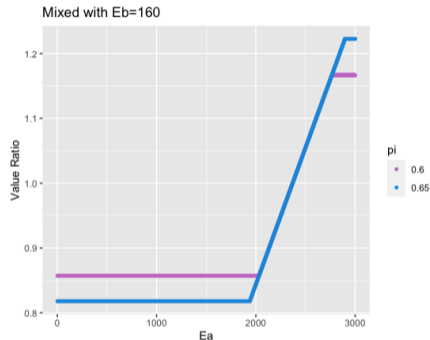
- Linear preferences \Rightarrow expect (and find) corner solutions

Optimal Liquidity: Comparative Statics



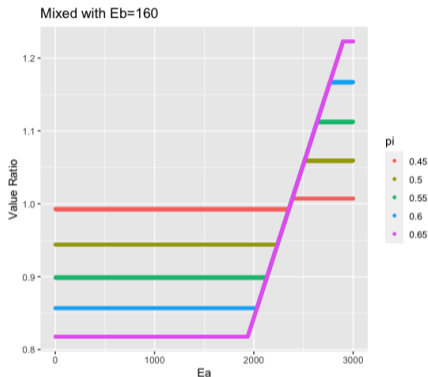
1. *Change in Endowments:* Value ratio $\mu_a e_a / \mu_b e_b$ rises with E_a

Optimal Liquidity: Comparative Statics



1. *Change in Endowments*: Value ratio $\mu_a e_a / \mu_b e_b$ rises with E_a
2. *Change in Informed Trading*: Value ratio $\mu_a e_a / \mu_b e_b$ closer to 1 with more *informed* trade

Optimal Liquidity: Comparative Statics



1. *Change in Endowments*: Value ratio $\mu_a e_a / \mu_b e_b$ rises with E_a
 2. *Change in Informed Trading*: Value ratio $\mu_a e_a / \mu_b e_b$ closer to 1 with more *informed* trade
- ☞ Adverse Selection distorts intermediation quantities

EFFICIENCY

Implications for AMM Design

- How should the price schedule be designed?
- Framework offers a new tradeoff:
 - Convexity hinders trading volume and reduces realized gains to trade
 - Convexity offers protection from informed trading

Local Convexity of the Price Function

- Consider a class of of price functions that differ by local convexity:

$$(e_a + (1 - \tau)q_a)(e_b - (1 - \tau)q_b) = e_a e_b$$

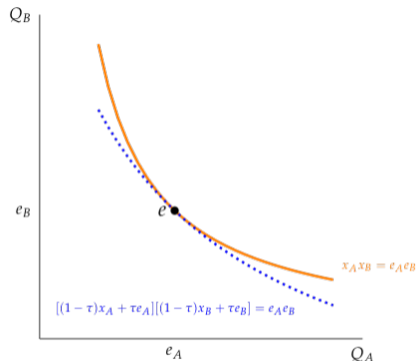
- Re-write in ex post portfolios for LP

$$((1 - \tau)x_a + \tau e_a)((1 - \tau)x_b - \tau e_b) = e_a e_b$$

Lemma

If LT's beliefs are bounded, there exists $\delta > 0$ such that for all $\tau \leq \delta$, the LP's optimal deposit does not vary with τ .

Local Convexity of the Price Function



- Increasing τ lowers convexity locally (more linear) around LP's deposit choice

Local Convexity of the Price Function

Proposition (*Efficient Price Design*)

If the LT's beliefs $\hat{\mu}_i$ are bounded, then for any convex, smoothly decreasing price function $G(\cdot)$, there exists $\delta > 0$ such that for $\tau < \delta$ the price function implicitly defined by

$$(1 - \tau)y + \tau e_b = G((1 - \tau)x + \tau e_a)$$

increases both the LP's and the LT's expected returns proportionally by $\frac{\tau}{1-\tau}$.

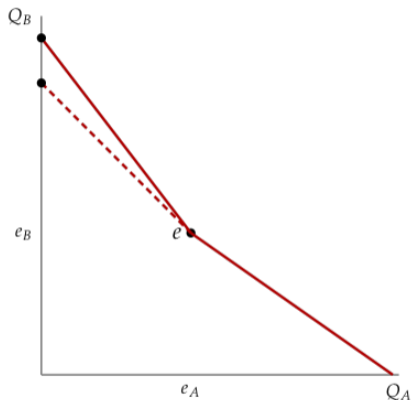
- Convex prices limit trade
- If liquidity provision profitable with convex prices, increasing trade is profitable
- ☞ A small reduction in (local) convexity raises market efficiency

Global Convexity of the Price Function

- *Claim:* If “extreme” beliefs possible, reducing local convexity is not always optimal
- Intuition:
 - Reducing local convexity promotes more (extensive) trading volume (volume effect)
 - Reducing local convexity implies lower prices for extreme trades (price effect)
 - At globally linear prices, price effect dominates, reduces profits
 - Easy to show using piece-wise linear approximation to the price function
 - Adverse selection strengthens this results

☞ Globally linear prices are not efficient

Global Convexity of the Price Function



- Consider how reducing local convexity (around (e_a, e_b)) impacts profits at the boundary

Global Convexity of the Price Function

- Let $-p_h$ be the slope of the price function for $x_a < e_a$
- Linear pricing \Rightarrow LTs trade to the boundary if $\hat{\mu}_a / \mu_a > p_h$
- Marginal effect on profits from reducing $|p_h|$:
 - Reduces prices for all uninformed LTs who trade: $-[1 - F(p_h)]$
 - Increases volume with uninformed LTs: $+(p_h - 1)f(p_h)$
 - Reduces prices for all informed LTs who trade: $-[1 - F(p_h)]$
- Net effect strictly negative as $p_h \rightarrow 1$

$$-[(1 - F(p_h)) - \pi((p_h - 1)f(p_h))]$$

☞ Globally linear prices are not efficient

WRAP UP

To do

- Exploring consequences of CPMM for allocation and gains to trade
- Extending analysis to dynamic framework
- Connecting model gains to trade to empirics from existing AMMs
- Use framework to conduct Robust Mechanism Design for AMMs

APPENDIX