Background

The aim of this project is to augment the existing code of a Uniswap AMM to calculate the properties describing the "goodness" of a proposed trade as defined in this paper.

More specifically, the Uniswap AMM contracts have been augmented to derive the following properties:

- Divergence Loss
- Linear Slippage
- Angular Slippage
- Load

To intuitively understand these properties, some brief (and informal) background can be given.

AMM

An AMM, or automated market maker, is a smart contract that provides a pool of liquidity between two resources, such as Ethereum and ERC 21 tokens.

The relevant AMM in this project is Uniswap, which maintains the invariant that $x * y = k$, where x and y are the initial amounts of the two resources, e.g. $f(x) = k/x$.

Stable state

A mathematical definition of a stable state, $(x, f(x))$, is provided in the paper. Intuitively, a stable state is one in which no profitable arbitrage can be performed by a trader.

Valuation

Valuation, $v \in (0, 1)$, assigns relative values to an AMM's assets. According to the paper, " v units of X are deemed worth $(1 - v)$ units of Y." In my opinion, this statement is actually misleading, as it implies that a higher valuation, v, means X is worth less, whereas the equations in the paper assume the opposite — that a higher valuation means X is worth more.

This is clearly shown in the equation of the $df(x)/dx$ at a stable state — when $v = 0.9$, $df(x)/dx = -9$, implying at that point, one unit of X is actually worth nine units of Y.

This can also be seen in the determination of an AMM's market cap: $vx + (1 - v)f(x)$.

Setup

Two functions are crucial to the calculation of the four properties of interest: they are defined as ϕ and ψ in the paper.

These functions are inverse of each other. Given an AMM:

- $\phi(v)$ determines the corresponding stable state $(x, f(x))$, from which no profitable arbitrage can occur
- $\psi(x)$ is the inverse. Given a stable state, $(x, f(x))$, ψ determines the corresponding valuation

In my implementation, these functions are defined within `UniswapExchange.sol` as `valueToStableState` (ϕ) and `stableStateToValue` (ψ), based on the equations provided in the paper.

Implementation

Divergence Loss & Linear Slippage

The calculation of divergence loss and linear slippage is relatively straightforward, given the `valueToStableState` and `stableStateToValue` utilities. Their implementations strictly follow the equations provided in the paper.

Relevant functions: `getEthToTokenDivergenceLoss` , `getEthToTokenDivergenceLossWrapper` , `getEthToTokenLinearSlippage` , `getTokenToEthLinearSlippage`

Angular Slippage

The calculation of angular slippage is also relatively straightforward, with the caveat that it requires the `arctan` function.

My implementation utilizes the identity, $arctan(x) = arcsin(x/(\sqrt{1+x^2}))$, where the implementation for arcsin is borrowed from an open source solidity implementation. To validate my implementation, these trigonometric functions are tested in `tests/exchange/test_trig.py` .

Relevant functions: `getEthToTokenAngularSlippage` and `getTokenToEthAngularSlippage`

Load

Since load can be derived easily from divergence loss and linear slippage, the load function is trivial after the implementation of the original functions.

Relevant functions: `getEthToTokenLoad` and `getTokenToEthLoad`

Testing

All mentioned functions have tests in `tests/exchange` :

- `test_div_loss.py`
- `test_lin_slip.py`
- `test_ang_slip.py`
- `test_load.py`
- `test_trig.py`

While these tests use contrived examples, they give confidence in the correctness of the implementation.

Notable challenges

Understanding intuition

The math definitely took a bit to work through! As the handout alludes to, I think this is likely the most difficult part of this assignment, e.g. understanding conceptually what we are calculating here. Synthesizing, here is my understanding:

- Divergence loss: The loss in market capitalization when a trade is made on an AMM that takes it from a non stable state to a stable state
- Linear slip: How the size of a trade negatively impacts its rate of return
- Angular slip: How the size of a trade negatively impacts trades that come after it.
- Load: A metric balance of cost between traders and liquidity providers.

Setting up the project

Since the open source project for this is pretty old, a lot of the dependencies are deprecated. I eventually got it working, but this was pretty annoying. In the future, it may be easier for students to just write a separate solidity script, not tied to an AMM, as I think I didn't use any of the AMM's primitives in my implementation.

Working with floating points

Since solidity doesn't natively support floats, to obtain accuracy, a common strategy is to use large integers, e.g. at $1e18$ scale.

This wasn't so much a challenge as much as it was a pain in the butt. In several instances, this caused bugs where I wasn't dividing or multiplying by $1e18$ when I needed to!