Calculating merit of trades made on a Uniswap-like Automated Market Maker

CS1951L, Blockchains & Cryptocurrencies

Capstone: May 2022

Full Name	Student ID
Diana Na Kyoung Lee	B01531157

Contents

1 | Goals

The goal of this capstone project was to edit code for an existing Automated Market Maker to implement functions that would allow users to query metrics that measure the "merit" of a trade, as defined by Engel and Herlihy [1]. These metrics include:

- Divergence Loss
- Linear Slippage
- Angular Slippage
- Load

I implemented eight functions, four for each of two trade directions, that return these figures given the input size of a proposed trade. Code for these edited functions can be found at this private repository. The rest of the unedited AMM code from the original repository can be found at the link in Section 2.1.

2 | Exisiting Work

2.1 | PhABC/uniswap-solidity

The project is based from a AMM implemented in Solidity by PhABC on Github. This constant-state AMM has similar functionality to Uniswap-v1, which was implemented in Vyper. While Uniswap-v3 is the most popular AMM in use today, its source code was too complicated for the scope of this class, and Uniswap-v1, while simpler, was written in Vyper.

2.2 | NovakDistributed/macroverse

In order to calculate arctangents, I used a Solidity library called RealMath, implemented as part of NovakDistributed macroverse project. The code is likewise available on Github. Since my project required only a small portion of the functionality, I copied only the functions I needed from RealMath into the AMM contracts. While I would rather not have copied and pasted code without permission, the two respositories I used were incompatible in Solidity versions and resulted in error without modifications. I will reach out to the authors if I ever intend to use this code for anything other than this assignment.

3 | Functions

3.1 | Divergence Loss

All of the functions below are implemented in UniswapExchange.sol, under the "Getter Functions" section. For divergence loss, I calculate the current state of the market, as defined by Engel and Herlihy, as well as the expected state after the proposed trade. Using these measures, I derive the current valuation v, as well as valuation post-trade, v'. I then use the formula provided in the paper to calculate the divergence loss of the propsed trade.

3.2 | Linear Slippage

Linear slippage is calculated in a similar fashion. However, it is worth noting that in the paper, linear slippage is defined to be a negative number. As SafeMath only works with unsigned integers, I chose to return the absolute value of the linear slippage (which would be non-negative) instead. While this value is not exactly the linear slippage, it allows for protection against overflows and underflows.

3.3 | Angular Slippage

Computing angular slippage requires calculating arctangents. Therefore, I imported RealMath's atan2 function. Since RealMath works with int128, the final calculations in this function do not use SafeMath, and this function returns an int128, instead of uint256 like the others.

3.4 | Load

The load is simply a product of the divergence loss and linear slippage. However, as noted above, the linear slippage function returns an absolute value in order to avoid using negative numbers. Therefore, the load function also returns the absolute value of the load.

3.5 | Token to Ether

I first implemented functions that generate these metrics for trading Ether in for an arbitrary token. When considering the other trade direction, I chose to see the market state as (x', y'), where x' is the amount of the token and y is the amount of Ether available. Once the market state was redefined in this way, I could calculate the figures of merit in the same ways.

```
============================= test session starts ==============================
platform darwin -- Python 3.7.3, pytest-7.1.2, pluggy-1.0.0 -- /Users/dianalee/Desktop/cs1951l/uniswap-solidity/
env/bin/python3
cachedir: .pytest_cache
rootdir: /Users/dianalee/Desktop/cs1951l/uniswap-solidity
collected 29 items

tests/exchange/test_ERC20.py::test_ERC20 PASSED                          [  3%]
tests/exchange/test_eth_to_token.py::test_swap_default PASSED            [  6%]
tests/exchange/test_eth_to_token.py::test_swap_input PASSED              [ 10%]
tests/exchange/test_eth_to_token.py::test_transfer_input PASSED          [ 13%]
tests/exchange/test_eth_to_token.py::test_swap_output PASSED             [ 17%]
tests/exchange/test_eth_to_token.py::test_transfer_output PASSED         [ 20%]
tests/exchange/test_factory.py::test_factory PASSED                      [ 24%]
tests/exchange/test_liquidity_pool.py::test_initial_balances PASSED      [ 27%]
tests/exchange/test_liquidity_pool.py::test_liquidity_pool PASSED        [ 31%]
tests/exchange/test_merit_eth_to_token.py::test_get_eth_to_token_divergence_loss PASSED [ 34%]
tests/exchange/test_merit_eth_to_token.py::test_get_eth_to_token_linear_slippage PASSED [ 37%]
tests/exchange/test_merit_eth_to_token.py::test_get_eth_to_token_angular_slippage PASSED [ 41%]
tests/exchange/test_merit_eth_to_token.py::test_get_eth_to_token_load PASSED [ 44%]
tests/exchange/test_merit_token_to_eth.py::test_get_token_to_eth_divergence_loss PASSED [ 48%]
tests/exchange/test_merit_token_to_eth.py::test_get_token_to_eth_linear_slippage PASSED [ 51%]
tests/exchange/test_merit_token_to_eth.py::test_get_token_to_eth_angular_slippage PASSED [ 55%]
tests/exchange/test_merit_token_to_eth.py::test_get_token_to_eth_load PASSED [ 58%]
tests/exchange/test_token_to_eth.py::test_swap_input PASSED              [ 62%]
tests/exchange/test_token_to_eth.py::test_transfer_input PASSED          [ 65%]
tests/exchange/test_token_to_eth.py::test_swap_output PASSED             [ 68%]
tests/exchange/test_token_to_eth.py::test_transfer_output PASSED         [ 72%]
tests/exchange/test_token_to_exchange.py::test_swap_input PASSED         [ 75%]
tests/exchange/test_token_to_exchange.py::test_transfer_input PASSED     [ 79%]
tests/exchange/test_token_to_exchange.py::test_swap_output PASSED        [ 82%]
tests/exchange/test_token_to_exchange.py::test_transfer_output PASSED    [ 86%]
tests/exchange/test_token_to_token.py::test_swap_input PASSED            [ 89%]
tests/exchange/test_token_to_token.py::test_transfer_input PASSED        [ 93%]
tests/exchange/test_token_to_token.py::test_swap_output PASSED           [ 96%]
tests/exchange/test_token_to_token.py::test_transfer_output PASSED       [100%]

============================== warnings summary ===============================
```

4 | Testing

I implemented eight tests, one for each new function, in the tests folder. Since PhABC's source code used pytest, my tests do also. Each test sets up a sample AMM with Ether and HAY tokens. Given a set amount of ether (or HAY token) to sell, the tests calculate the correct merit metric using Python. The values are then compared to calling the actual functions from the exchange's smart contract. It should be noted that the Python calculations use integer division, as Solidity's SafeMath's div function uses integer division. See the test_merit.py files above.

5 | References

[1] Daniel Engel and Maurice Herlihy. Presentation and publication: Loss and slippage in networks of automated market makers. *CoRR*, abs/2110.09872, 2021.