LVR Project

Hardships

The first thing I did after I left your office was to look up Remix, that you mentioned should help me.

At first, I just found information regarding web development and didn't realize what it has to do with me. Then I found remix.ethereum.org, and almost immediately I realized that it had the ability to help me a lot.
Nonetheless, I was not familiar with it, and it is almost unnecessary to state I am new to Solidity as well, and since I didn't manage to get my local project to work on Remix, I decided to take some tutorials to understand better what I'm facing. It took me about a day and a half, but I understood how to work with it eventually.

Then my problem was - I still didn't manage to work with any online AMM: I needed specific behaviour that I couldn't find anywhere online, like setting the exchange rate of the liquidity pool to check the changes it implied on the pool.
Hence, I gathered several contracts online, read Uniswap-V2 documentation, and worked closely with ChatGPT, to finally hold a working basic CFMM, that allows me to create a liquidity pool of two ERC20 tokens and change their rate at my will to see the impact on the pool.
Attached to the email is the ConstantProductAMM.

Implementing the LVR Method

To make the AMM use the LVR method, I changed the function that rebalances the pool after the exchange rate is changed.
Normally in liquidity pools - the prices of the pool are changed after every transaction. However, the paper specifically states: "We assume the CEX is infinitely deep, so trades have no price impact", therefore I ignored trades in my tests other than to check the contract actually works.

The function in a regular CFMM looks like this:

```
function balancePool(uint256 oldPrice, uint256 newPrice) internal {    infinite gas
    if (oldPrice == newPrice){
        return;
    }
    uint256 newReserve0 = sqrt(K/newPrice);
    uint256 newReserve1 = newReserve0 * newPrice;
    reserve0 = newReserve0;
    reserve1 = newReserve1;
}
```

The constant product is K, and it is used to maintain the Liquidity Pool at equilibrium - the reserves of the first token (reserve0) and the reserves for the second token (reserv1) both have the same worth, but the trade is sone only at the AMM values, without considering the new price.

In an AMM that implements the LVR mitigation technique - the function must consider the new price, and trade the tokens for their real value, out of the AMM:

```
function balancePool(uint256 oldPrice, uint256 newPrice) internal {    infinite gas
    if (oldPrice == newPrice){
        return;
    }
    if (oldPrice < newPrice) {//selling TOK0
        uint256 amountToSell = reserve0 - sqrt(K/newPrice);
        uint256 amountEarned = amountToSell * newPrice;
        reserve0 -= amountToSell;
        reserve1 += amountEarned;
    } else {//buying TOK0
        uint256 amountToBuy = sqrt(K/newPrice)-reserve0;
        uint256 amountSpent = amountToBuy * newPrice;
        reserve1 -= amountSpent;
        reserve0 += amountToBuy;
    }
}
```

I first check if the action needed is selling or buying, then calculate the amount using the new price, and forward it to calculate the amount of TOK1 to be added/subtracted.

It seems like such a small detail, but it certainly has an effect.

Testing

I ran a test to check the differences between the two contracts, with the following scenario: the pool is initialized with 1 TOK0 and 1000 TOK1, and that is their exchange rate - 1 TOK0 = 1000 TOK1.
At first both contracts behave identically, with a total pool value of 2000 TOK1.
However, when the price changes, there are differences:

Regular CFMM

Now 1 TOK0 = 3000 TOK1:

There is 0.577 TOK0 and 1731 TOK1s in the pool (there are no decimals in Solidity apparently, and scaling is the common fix-up. In this case the contract scales by 1000), and the total value in TOK1 is 3462.
How much of loss is there? If the LP would have kept his 1 TOK0 and 1000 TOK1, he now would have had the worth of 4000 TOK1, so his loss for not holding (loss-versus-holding) is 4000-3462= 532 TOK1.

Then, the price drops to 2000, and so:

The loss has now reduced and it is 3000-2828=172 TOK1s.

Then the price returns to 1000, and so:

Now the loss is 0.

LVR-AMM

After the price went up 1000=>3000:

There is no loss - just as they state in the paper: "that rebalancing arbitrage profits are equal to LVR".

Then, when the price goes down to 2000:


```
                "from": "0xEC5ef95575F1c4A56C7c576F8a18C14B73A7f188",
                "topic":
"0x1b331f62b48b3a4d57aaf6ebaba3b4629a358fc9c75e3b4696d65699829f84d2",
                "event": "LogReservesAndPoolValue",
                "args": {
                        "0": "707",
                        "1": "2009000",
                        "2": "3423",
                        "reserve0": "707",
                        "reserve1": "2009000",
                        "poolValue": "3423"
                }
```


Not only there is no loss, there is gain for the liquidity providers of 423 TOK1s.

And finally, when the price goes back down:


```
                "from": "0xEC5ef95575F1c4A56C7c576F8a18C14B73A7f188",
                "topic":
"0x1b331f62b48b3a4d57aaf6ebaba3b4629a358fc9c75e3b4696d65699829f84d2",
                "event": "LogReservesAndPoolValue",
                "args": {
                        "0": "1000",
                        "1": "1716000",
                        "2": "2716",
                        "reserve0": "1000",
                        "reserve1": "1716000",
                        "poolValue": "2716"
                }
```


The profit keeps on growing and is now on 716 TOK1s.

Implementing the LVR-CPMM Method

The LVR method diverts from the CPMM - and it is acknowledged also in the paper. Several ways are suggested to deal with it, one of which is "moving" the excess tokens to a "third party wallet" that belongs to the LP. That way, the pool maintains the constant product, and the LP still gains from his pool more than he would from a regular CPAMM.

Therefore, I changed the LVR-AMM contract, and implemented an AP wallet as described in the paper. Other than implementing the wallet, the only changes took place in the balancePool method (same function as before):

```
function balancePool(uint256 oldPrice, uint256 newPrice) internal {    infinite gas
    if (oldPrice == newPrice){
        return;
    }
    if (oldPrice < newPrice) {//selling TOK0
        uint256 amountToSell = reserve0 - sqrt(K/newPrice);
        uint256 amountEarned = amountToSell * newPrice;
        reserve0 -= amountToSell;
        reserve1 += amountEarned;
    } else {//buying TOK0
        uint256 amountToBuy = sqrt(K/newPrice)-reserve0;
        uint256 amountSpent = amountToBuy * newPrice;
        reserve1 -= amountSpent;
        reserve0 += amountToBuy;
    }
}
```

The results for the same scenario ran on the other 2 AMMs are as follows:

Testing

LVR-CFMM

Price 1000=>3000:


```
"from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
"topic": "0x1b331f62b48b3a4d57aaf6ebaba3b4629a358fc9c75e3b4696d65699829f84d2",
"event": "LogReservesAndPoolValue",
"args": {
        "0": "577",
        "1": "2269000",
        "2": "4000",
        "reserve0": "577",
        "reserve1": "2269000",
        "poolValue": "4000"
}


"from": "0xb5465ED8EcD4F79dD4BE10A7C8e7a50664e5eeEB",
"topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
"event": "Transfer",
"args": {
        "0": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
        "1": "0xB9B678B56D243e5d4a9Dff43458226c06557EA2b",
        "2": "538",
        "from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
        "to": "0xB9B678B56D243e5d4a9Dff43458226c06557EA2b",
        "value": "538"
}


"from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
"topic": "0x1b331f62b48b3a4d57aaf6ebaba3b4629a358fc9c75e3b4696d65699829f84d2",
"event": "LogReservesAndPoolValue",
"args": {
        "0": "577",
        "1": "1731000",
        "2": "3462",
        "reserve0": "577",
        "reserve1": "1731000",
        "poolValue": "3462"
}
```

We notice the pool is balanced both by the tokens' worth - which is exactly 1731 TOK1s each (0.577*3000=1731), and their reserves product holds the Constant Product - 0.577*1731 ~ 1000.
What about the loss? 577 TOK1s are transferred to the wallet, and so 4000 - (3462 + 577) = -39, meaning there is a profit of 39 TOK1s.

When 3000=>2000:


```
"from": "0xb5465ED8EcD4F79dD4BE10A7C8e7a50664e5eeEB",
"topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
"event": "Transfer",
"args": {
        "0": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
        "1": "0xB9B678B56D243e5d4a9Dff43458226c06557EA2b",
        "2": "57",
        "from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
        "to": "0xB9B678B56D243e5d4a9Dff43458226c06557EA2b",
        "value": "57"
}


"from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
"topic": "0x1b331f62b48b3a4d57aaf6ebaba3b4629a358fc9c75e3b4696d65699829f84d2",
"event": "LogReservesAndPoolValue",
"args": {
        "0": "707",
        "1": "1414000",
        "2": "2828",
        "reserve0": "707",
        "reserve1": "1414000",
        "poolValue": "2828"
}
```


Additional 57 TOK1s are transferred to the wallet, so now there are 634 TOK1s in the wallet, and with the pool value of 2828, the loss in all is 3000-(2828+634)=-462, the profit of the LP is now significant.

Lastly, when the prices return to their initial state:


```
"from": "0xb5465ED8EcD4F79dD4BE10A7C8e7a50664e5eeEB",
"topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
"event": "Transfer",
"args": {
        "0": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
        "1": "0xB9B678B56D243e5d4a9Dff43458226c06557EA2b",
        "2": "121",
        "from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
        "to": "0xB9B678B56D243e5d4a9Dff43458226c06557EA2b",
        "value": "121"
}


"from": "0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c",
"topic": "0x1b331f62b48b3a4d57aaf6ebaba3b4629a358fc9c75e3b4696d65699829f84d2",
"event": "LogReservesAndPoolValue",
"args": {
        "0": "1000",
        "1": "1000000",
        "2": "2000",
        "reserve0": "1000",
        "reserve1": "1000000",
        "poolValue": "2000"
```

There are 121 excess TOK1s to transfer to the wallet, making the wallet's asset 755 TOK1, and the LVR is 2000-(2000+755) =-755, a significant profit, considering that the loss-versus-holding is 0.

Results

Regular CPAMM – 0 Tokens of profit for the LP.
LVR-AMM – 716 TOK1s profit for the LP.
LVR-CPAMM – 755 TOK1s profit for the LP.

The results clearly show that the Constant Product LVR mechanism benefits the Liquidity Providers the most, and thus an enhanced use of it will probably implore more people to become LPs, and push the economy forward.