Market's protocol fees

We would like to capture a percentage of the swap fees as protocol fees. The best way to do it is by minting LPs to the protocol, proportional to the "increase in the market's liquidity due to swap fees".

First, let's see how Uniswap does it:

- In Uniswap,
  - `x * y = k`, where x and y are amounts of the two tokens
  - Let's define `sqrt(k)` as the "liquidity" of the pool, or basically the geometric mean of the two token balances.
  - When first bootstrapped at `t0`, the initial LPs minted is exactly the liquidity in the pool `lp_total = sqrt(x_0 * y_0) = sqrt(k) = pool's liquidity`
  - Lets say there is no liquidity addition or removal from `t0` to `t1`
    - At `t0`: `x_0 * y_0 = k_0`
    - There are multiple swaps happening from `t0` to `t1` . Due to the 0.3% fees for each swap, `k` actually increases (due to the 0.3% of the tokens being retained in the pool for each swap)
    - At `t1`: `x_1 * y_1 = k_1`
      - Now, the liquidity in the pool increases by an amount: `sqrt(k_1) - sqrt(k_0)`
        - For the sake of understanding, lets use the example that `k_1 = 1.21 * k_0`
        - Basically, there is a 10% increase in liquidity
        - As such, the `lp_total` LPs will enjoy this 10% increase in liquidity (claim to a higher geometric mean of the two token balances by 10%)
  - At `t1`, before another event (add/remove liquidity) that changes `k` that is not due to the 0.30% fees from swapping:

    - Uniswap will mint a certain amount of `s` LPs to the protocol, right before doing the k-changing event, such that due to this effect, the protocol will receive 1/6 of the increase in liquidity `sqrt(k_1) - sqrt(k_0)`. Now to calculate `s`, we can refer to the Uniswap whitepaper here: https://uniswap.org/whitepaper.pdf at page 5.
        - This will lead to the protocol enjoys 1/6 of the value brought about by the swap fees from `t0` to `t1`

- Basically, the increase in liquidity due to swap fees are converted to LP tokens for the protocols. As such, all these `mintLpFeesForProtocol` functions are done right before any other actions that change `k` due to a non-swapping event.
    - Details:
        - Uniswap saves the `kLast` right after the last non-swapping event
        - Uniswap makes sure that when `mintLpFeesForProtocol` function(`_mintFee`) is called, the liquidity of the pool is only changed due to swap fees
            - Basically, Uniswap calls `mintLpFeesForProtocol` right before any non-swapping event that changes k.

For us:

- Our AMM formula is as such:
    - `x^(alpha) * y^(1-alpha) = k`
    - Initially, alpha =0.5 and we mint exactly `k` LPs to the bootstrapper
- As such, the "liquidity" definition in our case is exactly `k`, or the weighted geometric mean of the token balances.
- When there are no curve shifts, (and alpha is fixed), `k` will also increase due to 0.35% of the token swapped being retained in the market.
    - As such, we can adopt a similar approach, to mint a percentage of the increases in `k` to the protocol
    - We will also need to save the last k right after any non-swapping event, and call a function `mintProtocolFees` to mint the protocol LPs right before k is changed due to a non-swapping event

- `mintProtocolFees`:

```
currentK = x^(alpha) * y^(1-alpha)
toMint = (currentK - lastK) / ( (1/feesPortion - 1) * currentK + lastK ) * totalLP
mint(protocol, toMint)
```

- Basically, we need to make sure that:

> 💡 **INVARIANT 1:** from the moment `lastK` is saved, until the moment `mintProtocolFees` is called, the only way k was changed is due to swapping.

- In our protocol, other than adding liquidity and removing liquidity, another non-swapping event that would change `k` is the curve shift, where:
  - At time `t`, when a curve shift happens, the balances of XYT is `x_t` and balance of base token is `y_t`
    - Before curve shifting:
      - `x_t^(alpha) * y_t^(1-alpha) = k_before`
    - After curve shifting:
      - `x_t^(alpha - deltaAlpha) * y_t^(1-alpha+deltaAlpha) = k_after`
  - `k_after` is clearly different from `k_before`, so:
    - before doing a curve shift, we will need to call `mintProtocolFees()`
    - after doing a curve shift, we need to save the `lastK`
- Similarly for adding liquidity and removing liquidity, we will also need to:
  - Call `mintProtocolFees()` before the action
  - Save `lastK` after the action
- INVARIANT 1 makes sure that we don't wrongly count non-swapping changes in k to the protocol fees.
- There is also this INVARIANT 2 to make sure we capture all changes in k due to swapping:

💡 **INVARIANT 2**: For every swaps, the increases in k due to the swap must be captured due to an eventual execution of mintProtocolFees()