深入 Ref-Finance

合约解读

Marco

exchange

概览

https://app.ref.finance/

https://github.com/ref-finance/ref-contracts/tree/main/ref-exchange

https://stats.ref.finance/stats/dex

AMM 基础

线性不变量 Linear Invariant

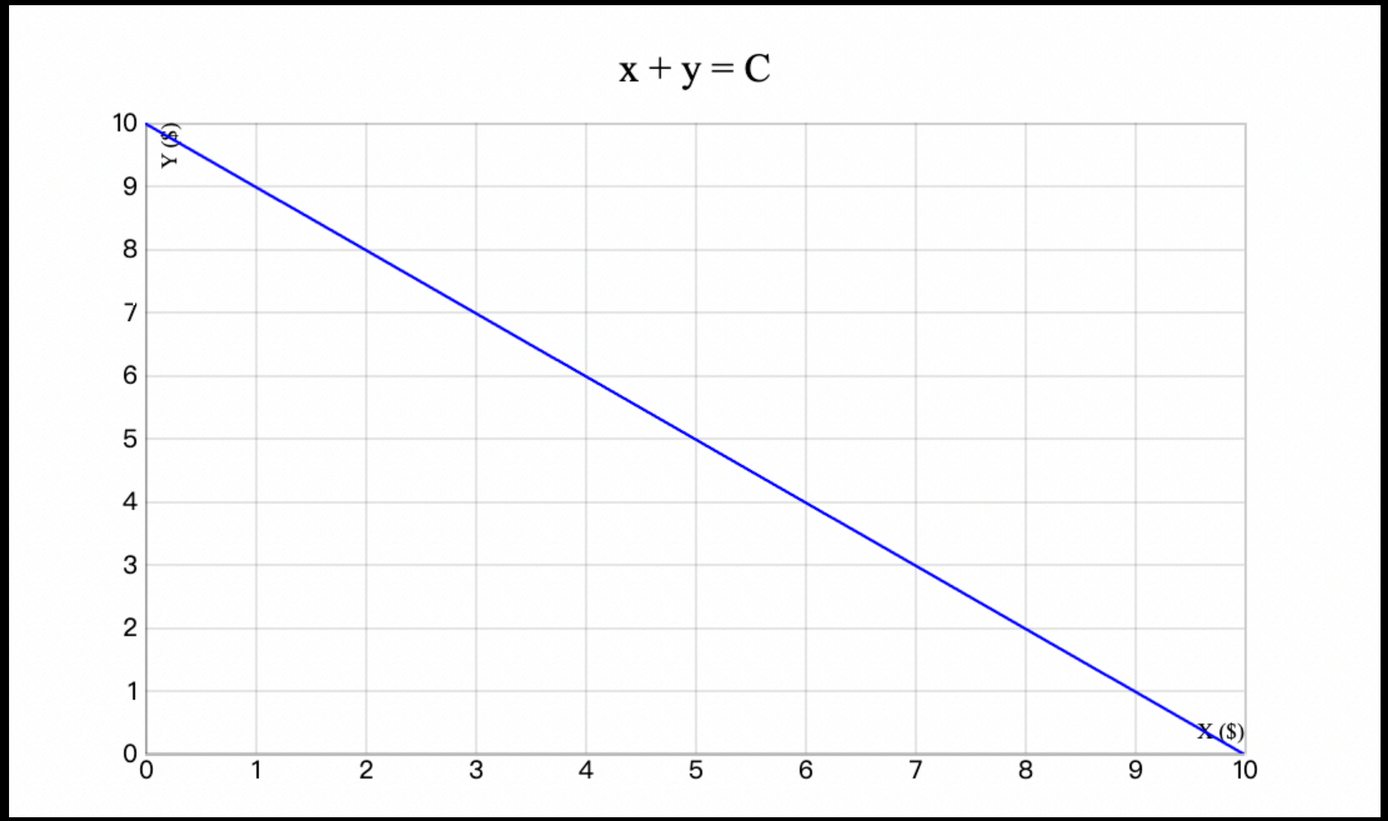
$$x + y = C$$

100 A + 100 B = 200

张三 用25枚A可以换得25枚B:

125 A + 75 B = 200

滑点 slippage = 0



AMM 基础

乘积不变量 Product Invariant

$$x * y = K$$

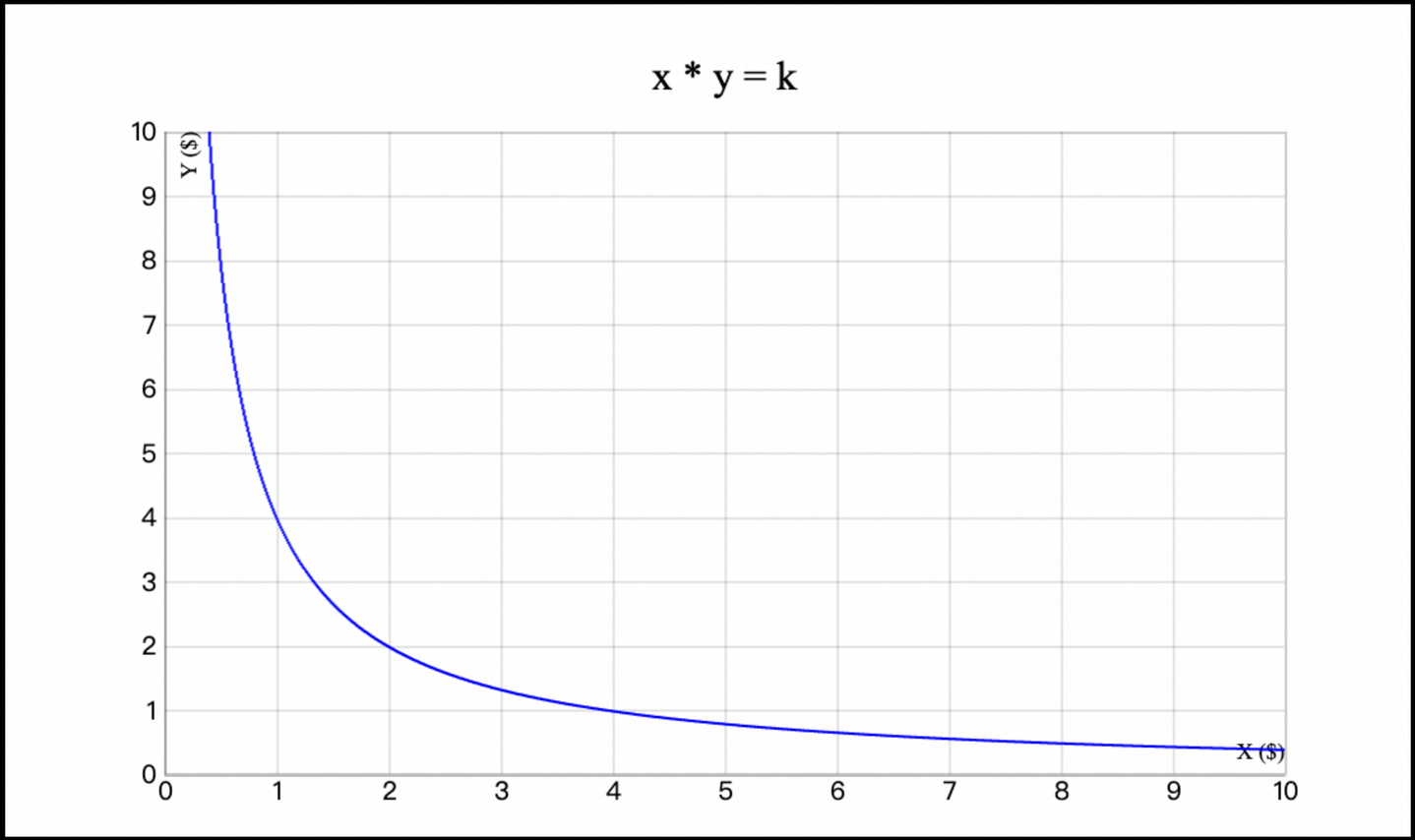
$$y = K/x$$

100 A * 100 B = 10K

张三用25枚A可以换得20枚B (price=0.8):

125 A * 80 B = 10K

滑点 slippage = 0.2 = (1.0 - 0.8)/1.0



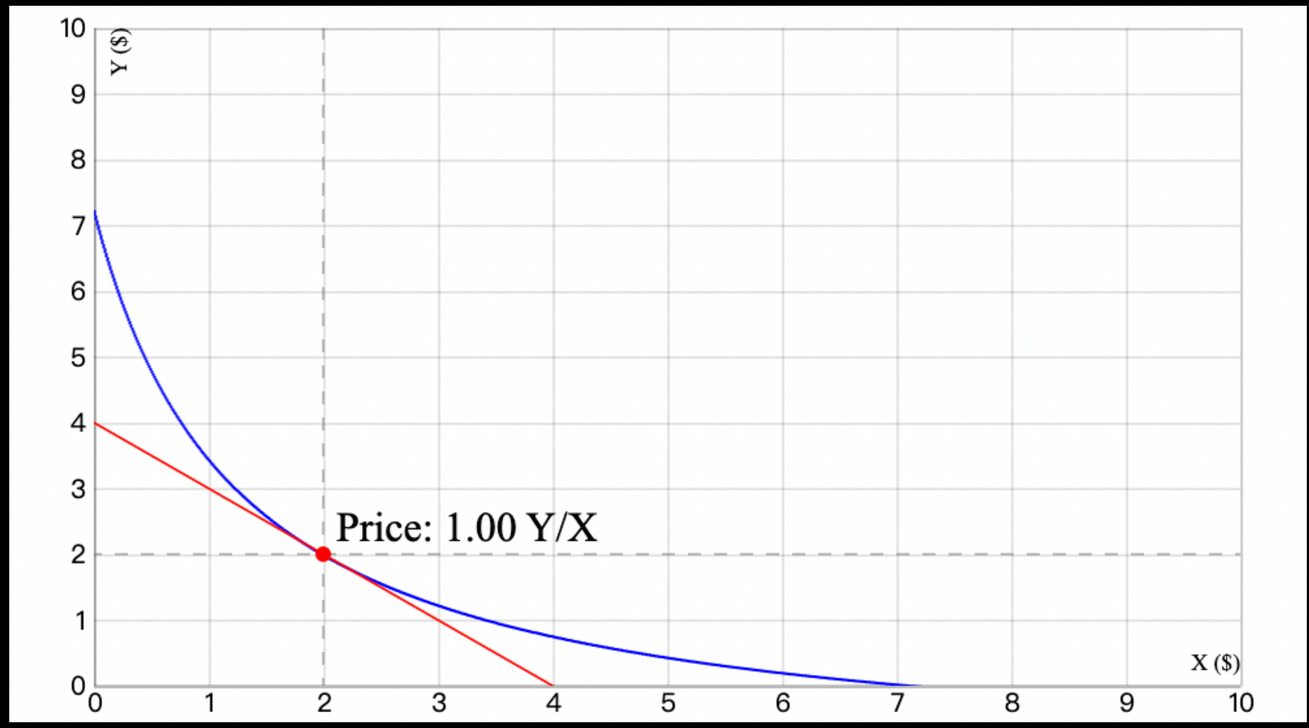
AMM 基础

稳定池模型

$$An^n \sum x_i + D = ADn^n + \frac{D^{n+1}}{n^n \prod x_i}$$

$$4A(x+y) + D = 4AD + \frac{D^3}{4xy}$$

https://curve.fi/files/stableswap-paper.pdf



AMM 基础

稳定池模型

Invairant D

$$An^n \sum x_i + D = ADn^n + \frac{D^{n+1}}{n^n \prod x_i}$$

Newton迭代法演算：

$$\begin{cases} f(D) = \frac{D^{n+1}}{n^n \prod x_i} + (An^n - 1)D - An^n \sum x_i = 0 \\ f'(D) = \frac{n+1}{n^n \prod x_i} D^n + An^n - 1 = 0 \\ D_{k+1} = D_k - \frac{f(D_k)}{f'(D_k)} \end{cases}$$

记：

$$D_{prod} = \frac{D^{n+1}}{n^n \prod x_i}$$

则：

$$D_{k+1} = \frac{D_k(An^n \sum x_i + nD_{k,prod})}{D_k(An^n - 1) + (n+1)D_{k,prod}}$$

记：

$$\begin{cases} \sum x_i = y + \sum x'_i \\ \prod x_i = y \prod x'_i \end{cases}$$

则：

$$An^n(y + \sum x'_i) + D = ADn^n + \frac{D^{n+1}}{n^n y \prod x'_i}$$

Newton迭代法演算：

$$\begin{cases} An^n y^2 + [An^n \sum x'_i - (An^n - 1)D]y = \frac{D^{n+1}}{n^n \prod x'_i} \\ y^2 + (\sum x'_i + \frac{D}{An^n} - D)y = \frac{D^{n+1}}{An^{2n} \prod x'_i} \\ 2y + \sum x'_i + \frac{D}{An^n} - D = 0 \\ y_{k+1} = y_k - \frac{f(y_k)}{f'(y_k)} \end{cases}$$

记：

$$\begin{cases} b = \sum x'_i + \frac{D}{An^n} \\ c = \frac{D^{n+1}}{An^{2n} \prod x'_i} \end{cases}$$

则：

$$y_{k+1} = \frac{y_k^2 + c}{2y_k + b - D}$$

AMM 基础

交易费

swap fee

用 dx 换 dy，部分dx作为fee预先扣除，留在 pool 中，D 因此缓慢增加

$$(x + (1 - f)dx)(y - dy) = xy \quad => \quad dy = \frac{(1 - f)dx * y}{(x + (1 - f)dx)}$$

protocol fee

从swap fee中收取一定比例，以流动性的方式归属合约本身

LP Token 流动性代币

Nep-141 扩展

流动性的添加(mint)和移除(burn)

初始流动性的定价作用

https://stats.ref.finance/leaderboards/liquidityproviders

流动性代币的代币特性

ref-exchange/src/multi_fungible_token.rs

存储分析

存储费管理和存储分布

内部用户存储 与 instant swap

ref-exchange/src/storage_impl.rs

LP的存储

添加流动性: 预付+返还机制+auto-register　internal_check_storage

token层面: register机制

Farming

概览

https://github.com/ref-finance/ref-contracts/tree/main/ref-farming

https://app.ref.finance/farms

https://stats.ref.finance/leaderboards/farmers

Farming
一币多挖

Seed

LP Token, NEP-141

```rust
#[derive(Debug, Serialize, Deserialize
#[serde(crate = "near_sdk::serde")]
4 implementations
pub struct SeedInfo {
    pub seed_id: SeedId,
    pub seed_type: String,
    pub farms: Vec<FarmId>,
    pub next_index: u32,
    pub amount: U128,
    pub min_deposit: U128,
}
```

Farm

Seed->Farm => 1:N

Reward

```rust
#[derive(BorshSerialize, BorshDeserialize, Clone)]
2 implementations
pub struct SimpleFarmTerms {
    pub seed_id: SeedId,
    pub reward_token: AccountId,
    pub start_at: TimestampSec,
    pub reward_per_session: Balance,
    pub session_interval: TimestampSec,
}
```

```rust
#[derive(BorshSerialize, BorshDeserialize)]
1 implementation
pub struct SimpleFarm {

    pub farm_id: FarmId,

    pub terms: SimpleFarmTerms,

    pub status: SimpleFarmStatus,

    pub last_distribution: SimpleFarmRewardDistribution,

    /// total reward send into this farm by far,
    /// every time reward deposited in, add to this field
    pub amount_of_reward: Balance,
    /// reward token has been claimed by farmer by far
    pub amount_of_claimed: Balance,
    /// when there is no seed token staked, reward goes to beneficiary
    pub amount_of_beneficiary: Balance,

}
```

Farming

奖励分配

基于 per-seed-reward 的分配机制

https://github.com/ref-finance/ref-contracts/blob/main/ref-farming/how_does_it_work.md#reward-distribution

```rust
/// Account deposits information and storage cost.
#[derive(BorshSerialize, BorshDeserialize)]
#[cfg_attr(feature = "test", derive(Clone))]
1 implementation
pub struct Farmer {
    /// Native NEAR amount sent to this contract.
    /// Used for storage.
    pub amount: Balance,
    /// Amounts of various reward tokens the farmer claimed.
    pub rewards: HashMap<AccountId, Balance>,
    /// Amounts of various seed tokens the farmer staked.
    pub seeds: HashMap<SeedId, Balance>,
    /// record user_last_rps of farms
    pub user_rps: LookupMap<FarmId, RPS>,
    pub rps_count: u32,
}
```

```rust
/// Reward Distribution Record
#[derive(BorshSerialize, BorshDeserialize, Clone, Default)]
2 implementations
pub struct SimpleFarmRewardDistribution {
    /// unreleased reward
    pub undistributed: Balance,
    /// the total rewards distributed but not yet claimed by farmers.
    pub unclaimed: Balance,
    /// Reward_Per_Seed
    /// rps(cur) = rps(prev) + distributing_reward / total_seed_staked
    pub rps: RPS,
    /// Reward_Round
    /// rr = (cur_block_timestamp in sec - start_at) / session_interval
    pub rr: u32,
}
```

谢谢观看

May 2022