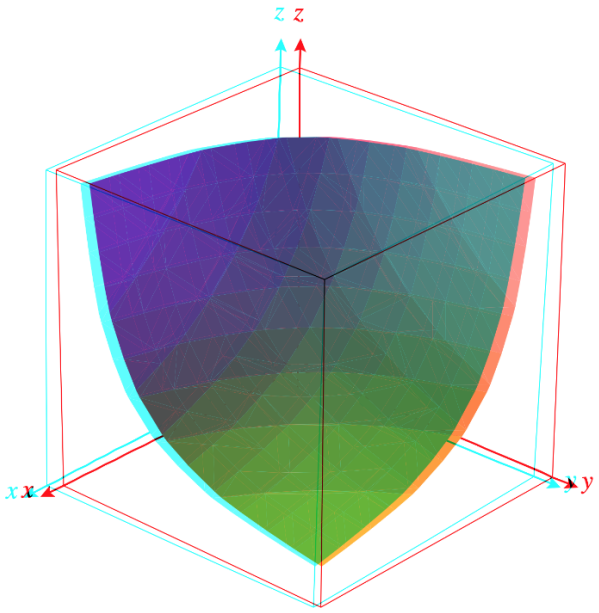
Whitepaper



A non-custodial portfolio manager, liquidity provider, and price sensor.

by:
Fernando Martinelli
Nikolai Mushegian

v2019-09-19

contact@balancer.finance (mailto:contact@balancer.finance)

Introduction

A Balancer Pool is an automated market maker with certain key properties that cause it to function as a *self-balancing weighted portfolio* and *price sensor*.

Balancer turns the concept of an index fund on its head: instead of paying fees to portfolio managers to rebalance your portfolio, you collect fees from traders, who rebalance your portfolio by following arbitrage opportunities.

Balancer is based on a particular N-dimensional surface which defines a cost function for the exchange of any pair of tokens held in a Balancer Pool. This approach was first described by V. Buterin[0] (https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/),

generalized by Alan Lu[1] (https://blog.gnosis.pm/building-a-decentralized-exchange-in-ethereum-eea4e7452d6e), and proven viable for market making by the popular Uniswap[2] (https://uniswap.io) dapp.

We independently arrived at the same surface definition by starting with the requirement that any trade must maintain a constant proportion of value in each asset of the portfolio. We applied an invariant-based modeling approach described by Zargham et al[3] (https://arxiv.org/pdf/1807.00955.pdf) to construct this solution. We will prove that these constant-value market makers have this property.

Table of Contents

Present Work

Index funds are a common financial instrument. The first index fund became effective in 1972. Ever since, investors rely heavily on different portfolio strategies to hedge risk and achieve diversification. Index funds guarantee investors a constant and controlled exposure to a portfolio. If one of its assets out- or under-performs, it is respectively sold or bought to keep its value share of the total portfolio constant.

Both in the conventional financial system as well as in the blockchain context, index funds and other types of investment portfolios charge investors fees for managing and holding their funds. These fees are necessary to pay for the costs of actively rebalancing the index funds, be it by manual traders or automatic bots.

There are many centralized solutions for portfolio management and for investing in index funds. These all share some form of custodial risk.

We are aware of one decentralized (read: non-custodial) solution that shares all the fundamental characteristics Balancer was designed to have: Uniswap (https://uniswap.io). This approach was first described by V. Buterin (https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/) and generalized by Alan Lu (https://blog.gnosis.pm/building-a-decentralized-exchange-in-ethereum-eea4e7452d6e).

We independently arrived at the same surface definition by *starting* with the requirement that any trade must maintain a constant proportion of value in each asset of the portfolio. We applied an invariant-based modeling approach described by Zargham et al (https://arxiv.org/pdf/1807.00955.pdf) to construct this solution. We will prove that these constant-value market makers have this property.

Theory

Throughout this paper, we use the term "token" to refer to a generic asset because our first implementation is a contract system that manipulates ERC20 tokens on the Ethereum network. However, there is nothing fundamental about the Ethereum execution context that enables this market-making algorithm, which could be offered by a traditional financial institution as a centralized (custodial) product.

Value Function

The bedrock of Balancer's exchange functions is a surface defined by constraining a value function V — a function of the pool's weights and balances — to a constant. We will prove that this surface implies a spot price at each point such that, no matter what exchanges are carried out, the share of value of each token in the pool remains constant.

The value function V is defined as:

$$V = \prod_t B_t^{W_t} \tag{1}$$

Where

- t ranges over the tokens in the pool;
- B_t is the balance of the token in the pool;
- W_t is the normalized weight of the token, such that the sum of all normalized weights is 1.

By making V constant we can define an invariant-value surface as illustrated in Fig.0.

Spot Price

Each pair of tokens in a pool has a spot price defined entirely by the weights and balances of just that pair of tokens. The spot price between any two tokens, $SpotPrice_i^o$, or in short SP_i^o, is the the ratio of the token balances normalized by their weights:

$$SP_i^o = \frac{\frac{B_i}{W_i}}{\frac{B_o}{W_o}} \tag{2}$$

Where:

- B_i is the balance of token *i*, the token being sold by the trader which is going *into* the pool.
- B_o is the balance of token *o*, the token being bought by the trader which is going *out* of the pool.
- W_i is the weight of token *i*
- W_o is the weight of token *o*

From this definition it is easy to see that if weights are held constant, the spot prices offered by Balancer Pools only change with changing token balances. If the pool owner does not add or remove tokens to/from the pool, token balances can only change through trades. The constant surface causes the price of tokens being bought by the trader (token *o*) to increase and price of tokens being sold by the trader (token i) to decrease. One can prove that whenever external market prices are different from those offered by a Balancer Pool, an arbitrageur will make the most profit by trading with that pool until its prices equal those on the external market. When this happens there is no more arbitrage opportunity. These arbitrage opportunities guarantee that, in a rational market, prices offered by any Balancer Pool move in lockstep with the rest of the market.

Effective Price

It is important to bear in mind that SP_i^o is the *spot* price, which is the theoretical price for infinitesimal trades, which would incur no slippage. In reality, the effective price for any trade depends on the amount being traded, which always causes a price change. If we define A_o as the amount of token *o* being bought by the trader and A_i as the amount of token i being sold by the trader, then we can define the Effective Price as:

$$EP_i^o = \frac{A_i}{A_o} \tag{3}$$

And as mentioned above, EP tends to SP when traded amounts tend to 0:

$$SP_i^o = \lim_{A_o, A_i \to 0} EP_i^o \tag{4}$$

Spot Price Proof

Let's now prove that this choice of V entails Eq.2.

First of all, we know that what the trader buys, A_o, is subtracted from the contract's balance. Therefore $A_o = -\Delta B_o$. Likewise, what the trader sells, A_i, is added to the contract's balance. Therefore $A_i = \Delta B_i$.

Substituting in Eq.2 and Eq.3 we get:

$$SP_i^o = \lim_{A_o, A_i \to 0} EP_i^o = \lim_{\Delta B_o, \Delta B_i \to 0} \frac{\Delta B_i}{-\Delta B_o} \tag{5}$$

This limit is, by definition, minus the partial derivative of B_i in function of B_o:

$$SP_i^o = -\frac{\partial B_i}{\partial B_o} \tag{6}$$

From the value function definition in Eq.1 we can isolate B_i:

$$B_i^{W_i} = \frac{V}{\left(\prod_{k \neq i,o} B_k^{W_k}\right) . B_o^{W_o}}$$

$$B_i = \left(\frac{V}{\left(\prod_{k \neq i,o} B_k^{W_k}\right) . B_o^{W_o}}\right)^{\frac{1}{W_i}} \tag{7}$$

Now we use Eq.7 to expand the partial derivative in Eq.6:

$$SP_i^o = -\frac{\partial B_i}{\partial B_o} = -\frac{\partial}{\partial B_o}\left(\left(\frac{V}{\left(\prod_{k \neq i,o} (B_k)^{W_k}\right) \cdot (B_o)^{W_o}}\right)^{\frac{1}{W_i}}\right) =$$

$$-\left(\frac{V}{\prod_{k \neq i,o} (B_k)^{W_k}}\right)^{\frac{1}{W_i}} \cdot \frac{\partial}{\partial B_o}\left(B_o^{-\frac{W_o}{W_i}}\right) =$$

$$-\left(\frac{V}{\prod_{k \neq i,o} (B_k)^{W_k}}\right)^{\frac{1}{W_i}} \cdot -\frac{W_o}{W_i} \cdot B_o^{-\frac{W_o}{W_i}-1} =$$

$$\left(\frac{V}{\prod_{k} (B_k)^{W_k}}\right)^{\frac{1}{W_i}} \cdot B_o^{\frac{W_o}{W_i}} \cdot B_i \cdot \frac{W_o}{W_i} \cdot B_o^{-\frac{W_o}{W_i}-1} =$$

$$\left(\frac{V}{V}\right)^{\frac{1}{W_i}} \cdot B_o^{\frac{W_o}{W_i}} \cdot B_o^{-\frac{W_o}{W_i}} \cdot \frac{B_i}{W_i} \cdot \frac{W_o}{B_o} = \frac{\frac{B_i}{W_i}}{\frac{B_o}{W_o}}$$

which concludes our proof.

Constant Value Distribution Proof

We will now prove that:

1. Balancer Pools maintain a constant share of value across all tokens in the pool and;
2. These shares of value are equal to the weights associated to each token.

Let's calculate V^t, the total pool value in terms of an arbitrary token t from the pool. Since we already know that the pool has B_t tokens t, let's calculate how many tokens t all the other remaining tokens are worth. It does not make sense to use their Effective Price relative to token t since we are not going to do any actual trade. Instead, to calculate the theoretical value we use their Spot Price relative to token t.

From Eq.2 we can calculate V_n^t, i.e how many tokens t the balance of each token n is worth:

$$V_n^t = \frac{B_n}{SP_n^t} = B_n \cdot \frac{\frac{B_t}{W_t}}{\frac{B_n}{W_n}} = B_t \cdot \frac{W_n}{W_t} \tag{8}$$

We know that the total pool value in terms of tokens t is the sum of the values of each token in terms of tokens t:

$$V^t = \sum_k V_k^t = B_t + \sum_{k \neq t} V_k^t = B_t + \frac{B_t}{W_t} \cdot \sum_{k \neq t} W_n = \frac{B_t}{W_t} \cdot (W_t + \sum_{k \neq t} W_n) = \frac{B_t}{W_t} \tag{9}$$

Now to calculate S_n, the share of value each token n represents in the pool, all we have to do is divide the value of each token n, V_n^t, by the total pool value, V^t:

$$S_n = \frac{V_n^t}{V^t} = W_n \tag{10}$$

which proves both that the share each token represents of the total pool value is constant and also that it is equal to the weight of that token.

Trading Formulas

Calculating the trade outcomes for any given Balancer Pool is easy if we consider that the Value Function must remain invariant, i.e. V must have the same value before and after any trade.
In reality, V will increase as a result of trading fees applied after a trade state transition.
For more details on fees, see Implementation: Swap and Exit Fees

Out-Given-In

When a user sends tokens i to get tokens o, all other token balances remain the same. Therefore, if we define A_i and A_o as the amount of tokens i and o exchanged, we can calculate the amount A_o a users gets when sending A_i. Knowing the value function after the trade should be the same as before the trade, we can write:

$$\prod_{k \neq i,o} (B_k)^{W_k} \cdot (B_o - A_o)^{W_o} \cdot (B_i + A_i)^{W_i} = \prod_k (B_k)^{W_k} \tag{11}$$

$$\prod_{k \neq i,o} (B_k)^{W_k} \cdot (B_o - A_o)^{W_o} \cdot (B_i + A_i)^{W_i} = \prod_{k \neq i,o} (B_k)^{W_k} \cdot B_o^{W_o} \cdot B_i^{W_i} \tag{12}$$

$$(B_o - A_o)^{W_o} \cdot (B_i + A_i)^{W_i} = B_o^{W_o} \cdot B_i^{W_i} \tag{13}$$

$$B_o - A_o = \frac{B_i^{\frac{W_i}{W_o}} \cdot B_o}{(B_i + A_i)^{\frac{W_i}{W_o}}} \tag{14}$$

$$A_o = B_o \cdot \left(1 - \left(\frac{B_i}{B_i + A_i}\right)^{\frac{W_i}{W_o}}\right) \tag{15}$$

In-Given-Out

It is also very useful for traders to know how much they need to send of the input token A_i to get a desired amount of output token A_o. We can calculate the amount A_i as a function of A_o similarly as follows:

$$\prod_{k \neq i,o} (B_k)^{W_k} \cdot (B_o - A_o)^{W_o} \cdot (B_i + A_i)^{W_i} = \prod_{k} (B_k)^{W_k} \tag{16}$$

$$\prod_{k \neq i,o} (B_k)^{W_k} \cdot (B_o - A_o)^{W_o} \cdot (B_i + A_i)^{W_i} = \prod_{k \neq i,o} (B_k)^{W_k} \cdot B_o^{W_o} \cdot B_i^{W_i} \tag{17}$$

$$(B_o - A_o)^{W_o} \cdot (B_i + A_i)^{W_i} = B_o^{W_o} \cdot B_i^{W_i} \tag{18}$$

$$B_i + A_i = \frac{B_o^{\frac{W_o}{W_i}} \cdot B_i}{(B_o - A_o)^{\frac{W_o}{W_i}}} \tag{19}$$

$$A_i = B_i \cdot \left(\left(\frac{B_o}{B_o - A_o}\right)^{\frac{W_o}{W_i}} - 1\right) \tag{20}$$

Notice that A_o as defined by Eq.11 tends to $SP_i^o \cdot A_i$ when $A_i << B_i$, as expected. This can be proved by using L'Hopital's rule, but this proof is out of the scope of this paper.

In-Given-Price

For practical purposes, traders intending to use our contract for arbitrage will like to know what amount of tokens i – A_i – they will have to send to the contract to change the current spot price SP_i^o to another desired one $SP_i'^o$. The desired spot price will usually be the external market price and, so long as the

contract spot price differs from that of the external market, any arbitrageur can profit by trading with the contract and bringing the contract price closer to that of the external market.

The highest profit possible by an arbitrageur is when they bring the contract spot price exactly to that of the external market. As already mentioned, this is the main reason why our design is successful in keeping track of the market prices. This makes it a reliable on-chain price sensor when implemented on a blockchain.

It can be proven that the amount of tokens i – A_i – a user needs to trade against tokens o so that the pool's spot price changes from SP_i^o to $SP_i'^o$ is:

$$A_i = B_i \cdot \left(\left(\frac{SP_i'^o}{SP_i^o} \right)^{\left(\frac{W_o}{W_o + W_i} \right)} - 1 \right) \tag{21}$$

Liquidity Providing Formulas

Pool Tokens

Pools can aggregate the liquidity provided by several different users. In order for them to be able to freely deposit and withdraw assets from the pool, Balancer Protocol has the concept of pool tokens. Pool tokens represent ownership of the assets contained in the pool. The outstanding supply of pool tokens is directly proportional to the Value Function of the pool. If a deposit of assets increases the pool Value Function by 10%, then the outstanding supply of pool tokens also increases by 10%. This happens because the depositor is issued 10% of new pool tokens in return for the deposit.

There are two ways in which one can deposit assets to the pool in return for pool tokens or redeem pool tokens in return for pool assets:

- Weighted-asset deposit/withdrawal
- Single-asset deposit/withdrawal

All-Asset Deposit/Withdrawal

An "all-asset" deposit has to follow the distribution of existing assets in the pool. If the deposit contains 10% of each of the assets already in the pool, then the Value Function will increase by 10% and the depositor will be minted 10% of the current outstanding pool token supply. So to receive P_{issued} pool tokens given an existing total supply of P_{supply}, one needs to deposit D_k tokens k for each of the tokens in the pool:

$$D_k = \left(\frac{P_{supply} + P_{issued}}{P_{supply}} - 1 \right) \cdot B_k \tag{22}$$

Where B_k is the token balance of token k before the deposit.

Similarly, a weighted-asset withdrawal is the reverse operation where a pool token holder redeems their pool tokens in return for a proportional share of each of the assets held by the pool. By redeeming $P_{redeemed}$ pool tokens given an existing total supply of P_{supply}, one withdraws from the pool an amount A_k of token k for each of the tokens in the pool:

$$A_k = \left(1 - \frac{P_{supply} - P_{redeemed}}{P_{supply}}\right) \cdot B_k \tag{23}$$

Where B_k is the token balance of token k before the withdrawal.

Single-Asset Deposit/Withdrawal

When a user wants to provide liquidity to a pool because they find its distribution of assets interesting, they may likely not have all of the assets in the right proportions required for a weighted-asset deposit.

Balancer allows anyone to get pool tokens from a shared pool by depositing a single asset to it, provided that the pool contains that asset.
Depositing a single asset A to a shared pool is equivalent to depositing all pool assets proportionally and then selling more of asset A to get back all the other tokens deposited. This way a depositor would end up spending only asset A, since the amounts of the other tokens deposited would be returned through the trades.

The amount of pool tokens one gets for depositing a single asset to a shared pool can be derived from the Value Function described above.

Single-Asset Deposit

The increase in the pool token supply proportional to the increase in the Value Function. If we define P_{issued} as the amount of pool tokens issued in return for the deposit, then:

$$\frac{V'}{V} = \frac{P'_{supply}}{P_{supply}} = \frac{P_{supply} + P_{issued}}{P_{supply}}$$

$$P_{issued} = P_{supply} \cdot \left(\frac{V'}{V} - 1\right) \tag{24}$$

Where V' is the Value Function after the deposit and V is the Value Function before the deposit.
Considering also B'_k the balance of asset k after the deposit and B_k its balance before the deposit, we have:

$$\frac{V'}{V} = \frac{\prod_k (B'_k)^{W_k}}{\prod_k (B_k)^{W_k}}$$

Let's say the single-asset deposit was done in asset t, then the balances of all other tokens do not change after the deposit. We can then write:

$$\frac{V'}{V} = \frac{\prod_k (B'_k)^{W_k}}{\prod_k (B_k)^{W_k}} = \frac{(B'_t)^{W_t}}{(B_t)^{W_t}} = \left(\frac{B'_t}{B_t}\right)^{W_t}$$

If we define A_t as the amount deposited in asset t, then the new pool balance of asset t is $$B'*t = B_t + A_t$$. We can then substitute and get the final formula for the amount of new pool tokens issued $P*{issued} $in return for a single-asset depositI_t:

$$P_{issued} = P_{supply} \cdot \left(\left(1 + \frac{A_t}{B_t} \right)^{W_t} - 1 \right) \tag{25}$$

Single-Asset Withdrawal

When a pool token holder wants to redeem their pool tokens $P_{redeemed}$ in return for a single asset t, the amount withdrawn in asset t, A_t, is:

$$A_t = B_t \cdot \left(1 - \left(1 - \frac{P_{redeemed}}{P_{supply}} \right)^{\frac{1}{W_t}} \right) \tag{26}$$

Where B_t is the pool balance of asset t before the withdrawal.

Indeed, using the formulas of deposit and withdrawal defined above, not considering any fees, if one deposits A_t asset t for P_{issued} pool tokens and then redeems that same amount of pool tokens for asset t, they will get the same initial A_t back.

Trading Fees for Single-Asset Deposit Withdrawal

Depositing or withdrawing to/from a shared pool in a single asset t is equivalent to trading $(1 - W_t)$ of the amount deposited for all the other assets in the pool. W_t of the amount deposited is held by the pool already in the form of asset t, so charging a trading fee on that share would be unfair.

Indeed, if we disregard any possible pool exit fees, depositing only asset i and instantly withdrawing asset o will incur in the same trading fees as doing the trade from i to o using the trade function the pool offers.

Implementation

There are a few initial notes regarding the first release of Balancer. We will release a much more detailed explanation of the system at the same time that the source code is released.

Free Software on Ethereum

Balancer is implemented as a GPL3-licensed Ethereum smart contract system.

Releases

The 🍂**Bronze Release**🍂 is the first of 3 planned releases of the Balancer Protocol. Bronze emphasizes code clarity for audit and verification, and does not go to great lengths to optimize for gas.

The ❄️**Silver Release**❄️ will bring many gas optimizations and architecture changes that will reduce transaction overhead and enable more flexibility for controlled pools.

The ☀️**Golden Release**☀️ will introduce several new features to tie the whole system together.

Numerical Algorithms

The formulas in the Theory section are sufficient to describe the functional specification, but they are not straightforward to implement for the EVM, in part due to a lack of mature fixed-point math libraries.

Our implementation uses a combination of a few algebraic transformations, approximation functions, and numerical hacks to compute these formulas with bounded maximum error and reasonable gas cost.

The rest of this section will be released at the same time as the Bronze release source code.

Controlled vs Finalized Pools

The 🍂Bronze Release🍂 allows two basic tiers of trust with respect to pools:

1. *Controlled* pools are configurable by a "controller" address. Only this address can add or remove liquidity to the pool (call `join` or `exit`). This type of pool allows the change of pool assets types and their weights. Note that since the controller is an address, this could in principle implement arbitrary logic, like managing public deposits in a manner similar to a finalized pool. The key difference is that official tooling will not recognize it as a "trustless" pool. Controlled pools with increased trust requirements will be possible with the ❄️Silver Release❄️.
2. *Finalized* pools have fixed pool asset types, weights, and fees. Crucially, this enables `join` and `exit` to be publicly accessible in a safe, trustless manner while keeping a minimal implementation.

Swaps and Exit Fees

The 🍂Bronze Release🍂 charges fees in two situations: When traders exchange tokens (via `swap` and its variants), and when liquidity providers remove their liquidity from the pool (via `exit` and its variants).

Both of these fees are configurable by the controller, but they are also fixed when the pool becomes finalized.

100% of the swap fee goes to the liquidity providers — the amount of the underlying token that can be redeemed by each pool token increases.

Most of the exit fee is returned to the liquidity providers who remain in the pool.
This is similar in spirit to a swap fee charged for exchanging pool tokens with underlying tokens.

The rest of the exit fee is transferred to an account controlled by Balancer Labs, Inc, for the development of ❄️Future Releases☀️.

References

[0] Vitalik Buterin: Let's run on-chain decentralized exchanges the way we run prediction markets (https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/)

[1] Alan Wu: Building a Decentralized Exchange in Ethereum (https://blog.gnosis.pm/building-a-decentralized-exchange-in-ethereum-eea4e7452d6e)

[2] https://uniswap.io/ (https://uniswap.io)

[3] Zargham, M., Zhang, Z., Preciado, V.: A State-Space Modeling Framework for Engineering Blockchain-Enabled Economic Systems. New England Complex Systems Institute (2018) (https://arxiv.org/pdf/1807.00955.pdf)

 (mailto:contact@balancer.finance)  (https://twitter.com/BalancerLabs) 

(https://medium.com/balancer-protocol) (https://discord.gg/ARJWaeF)  (https://github.com/balancer-labs/)  (https://defipulse.com)