Dexy - A Stablecoin With Algorithmic Bank Interventions

Alexander Chepurnoy, Amitabh Saxena

May 24, 2022

Abstract

Inspired by the success of Bitcoin, many clients for the Bitcoin protocol as well as for alternative blockchain protocols have been implemented. However, implementations may contain errors, and the cost of an error in the case of a cryptocurrency can be extremely high.

We propose to tackle this problem with a suite of abstract property tests that check whether a blockchain system satisfies laws that most blockchain and blockchain-like systems should satisfy. To test a new blockchain system, its developers need to instantiate generators of random objects to be used by the tests. The test suite then checks the satisfaction of the laws over many random cases. We provide examples of laws in the paper.

1 Introduction

Algorithmic stablecoins is a natural extension of cryptocurrencies, trying to solve problems with volatility of their prices by pegging stablecoin price to an asset which price believed to be more or less stable with time (e.g. 1 gram of gold).

Stable pricing could be useful for:

1. securing fundraising;
2. doing business with predictable results. For example, a hosting provider knows
3. shorting: .
4. lending and other decentralized finance applications.

Algorithmic stablecoins are different from centralized stablecoins, such as USDT and USDC, which are convertible into underlying (pegged asset) via a trusted party. In case of an algorithmic stablecoin, its pegging is done via rebasement of total supply, or via imitating the trusted party, which holds.

2 General Design

Unlike popular algorithmic stablecoins based on two tokens (instruments), Dexy is based on one token but two protocols. In the first place, there is market where trading of Dexy vs the base currency (ERG) happens. In the second place, if market price is way too different from target price (reported by an oracle), there is an algorithmic central bank which makes interventions. The central bank can also mint new Dexy tokens, selling them for ERG. The bank is using reserves in ERG it is buying for interventions then.

As a simple solution for the market, we are using constant-factor Automated Market Maker (CF-AMM) liquidity pool, similar to ErgoDEX and UniSwap. The pool has ERG on one side and Dexy on another. For CF-AMM pool, multiplication of ERG and Dexy amounts before and after a trade should be preserved, so $e * u = e' * u'$, where e and u are amounts of ERG and Dexy in the pool before the trade, and e' and u' are amounts of ERG and Dexy in the pool after the trade. It is also possible to put liquidity into the pool, and remove liquidity from it.

The bank has two basic operations. It can mint new Dexy tokens and

Now we are going to consider how to put restrictions and design rules for the system to have stable price of Dexy.

3 Stability

What provides stability for the stablecoin when we have the design sketched in the previous section?

4 Worst Scenario and Bank Reserves

The bank is doing interventions when the situation is far from normal on the markets, and enough time passed for markets to stabilize themselves with no interventions. In our case, the bank is doing interventions based on stablecoin price in the liquidity pool in comparison with oracle provided price (we can assume that price on other markets is similar to liquidity poolś due to arbitrage). The bankś intervention then is about injecting its ERG reserves into the pool.

We start with introducing notation:

- T - period before intervention starts. After one intervention the bank can start another one after T.
- p - price reported by the oracle at the moment (for example, 20 USD per ERG)
- s - price which the bank should stand in case of price crash. For example, we can assume that $s = \frac{p}{4}$ (so if p is 20 USD per ERG, then s is 5 USD per ERG, means the bank needs to have enough reserves to save the markets when the price is crashing from 20 to 5 USD per ERG)

5 Implementation

6 Simulations

7 Extensions

Acknowledgments

Authors would like to thank.