Primitive RMM-01

0xEstelle
estelle@primitive.xyz

experience
experience@learnedtrustlessness.io

Alexander Angel
alex@primitive.xyz

Matt Czernik
matt@primitive.xyz

October 2021

Abstract

Constant function market makers (CFMMs) have evolved from a small group of decentralized exchanges (DEXs) to a broad and largely undiscovered class of automated market makers (AMMs). CFMMs are decentralized exchanges fully backed by a community of liquidity providers seeking to earn a yield on their deposited assets. The portfolio of a liquidity provider follows a payoff structure specific to the CFMM they are providing liquidity too. It was recently shown that the space of concave, non-negative, non-decreasing, 1-homogeneous payoff functions and the space of convex CFMMs are equivalent, along with a method to convert between a given payoff of the above type with an associated CFMM. These CFMMs, which replicate specific, desired payoff functions, are called replicating market makers (RMMs). In this paper, we present an implementation of an RMM that approximates a Black–Scholes covered call, which we call RMM-01.

Contents

1 Introduction **2**
- 1.1 Constant Function Market Makers . . . 2
- 1.2 Replicating Market Making . . . 4
- 1.3 Options Fundamentals . . . 4

2 RMM-01 **6**
- 2.1 Payoff Structure . . . 7
- 2.2 Reported Price . . . 7
- 2.3 Swap Calculations . . . 8
- 2.4 Price of Swaps . . . 8
  - 2.4.1 Price Impact . . . 9
- 2.5 Analysis . . . 10
  - 2.5.1 Constraints and Bounds . . . 13

3 Solidity Implementation **13**
3.1 Trading Function Approximations 13
3.2 Integers, Fixed Point Numbers, and Decimals 13
3.3 Two-Token Pairs 14
3.4 Multiple Pool Configurations Per Pair 14
3.5 Liquidity Initialization 14
3.6 Adding & Removing Liquidity 15
3.7 Swapping 15
3.8 Optimistic Transfers 16
3.9 Agnostic Payments 16
3.10 Internal Balances 16
3.11 Swap Fee & Fee Adjustments 17
3.12 The Invariant 17

4 Implications **17**
4.1 Exotic Options 17

5 Conclusion **18**

1 Introduction

The design space of CFMMs[4] and their potential impact on on-chain derivative design is limited by the lack of modularity for each new CFMM. No solid framework for converting from a payoff structure to a CFMM existed until recently [5]. The paper proved the equivalence of the space of concave, non-negative, non-decreasing, 1-homogeneous payoff functions and the set of convex CFMMs. It provided a method to convert a particular payoff function to a corresponding CFMM. It also provides the first few examples of replicating market makers (RMMs).

In this paper, we describe the implementation of an RMM that replicates a Black–Scholes covered call payoff structure, called RMM-01. The particular RMM was originally defined in the *Replicating Market Makers* paper. However, it wasn't able to achieve self-financing replication due to the CFMM's inability to capture theta decay[5]. We show that theta is recoverable within a probabilistic range with a fixed fee, and thus, a covered call payoff is obtainable. Before diving into this implementation, some preliminary concepts deserve review.

1.1 Constant Function Market Makers

Constant Function Market Makers (CFMMs) have grown from a simple solution to decentralized trading to a highly dynamic and broad class of decentralized exchanges (DEXs), a significant research focus within decentralized finance, and a convenient building block for more complex decentralized financial derivatives[3]. They now host tens of billions of dollars

in available liquidity across multiple different blockchains, and hundreds of billions of volume traded, making them some of the most liquid and widely used decentralized applications. The available liquidity in a CFMM is supplied by the liquidity providers, who deposit assets in the contracts to passively earn yield and potentially hedge the assets' relative price motion. Each CFMM gives its liquidity providers a corresponding payoff structure, exposing them to different risks and yields on their assets based on the underlying price motion and trading flow [4]. For example, Balancer[10] maintains a personalized weighted balance on the pooled assets; this includes up to 8 assets. To further understand CFMMs and the payoff structures, it would be beneficial to define them again briefly formally.

CFMMs of two assets are defined via functions of the asset reserves, x and y respectively, $\varphi : (R_x, R_y) \to \mathbb{R}$ that defines a trading rule between the two assets [3]. Namely, in the case of CFMMs, for a given Δ amount of tokens swapped in, the trader will receive Δ' of the other token such that,

$$\varphi(x, y) = \varphi(x + \Delta, y - \Delta') = k$$

for some constant $k \in \mathbb{R}$ known as the *invariant*. In a sense, the value of the function φ should not change with trading volume given that the pool has no additional structures, such as a swap fee. The swap fee comes into effect by only calculating the amount out with a fraction of the amount, then recalculating k with the total amount added to the pool. This will result in a smaller amount out to the trader with the total amount added to the pool.

The reported price of a CFMM is contained in the invariant, as the marginal price of the pair for an infinitesimal trade size. For instance, assume a trader is looking to swap in some amount of token X for token Y. For a two token CFMM, the trading function can often be re-written as $y(x)$ where x is the reserve quantity of token X and y the reserve quantity of token Y. The marginal price can be expressed as,

$$S(x) = -\frac{dy}{dx}$$

the negative rate of change of the Y reserve with respect to the X reserve. The negative applies since the Y reserve decreases whenever the X increases, yielding a negative rate of change, and the price must be positive. This returns us the reported price of the CFMM and the price received on a tiny trade. However, this is not the price obtained on a regular trade. For more on the math associated with CFMMs, refer to [3]. We now turn to the liquidity provider and the payoff they receive.

Each CFMM yields its liquidity providers a corresponding payoff structure. In the above case, the payoff structure of the liquidity provider's position can be described as,

$$V(p_x, p_y) = p_x x + p_y y(x) = p_x(x + S(x)y(x)) = p_x(x - \frac{dy}{dx}y(x))$$

Where p_x, p_y are the prices of the two assets on some reference market, concerning some different currency, and x, y are the pool reserves, respectively. This effectively describes the pooled value but represents that of the liquidity provider under the assumption of only one provider in the pool. The nature of this relationship makes it simple to recover the payoff structure given a CFMM, but not to find a CFMM given some payoff structure.

1.2 Replicating Market Making

There has been increasing attention in CFMM research focused on replicating particular payoff structures as CFMM shares and what actually can be replicated as CFMM shares. To address this, [5] introduced two key results. First, it shows that convex CFMMs are equivalent to the space of concave, non-negative, non-decreasing, 1-homogeneous functions. This tells us what payoff functions can be replicated as shares directly. Second, it directly converts a payoff function of the stated type to an associated CFMM. As discouraging as it may seem for its inability to replicate convex payoffs, it should be noted that convexity can be achieved through leverage. For more on recovering convexity see [8]. However, convex payoffs are not the focus or point of the paper. The key aspect is there is now a method of converting from payoff function to an associated CFMM, which has broad implications in on-chain derivative design.

In this paper, we focus on the implementation of RMMs that roughly replicate a Black–Scholes covered call: exhibiting a strike price, time of expiry, and implied volatility over the duration, but without the same payoff as a covered call[5]. Due to its relation to Black–Scholes options, it would be beneficial to review options as a whole. Associated terminology such as The Greeks, as framing the product from that perspective allows for more profound insight in the behavior of its payoff given changes in the options parameters and external market price of the risky asset.

1.3 Options Fundamentals

An option is a derivative instrument composed of a risky asset and a risk-less reserve, giving the right to the option purchaser to buy or sell the risky asset within a specific period, subject to certain conditional requirements[7]. Options allow for asymmetric directional exposure, increased leverage, the ability to hedge risk, and a composable framework for achieving the desired payoff structure based on the underlying asset's price motion.

Options are priced through a purchase premium using a pricing model based on the intrinsic value the contract offers and the extrinsic value dictated by the probability that the conditional requirements of the contract are met by the expiration date. Due to the random and often unpredictable nature of volatile assets' price motion, different pricing models arise from unique probability models. These conditional requirements will usually involve a strike price (K) that the underlying asset's price must be above or below, depending on the contract type, for the purchaser to exercise the contract. In a call option, the spot must be above the strike for exercise ($S > K$), and in a put option, the spot must be below the strike ($S < K$). In the case that the expiration of the contract doesn't meet the conditional requirements, the option expires worthless, leaving the premium paid as a sunk cost.

The two major groups of options are European options, where the purchaser only has the right to exercise at the expiration date, and American options, where the purchaser can exercise any time prior to[6]. We will focus on European options using the Black–Scholes pricing model. This pricing model depends on a strike price K, a constant implied volatility σ (this is usually defined over a given period, such as 80% implied volatility annually), the

risk-free return rate or bond rate r, time to expiry τ (defined simply as the difference between the expiration time T and the current time t, $\tau = T - t$), and of course the spot price of the underlying S; exhibiting a value of V[7].

The following list of terms, which includes the Greeks, is used to describe the behavior of an option contract in an efficient manner:

i. *In-the-money (ITM)*: refers to when a contract meets its conditional requirements

ii. *Out-of-the-money (OTM)*: refers to when a contract is not in-the-money

iii. *Delta* $\Delta = \frac{\partial V}{\partial S}$: Rate of change of the option's value for an infinitesimal change in the spot

iv. *Theta* $\theta = \frac{\partial V}{\partial \tau}$: Rate of change of the option's value for an infinitesimal change in the time to expiry

v. *Gamma* $\gamma = \frac{\partial^2 V}{\partial S^2}$: Rate of change of the option's value for an infinitesimal change in the option's Delta Δ. It can be thought of as the rate of acceleration of the option's spot price.

vi. *Vega* $\nu = \frac{\partial V}{\partial \sigma}$: Rate of change of the option's value with respect to an infinitesimal change in the implied volatility.

vii. *Rho* $\rho = \frac{\partial V}{\partial r}$: Rate of change of the option's value for an infinitesimal change in the bond rate.

Since we are only focusing on Black–Scholes priced covered calls in this paper, the risk-free return rate and implied volatility are assumed to be constant, meaning we will not focus on *Rho* or *Vega*. The Greeks, however, are vital tools for understanding an option's behavior and managing risk accordingly. One particularly unique insight brought forth by the Greeks is known as *Theta decay*; the decline in an option's value due to the approaching time to expiry.

An option's value can be broken up into two components: the option's intrinsic value $V_{Intrinsic}$ and extrinsic/time value V_{Time}. $V_{Intrinsic}$ is the value the option would have if it were exercised today, where the time value can be computed as the difference between the option's current value and its intrinsic value, $V_{time} = V - V_{Intrinsic}$. The time value represents the contributions to the option's value due to the probability of expiring profitably. This is where theta decay arises.

Theta decay declines the time value portion of the option's value as expiry approaches. Intuitively, as the time to expiry decreases, the probability of profitability from purchasing the contract decreases since there is less time for favorable price motion. Meaning the magnitude of the time value should reflect a decrease as well. The result of this is an accelerating decay of the time value until $V_{time} = 0$ at $\tau = 0$ (at expiry), so that $V = V_{Intrinsic}$ as expected.

2 RMM-01

RMM-01 is originally defined in [5] as:

$$y - K\Phi(\Phi^{-1}(1-x) - \sigma\sqrt{\tau}) = k$$

where $k = 0$ in the no-fees case, and K, σ, and τ are the option's inputs. Under the assumption of arbitrage-free spot pricing, the value of the reserves of RMM-01 replicates closely the value of a Black-Scholes covered call with the given *static* options inputs, as shown below in Figure 1.

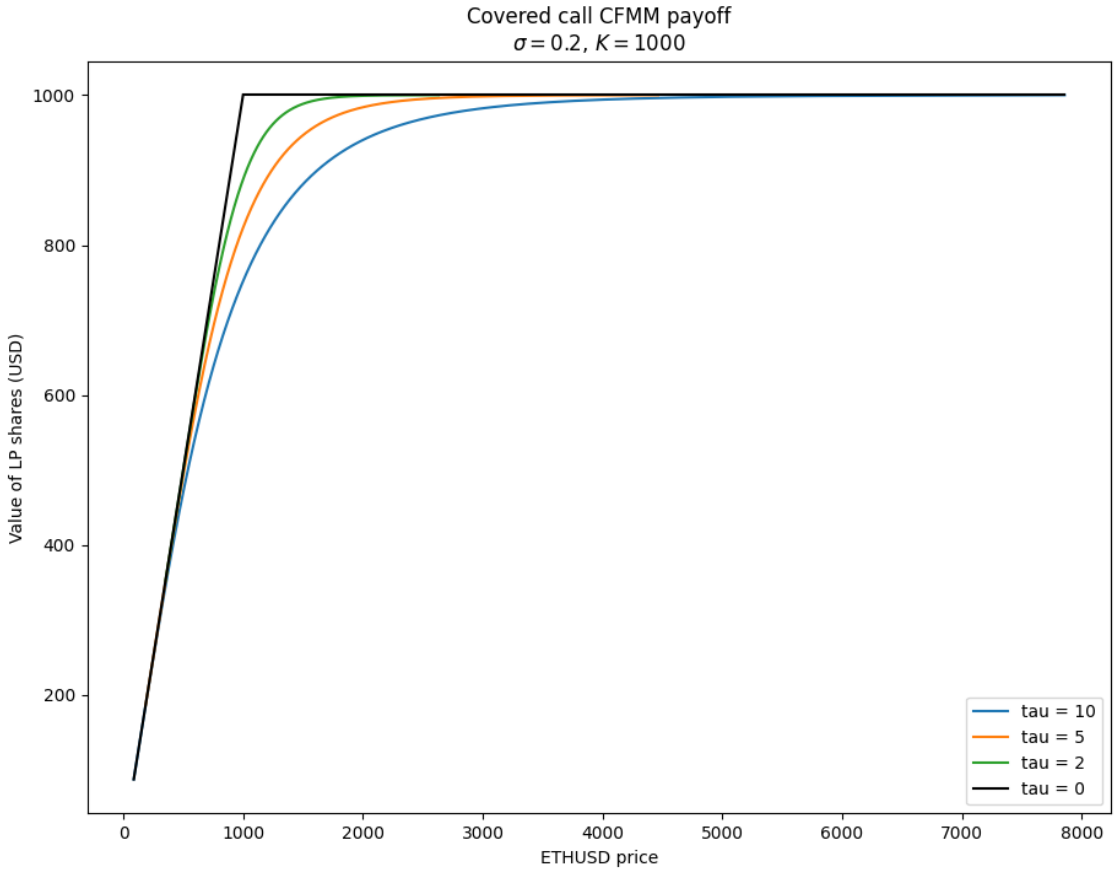

Figure 1: *The value of the RMM-01 LP share, assuming no-arbitrage, as a function of the risky asset price and τ.*

Due to the nature of the time dependence of a covered call, causing continual upward pressure on the value until expiry, combined with the inherently static nature of a CFMM under no trading flow, there will always be a portfolio replication problem without external funding when updating τ. In short, the CFMM cannot capture the value accrued due to theta decay on its own.

As such, we are applying a fee regime, $\gamma = 1 - fee$. Note that the fee can be optimally adjusted for each pool to best recover the deviation in value between the theoretical payoff and practical payoff, given a set of environmental expectations such as trade frequency, as shown in section *2.1 Application*.

We will denote any ERC-20 volatile assets as risky assets and any stable ERC-20 assets as risk-less assets. The CFMM can support any token pair. However, due to Black-Scholes pricing, it is common practice to have one stable token and one risky token. We can now calculate all the required trading quantities using this invariant function and general fee regime γ.

2.1 Payoff Structure

It is beneficial to review the theoretical payoff of the liquidity share. Since RMM-01 intends to replicate a Black-Scholes covered call, the theoretical payoff follows the same rule:

$$V(S) = S(1 - \Phi(d_1)) + K\Phi(d_2)$$

where S is the reported price of the pair, $d_1 = \frac{\log(S/K) + (\sigma^2/2)\tau}{\sigma\sqrt{\tau}}$ and $d_2 = d_1 - \sigma\sqrt{\tau}$ [5]. The results of this equation can be seen in *Figure 1*, with the following expiration outcomes:

i. If the reported price is greater than the strike, $S > K$, the LP receives a payoff of $V = K$ risk-less, fully denominated as risk-less; Leaving a position of K risk-less.

ii. If the reported price is less than the strike, $S < K$, the LP receives a payoff of $V = S$ risk-less, fully denominated as risky; leaving a position of 1 unit of the risky asset.

The payoff of the RMM-01 has a portfolio tracking error due to the inability to capture theta decay properly[5]. The fee has the intention of re-capturing the accrual. The result of this is an error term in the payoff structure, $V_{eff} = V - \varepsilon_0$ for some $\varepsilon \in (-\infty, V)$. An in-depth discussion of this error can be found in *2.1 Applications*.

2.2 Reported Price

Rearranging the invariant equation to isolate $y(x)$,

$$y = K\Phi(\Phi^{-1}(1 - x) - \sigma\sqrt{\tau}) + k$$

Taking the negative derivative will return the marginal price $S(x)$,

$$S(x) = K\phi(\Phi^{-1}(1 - x) - \sigma\sqrt{\tau}) \times (\Phi^{-1})'(1 - x)$$

where ϕ is the standard normal probability distribution function. Using the inverse function theorem, this is equivalent to,

$$S(x) = K\frac{\phi(\Phi^{-1}(1 - x) - \sigma\sqrt{\tau})}{\phi(\Phi^{-1}(1 - x))}$$

Using the definition of the standard normal distribution ϕ and the relation:

$$\phi(\Phi^{-1}(1-x)-\sigma\sqrt{\tau}) = \phi(\Phi^{-1}(1-x))e^{\Phi^{-1}(1-x)\sigma\sqrt{\tau}}e^{-\frac{1}{2}\sigma^2\tau}$$

We can obtain the reported marginal price as,

$$S(x) = Ke^{\Phi^{-1}(1-x)\sigma\sqrt{\tau}}e^{-\frac{1}{2}\sigma^2\tau}$$

This function supports all prices in its range since $\lim_{x\to 0} S(x) = +\infty$ and $\lim_{x\to 1} S(x) = 0$ for any $\tau > 0$ and $\sigma > 0$. Due to the standard normal CDF dependence, this function has anomaly end behavior. The end behavior shows that some prices cannot be modeled in practice. In this case, the RMM-01 will yield an upper/lower bound for prices above some threshold point[9].

2.3 Swap Calculations

Assume the pool initially has x, y liquidity of the constituent risky asset and risk-less asset respectively. Consider the case the trader wants to swap in Δ of the risky asset. The new reserves become $x' = x + \Delta$ and

$$y' = K\Phi(\Phi^{-1}(1-(x+\gamma\Delta)) - \sigma\sqrt{\tau}) + k$$

with the amount of the risk-less asset received by the trader being $\Delta' = y - y'$. The updated invariant value becomes,

$$k' = y' - K\Phi(\Phi^{-1}(1-x') - \sigma\sqrt{\tau})$$

In the case the trader wants to swap in Δ of the risk-less asset. The new reserves become $y' = y + \Delta$ and

$$x' = 1 - \Phi\left(\Phi^{-1}\left(\frac{y+\gamma\Delta-k}{K}\right) + \sigma\sqrt{\tau}\right)$$

where the amount of the risky asset received by the trader is $\Delta' = x - x'$, the invariant is subsequently updated as above.

2.4 Price of Swaps

We can obtain a trade price using the swap procedures described above. Consider the case that the trader swaps in Δ risk-less to buy the risky asset, under a fee regime γ. The invariant reads,

$$y + \gamma\Delta - K\Phi(\Phi^{-1}(1-(x-\Delta')) - \sigma\sqrt{\tau}) = k$$

Isolating for the quantity of the risky asset received Δ',

$$\Delta' = x - 1 + \Phi\left(\Phi^{-1}\left(\frac{y+\gamma\Delta-k}{K}\right) + \sigma\sqrt{\tau}\right)$$

This has put the the quantity of the risky asset received as a function of the quantity of the risk-less asset swapped in. Taking the derivative of this function will yield the marginal price for a particular trade size,

$$\frac{d\Delta'}{d\Delta}(\Delta) = \frac{\gamma}{K}\phi\left(\Phi^{-1}\left(\frac{y+\gamma\Delta-k}{K}\right)+\sigma\sqrt{\tau}\right)\times(\Phi^{-1})'\left(\frac{y+\gamma\Delta-k}{K}\right)$$

We can repeat this procedure for the second case, where the trader is looking to sell Δ risky asset for the risk-less asset. The quantity received by the trader is,

$$\Delta' = y - K\Phi(\Phi^{-1}(1-x-\gamma\Delta)-\sigma\sqrt{\tau})-k$$

Deriving this with respect to the quantity of the risky asset swapped in Δ, we yield the marginal price of the exchange as

$$\frac{d\Delta'}{d\Delta}(\Delta) = \gamma K\phi(\Phi^{-1}(1-x-\gamma\Delta)-\sigma\sqrt{\tau})\times(\Phi^{-1})'(1-x-\gamma\Delta)$$

2.4.1 Price Impact

We can calculate the price impact of a trade by running through the swap procedure generally, then recalculating the price. Take a swap of Δ risky in, under the same γ fee regime. The updated reserves become,

$$y' = K\Phi(\Phi^{-1}(1-(x+\gamma\Delta))-\sigma\sqrt{\tau})+k$$

with $y' = y - \Delta'$. We can derive y' with respect to x' to obtain the new marginal price.

$$S(x+\Delta) = Ke^{\Phi^{-1}(1-(x+\gamma\Delta))\sigma\sqrt{\tau}}e^{-\frac{1}{2}\sigma^2\tau}$$

If S_0 is the price before the trade, $S(\Delta) < S_0$ for all $\Delta > 0$. Implying that selling always has a negative price impact, as expected. To calculate the price impact, compute:

$$\frac{S(\Delta)-S_0}{S_0}$$

On the other hand, purchasing some of the risky asset with Δ risk-less, we can calculate the reported price on x' and y'. Using the same price function, we can substitute in $x' = x - \Delta'$,

$$S(\Delta') = Ke^{\Phi^{-1}(1-(x-\Delta'))\sigma\sqrt{\tau}}e^{-\frac{1}{2}\sigma^2\tau}$$

Since for every $\Delta > 0$ of risk-less in, the trader receives some $\Delta' > 0$ units of the risky asset. So $x - \Delta' < x$ and thus $S(\Delta') > S_0$ for all $\Delta' > 0$, which occurs whenever $\Delta > 0$. Implying that purchasing the risky asset will always lead to an increased price. To calculate the exact price impact, compute:

$$\frac{S(\Delta')-S_0}{S_0}$$

2.5 Analysis

A few aspects of the model particularly merit further discussion. The portfolio tracking problem and resulting fee structure, a few subtle constraints and bounds of the RMM, the solidity implementation, and the product's implications are of interest.

To further expand on the portfolio tracking problem, assume that a covered call option is currently at τ_i to expiry, with an underlying spot price of S, and the RMM is getting no trading flow. If at $\tau_f < \tau_i$, the underlying has the same spot price S, then the value of that covered call will increase ($V_i < V_f$). Since the reserves are left unchanged, but the value of a covered call has increased, a correction function is necessary. The replication error can be drastically reduced or closed by adding a fixed swap fee to the pool.

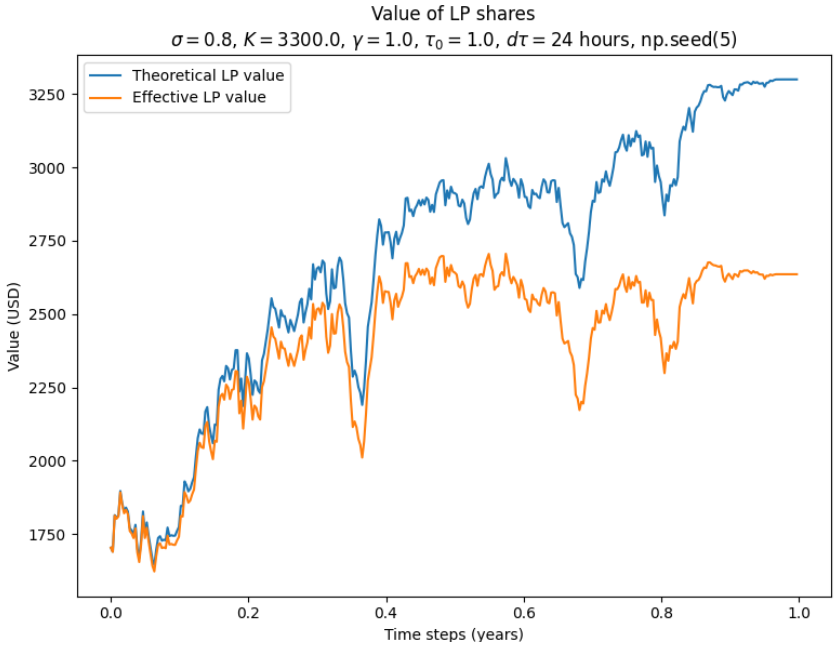

Figure 2: *No fee case* ($\gamma = 1.0$) *with sample strike, volatility, and price path. In the no fees case, the RMM fails to self-finance the replication of the covered call due to the inability to capture theta-decay, as seen above. See [9] for more information on the simulations presented.*

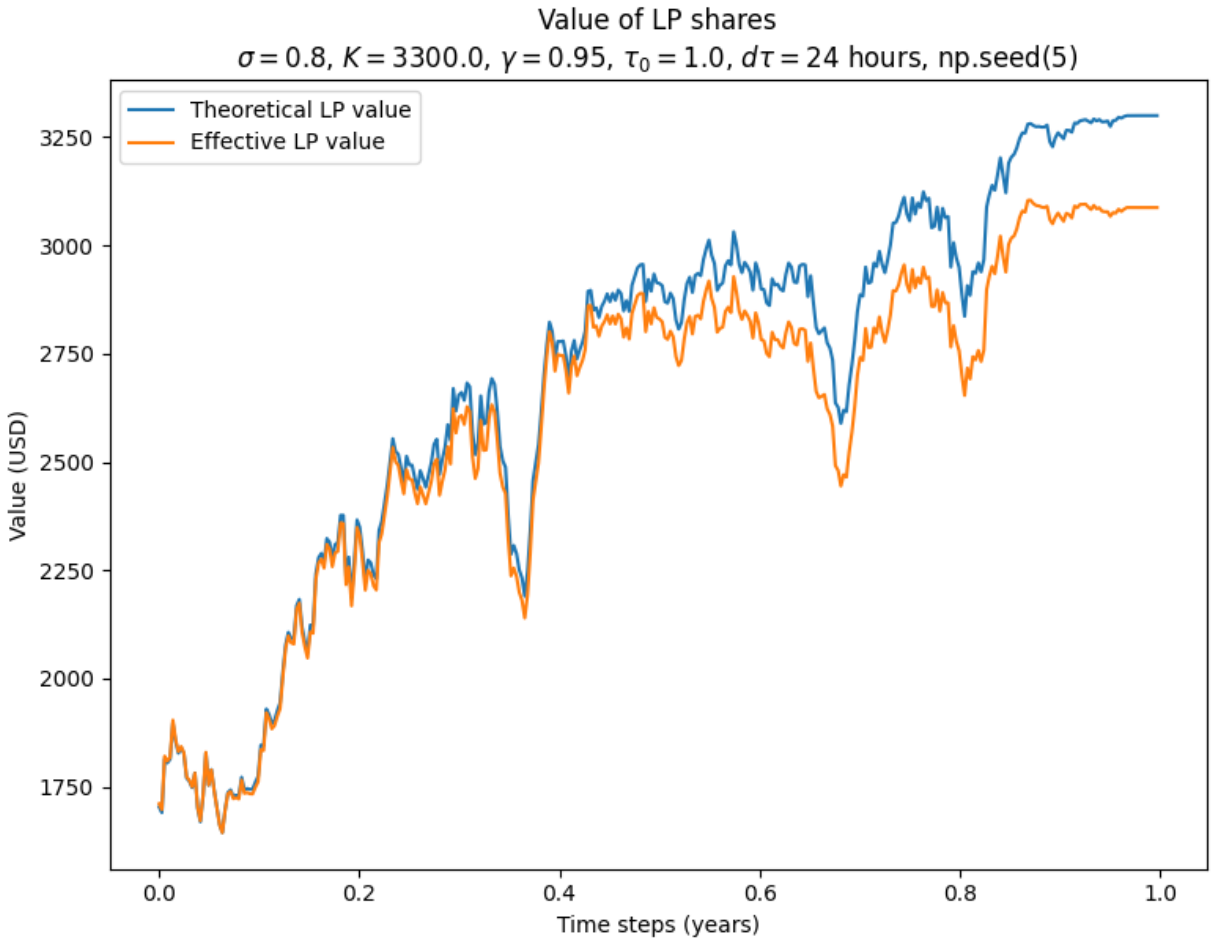

Figure 3: *5% fee case* ($\gamma = 0.95$) *with the same sample data, the RMM still fails to replicate the covered call, but the gap between theoretical and effective payoff is drastically reduced [9].*

The above charts are for a single price path and option inputs $K = 3300$, $\sigma = 0.8$, $\tau = 1$ year. Given some options inputs, many price paths, and a set fixed fee, is there a particular swap fee that probabilistically minimizes this terminal payoff error? Taking the mean of the terminal payoff errors over all the price paths and optimizing for a minimum bound has proven to reduce error.

Different assumed arbitrage frequencies in the simulations will yield different results and thus require a separate minimum fee. More frequent trading events will require a smaller fee to recover the covered call payoff. In contrast, spread-out trading events will require a more significant fee to capture the effect of theta decay over the period. To accurately simulate end behavior and achieve a static optimal fee on the pool, simulations start from the boundary case perspective for expected arbitrage frequency [9].

To find a working static fee there are two main approaches. The first approach sets

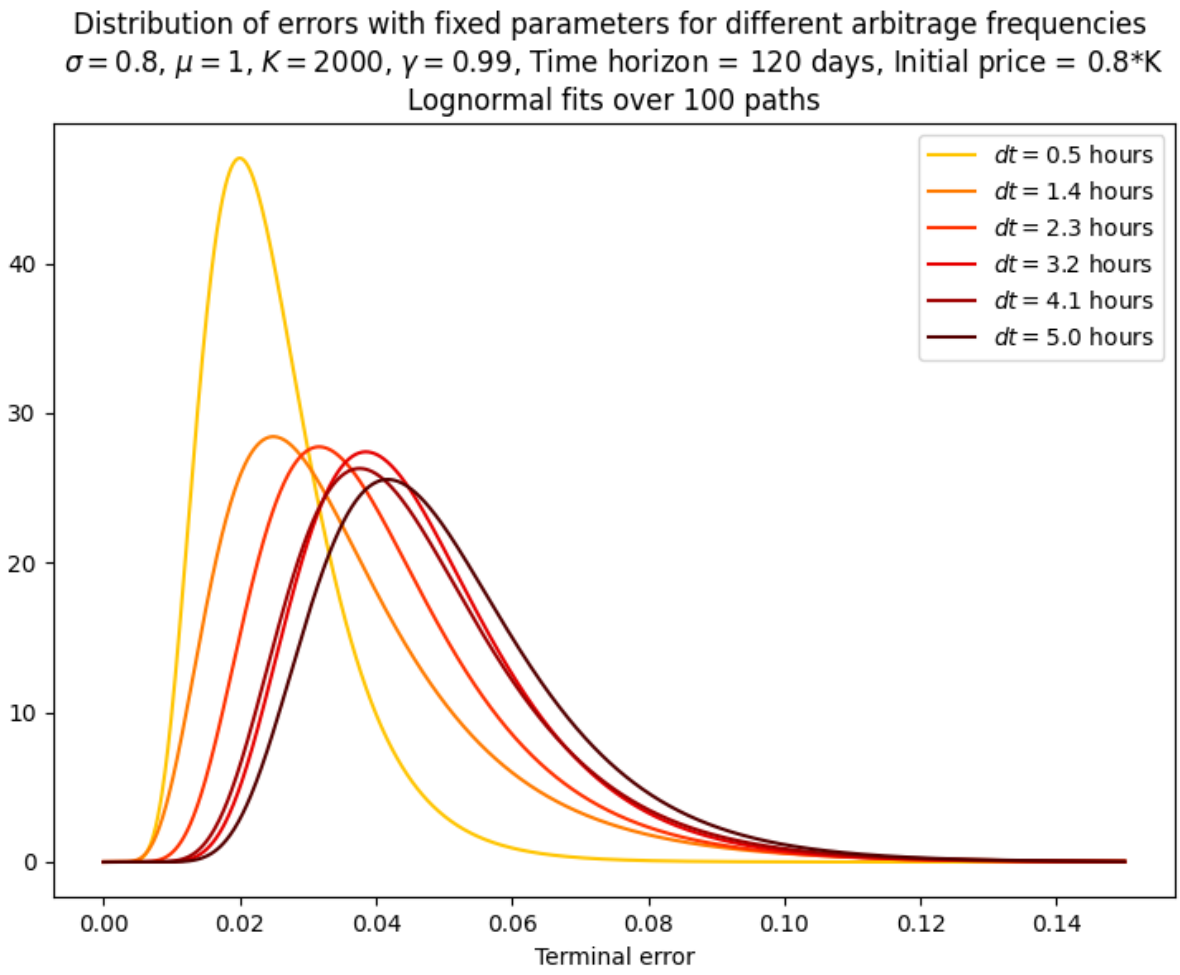

Figure 4: *The terminal payoff error distributions attained by varying the arbitrage frequency over the same pool inputs (strike, implied volatility, duration, initial price and fee)*

the fixed fee to the optimized fee obtained where only arbitrage trades occur with a low frequency ζ. Replication error is minimized if all trades occur with frequency ζ.

Consequently, this approach commonly yields a high fee preventing the pool liquidity from being utilized for standard trading flow. Furthermore, implementing this approach in a permissionless system requires trade-offs elsewhere. The second approach is to set a low fee but rely on indiscriminate trading volume to compensate for the difference in fee income. The advantage is that the protocol liquidity now becomes accessible to traders instead of just arbitrageurs. In the former case, the fee acts as a barrier to traders. The disadvantage is that this is more heavily dependent on indiscriminate trading volume, which may not be the most dependable than the arbitrage-only volume required by the first scenario. There should be no implicit assumption of indiscriminate trading volume. Many CFMM implementations don't see trading volume beyond arbitrageurs.

While a fixed fee can work at recovering theta decay over the pool's lifespan, we posit

that a dynamic fee would also be a viable alternative, barring any gas conditions. As a covered call approaches expiry, the effect of theta decay accelerates. Consequently, the pools benefit from a fee proportionally compensating for the replication error. The disadvantage is that the fees increase proportionally as the replication error grows, potentially thinning trading volume further.

2.5.1 Constraints and Bounds

The invariant function of the RMM contains both a standard normal cumulative distribution function, Φ, and its inverse. The domain of the standard normal quantile function is $\mathcal{D}(\Phi^{-1}) = (0, 1)$. This domain implies that the risky asset reserve x is bounded above 1. The bound on the risky asset is a consequence of the theoretical payoff of a covered call contract. To compensate, scale the inputs with the pool's global liquidity as shown in *2.2 Solidity Implementation*.

3 Solidity Implementation

RMM-01 is implemented as a single smart contract written in solidity, deployable by a factory contract. The factory contract deploys new RMM-01 engines per token pair, which create any amount of pools.

3.1 Trading Function Approximations

The compute constrained environment of the Ethereum Virtual Machine (EVM) forces the use of approximations for transcendental functions, such as the standard normal cumulative distribution function (CDF) and the quantile function. As a result, the trading function of RMM-01 is approximate, affecting the price slippage of trades on the curve. The trading function requires convexity, requiring the approximate function to bo convex as well, preventing the siphoning of funds. We are implementing the approximations [1], and [11] for the standard normal CDF and quantile function, respectively. The precision and exact implementation are broken down within[2].

3.2 Integers, Fixed Point Numbers, and Decimals

The trading function math uses a constant of `1`, which is a float —resulting in the other function variables, like the reserves, to have the same number of decimals. Floats are not available in Solidity 0.8.6. Therefore the ABDK 64.64-bit fixed-point number library is used for all trading function-related math. A signed 64.64-bit fixed-point number is a simple fraction whose numerator is a signed 128-bit integer and denominator is 2**64.

The math executes using fixed point 64.64-bit numbers, and once a result is calculated, it is scaled back to an unsigned 256-bit integer. The value of a reserve starts as an unsigned integer with decimals equal to the decimals of the token. To properly scale these to have the

same decimal places as the `1` constant, an immutable scalar value is stored for the respective tokens. This scalar value is multiplied or divided against the token amount so that it can be used in the trading function math or when transferring the tokens.

The `Units.sol` library has several utility functions to scale between these different types.

3.3 Two-Token Pairs

Each `PrimitiveEngine.sol` contract is deployed from the Factory by specifying two tokens: a `risky` and `stable`. As such, each Engine is responsible for the pools of those tokens.

Since not every token has the same decimal places, the scalar values are calculated as `10**(18 - decimals)` by the `PrimitiveFactory.sol` contract during a `deploy` call. Any token can be used, with the condition that its `decimals` are between 6 and 18, inclusive.

The other immutable variable that the factory calculates is `MIN_LIQUIDITY`, a constant amount of liquidity that is burned when a new pool is created. This value equals the lowest decimal token out of the risky and stable divided by 6. Since the lowest token decimals supported is 6, the `MIN_LIQUIDITY` is at least one wei. Small quantities of liquidity must be burned, avoiding the scenario in which all the liquidity of the pool is burned. A pool with zero liquidity is not defined.

3.4 Multiple Pool Configurations Per Pair

Each Engine contract has a `calibrations` mapping. This is a `poolId` mapped to a `Calibration` struct, in which the parameters of the pool are stored as state.

Type	Variable	Notation
uint128	strike	K
uint32	sigma	σ
uint32	maturity	T
uint32	lastTimestamp	t
uint32	gamma	γ

The `poolId` is simply a keccak256 hash of the packed parameters: `engine address`, `strike`, `sigma`, `maturity`, and `gamma`. It is important to make sure the correct types are used when packing and hashing the variables.

If a pool has a `lastTimestamp` that is greater than zero, it is considered initialized. This is because this variable is always set using `block.timestamp`, which should not be zero.

Each pool's trading function uses these static parameters to calculate the reserves and the invariant.

3.5 Liquidity Initialization

Pools are first created using the specified arguments in the `calibration` and an amount of initial liquidity to mint, which the caller must pay. Additionally, the initial `R1`, risky reserve

per liquidity, is an argument that should be computed off-chain. This is the most critical parameter, as it effectively determines the spot price of the pool on creation.

This `R1` variable should be initialized to `1 - delta`, where `delta` is the delta of the call option being replicated from the pool's parameters.

$$\Delta = N(d_1),\ d_1 = \frac{ln(S/K) + r\sigma^2/2}{\sigma\sqrt{\tau}}$$

If initialized with the fair value, the implied spot price of the new pool should match the reference market price, `S`, used to calculate the `delta`.

The `calibrations` state is updated, mapping the `poolId` to the `calibration` arguments with the `lastTimestamp` equal to the `block.timestamp`, making the pool initialized.

3.6 Adding & Removing Liquidity

The trading function is defined to replicate a single covered call payoff, resulting in a bounded range on the risky reserve where the trading function is well-defined. This is a result of the quantile term in the trading function. As such, given n LP shares (equivalent to covered call contracts), liquidity can be scaled, and the trading function can be re-defined as follows:

$$y - K\Phi(\Phi^{-1}(\frac{n-x}{n}) - \sigma\sqrt{\tau}) = k$$

Providing liquidity takes arguments for the number of tokens to add to each side of the pool, which means that an `optimal` amount of liquidity can be minted. The pro-rata amount of liquidity is calculated for each side of the pool, and the lesser amount is the chosen liquidity to mint. Therefore, the desired amount of liquidity to allocate should be used to calculate each side of the pool. Both liquidity amounts are the same when calculated in the Engine.

This is to prevent value siphoning from specifying a liquidity amount that rounds each side of the pool up. While for 18 decimal tokens, this might only be a one Wei difference, for high-value tokens like WBTC, one Wei is worth a non-trivial amount.

Removing liquidity will take the liquidity off the curve and deposit the two tokens into the `msg.sender`'s `margin` account. As such, a higher-level contract should perform a multicall with two function calls: `remove` then `withdraw` to receive the underlying tokens into a target account.

3.7 Swapping

Either the `risky` or `stable` token can be swapped for the other, as long as the reserves and their corresponding invariant increase or stay the same immediately prior- and post-swap.

The function argues the desired amount of output tokens and the amount to input as payment. Therefore, optimal amounts to swap are possible and should be pre-computed off-chain and executed through a high-level contract with a price impact check.

The trading invariant is time-based. As a result, the theoretical *curve* in which the pool is trading on is continuously changing its curvature.

For a visual example, you can think of a string that makes a U shape, and as time passes, each end of the string is being pulled more and more until it makes a straight line. This straight line is the marginal price of a swap which is equal to `K`, the strike price. Therefore, beyond expiration, `K` units of the `stable` token can be swapped for `1` unit of `risky` token and vice versa.

The curve needs to be updated to the `block.timestamp` before a swap; else the swap would be trading on an old curve that does not reflect the time which has passed. In the `swap` call itself, there is a timestamp update before any swap-related operation takes place. With the now updated curve, the current `invariant` can be stored in memory to compare it against the post-swap invariant.

The post-swap invariant is calculated using the input token reserve plus the amount in *with the fee applied* and output token reserve less the amount out. However, the actual token payments, and thus the amounts to update the reserves, are the amount in and amount out. This difference in the invariant check has the effect of re-investing the swap fee into the liquidity pool. The check will pass if the post-swap invariant is greater than or equal to the pre-swap invariant in memory (post time update).

3.8 Optimistic Transfers

Swaps can be paid for by externally transferring tokens or debiting the `margin` account of the caller. In the former case, the output token `deltaOut` amount is transferred to the caller before payment is required. This allows the caller to use the swap revenue to pay for itself, similar to a flash swap. If the required `deltaIn` amount of tokens is not paid, the swap will revert.

3.9 Agnostic Payments

For swaps that are not paid using a `margin` account, a callback function is triggered, calling back into the `msg.sender` of the swap call. For this reason, a higher-level smart contract that implements the `swapCallback` must be used to pay for swaps from external accounts. This benefits from paying for the swaps by executing arbitrary logic in the callback, an ideal scenario for composability with other protocols.

3.10 Internal Balances

Mentioned in the previous section, higher-level smart contracts can deposit the Engine's `risky` and `stable` into a `margin` account. The primary use of this is to preemptively fund a `margin` account to use when adding/removing liquidity, or doing swaps. Using the `margin` account for these operations will prevent one or two token transfers, which are often more than 15% of the total gas used per stateful function. This feature can be leveraged to fund an account for arbitrage opportunities for arbitrageurs preemptively.

3.11 Swap Fee & Fee Adjustments

While the theoretical curve is being adjusted over time, the effective curve in practice will only follow this behavior with a non-zero swap fee due to the portfolio tracking problem described in *2.1 Application*. For each pool's parameters and underlying token pair, an optimal fee should exist based on the pool's expected liquidity and volume that probabilistic-ally reduced the terminal payoff difference.

When a new pool is created, a `gamma` argument can be specified by the caller to set the pool's swap fee for its lifetime. The fee itself is not stored or directly referenced because only the `gamma` is used in the actual `swap` function, which is stored instead. A pool's `gamma` is a 32-bit unsigned integer with four decimal places, e.g., 10,000 is equal to 100 percent in the smart contract.

Fee revenue is re-invested into the liquidity tokens at the end of the swap. In application, the purchasing power of the input tokens is multiplied by `gamma`, which is `1 - fee %`. This will result in fewer output tokens for a valid (invariant check passing) trade. The new invariant is calculated using the amount in with the fee applied, which means the `deltaOut` argument needs to be estimated using a fee-included amount.

The full swap-in amount is paid from the caller and added to the respective reserves. This has the effect of paying an additional amount of tokens (fee amount), which will increase the invariant.

3.12 The Invariant

On that note, the `invariant` is denominated in units of the stable token. At maturity, this will be negative or positive, but ideally zero. This does not imply zero profit, but rather the pool captured all theta decay through the swap fees. Over time, the `invariant` is decreasing, all else equal, and it is the swap fee revenue that increases it back towards zero.

4 Implications

Since a covered call position is concave, convexity is achieved by shorting the LP position. Shorting LP positions consists of borrowing the shares at $1 - V$, where V is the value of the share in the risky asset, then re-denominating the value into either the numéraire or the risky asset, as shown in[8]. Upon exiting the position, the liquidity shares are re-minted and repaid (or at expiry, the equivalent position is returned), resulting in a long put position if the re-denomination was to the numéraire and a long call if to the risky.

4.1 Exotic Options

A much simpler implication is in exotic options. In a covered call position, one can allow users to purchase the right to either the risky asset or stable asset of the LP position, given

that the spot price is below or above the strike. This leads to the creation of asset-or-nothing put options and cash-or-nothing call options. This allows for much more diverse hedging plays that traditional options would fail to capture on their own.

5 Conclusion

In this paper, we defined and implemented a RMM, called RMM-01, that replicates a payoff similar to that of a Black–Scholes covered call option. This CFMM was originally defined in [5]. However, it failed at self-financing replication due to the inability to capture theta decay in the no-fee case. We showed that an optimal static fee can be used to successfully recover the missed theta decay accrual, and thus allow the RMM to achieve payoffs very similar to that of covered call options for its liquidity providers. RMM-01 has broad implications in derivatives beyond just the covered call, such as recovering long options, resulting strategies, and exotics, and allows passive management without the need of re-balancing on more complex hedging positions on-chain. These implications on further derivatives are well worth exploring in-depth and as such will be addressed in a separate paper. This is just one of many potential RMM implementations that could serve beneficial for constructing more complex portfolios, and as such we aim to further explore RMMs in the future.

References

[1] Milton Abramowitz and Irene A Stegun. Handbook of mathematical functions with formulas, graphs, and mathematical table. *US Department of Commerce; National Bureau of Standards Applied Mathematics Series*, 55, 1965.

[2] Alex Angel. rmm-core, 3 2022.

[3] Guillermo Angeris, Akshay Agrawal, Alex Evans, Tarun Chitra, and Stephen Boyd. Constant function market makers: Multi-asset trades via convex optimization. *arXiv preprint arXiv:2107.12484*, 2021.

[4] Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 80–91, 2020.

[5] Guillermo Angeris, Alex Evans, and Tarun Chitra. Replicating market makers. *arXiv preprint arXiv:2103.14769*, 2021.

[6] Alain Bensoussan. On the theory of option pricing. *Acta Applicandae Mathematicae*, 2:139–158, 06 1984.

[7] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. In *World Scientific Reference on Contingent Claims Analysis in Corporate Finance:*

Volume 1: Foundations of CCA and Equity Valuation, pages 3–21. World Scientific, 1973.

[8] Tarun Chitra, Guillermo Angeris, Alex Evans, and Hsien-Tang Kao. A note on borrowing constant function market maker shares. 2021.

[9] Anonymous experience. Simulating rmms - documentation.

[10] Fernando Martinelli and Nikolai Mushegian. A non-custodial portfolio manager, liquidity provider, and price sensor. *URl: https://balancer.finance/whitepaper*, 2019.

[11] Paul Voutier. A new approximation to the normal distribution quantile function. *arXiv.org, Quantitative Finance Papers*, 02 2010.