Primitive RMM-01

0xEstelle
estelle@primitive.finance

experience
experience@learnedtrustlessness.io

Alexander Angel
alex@primitive.finance

Matt Czernik
matt@primitive.finance

October 2021

Abstract

In a few short years, 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 which are fully backed by a community of liquidity providers seeking to earn yield on their deposited assets. The portfolio of a liquidity provider follows a payoff structure specific to the CFMM they are providing to. 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, or RMMs, for short. In this paper, we present an implementation of a RMM that replicates a payoff similar to that of a Black–Scholes covered call, which we call RMM-01.

Contents

1 Introduction

The design space of CFMMs [AC20] and their potential impact on on-chain derivative design has been limited by the slow process of starting from scratch each time for each new CFMM use case. This has long been a problem in the space, as no solid framework for converting from a payoff structure to a CFMM existed, until recently with the publication of [AEC21]. The paper proved the equivalence of the space of concave, non-negative, non-decreasing, 1-homogeneous payoff functions and the space of convex CFMMs, as well as provided a method to convert from 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 a 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 wasn't able to achieve self-financing replication due to the CFMM's inability to capture theta decay [AEC21]. We show that a static fee can be implemented in such a way that theta is recoverable within a probabilistic range, and thus a covered call payoff can be replicated. Before diving into this implementation, there are some preliminary concepts that 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 major research focus within decentralized finance, and a convenient building block for more complex decentralized financial derivatives [AAE+21]. 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 with the intention of passively earning yield and potentially hedging 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 [AEC21]. For example, Balancer [MM19] maintains a personalized weighted balance on the pooled assets; this includes up to 8 assets. To further understand CFMMs and the payoff structures they provide, it would be beneficial to briefly formally define them again.

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 [AAE+21]. 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 in, then recalculating k with the full amount in added to the pool. This will result in a smaller amount out to the trader with the full amount added to the pools still.

The reported price of a CFMM is also 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 is decreasing whenever the X increases yielding a negative rate of change, and price must be positive. This returns us the reported price of the CFMM, and the price received on an infinitesimal trade, however this is not the price received on a regular trade. For more on the math associated with CFMMs, refer to [AAE+21]. We now turn to the liquidity provider and more specifically, 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, with respect to some separate 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; thus, in comes [AEC21].

1.2 Replicating Market Making

There has been increasing attention in CFMM research focused on replicating particular payoff structures as CFMM shares, as well as what actually can be replicated as CFMM shares. To address this, [AEC21] was introduced with two key results. First, it shows the space of convex CFMMs is equivalent to the space of concave, non-negative, non-decreasing, 1-homogeneous functions. This tells us in particular what payoff functions can be replicated as shares directly. Second, it gives a direct method of converting between 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 the use of leverage. For more on recovering convexity see [CAEK21]. 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 replicates a Black–Scholes covered call: exhibiting a strike price, time of expiry and implied volatility over the duration, but without the exact same payoff as a covered call [AEC21]. Due to its relation to Black–Scholes options, it would be beneficial to review options as a whole and associated terminology such as *The Greeks*, as framing the product from that perspective allows for deeper 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 of time, subject to certain conditional requirements [BS73]. Options allow for asymmetric directional exposure, increased leverage, the ability to hedge risk, and a composable framework for achieving a 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 random and often unpredictable nature of volatile assets' price motion, many different pricing models arise from unique probability models. These conditional requirements will often involve a *strike price* (K) that the underlying asset's price must be above or below, depending on the contract type, in order 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 conditional requirements aren't met by the expiration of the contract, the option expires worthless leaving the premium paid as 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 [B84]. We will be focus on European options priced 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 [BS73].

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 with respect to an infinitesimal change in the spot

iv. *Theta* $\theta = \frac{\partial V}{\partial \tau}$: Rate of change of the option's value with respect to 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 with respect to an infinitesimal change in the option's Delta Δ. Can be thought of as the rate of acceleration of the option's value with respect to spot.

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 with respect to 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, as a whole, are vital tools for understanding an option's behaviour and managing risk accordingly. One particularly notable 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 as a result of the probability of expiring profitably. This is where theta decay arises.

Theta decay is the decline of the time value portion of the option's value as expiry approaches. Intuitively, as time to expiry decreases, the probability of profitability from purchasing the contract decreases as well, 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 [AEC21] 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.
However, 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

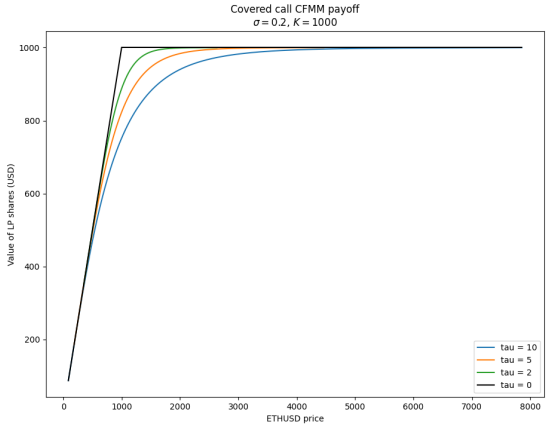

Figure 1: *The value of the RMM-01 LP share, assuming no-arbitrage, as a function of the risky asset price and* τ.

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, $\gamma = 1 - fee$. It should be noted that the fee can be optimally adjusted for each pool to best recover the deviation in value between the theoretical payoff and effective payoff, given a set of environmental expectations such as trade frequency, as shown in section *2.1 Application*.

In our context, we will denote any ERC-20, with the exception of "stable" assets such as USDC and DAI, as risky assets, as well as any "stable" ERC-20 assets as risk-less assets. In theory, the CFMM can support any token pair, however, due to the nature of 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.

Payoff Structure

First, it would be 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 $d_2 = d_1 - \sigma\sqrt{\tau}$ [AEC21]. 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, $S > K$, then the LP receives a payoff of $V = K$ risk-less, fully denominated as risk-less. This leaves a position of K risk-less.

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

As mentioned above, however, the payoff of the RMM-01 has a portfolio tracking error due to the inability to capture theta decay properly [AEC21]. The fee has the intention of re-capturing the accrual, however regardless of the outcome, 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*.

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 $\sigma > 0$. However, due to the usage/dependence on the standard normal CDF, this function exhibits significant kinks at the extreme ends of the scale, meaning some prices cannot feasibly be reached in a real-world setting; the RMM-01 will simply yield an upper/lower bound for prices above some threshold point [e21].

Swap Calculations

Assume the pool initially has x, y liquidity of the constituent risky asset and risk-less asset respectively. First, 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,

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

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

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

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

Price of Swaps

We can obtain the price of a trade using the swap procedures described above. First, 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. Simply 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 too, where the trader is looking to sell Δ risky asset for the risk-less asset. The quantity received by the trader is,

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

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

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

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 $\Delta > 0$. This implies that selling always has a negative price impact, as expected. To calculate the price impact simply 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 $x' = x - \Delta'$,

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

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 $\Delta > 0$. This implies purchasing the risky asset will always lead to an increased price, as expected. Again to calculate the exact price impact simply compute:

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

2.1 Application

There are a few aspects of the model that particularly merit further discussion, such as the portfolio tracking problem and resulting fee structure, a few subtle constraints and bounds of the RMM, the implementation in solidity, and the implications of the product.

To further expand on the portfolio tracking problem, assume the following: 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 actually increase ($V_i < V_f$). However since no trading flow has occurred on the RMM to cause updates, the reserves of the pool, and thus the composition and value of the LP share are left unchanged. This presents a difference in value between the covered call and that of the RMM share, requiring external funding for a correction.
By simply adding a static swap fee to the pool, this gap can be drastically reduced or closed, and in some scenarios, even yield an effective payoff higher than that of a covered call with the corresponding inputs (although this is neither the point or goal; rather simply a demonstration of the fact that the self-replication is never exact).

It should be noted that the above charts are for a single price path and set of option inputs (strike $K = 3300$, implied volatility $\sigma = 0.8$, and expiry after 1 year). A natural question arises: given some options inputs and assumed market conditions, is there a particular swap fee that probabilistic-ally minimizes this terminal payoff error? To test this for some given options inputs, we've generated a large number of price paths, set a static fee, taking the mean of the terminal payoff errors over all the runs, and applied an optimization

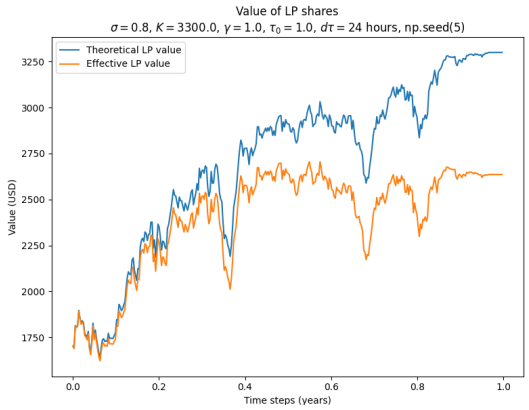

Figure 2: *No fee case* ($\gamma = 1.0$) *with sample strike, volatility, and price path. In the no fees case, the RMM fails to self-finance the replication of the covered call due to the inability to capture theta-decay, as seen above. See [e21] for more information on the simulations presented.*

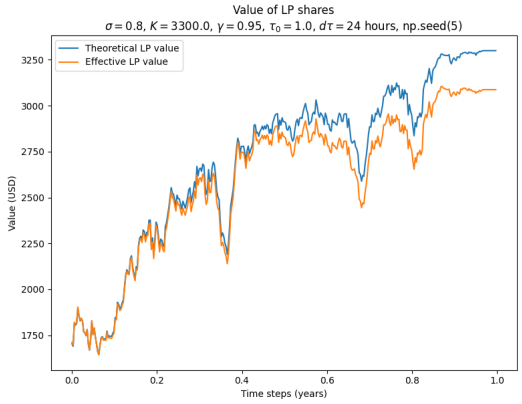

Figure 3: *5% fee case* ($\gamma = 0.95$) *with the same sample data. The RMM still fails to replicate the covered call but the gap between theoretical and effective payoff is drastically reduced. See [e21].*

routine to yield a fee with minimized expected terminal error (in-depth explanation of the fee optimization routine is described in *Simulating RMMs* [e21].

It should also be noted that different assumed arbitrage frequencies in the simulations *will* yield different results and thus require a different minimum fee. More frequent trading events will require a smaller fee to recover the covered call payoff, while spread-out trading events will require a larger fee to capture the effect of theta decay over the period between events. To accurately simulate something with so much variability and achieve a static optimal fee on the pool, simulations must be run from the boundary case perspective for *expected* arbitrage frequency (to which multiple estimates were simulated upon) [e21].

To find a working static fee there are two main approaches. First approach is to set the

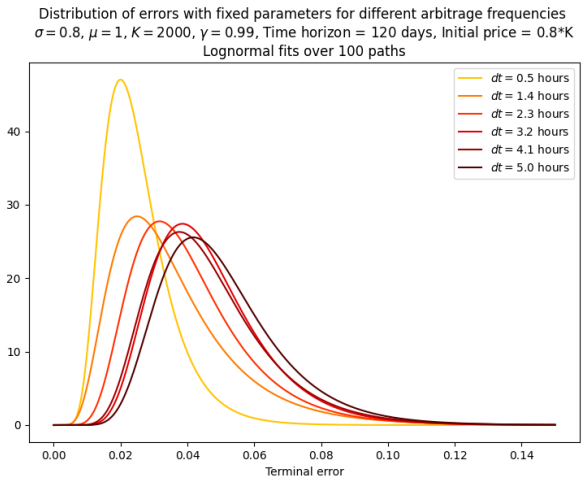

Figure 4: *The terminal payoff error distributions attained by varying the arbitrage frequency over the same pool inputs (strike, implied volatility, duration, initial price and fee)*

static fee to the optimized fee obtained in the case where only arbitrage trades occur with extremely low arbitrage frequency. The advantage of this is if trades can occur simply as often as the low arbitrage frequency, theta decay will be captured. Disadvantage however is this likely will yield a very high fee preventing it from being useful for standard trading flow and cannot easily be implemented trustlessly without sacrifices elsewhere. The second approach is to set a low fee but rely on retail trading flow to compensate the difference in fee income. The advantage of this is the protocol liquidity now becomes accessible to traders instead of just arbitrageurs, whereas in the former case the fee likely acts as a barrier to traders. The disadvantage is that this is more heavily dependent on retail trading volume which may not be the most dependable as opposed to the arbitrage-only volume required by the first scenario. For safety purposes, however, there should be no implicit assumption of retail trading volume, as many CFMM implementations don't see any beyond arbitrageurs, especially during the early stages.

While a static fee can work at recovering theta decay over the lifespan of the pool, we posit that a dynamic fee would also be a viable alternative, barring any gas conditions. The effect of theta decay largely accelerates the closer to expiry it gets, and as such the pools could benefit from a fee matching the required gap to fill. The disadvantage of this, however, would be that the fee could grow quite large nearer to expiry or if trading volume spread out.

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 implies the risky asset reserve x is bounded above by 1. This is the result of having the same theoretical payoff as a covered call contract of one unit (barring

the tracking error). To get around this issue, simply scale the inputs with global liquidity as shown in *2.2 Solidity Implementation*.

Turning to a rather subtle constraint of the RMM, the range of achievable prices is bounded, leaving some prices at the far extremes unachievable. Due to the nature of the standard normal quantile function, there are significant kinks at the extreme ends of the price scale, leaving some prices at the extreme ends practically unachievable on the RMM. For an example of this property, see [e21]. Now, we turn our focus to the implementation of RMM-01 in solidity.

2.2 Solidity Implementation

RMM-01 is implemented as a single smart contract written in solidity, deploy-able by a factory contract. The factory is used to permissionlessly deploy new RMM-01 *engines* per token pair, which can be used to create any amount of pools.

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 quantile function. As a result, the trading function of RMM-01 is approximate, affecting the price slippage of trades on the curve as well as the quantity received. The trading function requires convexity and as such, the approximate function being used in place must be convex as well to prevent the siphoning of funds. We are implementing the approximations [AS64] and [V10] for the standard normal CDF and quantile function respectively. The precision and exact implementation are broken down within [AL21].

Integers, Fixed Point Numbers, and Decimals

The trading function math uses a constant of `1`, which is a float. This requires the other variables of the function, like the reserves, to be scaled to a float with the same 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 basically a simple fraction whose numerator is a signed 128-bit integer and denominator is 2**64.

The math is executed using fixed point 64.64-bit numbers, and once a result is calculated its scaled back to an unsigned 256-bit integer. However, the raw value of a reserve starts as an unsigned integer with an amount of decimals equal to the decimals of the token. Therefore, 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 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.

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 is calculated by the Factory is `MIN_LIQUIDITY`, a constant amount of liquidity that is burned when a new pool is created. This value is equal to 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. A small amount of liquidity must be burned to avoid the scenario in which all the liquidity of the pool is burned, as a pool with zero liquidity is not defined.

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.

Liquidity Initialization

Pools are first created using the specified arguments in the `calibration`, and an amount of initial liquidity to mint, which must be paid by the caller. Additionally, the initial `R1`, risky reserve per liquidity is an argument which should be computed off-chain. This is the most important 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 proper 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.

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 the 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 amount of tokens to add to each side of the pool, which means that an `optimal` amount of liquidity can be minted. For each side of the pool, the pro-rata amount of liquidity is calculated, 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, such that both liquidity amounts are the same when calculated in the Engine.

The purpose of this is to prevent value siphoning from specifying a liquidity amount which 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.

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.

Both the desired amount of output tokens and the amount to input as payment are arguments in the function. 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` prior to 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 prior to any swap related operation taking 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 by, 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. If the post-swap invariant is greater than or equal to the pre-swap invariant in memory (post time update), the check will pass.

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 gives the caller the opportunity 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.

Agnostic Payments

For swaps which 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 which implements the `swapCallback` must be used to pay for swaps from external accounts. This has the benefit of paying for the swaps by executing arbitrary logic in the callback, an ideal scenario for composability with other protocols.

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. For arbitrageurs, this feature can be leveraged to preemptively fund an account for arbitrage opportunities.

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, there should exist an optimal fee based on the expected liquidity and volume of the pool, 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. To apply the fees in this way, the *purchasing power* of the input tokens is multiplied by `gamma`, which is `1 - fee %`. This will result in less 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.

The Invariant

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

3 Implications

Given the nature of a covered call position being short a call option while holding spot, as well as the opposite relationship between convexity and concavity, convexity can be achieved by shorting the LP position. This is done through borrowing the shares at a cost of $1-V$, where V is the value of the share in the risky asset, and re-denominating the value into either the numéraire or the risky asset, as shown in [CAEK21]. 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. This has implications in and of itself, however we will leave this for a later paper.

A much simpler implication is in exotic options. Due to this being 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 respectively. 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. These are just two of many possible implications of RMM-01 in further derivative design, and both deserve much more in-depth attention; however they will be left to a later paper.

4 Summary

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 [AEC21]. 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

[AC20] Guillermo Angeris and Tarun Chitra. Improved Price Oracles: Constant Function Market Makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, page 80–91, New York, NY, USA, 2020. Association for Computing Machinery.

[AEC21] Guillermo Angeris, Alex Evans, and Tarun Chitra. Replicating Market Makers. *arXiv preprint arXiv:2103.14769*, 2021.

[AAE+21] 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.

[MM19] Fernando Martinelli and Nikolai Mushegian. A non-custodial portfolio manager, liquidity provider, and price sensor. 2019.

[CAEK21] Tarun Chitra, Guillermo Angeris, Alex Evans, and Hsien-Tang Kao. A Note on Borrowing Constant Function Market Maker Shares. 2021.

[BS73] Fischer Black and Myron Scholes. The Pricing of Options and Corporate Liabilities. In *The Journal of Political Economy*, Vol. 81, No. 3 (May - Jun., 1973), pp. 637-654.

[B83] Alain Bensoussan. On The Theory of Pricing Options. In *Acta Applicandae Mathematicae* 2, pp. 139-158. 1984.

[e21] experience. Simulating RMMs - Documentation. 2021.

[AS64] Milton Abramowitz and Irene Stegun. Handbook of Mathematical Functions. 1964.

[V10] Paul Voutier. A New Approximation to the Normal Distribution Quantile Function. 2010.

[AL21] Alexander Angel and Clément Lakhal. CumulativeNormalDistribution.sol. 2021.