Rho: Automated Interest Rate Swaps

Authored by:
Max Wolff

Revised by:
Brandon McFarland
Nathaniel Mendoza

Abstract

Rho is a specification for an on-chain interest rate swaps protocol designed as an automated market maker. Rho allows traders to open swaps directly with the protocol, determining the fixed rate to charge or pay on incoming swaps with an interest rate model. Opening interest rate swaps allows traders to speculate or hedge risk on an underlying interest rate. Abstracting individual swaps away from liquidity providers allows them to simply supply funds into the protocol and collect trading fees.

Contents

I. Introduction

Rho is a low-friction automatic market maker for interest rate swaps. It offers to charge or pay traders a fixed rate on a notional amount in return for paying or charging a floating rate on that amount using an interest rate model. Interest rate swaps can be used to hedge risk or speculate on the volatile interest rates of an on-chain borrow. Combining an interest rate swap with a floating-rate borrowing position can be used to net out to a fixed-rate borrow. The floating leg of Rho interest rate swaps can be benchmarked to any rate, but benchmarking to Compound cToken [1] interest rates allows the protocol to be run with minimal counterparty risk while indexing the on-chain borrowing market.

Instead of traders opening swaps directly with each other, they trade against the protocol's liquidity providers. Abstracting individual swaps away from liquidity providers allows them to simply supply funds into the protocol and collect trading fees. It also smoothes demand for swaps, allowing traders to open positions without having to wait for a counterparty. Liquidity providers can withdraw their funds from the protocol at any time, provided that not all of the liquidity is locked as swap collateral. A dynamically set fee based on protocol usage incentivizes some liquidity to remain unutilized and available for withdrawal.

II. Prior Work

Several interest rate derivatives have been explored, but none have significant deployments on mainnet. Proposed designs include schemes for bonds, like Yield [2] and Maple[3], and exotic derivatives like LSDai [4] and Cherry Swaps [5]. However, these approaches lack pure and precise exposure to interest rates. Some force traders to speculate on the future trade activity of the protocol itself. For example, a trader's payout in a Cherry swap position depends not only on the underlying interest rate, but on trade activity on Cherry Swaps that occurs in the future. Similarly, short-term

yToken trades on longer term Yield contracts carry complex exposure. Traders are exposed to expectations of future interest rates up until the entire market's expiration. Traders must also bear liquidity risk, since yToken positions are settled by trading yTokens on an open market that may be too shallow to accommodate high trade volume.

In contrast, Rho traders are only exposed to interest rate variability. Once a position is opened, only the underlying interest rate during the swap impacts the payout. RateLock [6] was the only prototype reviewed with a proposed design for interest rate swaps. However, its order book design could suffer from weak price discovery and liquidity the same way that early p2p protocols like the early Dharma prototype did. Rho pools market makers and traders into a single perpetual market, emulating Compound and Uniswap [7], two protocols that sacrificed flexible terms for greater user-friendliness and liquidity.

III.Protocol

a. Payouts and Collateral

Upon opening a swap, a trader decides upon the swap's notional amount and if they would like to pay or receive the fixed rate. Receiving the floating rate and paying the fixed rate puts the trader long on the floating rate, and doing the opposite constitutes a short position. The protocol offers all swaps at a single duration for simplicity. Once a swap has expired, anyone can call a function to close the swap. The trader's net profit from the swap is calculated and exchanged when it is closed:

$$fixedLeg = swapDuration \times notionalAmount \times fixedRate$$

$$floatLeg = swapDuration \times notionalAmount \times floatRate$$

$$payout_{payFixedSwap} = floatLeg - fixedLeg$$

$$payout_{receieveFixedSwap} = fixedLeg - floatLeg$$

Each swap is fully collateralized, meaning that both the trader and the protocol must lock up collateral with the value of the worst-case payment:

$$collateral_{payFixedSwap} = duration \times notional(swapRate - cToken_{minRate})^{1}$$

$$collateral_{receiveFixedSwap} = duration \times notional(cToken_{maxRate} - swapRate)$$

b. Supplying Liquidity

Rho takes the opposite side of every trader's swap, charging fees to ensure that it remains solvent. Anyone can supply liquidity to protocol, in return they have the right to withdraw their funds later and take a proportional share of the protocol's profits or losses. Providing liquidity should be roughly market-neutral with respect to interest rates, as the interest rate model adjusts its offered rate continuously to follow the market.

The protocol allows liquidity providers to enter or exit freely. Upon a withdrawal transaction, the protocol calculates its total funds by accruing interest payments from outstanding swaps. The protocol's total liquidity is accounted with a $supplyIndex$:

$$profitAccrued = fixedReceived + floatReceived - fixedPaid - floatPaid$$

$$totalLiquidity += profitAccrued$$

$$supplyIndex_{new} = supplyIndex_{old}\left(1 + \frac{protocolProfitAccrued}{totalLiquidity}\right)$$

Liquidity providers earn a proportional share of the protocol's profits or losses while their funds are supplied:

$$accountValue_{liquidityProvider} = initialSupply \times \frac{supplyIndex_{withdrawal}}{supplyIndex_{supply}}$$

[1] Collateral efficiency can be improved by using a bound around the likely values of a cToken's interest rates.

Liquidity providers can not withdraw funds currently set aside as collateral for active swaps. $ActiveCollateral$ is calculated by finding the protocol's worst case liabilities of all of its outstanding swaps:

$$activeCollateral = fixedToPay + maxFloatToPay - fixedToReceive - minFloatToReceive$$

When the protocol's payouts from the swaps that it owns surpass the absolute worst case, $activeCollateral$ decreases.

c. Interest Rate Model

Rho uses an interest rate model to determine the fixed rate it will offer to pay or charge traders on new swaps. It reacts to demand to pay or receive the fixed rate by shifting its offered rate, approximating the market's expectation of the benchmarked floating rate. The interest rate model reacts to incoming swaps by modifying the $rateFactor$, a value which is filtered through an s-curve to yield the final interest rate. The amount it reacts is proportional to the size of the swap, $swapNotionalAmount$, and the protocol's total funds $totalLiquidity$. For example:

$$rateFactor = rateFactor_{prev} \pm k \times \frac{swapNotionalAmount}{totalLiquidity}$$

$$k = rateFactorSensitivity \times contractDuration$$

A few constants are added to put the graph in the correct domain. For example:[2]

$$baseRate = \frac{range \times rateFactor}{\sqrt{rateFactor^2 + slopeFactor}} + yOffset \pm fee$$

The protocol charges a fee on all swaps on top of the $baseRate$ in order to reward liquidity providers. The fee increases when usage of the protocol's liquidity is high, to

[2] Described in more detail in this [notebook]

incentivize more liquidity providers to enter. The fee is a function of the portion of the protocol's funds that are locked up as collateral, or its $utilizationRate$. For example:

$$utilizationRate = \frac{activeCollateral}{totalLiquidity}$$

$$fee = basefee + feeMultiplier \times utilizationRate$$

This allows Rho to write more swaps, and provides a liquidity buffer that allows existing liquidity providers to exit.

IV.Interface

Function ABI	**Description**
openReceiveFixedSwap(uint notionalAmount)	Opens a swap where the trader receives a fixed rate and receives the benchmark floating rate. Pulls collateral from msg.sender, locks some of the protocol's funds as collateral.
openPayFixedSwap(uint notionalAmount)	Opens a swap where the trader pays a fixed rate and receives the benchmark floating rate. Pulls collateral from msg.sender, locks some of the protocol's funds as collateral.
closeSwap(uint orderNumber)	Determines a swap's payout once it has expired. Transfers the payout and any excess collateral to the trader.
addLiquidity(uint supplyAmount)	Transfers funds to the protocol, accrues interest from outstanding swaps and records the current supply index.
removeLiquidity(uint withdrawAmount)	Withdraws previously supplied liquidity from the protocol, in addition to the liquidity provider's share of the protocol's profits or losses since they supplied.
updateModel(uint midpoint, uint range, uint slope, uint baseFee, uint feeMultiplier)	Updates parameters for the protocol's interest rate model.

V. Future Work

A few features have been omitted from the specification for brevity and simplicity:

a. Proportional ownership of liquidity shares in Rho be tokenized, similar to Uniswap's UNI-tokens or Compound cTokens.
b. Allowing undercollateralized positions to be liquidated would allow for Rho to have much lower collateral requirements.
c. Rho can accommodate custom swap durations either by representing active swaps as tradable non-fungible tokens. It could also allow traders to settle swaps before they expire, though it would require reworking the interest rate model.
d. If long term interest rates could be approximated by reading a cToken's interest rate model parameters, a trustless interest rate model could be constructed. A public "poke" function would allow traders to move Rho's interest rate model's midpoint when the underlying cToken's parameters change.
e. Rho could withhold a portion of interest payments as reserves, serving as a liquidity cushion for withdrawals. This could also serve as a business model for the protocol. Interest-bearing tokens (such as Compound cTokens) could be used for swaps, reducing the opportunity cost of opening swaps and of supplying liquidity.

VI.Summary

a. Rho is a low-friction automatic market maker for interest rate swaps
b. Each swap has a fixed leg and a floating leg which is benchmarked on cToken rates, and a single standard duration
c. Traders can take out swaps to go long or short on interest rates, or hedge risk of an existing Compound borrow to net out to a fixed-rate borrow
d. Traders can supply liquidity to capture the profits of the protocol through fees while remaining roughly market neutral
e. Fees are dynamically adjusted based on protocol liquidity

Reference Implementation

I. Full Spec
II. Reference Implementation
III. Worked Example Spreadsheet
IV. Interest Rate Model

Acknowledgements

A big thanks to Hart Lambur, Regina Cai, Geoffrey Hayes, Jared Flatow, Coburn Berry, Dan Robinson, Mike McDonald and Alex Evans for conversations that inspired these ideas, and for reviewing early drafts of Rho.

References

[1] Robert Leshner and Geoffrey Hayes. Compound: The Money Market Protocol.
https://compound.finance/documents/Compound.Whitepaper.pdf
[2] Dan Robinson. The Yield Protocol: On-Chain Lending With Interest Rate Discovery.
http://research.paradigm.xyz/Yield.pdf
[3] Sidney Powell. Maple Finance.
https://maple.finance/app/uploads/2019/10/Maple-Whitepaper-v5.0-2019.10.16.pdf
[4] Sabine Bertram, Christopher Maree, Leisel Eichholz, ChrisJGF, Haythem Sellami
https://github.com/NeapolitanSwaps/CherrySwap
[5] David Leitert, Tina Zhen, Miao ZhiCheng, Dan Matthews, Kobuta23
https://devpost.com/software/lsdai
[6] Andy Chorlian, Anurag Angara, Noah Zinsmeister. Rateswap
https://devpost.com/software/rateswap
[7] Hayden Adams. Uniswap.
https://hackmd.io/@Uniswap/HJ9jLsfTz