Pendle Boros - an On-chain System for Margin Interest Rate Swap Markets

Pendle Finance

August 11, 2025

Abstract

Interest rate swap markets play a crucial part in a financial system, as apparent from its relative size in Traditional Finance (TradFi).

In this paper, we present Pendle Boros - an on-chain system for capital-efficient interest rate swap markets for any kind of interest that has an on-chain oracle, including TradFi interest rates. This system complements Pendle V2 to allow for much higher capital efficiency in speculation and management of interests/yields.

1 Overview

There are several components of Pendle Boros, one building on top of another:

- Interest rate swap accounting
- Interest rate trading zones and orderbook

The rest of the paper will discuss the components in details in chapter 2 and 3.

2 Interest rate swap accounting

2.1 High level overview

As with the traditional definition, an interest rate swap is an agreement between two parties to exchange one stream of interest payments for another over a set period of time.

This chapter describes a model for accounting the PnL of interest rate swaps among users, without regard for how the user can settle the profits and losses. These aspects will be discussed in Chapters 3.

There is always a **maturity** in the context of an interest rate swap.

In Pendle Boros, "the system" is always the direct counter-party to each user in the context of an interest rate swap. The system will make sure that the sum of all floating interest streams of all users is zero, such that the system takes zero exposure to interest rates fluctuations.

Central to Pendle Boros's interest rate swap mechanism is a **floating interest** on a **base asset** that fluctuates in real-time. For example, the short-term LIBOR rate on USD, the funding rates of ETHUSDT Perp market on Binance, or the stETH interest rate on stETH.

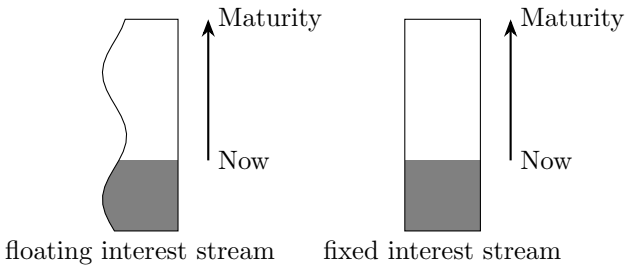

Figure 1: Graphical representation of interest streams

In Pendle Boros's interest rate swap, one interest stream is always the **floating rate**, which will be defined by an external oracle. The other interest stream is always a **fixed rate** that is specific to each interest rate swap position between a user and the system.

2.2 Conceptualising interest rate swaps

In this section, we discuss the two ways to conceptualize an interest rate swap position, which will become useful in thinking about the system.

We will use the example of a Binance ETH-USDT Perp Funding Rate market, expiring in 1 year, where the base asset is ETH.

2.2.1 Floating interest streams and fixed interest streams

A floating interest stream on x ETH represents the right to receive funding rate on x ETH over time, being paid in real-time as the funding rates are updated. An example of a tokenized version of a floating interest stream is the YT token in Pendle V2. 1 YT-stETH represents the right to receive the continuous yields on 1 stETH over time until a maturity.

A fixed interest stream of y% on x ETH represents the right to receive y% APR on x ETH, being paid in real-time together with the floating stream payments.

Figure 1 shows a graphical representation of the two types of interest streams.

2.2.2 Interest rate swap position - the traditional definition

The traditional definition of an interest rate swap position is to exchange an interest stream for another interest stream.

There are only two types of interest rate swaps:

- A fixed-to-float position, or a long rate position: receiving a floating interest stream and paying a fixed interest stream. Every time period, the user gets the floating interest payment and pays the fixed interest.
- A float-to-fixed position, or a short rate position: receiving a fixed interest stream and paying a floating interest stream. Every time period, the user pays the floating interest payment and receive the fixed interest.

Figure 2 shows a graphical representation of the two types of interest rate swaps. The black area represents the realised PnL over time.

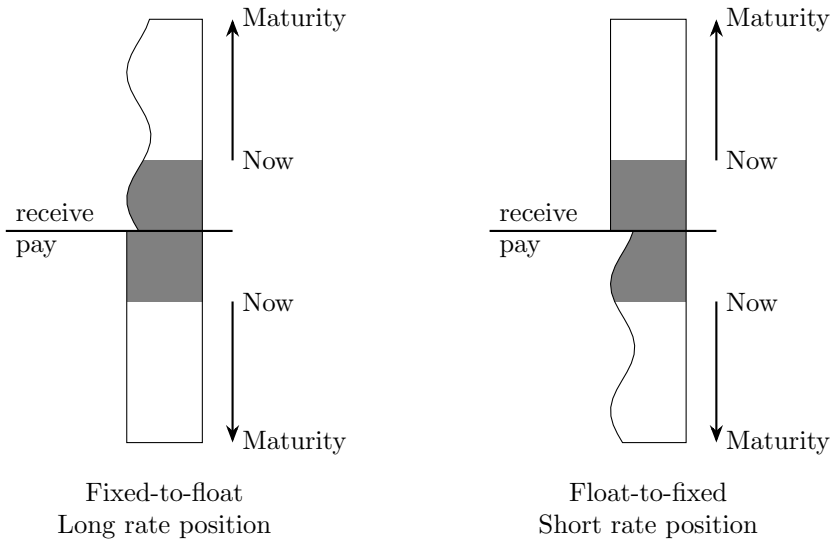

Figure 2: Graphical representation of interest rate swaps

2.2.3 Interest rate swap position - buying/short selling a floating interest stream

One observation is that when a swap is opened, we already know exactly how much the fixed stream will be worth: it is exactly

$$fixedAPR \times notionalSize \times timeToMaturity$$

In Boros' contract system, to chose to realise the value of the fixed interest stream immediately when the swap is opened:

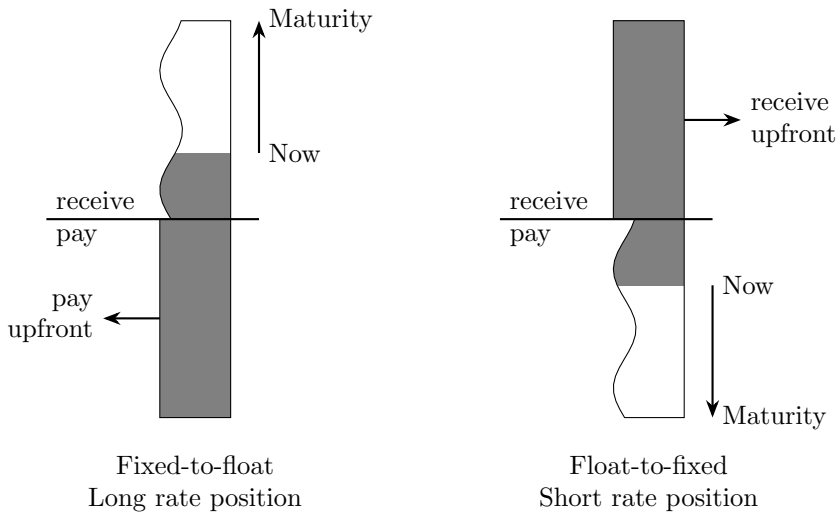

Figure 3: Upfront accounting of fixed rate streams

- When opening a fixed-to-float (long rate) position: the user pays the fixed interest debt immediately to obtain the floating interest stream. In a sense, the user "buys" the floating

interest stream using a fixed amount of capital paid upfront. Over time, the user will receive the floating interest payments.

- When opening a float-to-fixed (short rate) position: the user receive upfront the fixed interest payments immediately, while incurring a debt of a floating interest stream. In a sense, the user borrows and sells the floating interest stream to get the capital upfront. Over time, the user needs to pay back the floating interest payments.

Figure 3 shows a graphical representation of how the fixed stream's value is realised upfront in Boros' implementation.

2.3 Notation

time t represent time, in terms of years, from an arbitrary starting point (for example, Unix time 0 at 0 UTC 1 Jan 1970)

time delta represent the duration between two legs distributions of the swap, in terms of fraction of a year. A duration of one day is corresponding to $\frac{1}{365}$

maturity t_{end} is the maturity of the interest rate swap

base asset is the base asset whose interest is being exchanged in the interest rate swap

floating index $floatingIndex(t)$ is how much 1 unit of **base asset** would have grown into from an arbitrary time t_{start} until time t. This index defines the floating interest rates over time.

settlement fees $F_{settlement}$ is a fees charged on all open interest rate swap positions, at every settlement every **time delta**. It is represented as a real number. A fees of 5% will be represented by a value 0.05.

Note that in practice:

- **floating index** is obtained from an oracle. For DeFi yield, this floating index could be exactly the "exchangeRate" like wstETH/stETH rate.
- **settlement fees** is stored in the system contract as a configuration. It could be changed by the governance. We denote $F_{settlement}(t)$ as the settlement fees at time t.

Let's define a system of interest rate swaps among a group of users u from u_1 to u_N. Each interest rate swap market in this context is defined by a few parameters:

- A specific **base asset**
- A specific **maturity** t_{end} and **start time** t_{start}.
- The **floating interest** over time is specified via an increasing function $floatingIndex(t)$, with an arbitrary t_{start}.

Each **swap position** is defined by a few parameters:

- **open time** t_0 is the timestamp where the swap is opened.
- **position size** $s(t)$ is the size of the user's floating stream at time t. A positive position where $s(t) > 0$ means user u is getting the **floating rate**, while $s(t) < 0$ means the reverse.
- **fixed rate debt** $debt(t)$ at time t is the total obligation that the position needs to pay in its fixed rate leg. This obligation is conceptually "realised" every time a swap is open. It is as if the user has paid upfront all the fixed rate obligation.

sign of a position size is either 1 for long, 0 for 0, or -1 for short.

$$sign(s) = \frac{s}{|s|}$$
$$sign(0) = 0$$

2.3.1 Realised profit and loss

The **realised profit and loss** $p(t)$ of user u's **position** at time t is basically the **fixed rate debt**, the accumulated payments from the **floating rate** and the settlement fees, which we will define as:

$$p_{realized}(t) = -debt(t) + pFloat(t) + pFees(t) \tag{1}$$
$$pFloat(t) = \sum_{i=0}^{n-1} (floatingIndex(t_{i+1}) - floatingIndex(t_i)) \times s(t_{i+1}) \tag{2}$$
$$pFees(t) = -\sum_{i=0}^{n-1} |s(t_{i+1})| \times F_{settlement}(t_{i+1}) \times (t_{i+1} - t_i) \tag{3}$$

where $t_0 = t_{start}$ to $t_n = t$ are the discrete timestamps where the settlements happen. These discrete timestamps are mostly regular (for example, once every 8 hours, which means $t_{i+1} - t_i$ is always 8 hours), but **the frequency could change over time** based on the setting of the market (for example, a CEX perp funding rate frequency could change from every 8 hours to every 4 hours).

What this means from the implementation perspective:

- Every "settlement period" of **time delta** (which is 8 hours for most funding rates for example), we update the current global floating index from an external oracle
- Along side the floating index, we also update a quantity called $feeIndex(t)$, which is the accumulated settlement fees from t_{start} to t.

$$\begin{aligned} feeIndex(t) &= \sum_{i=0}^{n-1} F_{settlement}(t_{i+1}) \times (t_{i+1} - t_i) \\ &= feeIndex(t_{n-1}) + F_{settlement}(t) \times (t_n - t_{n-1}) \end{aligned}$$

- For each user, we keep track of the last floatingIndex since we "touch their position" (changing the position sizes)
- Since s doesn't change for the user from t_{last} to t_{now}, the changes in floating rate PnL ($pFloat$) and settlement fees PnL ($pFees$) are relatively simple to calculate:

$$\Delta_{floatingPnL} = s \times (floatingIndex(t_{now}) - floatingIndex(t_{last})) \tag{4}$$
$$\Delta_{settlementFeesPnL} = -|s| \times (feeIndex(t_{now}) - feeIndex(t_{last})) \tag{5}$$

- In implementation we **do not** need to store the total debt $debt(t)$, the total accumulated PnL from floating rate $pFloat(t)$ and the total accumulated PnL from settlement fees $pFees(t)$. Whenever we update the states for a user, we simply apply the changes to these numbers directly to the collateral of the user immediately.

$$payment = -anyNewDebt + \Delta_{floatingPnL} + \Delta_{settlementFeesPnL} \tag{6}$$
$$collateral = collateral + payment \tag{7}$$

- Note how we don't need to keep track of the fixed rate of the position. Whenever a new position is opened, we realise these debt immediately into the collateral. In other words, the user pays (or receive) the total value of the fixed rate stream upfront.

The tuple $(t, floatingIndex(t), feeIndex(t))$ is known as the `FIndex` in our contract.

Every time user u opens new position on this market, it's aggregated into his current position based on mechanism mentioned in 2.4.2.

2.4 State changes

2.4.1 The initial state

At the start at time $t_0 \geq t_{start}$, for every user and market, their position has zero size and zero fixed rate debt:

$$s(t_0) = 0$$
$$debt(t_0) = 0$$

The notation of user u and market M_i will be used when necessary (e.g. $s_u^{(i)}$ and $debt_u^{(i)}(t_0)$).

2.4.2 Create a new pair of swaps

The only state change possible in the accounting system for interest rate swaps is to create a new pair of interest rate swaps for two users u_i and u_j with the fixed rate f_{new} and position size s_{new} for u_i and the same fixed rate f_{new} and opposite position size $-s_{new}$ for u_j.

When we talk about "a swap with fixed rate f_{new}, position size s_{new} opened at time t", it is equivalent to "a swap of position size s_{new} and $debt_{new} = s_{new} \times t \times f_{new}$, opened at time t", which is the standard way to represent swaps in our system.

The existing position of each user will be merged with the newly created swap positions. At any time $t > t_0$: (denote $t-$ as the moment right before t)

$$(s_{u_i}(t), debt_{u_i}(t)) = mergeSwaps(s_{u_i}(t-), debt_{u_i}(t-), s_{new}, debt_{new})$$
$$(s_{u_j}(t), debt_{u_j}(t)) = mergeSwaps(s_{u_j}(t-), debt_{u_j}(t-), -s_{new}, -debt_{new})$$

As the **debt** is updated for both users, their realised PnL will be updated accordingly as well.

Merging of two swaps When merging two swaps, we simply add the position sizes and debts:

$$mergeSwaps(s_1, debt_1, s_2, debt_2) = (s_1 + s_2, debt_1 + debt_2)$$

3 Interest rate swap trading zones and orderbooks

3.1 Overview

In Pendle Boros, an **interest rate swap trading zone** is a collection of one or multiple "interest rate swap markets" with the same **base asset** (but the interest rates could be different or the **maturity** could be different or both could be different).

Note: In practice, for each base asset: there are multiple "isolated zones" consisting of only one market each (intended for more risky markets), and a cross-collateral zone comprised of the more bluechip markets.

To be able to open an interest rate swap, users will need to post some collateral in terms of **base assets** into the corresponding **interest rate swap trading zone** to guarantee that they can fulfill their obligations in the system (paying for potential negative PnL).

A user could open positions in multiple interest rate swap markets in the same interest rate swap trading zone, using the same amount of collateral.

To open a new interest rate swap, a user must have enough **initial margin** for the position they want to open.

To maintain all of their positions, their collateral must be enough to cover the unrealised profits and losses and the **maintenance margin** required for all of their positions. If the **current total position value** is below the required **maintenance margin**, the user can be liquidated by any other user.

As time passes, the realised profits and losses of a user's positions will be reflected in their collateral value.

There is also an orderbook integrated into each market of the trading zone.

The orderbook consists two sub-orderbooks: a positive orderbook consisting of orders with positive sizes, and a negative orderbook consisting of orders with negative sizes.

From the perspective of the interest rate swap accounting system (in chapter 2), the positive orderbook and the negative orderbook are just two users that might open swaps with other users and realise some profit and loss over time.

However, from the trading zone's perspective, the positive orderbook and the negative orderbook are not users. They don't have any collateral and don't get liquidated. Any open position, realised PnL and unrealised PnL by the orderbooks will implicitly count toward the open positions, realised PnL and unrealised PnL of users whose orders have been filled.

3.2 Definitions

Collateral value $c_u(t)$ is the amount of **collateral** a user u hold at time t in the system, in terms of **base assets**.

Interest rate swap (IRS) market refers to a particular interest rate swap market. There are n_M interest rate swap markets from M_1 to M_{n_M}. For each interest rate swap market M_i, the unit to measure PnL/value is the **base asset** and the positive value indicates gains.

- The **maturity** is t_{end}.
- The **Oracle Mark rate** of the market at time t is $r_m(t)$.
- The **initial margin (IM)** of a position of user u in the market at time t depends on their size and the margin of the open orders. The initial margin's formula is configured to control the risks in opening new positions in the system.
- The **maintenance margin (MM)** of a position of user u in the market at time t depends on their size. A position of **size** s of user u will have the maintenance margin of $MM(s, u, t)$ at time t. This function is configured to control the risks of open positions in the system.

Positive orderbook Each IRS market has a **positive orderbook** that consists of N^+ orders in total, which are divided into a set X^+_{open} of open orders, a set X^+_{filled} of filled orders and a set $X^+_{settled}$ of settled orders. An order can move from the open set to the filled set and finally the settled set.

- The total position size of the positive order book is s^+. This is the same as the total size of the orders in X^+_{filled}

- Each user u have N_u^+ orders in total, each order $order_{u_i}$ have size $s_{u_i}^+$, fixed rate $f_{u_i}^+$ and filled timestamp $t_{u_i}^+$ (which is 0 if the order is not filled)
- The orders in the positive orderbooks are grouped into multiple ticks from k_{-Z} to k_Z, each tick consisting of orders with the same fixed interests.
- Z is a constant for each market, which defines how many ticks there are in the orderbook $(2Z+1)$
- The **tick interest** I_{k_i} of tick k_i follows an increasing function on integers

$$I_{k_i} = g(i)$$
$$\text{where } g(0) = 0 \text{ and } g(i) = -g(-i)$$

  An example of this function is:

$$\begin{aligned} g(i) &= (1 + I_{smallest})^i - 1 \\ g(-i) &= -g(i) \\ i &\in N^* \end{aligned}$$

$$\text{where } I_{smallest} \text{ is the interest rate of the first tick}$$

- As such, each order can only have certain fixed rates as defined by the ticks.
- The N_u^+ orders or user u are divided into an open set $X_{open_u}^+$, a filled set $X_{filled_u}^+$ and a settled set $X_{settled_u}^+$. The Union of all users' open sets, filled sets and settled sets is basically the book's open set, filled set and settled set:

$$\begin{aligned} \sum_u N_u^+ &= N^+ \\ \bigcup_u X_{open_u}^+ &= X_{open}^+ \\ \bigcup_u X_{filled_u}^+ &= X_{filled}^+ \\ \bigcup_u X_{settled_u}^+ &= X_{settled}^+ \end{aligned}$$

- Each order consumes a fixed "pre-scaling initial margin" amount that is set at the time when the order is placed, according to the pre-scaling initial margin function at the price of the order

$$pim_{u_i}^+ = pim(s_{u_i}^+, f_{u_i}^+, t_{placed})$$

- Each user u will have a **positive book pre-scaling initial margin** $pim_u^+(t)$ at time t, which is the sum of all pre-scaling initial margin of all **orders that have not been settled**.

$$pim_u^+(t) = \sum_{order_{u_i} \in X_{open_u}^+} pim_{u_i}^+ + \sum_{order_{u_i} \in X_{filled_u}^+} pim_{u_i}^+$$

  Note: In practice, we always settle filled orders before calculating margin, so there will be no filled orders when calculating margin.

- The **total positive book size** of user u is b_u^+, which is sum of the size of all unsettled orders in the positive orderbook:

$$b_u^+(t) = \sum_{order_{u_i} \in X_{open_u}^+} s_{u_i}^+ + \sum_{order_{u_i} \in X_{filled_u}^+} s_{u_i}^+$$

Note: In practice, we always settle filled orders before calculating margin, so there will be no filled orders when calculating margin.

- Each user u will have an implicit position from each of their filled order $order_{u_i} \in X^+_{filled_u}(t)$ at time t, which gives u a position of size $s^+_{u_i}$ and fixed rate debt $debt^+_{u_i}$.

Negative orderbook Each IRS market an a **negative orderbook** that works exactly like the positive orderbook. It consists of N^- orders in total, which are divided into a set X^-_{open} of open orders, a set X^-_{filled} of filled orders and a set $X^-_{settled}$ of settled orders. The orders are also grouped into ticks from k_{-Z} to k_Z. The concepts and formulas for each user u are exactly the same, where the terms are:

- u has N^-_u orders in total, each order $order_{u_i}$ have size $s^-_{u_i}$, fixed rate debt $debt^-_{u_i}$ and filled timestamp $t^-_{u_i}$ (which is 0 if the order is not filled)
- The N^-_u orders or user u are divided into an open set $X^-_{open_u}$, a filled set $X^-_{filled_u}$ and a settled set $X^+_{settled_u}$
- The **negative book pre-scaling initial margin** is $pim^-_u(t)$ at time t
- The **total negative book size** of user u is $b^-_u(t)$
- The total implicit combined position from all the filled orders in the negative orderbook has size $s^-_u(t)$ and fixed rate debt $debt^-_u(t)$

Combined implicit position from orderbooks At any point in time, a user u will have an implicit position of $(s^{book}_u(t), debt^{book}_u(t)) = mergeSwaps(s^+_{u_i}, debt^+_{u_i}, s^-_{u_i}, debt^-_{u_i})$ at time t. This implicit position will have realised PnL p^{book}_u and unrealised PnL $p^{book}_{unrealised}$ just like another user in the system.

- The realised PnL as defined in Chapter 2.3.1:

$$p^{book}(t) = -debt^{book}(t) + pFloat(t) + pFees(t)$$

Note: In practice, the implicit position will always be merged into the user's standalone position first (by settling all filled orders) before we do any action that requires the calculation of realised or unrealised PnL.

For each user and a market

- For each user u at time t:
  - Their **standalone position size** is $s_{alone_u}(t)$
  - Their standalone position's **fixed rate debt** is $debt_{alone_u}(t)$ at time t

    Notes: the total accumulated fixed rate debt defined here is for the sake of understanding the system and defining the model. In practice, we don't store this number because any new debt will just be realised into the collateral value of the user.
  - Their **position size** $s_u(t)$ and **fixed rate debt** $debt_u(t)$ are the results from merging the standalone position $(s_{alone_u}(t), debt_{alone_u}(t))$ and the implicit position from the orderbooks $(s^{book}_u(t), debt^{book}_u(t))$ according to the mechanics in chapter 2.
    Notes: In our implementation, we always settle all user's filled orders before any on-chain action that needs the calculation of maintenance margin, so $s_u(t)$ and $debt_u(t)$ will be exactly the same as $s_{alone_u}(t)$ and $debt_{alone_u}(t)$ by then.

– Their **realized PnL** at time t is p_u which is the result from holding $(s_u(t), debt_u(t))$ over time. Any changes in realised PnL will be realised into the collateral value immediately.

$$c_u(t) = c_u(t-) + p_u(t) - p_u(t-)$$

– Their **positive pre-scaling initial margin** at time t $pim^+_{total_u}(t)$ is the pre-scaling initial margin consumed by positive orders and positions. This is 0 if the user is currently shorting rate and the total positive book size is not more than current position size.

if $s_{alone_u} < 0$ and $b^+_u \leq |s_{alone_u}|$

$$pim^+_{total_u}(t) = 0$$

Else:

$$pim^+_{total_u}(t) = pim^+_u(t) \pm pim(s_{alone_u}(t), r_m(t), t)$$

Note It is + if $s_{alone_u} \geq 0$

– Their **negative pre-scaling initial margin** at time t $pim^-_{total_u}(t)$ is the pre-scaling initial margin by negative orders and positions. This is 0 if the user is currently longing rate and the total negative book size is not more than current position size.

if $s_{alone_u} > 0$ and $b^-_u \leq |s_{alone_u}|$

$$pim^-_{total_u}(t) = 0$$

Else:

$$pim^-_{total_u}(t) = pim^-_u(t) \mp pim(s_{alone_u}(t), r_m(t), t)$$

Note It is - if $s_{alone_u} \geq 0$

– Their **pre-scaling initial margin** is the bigger number between the two above

$$pim_u(t) = \max(pim^+_{total_u}(t), pim^-_{total_u}(t))$$

– The **initial margin** consumed by the user at time t is $im_u(t)$

$$im_u(t) = IM\left(pim_u(t), u, t\right)$$

– Their **normalised position size** is $s_{norm}(t)$ at time t

$$s_{norm_u}(t) = s_u(t) \times (t_{end} - t)$$

– Their **unrealised PnL** at time t is $p_{unrealized_u}(t)$

$$p_{unrealized_u}(t) = s_{norm_u}(t) \times r_m(t)$$

– The **maintenance margin** for the position at time t is $mm_u(t)$

$$mm_u(t) = MM\left(s_u(t), u, t\right)$$

Total initial margin consumed by all of a user u's positions at time t is $m_u^{total}(t)$. It is the sum of all the **initial margin** consumed across all the markets, in terms of the base asset.

$$im_u^{total}(t) = \sum_{i=1}^{n_M} im_u^{(i)}(t)$$

Total maintenance margin required by all of a user u's positions at time t is $m_{main_u}^{total}(t)$. It is the sum of all the **maintenance margin** required across all the markets, in terms of the base asset.

$$mm_u^{total}(t) = \sum_{i=1}^{n_M} mm_u^{(i)}(t)$$

Current total value of a user u at time t is the sum of their collateral value and all the unrealized profits and losses across all the markets

$$v_u(t) = c_u(t) + \sum_{i=1}^{n_M} p_{unrealized_u}^{(i)}(t)$$

Health ratio of a user u at time t $healthRatio_u$ is the ratio between their **current total value** and their **total maintenance margin**

$$healthRatio_u(t) = \frac{v_u(t)}{mm_u^{total}(t)}$$

A user is deemed **healthy** if their **health ratio** is larger than 1.

Fees There are four fee factors related to 4 different types of fees in Boros:

- **Settlement fees** $F_{settlement}$ as defined in chapter 2.3
- **Taker fees** F_{taker} is charged on market orders on the orderbook. It is represented as a real number. A fees of 5% will be represented by a value of 0.05. Each user u can have a taker fee factor $fee_{u_{taker}}$.
- **OTC swap fees** F_{otc} is charged on public opening of swaps between any two users. It is represented as a real number. A fees of 5% will be represented by a value of 0.05. Each user u can have an otc fee factor $fee_{u_{otc}}$.
- **OTC swap fees** F_{otc} is charged on OTC swaps between any two users. It is represented as a real number. A fees of 5% will be represented by a value of 0.05. Each user u can have an otc fee factor $fee_{u_{otc}}$.
- **liquidation fees** F_{liq} is charged from the liquidator after a liquidation. It is represented as a real number. A fees of 5% will be represented by a value of 0.05.

The notation for market M_i and user u is omitted when the context does not require the market or user to be specified.

3.3 State changes

$t-$ denotes the moment right before a state change at time t

3.3.1 The initial state

At the start at time t_0, for every user u and every market M_i:

$$\begin{aligned} s^{(i)}_{alone_u}(t_0) &= 0 \\ debt^{(i)}_{alone_u}(t_0) &= 0 \\ p^{(i)}_u(t_0) &= 0 \\ p^{(i)}_{unrealized_u}(t_0) &= 0 \end{aligned}$$

For every user:

$$\begin{aligned} c_u(t_0) &= 0 \\ v_u(t_0) &= 0 \end{aligned}$$

3.3.2 Add/remove collateral

This is the only state change that an "ineligible user" can do. In the contract, "becoming eligible" is equivalent to entering the very first market.

At any time t, a user u can adjust their **collateral value** by d_c, as long as the **strict initial margin requirement** satisfies after the change.

$$c_u(t) = c_u(t-) + d_c$$

After the change, the user must satisfy IM requirement of:

$$im^{total}_u(t) \leq v_u(t)$$

Min deposit amount to become eligible To become eligible (in the contract, its equivalent to entering the first market), the user must have deposited a minimum amount $MinDepositAmount$, which is a global setting per asset zone.

3.3.3 Create a new pair of swaps

This must be triggered by "OTC swap", "Market order", "Liquidation" or "Forced deleverage".

At any time t, for any market M_i, a new pair of swaps of size s_{new} can be created between users u_i and u_j (either u_i or u_j could be the positive orderbook or the negative orderbook) following the state changes described in 2.4.2

There is a maximum global **Hard OI Cap** S_{cap} on the total Open Interest of the system:

$$\begin{aligned} OI(t) &= s^+ + \sum_{s_{alone_u}>0} |s_{alone_u}(t)| \\ OI(t) &\leq S_{cap} \end{aligned}$$

Note that the OI is s^+ plus the sizes of all **long** rate positions, which is exactly the same as s^- plus the absolute sizes of all **short** rate positions. To simplify the implementation:

- Instead of $OI(t)$, we calculate and store the OI as $2 \times OI(t) = s^+ + s^- + \sum_u |s_{alone_u}(t)|$;

- Store $2 \times S_{cap}$ as the configuration;
- Compare $2 \times OI(t)$ against $2 \times S_{cap}$ to check the global cap.

Note: the action of a user u doing a market order on the orderbook is basically the user creating a pair of swap with either the negative or positive orderbook, filling one or more orders in the process

3.3.4 Realising payments

We define $t_{lastPayment}$ as the timestamp of the last time this state change happened. $t_{lastPayment}$ is initialized to be the timestamp of the first payment interval in the market.

This state change happens at discrete timestamps t (usually regular, like every 8 hrs). The payments will be settled for every user from $t_{lastPayment}$ to t.

For each user u (which could also be the positive orderbook or the negative orderbook) and market M_i, the realised PnL is updated consistently with the formulas in 2.3.1:

$$\begin{aligned} pFloat_u^{(i)}(t) &= pFloat_u^{(i)}(t_{lastPayment}) + d_{float_u}^{(i)} \\ pFees_u^{(i)}(t) &= pFees_u^{(i)}(t_{lastPayment}) + d_{fees_u}^{(i)} \\ p_u^{(i)}(t) &= p_u^{(i)}(t_{lastPayment}) + d_{combined_u}^{(i)} \end{aligned}$$

where:

- $d_{float_u}^{(i)}$ refers to the PnL from the floating leg for the last epoch.

$$d_{float_u}^{(i)} = s_u(t) \times (floatingIndex(t) - floatingIndex(t_{lastPayment}))$$

- $d_{fees_u}^{(i)}$ refers to the settlement fees paid for the last epoch.

$$d_{fees_u}^{(i)} = -|s_u(t)| \times F_{settlement}(t) \times (t - t_{lastPayment})$$

- $d_{combined_u}^{(i)}$ refers to the change in realised PnL due to the last epoch.

$$d_{combined_u}^{(i)} = d_{float_u}^{(i)} + d_{fees_u}^{(i)}$$

If u is not an orderbook: the collateral value of the user will be updated based on all the combined payments across the market (which is the change in realised PnL)

$$c_u(t) = c_u(t-) + \sum_{i=1}^{n_M} d_{combined_u}^{(i)}$$

If u is either the positive or negative orderbook, the realised PnL for the implicit orderbook position p_u^{book} just gets updated implicitly accordingly.

Note: in practice,

- The payment realisation for filled orders in the orderbook will be processed in batches, grouped by epoches that they were filled, when the user's positions are settled (as defined in a state change in a later section).

- Before any operation that we need to calculate a user's realised PnL, we will settle all the filled orders of the user first. As such, all implicit orderbook positions will already have been settled.
- We only update the user's realised PnL lazily, according to the "What this means from the implementation perspective" section in 2.3.1.

3.3.5 OTC swap

This must be triggered by "Execute a batch of orders"

A user u_i can trigger an OTC swap with another user u_j (with their consent) to create a new pair of swaps of size s_{new} between the two of them. A state change "Create a new pair of swaps" is triggered.

The initiator will pay an OTC fees of:

$$otcFees = F_{otc} \times fee_{u_{otc}} \times |s_{new}| \times (t_{end} - t)$$

3.3.6 Open a new limit order

This must be triggered by "Execute a batch of orders"

A user u can open a new limit order $order_{u_i}$ with fixed rate f_{u_i} and size s_{u_i} at either the positive or negative orderbook, where $s_{u_i} > 0$ for the positive orderbook and $s_{u_i} < 0$ for the negative orderbook. The fixed rate f_{u_i} must be among the fixed rates of the ticks.

The new order will be added to the open set X_{open_u} of the corresponding orderbook, at the corresponding tick.

$$\begin{aligned} X_{open}(t) &= X_{open}(t-) \cup \{order_{u_i}\} \\ X_{open_u}(t) &= X_{open_u}(t-) \cup \{order_{u_i}\} \\ l_u(t) &= l_u(t-) + s_{u_i} \end{aligned}$$

Note: other variables that are derived from the changed variables will be updated implicitly

3.3.7 Filling an open order

This state change involves filling an open order $order_{u_i}$ by user u in either the positive or negative orderbook. This state change can only be implicitly triggered by a market order (state change 3.3.10).

$$\begin{aligned} order_{u_i} &\in X_{open_u}(t-) \\ X_{open}(t) &= X_{open}(t-) \setminus \{order_{u_i}\} \\ X_{open_u}(t) &= X_{open_u}(t-) \setminus \{order_{u_i}\} \\ X_{filled}(t) &= X_{filled}(t-) \cup \{order_{u_i}\} \\ X_{filled_u}(t) &= X_{filled_u}(t-) \cup \{order_{u_i}\} \end{aligned}$$

3.3.8 Removing an open order

User u can remove an open order $order_{u_i}$, by setting the order size to be 0. The variables here are for the corresponding orderbook.

$$\begin{aligned} order_{u_i} &\in X_{open_u}(t-) \\ l_u(t) &= l_u(t-) - s_{u_i}(t-) \\ s_{u_i}(t) &= 0 \end{aligned}$$

Note: other variables that are derived from the changed variables will be updated implicitly.

3.3.9 Modifying an open order

User u can modify the size of an open order $order_{u_i}$ to s_{new}, by first removing the order and then adding a new order with the size of s_{new}.

3.3.10 A market order

This must be triggered by "Execute a batch of orders".

When a market order of size s is done by user u on either the positive or negative orderbook, a series of other state changes are triggered:

- A taker fee of $F_{taker} \times fee_{u_{taker}} \times s \times (t_{end} - t)$ would be charged from u's collateral to Fee Treasury.
- State change 3.3.3 (Create a new pair of swaps) will be triggered between u and the orderbook.
- Then, a series of state changes 3.3.7 will be triggered to fill the orders in the orderbook to fulfill the market order. The logic for which orders to fill is outside the concern of the model in this paper, but it will be filled based on (fixed rate, time opened) priority.
- If an order $order_x$ of size s_x and fixed rate f_x is partially filled for a size of s_{filled} as a result of this market order:
  - The original order will be "removed" in a state change 3.3.8 (new size will be 0)
  - A new order $order_{x_1}$ of size s_{filled} and fixed rate f_x is created for u in a state change 3.3.6. The time opened for this new order is set to be the same as the time opened for the original order.
  - A new order $order_{x_2}$ of size $s_x - s_{filled}$ and fixed rate f_x is opened for u in a state change 3.3.6. The time opened for this new order is set to be the same as the time opened for the original order.
  - $order_{x_1}$ is filled in a state change 3.3.7

Let $r_{lastTraded}$ as the rate of the last state change 3.3.7 (Filling an open order), we have a restriction on the max deviation between $r_{lastTraded}$ and the current mark rate:

$$|r_m - r_{lastTraded}| \leq maxRateDeviationFactor \times max(I_{threshold}, |r_m|)$$

where $maxRateDeviationFactor$ is a market specific config.

3.3.11 Execute a batch of orders

At any time t, for any market M_i, a set of orders can be executed, which will trigger one or more of the following state changes:

- OTC swap
- A market order
- Opening a new limit order

Note: we denote $r_{highestBid}$ as the highest bid price of any limit order being opened in this batch. $r_{lowestAsk}$ is the lowest ask price of any limit order being opened in this batch.

After all the changes are done:

- If $onlyClosingCheck = True$ (this is a market-specific setting)
  - The **only closing orders check** must satisfy
  - or the user is whitelisted $isExemptFromOnlyClosingCheck(u) = True$. This whitelist is meant for market makers to still be able to make orders when we turn on closing only mode. This mode is meant to be turned on when we are near the Hard OI Cap.
- Either the **strict IM requirement** must satisfy, or the **Margin check for closing orders** must satisfy

Strict IM requirement

- The total initial margin must not exceed user's total value

$$im_u^{total}(t) \leq v_u(t)$$

- The user's highest bid and lowest ask of any limit order being opened in this batch must be within a bound from the mark rate

$$r_{highestBid} < f^u(r_m)$$
$$r_{lowestAsk} > f^l(r_m)$$

  The exact function for f^u and f^l is in Appendix E

Margin check for closing orders are meant for situation where the user wants to close their position, either through limit, market or OTC orders (and doesn't necessarily pass strict IM requirement)

- The **only closing orders check** must satisfy.
- The worst rate for any closing order must be within a bound from the mark rate:

$$(r_m - r_{worst}) \times sign(s(t)) \leq max(I_{threshold}, |r_m|) \times closingOrderBound$$

  where r_{worst} is $r_{highestBid}$ if $s(t) < 0$, or $r_{lowestAsk}$ if $s(t) > 0$.
- The change in value and maintenance margin due to the batch must satisfy a condition:

$$v_u(t-) - v_u(t) \leq criticalHealthRatio \times (mm_u^{total}(t-) - mm_u^{total}(t))$$

  where $criticalHealthRatio$ is a market-specific setting. This check makes sure that if $healthRatio(t-) > criticalHealthRatio$, health ratio will always improve after the batch.
- If there was at least one OTC or market order, and $strictHealthCheck = True$ (this is a market-specific setting) for any of the market the user is in, we require that the health ratio after must be greater than the critical health threshold

$$healthRatio(t) > criticalHealthRatio$$

  Note: this check is to ensure that when a user's health ratio is already below the critical level (which violates the above check's assumption), they can't do OTC or market orders to worsen it. This check is relatively more expensive and always pass in normal market conditions, so we will only turn $strictHealthCheck$ on when necessary.

Only closing orders check

- The size of the position must be reduced (or stays the same)

$$|s(t)| \leq |s(t-)| \tag{8}$$
$$sign(s(t)) \times sign(s(t-)) \geq 0 \tag{9}$$

- The total book size for the same side of the position must stay exactly the same. This means that no more limit orders are allowed to open more position

  if $s > 0$

$$b^+(t) = b^+(t-)$$

  if $s < 0$

$$b^-(t) = b^+(t-)$$

  Note: in our contract implementation, we simply require the set of new limit orders on the same side is empty.

- The total book size for the opposite side of the position must not exceed the position size. if $s > 0$

$$b^-(t) \leq |s(t)|$$

  if $s < 0$

$$b^+(t) \leq |s(t)|$$

Market entrance fees When the user u does the very first action through this state change in this market, we charge a fixed $marketEntranceFees$ directly from the user's collateral to the treasury. This is a global setting per asset zone.

$$isFirstBatch = true$$
$$c_u(t) = c_u(t-) - marketEntranceFees$$

3.3.12 Settle filled orders

All filled orders of user u in a market M_i can be settled by anyone in one go, realizing the implicit position from the orderbooks into the user's collateral and merging the implicit position with the standalone position.

The filled orders will be settled in batches, grouped by the epoches where the orders were filled, in chronological orders.

For each of the filled order is in the positive orderbook $order^+_{u_i}$: (it will be the same for the negative orderbook)

$$order^+_{u_i} \in X^+_{filled_u}(t-)$$
$$X^+_{filled}(t) = X^+_{filled}(t-) \setminus \{order^+_{u_i}\}$$
$$X^+_{filled_u}(t) = X^+_{filled_u}(t-) \setminus \{order^+_{u_i}\}$$
$$X_{settled}(t)^+ = X^+_{settled}(t-) \cup \{order^+_{u_i}\}$$
$$X_{settled_u}(t)^+ = X^+_{settled_u}(t-) \cup \{order^+_{u_i}\}$$
$$l_u(t) = l_u(t-) - s^+_{u_i}$$

The combined implicit position $(s_u^{book}(t), debt_u^{book})$ will be combined with the stand alone position and the realised PnL from the combined implicit position will be realised:

$$(s_{alone_u}(t), debt_{alone_u}(t)) = mergeSwap(s_{alone_u}(t-), debt_{alone_u}(t-), s_u^{book}(t-), debt_u^{book}(t-)$$
$$c_u(t) = c_u(t-) + p_u^{book}(t-)$$

3.3.13 Forced deleverage

A forced deleverage can be triggered by permissioned admin accounts to deleverage two users, one "winner" u_{winner} and one "loser" u_{loser}, by an amount s_{de}

$$healthRatio_{winner} > healthRatio_{loser}$$
$$s_{winner} \times s_{loser} < 0$$
$$s_{de} \times s_{winner} < 0$$

Note: in our implementation, we implement these three checks at an external contract (that calls the core's deleverage function).

A forced deleverage is essentially a forced swap of size s_{de} at the **deleverage rate** $r_{deleverage}$ between u_{winner} and u_{loser} following a state change 3.3.3. This swap will always reduce their absolute position sizes.

The deleverage rate depends on whether the loser is already at bad debt:

- If the loser is already at bad debt, where $v_{loser} < 0$, we deleverage at the bankruptcy rate (of the loser)

  $$r_{deleverage} = r_{bankrupt} = r_{mark} - \frac{v_{loser} \times \alpha}{s_{loser} \times t}$$

  where $0 \leq \alpha \leq 1$ is an input to the deleverage function, indicating "how much this market is at fault for causing the bad debt". In our operation, when running the deleverage function, we will:

  - Set $\alpha = 1$ if the market is deemed to be 100% the cause for this user's bad debt (based on offchain logic).
  - Set $\alpha = 0$ for markets which we decide (in an offchain way) that didn't contribute to the bad debt, which leads to $r_{deleverage} = r_{mark}$
  - Set α at some other values if this market is deemed to partially contribute to the bad debt **Note:** In practice, we will use a contract to aggregate the force deleverage function calls for a user. The tentative specs is in Appendix F.

- If the loser is not at bad debt, where $v_{loser} \geq 0$, we deleverage at the mark rate

  $$r_{deleverage} = r_{mark}$$

This state change is intended for system admins to reduce risks in the whole system whenever there are excessive risks due to a lack of liquidity or too much leverage in the system.

3.3.14 Liquidation

At any time t, if a user u_i is not **healthy**, any user u_l can **liquidate** a portion $l \leq 1$ of u_i's position in any market M_j.

$$healthRatio_{u_i}(t-) \leq 1$$

Liquidation position size $s_{liquidation} = l \times s_{u_i}$

Before a liquidation happens, the user implicit positions must have been settled first.

$$s_u^{book}(t-) = 0$$

A liquidation involves 5 steps:

- Step 1: all open orders by the liquidated user are cancelled, as a series of state change 3.3.7 (to remove the orders).
- Step 2: Opening a pair of swap of size $s_{liquidation}$ between u_l and u_i at market rate $r_m(t)$, following the state change in 3.3.3. This is very similar to a forced deleverage, although u_l will not have to have any existing position.
- Step 3: A liquidation incentives $v_{liqIncentive}$ is transferred from u_i's collateral to u_l's collateral.
- Step 4: A protocol liquidation fee is transferred from the liquidator to the protocol treasury

  $$liqFees = F_{liq} \times |s_{liquidation}| \times t$$

  where F_{liq} is a market specific setting.
- Step 5: Make sure that after the liquidation, the liquidator satisfies either the Strict IM requirement or the Margin check for closing order (mentioned in 3.3.11). Note that the liquidator doesn't need to pass the Soft OI Cap check.

Liquidation incentive $v_{liqIncentive}$ is basically a factor of the change in maintenance margin of the liquidated user.

$$v_{liqIncentive} = \min(k(u_i, t), healthRatio_{u_i}(t-)) \times (mm_{u_i}^{total}(t-) - mm_{u_i}^{total}(t))$$

Liquidation incentive curve k is the default factor before the position becomes so unhealthy that all the factor will be exactly the health ratio, meaning all the collateral left will be proportionally used to cover for liquidation incentives (and if all user positions are liquidated the user will be left with 0 collateral). K will generally starts small when the health ratio is still close to 1, and will increase slowly as the position becomes more and more unhealthy. Refer to Appendix D for the function that will be used in practice for k.

$$c_{u_i}(t) = c_{u_i}(t-) - v_{liqIncentive}$$
$$c_{u_l}(t) = c_{u_l}(t-) + v_{liqIncentive}$$

3.3.15 Public canceling of limit orders for at-risk users

If a user's health ratio is below a certain threshold $noOrderHealthRatio$, an admin account can call a function to cancel all of the user's orders in the collateral zone.

$noOrderHealthRatio$ is a global setting of the asset zone.

Note that if $noOrderHealthRatio = 1$, this is already implicitly achieved since anyone can liquidate a single wei of the user's position to cancel all of their open orders.

3.3.16 Admin canceling of all out-of-bound orders

An admin can cancel all orders that fail the **bound on limit order prices** in Appendix E.

3.3.17 Admin canceling of particular orders

An admin account could cancel any particular order that could pose a threat to the system's risk (for example a super big order that could be filled that will trigger a super big liquidation)

3.3.18 Adjust a setting

At any time t, an admin account can adjust one or more of the settings.

3.3.19 Pause/unpause market

The market is by default in an unpaused state.

When the market is unpaused

- All state changes can happen
- An admin account can pause the system

When the market is paused

- An admin account can unpause the system
- All state changes are not allowed, except for:
  - Forced deleverage (3.3.13)
  - Public canceling of limit orders for at-risk users (3.3.15)
  - Admin canceling of all out-of-bound orders (3.3.16)
  - Admin canceling of particular orders (3.3.17)
  - Adjust a setting (3.3.18)
  - Unpause (3.3.19)

3.3.20 Ban/unban a user

A user is by default in an unbanned state.

When the user is unbanned

- All state changes regarding the user are allowed
- An admin account can ban the user.

When the user is banned

- An admin account can unban the user
- The user can't make any state change

3.3.21 Initiate/cancel/complete a withdrawal request

By default, $c_{withdrawalTotal} = 0$ for all users.

Initiate a withdrawal request At any time t, a user u can initiate a withdrawal request to withdraw $c_{withdraw}$ of cash from their collateral, as long as they satisfy the Strict IM requirement after the withdrawal:

$$\begin{aligned} c_u(t) &= c_u(t-) - c_{withdraw} \\ im_u^{total}(t) &\leq v_u(t) \\ c_{withdrawTotal}(t) &= c_{withdrawTotal}(t-) + c_{withdraw} \\ t_{withdraw} &= t \end{aligned}$$

Note: implications for repeated withdrawal requests is that $t_{withdraw}$ reset to the current time, and the total withdrawal amount increases by the amount being newly requested.

Cancel a withdrawal request A user can cancel their withdrawal request:

$$\begin{aligned} c_u(t) &= c_u(t-) + c_{withdrawTotal} \\ c_{withdrawTotal} &= 0 \end{aligned}$$

Complete a withdrawal After at least a cooldown period, the user will be able to withdraw their total withdrawal request amount $c_{withdrawTotal}$.

$$\begin{aligned} t &\geq t_{withdraw} + t_{cooldown} \\ c_{withdrawTotal} &= 0 \\ c_{userWallet}(t) &= c_{userWallet}(t-) + c_{withdrawalTotal} \end{aligned}$$

where $t_{cooldown}$ will follow a global setting by default, or a personal setting if that is set by admins for the particular user.

Appendices

A Maintenance margin function

In this section we will describe the specific maintenance margin function MM(s, u, t, r) that will be used in practice for each market.

At any time t, there are a few settings for the market regarding maintenance margin rate (that can be changed over time by governance)

- $I_{threshold}$ is the APR below which we will use a fixed margin requirement instead of a ratio of Mark APR. Note that this threshold applies for both maintenance margin and initial margin
- k_{MM} is the ratio of the Mark APR that we use as maintenance margin.
- $t_{threshold}$ is a time duration before maturity that margin requirements will stop going down over time. This is applicable for both Initial Margin and Maintenance Margin.

Each user u also has a **personal maintenance margin factor** $mm_u^{personal}(t)$ at time t, which is default to 1 and could be set by governance.

This Maintenance margin function is then:

- if $t < t_{threshold} \times k_{MM} \times mm_u^{personal}$ and $s \times r > 0$ and $|r| > I_{threshold}$:

$$MM(s, u, t, r) = s \times t \times r + |s| \times I_{threshold} \times (t_{threshold} \times k_{MM} \times mm_u^{personal} - t) \tag{10}$$

- Otherwise:

$$MM(s,u,t,r) = |s| \times \max(t, t_{threshold}) \times k_{MM} \times \max(I_{threshold}, |r|) \times mm_u^{personal} \tag{11}$$

B Pre-scaling Initial Margin function

In this section we will describe the pre-scaling initial margin function PIM(s, r, t), which is the Initial Margin before scaling with time, margin factor and personal margin factor.

At any time t, there is a market-specific setting $I_{threshold}$ which is the APR below which we will use a fixed margin requirement instead of a ratio of Mark APR. Note that this threshold applies for both maintenance margin and initial margin for the market regarding initial margin (that can be changed over time by governance)

The Prescaling initial margin function is then:

$$PIM(s,r,t) = |s| \times \max(I_{threshold}(t), |r|) \tag{12}$$

C Initial Margin function

In this section we will describe the specific initial margin function IM(pim, u, t) that will be used in practice for each market.

At any time t, there are a few settings for the market regarding initial margin (that can be changed over time by governance)

- $I_{threshold}$ which was described in the previous section.
- k_{IM} is the ratio of the Mark APR that we use as initial margin.
- $t_{threshold}$ is a time duration before maturity that margin requirements will stop going down over time. This is applicable for both Initial Margin and Maintenance Margin.

Each user u also has a **personal initial margin factors** $im_u^{personal}(t)$ at time t, which is default to 1 and could be set by governance.

The Initial margin function basically scales the pre-scaling initial margin with the margin factor, personal margin factor and time (but will fix t at $t_{threshold}$ when it reaches there.)

$$IM(pim,u,t) = pim \times k_{IM} \times im_u^{personal}(t) \times \max(t, t_{threshold}) \tag{13}$$

D Liquidation Incentive Factor

For any moment t, there are two settings that can be changed over time by governance for each market:

- Liquidation factor base $LiqFactor_{base}(t)$
- Liquidation factor slope $LiqFactor_{slope}(t)$

The Liquidation Incentive Factor function is just:

$$k(u,t) = LiqFactor_{base}(t) + LiqFactor_{slope}(1 - healthRatio_u(t))$$

E Bounds on limit order prices

When placing an order, a long order price must not exceed f^u and a short order must not be lower than f^l:

$$f^u(r_m) = \begin{cases} r_m \times upperLimitSlope & r_m \geq I_{threshold} \\ r_m + upperLimitConstant & 0 \leq r_m < I_{threshold} \\ -f^l(-r_m) & r_m < 0 \end{cases}$$

$$f^l(r_m) = \begin{cases} r_m \times lowerLimitSlope & r_m \geq I_{threshold} \\ r_m + lowerLimitConstant & 0 \leq r_m < I_{threshold} \\ -f^u(-r_m) & r_m < 0 \end{cases}$$

Note that there are 4 global configs per market here: $upperLimitSlope$, $upperLimitConstant$, $lowerLimitSlope$ and $lowerLimitConstant$

F Deleverager contract

In practice, we use a contract to aggregate the calls to force deleverage a user. **This contract could have its logic upgraded over time, and it shouldn't affect the risk assumptions for the protocol at all** (it is purely for the convenience of operation). This section describes a tentative mechanism we will use at launch.

An aggregated deleverage action here will have the following inputs:

- The losing user
- α_i for each market being deleveraged in this function $\alpha_1, \alpha_2, ...\alpha_n$.

$$\sum \alpha_i = 1$$

- For each market i, list of the winners and proportional size to be deleveraged with the winner $(winner_{i_1}, p_{i_1})$, $(winner_{i_2}, p_{i_2})$,..., such that sum of all p_{i_j} for this market i is 1.

Then, we go through each market i, and deleverage the loser with each of the $winner_{i_j}$, with size and alpha as such:

$$\begin{aligned} p_{done} &= p_{i_1} + ... + p_{i_{j-1}} \\ s_{de} &= s_{loser} \times p_{i_j} \\ \alpha &= \frac{\alpha_i(1 - p_{done})}{\alpha_i(1 - p_{done}) + \alpha_{i+1} + ... + \alpha_n} \end{aligned}$$

G Max deviation factor on funding rate oracle

To prevent malicious/wrong funding rate updates, we introduce a max deviation factor $MaxFRDeviationFactor$, and require that

$$\frac{|r_m - r_{oracle}|}{max(|r_m|, I_{threshold})} < MaxFRDeviationFactor$$

where r_{oracle} is the rate reported by the oracle for the settlement period.