Ajna Master Spec

Pool level attributes

- Price book is divided into buckets
    - The formula to generate the price for a bucket index shall be $1.005^{n-n'}$ where *n* is the index of the bucket being calculated, and *n'* is the median bucket index (3232).
    - 7388 buckets shall be provided and shall include the number 1, with 3231 to the left representing prices < 1, and 4156 buckets representing prices > 1.

Fenwick index	Bucket index	Price
1	4155	999969141.896623000000000000
1077	3079	4669863.090887930000000000
2154	2002	21699.795273866500000000
3232	924	100.332368143273000000
4156	0	1.000000000000000000
4166	-10	0.951347940696070000
4310	-154	0.463902261297398000
5260	-1104	0.004061325713752660
5388	-1232	0.002144924036174740
6466	-2310	0.000009917388865692
6647	-2491	0.000004021056399147
7108	-2952	0.000000403446260869
7388	-3232	0.000000099836282891

    - Buckets shall offer a minimum price around one millionth and a maximum price around one billion. (1,004,968,987.61) The formula above may be adjusted to accomplish this.
- Target utilization
    - The debt-weighted average collateralization ratio (DWAC) of loans in a pool is used to determine the Target Utilization TU. If D_i and C_i are the debt and collateral pledged for loan i, b is the borrower inflator, and T_i is the t0-equivalent debt, then DWAC is:

$$DWAC = \frac{\sum_i D_i \cdot \frac{D_i}{lup \cdot C_i}}{\sum_i D_i} = \frac{\sum_i b \cdot T_i \cdot \frac{b \cdot T_i}{lup \cdot C_i}}{\sum_i b \cdot T_i} = \frac{b \cdot \sum_i \cdot \frac{T_i^2}{C_i}}{lup \cdot \sum_i T_i}$$

    - Precisely, TU is the ratio of the 3.5 day exponentially weighted moving average (EMA) of the numerator of DWAC to the denominator of DWAC in this equation sampled every time there is interaction with the pool
    - The lambda used for the EMAs is $\lambda = exp\left(-\frac{\Delta t}{3.5\ days} ln\ 2\right)$, so that the EMA update equation is $EMA_{t+\Delta t}(x) = \lambda \cdot EMA_t(x) + (1 - \lambda) \cdot x_t$ (here, x_t could be either time series used in the definition of TU).
- Meaningful Actual utilization (MAU)
    - Define "Relevant Deposit" to be the amount of deposit above the debt-weighted average threshold price DWATP less the debt in auction. This is the average threshold price of all loans in the pool, where the loans are weighted by their debt.
    - Meaningful Actual Utilization is the the ratio of the 12-hour EMA of the total pool debt to the 12-hour EMA of relevant deposit
    - As above, this is updated using an EMA formula, with the relative weights of the new sample and previous EMA being a function of the elapsed time since the last update. Here, the lambda is $\lambda = exp\left(-\frac{\Delta t}{12\ hours} ln\ 2\right)$
- Highest Price Bucket (HPB)
    - The highest-priced bucket which contains deposit (not inclusive of claimable collateral).
- Interest rate
- Lowest Utilized Price (LUP)
- Total Encumbered Collateral
    - Debt / LUP
- Pool Collateralization
    - Total Available Collateral / Total Encumbered Collateral = (Total Collateral)*LUP/Debt
- Pool Threshold Price (PTP)
    - Total Debt / Total Collateral Posted
    - If no collateral has been deposited, the PTP is defined as 0.
- Highest Threshold Price (HTP)
    - Maintained in a heap which orders loans by collateralization
    - Threshold price of a particular loan is the debt/collateral
- LiquidationDebt is the total amount of debt currently being liquidated

Lender inputs:

- Price

- Quote token amount

Borrower inputs:
- Collateral amount
- Debt to withdraw (denominated in quote token)

Pool accounting:
- All deposits and balances are accounted for using a modified fenwick tree coupled with a scaling tree. Also see slide show.
- t0Debt: Debt is stored as a "t0-equivalent" debt quantity.
    - t0-equivalent debt is the actual current debt owed by the borrower divided by the current value of the pool borrower inflator.
    - Another way of thinking about this: the t0-equivalent debt of current debt amount is the amount of debt that would have had to have been taken out at the beginning of the pool to equal the current amount of debt, including interest.
    - While debt increased due to interest, t0-equivalent debt does NOT change due to interest. So, working with t0-equivalent debt quantities is often much more convenient.
    - Other debt related quantities are sometimes also processed and stored in t0-equivalent versions. For example, loans are stored in a heap based on their t0-equivalent threshold prices, which is the t0-equivalent debt divided by the amount of collateral posted.
    - In the code, debt related variables that are denominated in t0 term start with the prefix t0 (exampled: t0Debt, t0Np, etc).
    - The system uses t0Debt to reduce rounding issues that can occur during arithmetic.

Liquidity management

- The pool has a fixed number of price buckets.
- Depositors/lenders may deposit quote token or collateral into buckets. The quote token becomes deposit in that bucket, while collateral becomes claimable collateral for that bucket. Note: this collateral is distinct from borrower collateral that is used to collateralize a loan. Both award the actor with an LP balance in the bucket, though interest only accumulates on the amount of quote token (in the form of deposit) in the bucket. An actor may redeem their LP balance for quote token or collateral (priced at the bucket price). As such:
    - Lenders who deposited quote token may choose to redeem their deposit plus interest earned as either quote token or (if available) collateral.
    - Arbitrageurs who deposit collateral may immediately withdraw quote token to arb overpriced deposits off the book. Symmetrically, arbitrageurs may deposit quote token in buckets with available collateral to arb underpriced collateral off the book.
    - Although collateral does not earn interest, actors who place collateral into a bucket take a share of the interest earned by lenders' quote token.

    - When actors deposit NFT collateral into a bucket, they are awarded a requisite amount of LP without retaining any affinity to specific NFTs deposited. As such, they may choose to immediately redeem their LPB for quote token or to trade for another available NFT in any bucket. For example, if you deposit 1 QT to the 1 bucket, you'll get LPB worth 1 unit of collateral, burn this to withdraw the NFT from the 1 bucket, deposit this NFT in the 100 bucket, get LPB worth 1 unit of collateral, and withdraw the NFT that was in the 100 bucket.

  LP balance may not be redeemed for quote token if:
    - Doing so would push the LUP (lowest utilizable price) below the highest threshold price (HTP). The lender may liquidate loans with threshold prices between the current and desired LUP to facilitate such a withdrawal.
    - Buckets with prices greater than the bucket returned by the fenwick method findsum(liquidationDebt) are locked. In other words, all deposits within liquidation debt from the top-of-book down are frozen.
- The exchange rate (for a given bucket) shall be calculated as: $\frac{deposit + price_{bucket} \cdot collateral_{claimable}}{lpbalance_{outstanding}}$ If there is no outstanding LP balance, a 1-to-1 exchange rate is used
- Bucket bankruptcy
    - occurs on a withdraw of deposit or claimable collateral from a bucket if are zero and there is outstanding LP balance for the bucket when the withdrawal is made, the previous LP balances are forfeited. This can happen when there is bad debt, as the scaling factor may be 0.
    - May not add deposit or claimable collateral to a bucket that has been bankrupted in the same block
- Upon removing liquidity from a deposited bucket, a lender's LP position amount shall be decremented to reflect their current participation in the bucket.
- Users *may* mint an NFT of their position but are not required to do so, reducing gas costs.
- There is a global accumulator that tracks the total value of claimable collateral across all buckets in quote token terms, valued at the price of the bucket. This is used to compute the reserve.
- For the NFT case, there is a set of claimable NFTs maintained. These are the NFTs that may be claimed by LPB holders that were put into the book's claimable collateral by either auctions/liquidations or LP holders adding collateral to a bucket. These are disjoint from the posted collateral in the loans themselves. The sum of claimable collateral across all buckets should equal the number of NFTs in this set.

Borrowing

- Each borrower corresponds to a unique loan in a pool. Borrowers have two associated quantities:

    - Posted collateral, which are the assets they have deposited into the contract to collateralize their borrowing activity.
    - Debt, which is the amount of quote token they owe to the pool
- Any address can deposit collateral to any loan/borrower, and can repay the debt of any loan/borrower.
- As interest accrues, debt shall increase, causing more collateral to become encumbered.
- Borrowers must maintain their collateralization relative to the LUP.
- Each loan has a threshold price, which is defined as the loan's debt divided by the loan collateral
- All loans are kept in a sorted list with the HTP (Highest Threshold Price) loan, i.e. the least collateralised, the riskiest loan at the top.
- In NFT pools, specific NFTs pledged by the borrower retain their affinity to the borrower unless the borrower is liquidated.
- In addition to a mapping of borrower address to Borrower struct, borrower's also have data about their loan stored in a max heap. The max heap is updated to maintain that the head remains the highest threshold price. This head is maintained so we can restrict Lenders from freely withdrawing their deposit if the LUP falls below the HTP (highest threshold price).
- The Loans heap is a Max Heap data structure (complete binary tree), the root node is the loan with the highest threshold price (TP) at a given time. The heap is represented as an array, where the first element is a dummy element (Loan(address(0), 0)) and the first value of the heap starts at index 1, ROOT_INDEX. The threshold price of a loan's parent is always greater than or equal to the threshold price of the loan. Ajna's code was modified from the following source: https://github.com/zmitton/eth-heap/blob/master/contracts/Heap.sol . Some rules about the max heap:
    - A child can have a TP equal to but never greater than its parent
    - The head of the heap will always have the highest TP price

Flashloans

Quote token may be flashloaned up to the contract balance in two ways:

- **Implicitly:** Borrower may pledge collateral, borrow quote token, repay loan, and pull collateral in a single transaction using multicall or their own contract. In this scenario, borrower is charged the origination fee (discussed in *Manipulation Mitigations* below).
- **Explicitly:** To avoid collateral requirements and save gas, borrower may use the EIP-3156 interface for performing a flashloan. In this scenario, borrower is charged no fee.

Borrowers may flashloan ERC-20 collateral up to the contract balance using the EIP-3156 interface. Borrower is charged no fee for the flashloan. NFT collateral may not be flashloaned; EIP-3156 is only compatible with ERC-20 tokens.

Claiming NFT Collateral Across Multiple Buckets

MergeOrRemoveCollateral allows a lender to merge LPB in multiple price buckets to reconstitute an integral number of NFTs and withdraw.

1. User calls it with a list of bucket indices from which to remove collateral, number of NFTs to withdraw and the index to merge all of the collateral at, toIndex.
2. It then loops over the bucket indices provided by caller exiting if noOfNFTsToRemove is reached or max index is reached.
3. It then calls removeMaxCollateral removing the maximum amount of collateral that the user can remove from each index up until noOfNFTToRemove is reached.
4. After looping over all buckets if noOfNFTToRemove is not hit, the collateral is inserted into the merge index, toIndex.
5. If noOfNFTToRemove is hit, then transfer the NFTs to the caller.

Manipulation Mitigations

Penalties and Fees

Lenders

Unutilized Deposit Fee

Lenders are subject to a small penalty (24h of interest) for making a deposit below the LUP. This penalty will be calculated as 24 hours of interest. This penalty is imposed to mitigate against MEV attacks where a lender provides useless liquidity that cannot ever be borrowed.

Borrowers

Origination Fee

When borrowing quote token, a borrower is subject to an origination fee which is immediately added to the borrower's debt. The origination fee is calculated as the greater of the current annualized interest rate divided by 52 (one week of interest) or 5 bps.

Minimum Borrow Size

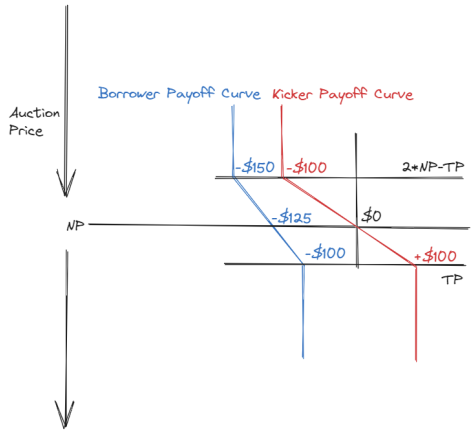
The minimum borrow will be 10% of the average loan size

- (Debt / Total Number of Loans) * 0.1
- This is to mitigate an attack where dust loans are opened to prop up the HTP and lock in depositors.
- This check is enforced only when there are 10 or more existing loans.

Take Penalty

If take is called above the NP of the loan, a take_penalty is assessed on the borrower and applied to the amount of debt cleared by the take, the penalty is calculated as:

Penalty = take_amount * (bond_factor * 5/4 - ¼ * bond_payoff_factor)



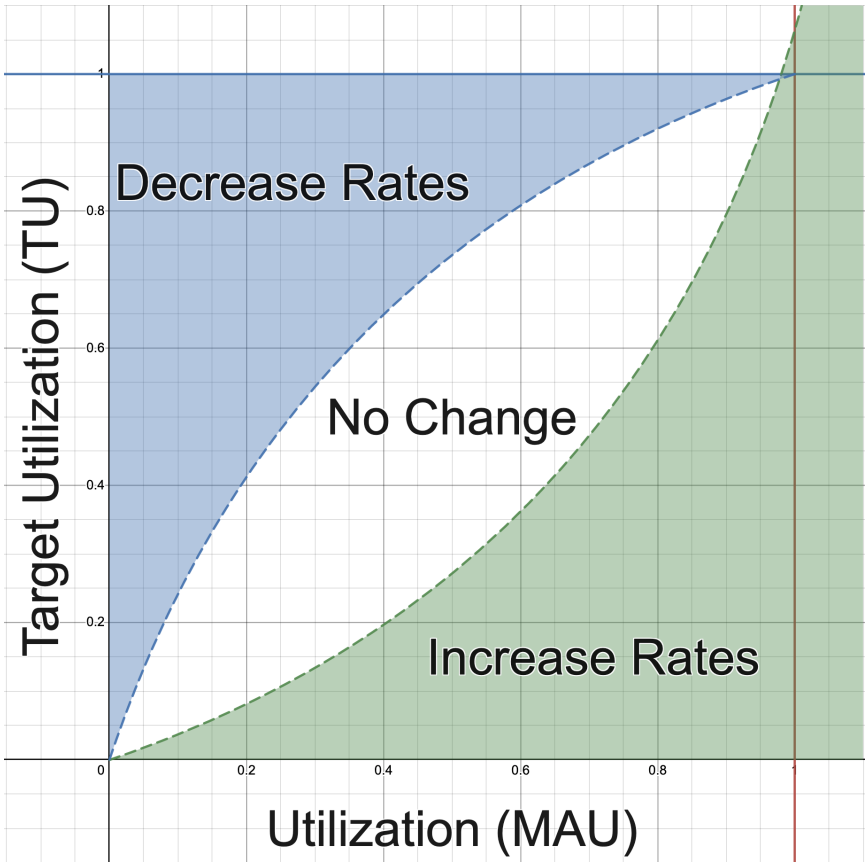
Interest rates

- Interest rate is initialized by the pool creator, at a rate between 1% and 10%.
- Rates have a minimum of 0.1% and a maximum of 400%
- Interest is applied to all debt as encumbered collateral and is denominated in quote token at the same rate.
- The proportion of interest rate paid on deposit above the PTP which goes to the reserve is calculated using the following formula:
  - $((1 - MAU)\text{^}(1/3)) * 0.15$
  - MAU = Meaningful Actual Utilization
  - Examples of net interest margin (NIM) at different MAU can be found below

Avg(MAU,TU)	Raw Spread	NIM (% of borrow rate)	Spread w/ NIM	To Lenders (% of borrow):	NIM Spread vs Raw Spread
99%	1.00%	3.23%	4.20%	95.80%	3.20%
95%	5.00%	5.53%	10.25%	89.75%	5.25%
90%	10.00%	6.96%	16.27%	83.73%	6.27%
80%	20.00%	8.77%	27.02%	72.98%	7.02%
70%	30.00%	10.04%	37.03%	62.97%	7.03%
60%	40.00%	11.05%	46.63%	53.37%	6.63%

50%	50.00%	11.91%	55.95%	44.05%	5.95%
40%	60.00%	12.65%	65.06%	34.94%	5.06%
30%	70.00%	13.32%	74.00%	26.00%	4.00%
20%	80.00%	13.92%	82.78%	17.22%	2.78%
10%	90.00%	14.48%	91.45%	8.55%	1.45%
1%	99.00%	14.95%	99.15%	0.85%	0.15%

- The Interest Rate *R* can be changed when *current utilization* and *target utilization* are in a certain relationship.
    - If the TU is higher than the MAU plus the “no change” window, the system will decrease rates.
    - If the MAU is higher than the TU plus the “no change” window, the system will increase rates.



To determine which region of the chart we are in with target utilization as *tu* and meaningful actual utilization as *mau*:
If

$$4(tu - 1.02 \cdot mau) < (tu + 1.02 \cdot mau - 1)^2 - 1$$

Then we raise rates. Else, if

$$4(tu - mau) > 1 - (tu + mau - 1)^2$$

Then we decrease rates.

- The rate changes in +- 10% increments - the current interest rate itself is multiplied by 1.1 or 0.9
- A check will be performed to update the rate implicitly when a deposit, remove and move function is called. It can only be updated 12-hours after the last update.
    - The above formula determines the pool state in which the rate will either be increased or decreased if an update is possible
- The check to alter interest rates occurs during a lend, borrow, repay, purchase, liquidate, move, withdraw quote and withdraw collateral call
- Interest is applied by amending the scaling tree to impact all buckets above the lesser of the HTP and the LUP
- Meaningful actual utilization is defined in section Pool Level Attributes

Tokenomics

- Origination fees, deposit penalties and a net interest margin are accumulated in Reserves

Reserves

- Origination fees, deposit penalties and a net interest margin are accumulated in Reserves

$$reserves = debt + balance_{contract} - deposit - \sum_{l \in Liquidations} bond(l) - balance_{CRA}$$

Liquidation

- Loans that are not fully collateralized with respect to the *lup* are eligible for liquidation.
    - `collateral * LUP < debt`
- Anyone can call `kick()` on a loan that is under-collateralized to send it into liquidation. This will require posting a bond. If the liquidation recovered much more than the debt, a portion (or all) of the bond will be forfeited. Otherwise, the caller of `kick()` will win some of the auction proceeds as well. Once a loan is kicked it cannot be re-collateralized, its debt must be extinguished via the liquidation auction

- When a loan enters liquidation and part of the debt is addressed (e.g. a partial take), debtInAuction is incremented or decremented as appropriate. debtInAuction is accounted for as a T0 equivalent debt number.
- Proceeds from a penalized liquidation bond (i.e. a bond which incurs a penalty) accumulate in the reserves.
- The `take() and bucketTake()` methods can be called by anyone to buy collateral from the loan at an auction price determined by the system.
- For each take, the borrower is punished by take(amount) * (bond_factor * 5/4 - ¼ * bond_payoff_factor), this also accumulates in the reserves
- The initial auction price is 256*reference_price. The reference_price = max(HTP, NP). The auction price decays exponentially from there.
- Auction price decay happens as follows: Begins with 6 twenty minute halvings, followed by 6 two hour halvings, followed by one hour halvings till the end of the 72 hour auction.
- Quote token raised from the liquidation process pays back debt and potentially rewards the kicker via their liquidation bond.
- Anyone can force deposit on the book to be used in a bucketTake() call on the auction. All deposit used as a bid must be in the same bucket. The price of the bucket must be greater than or equal to the auction price. Collateral taken by bucketTake is moved into the “claimable collateral” account for the bucket, and decrements the deposit in the bucket by the amount of the bid (as determined by the bucket’s price).
- An atomic swap mechanism allows bidders to access on-chain liquidity when calling a take for the collateral.
- While the technical possibility of triggering a liquidation occurs when the LUP is below the TP of the loan, actors are not economically incentivized to liquidate the loan until the market price is below the NP or if 256*reference_price (starting Auction price) is below the market price, where reference_price= max(HTP, NP).

Auction interface and methods

Loans that are not fully collateralized with respect to the LUP (meaning: `collateral_posted * LUP < debt`) are eligible for liquidation. In the case when the LUP moves down via user action, the withdrawing lender is restricted from withdrawing if this forces the LUP > HTP. To circumvent this an actor to kick blocking loans, then may call removeQuoteToken() if they wish to complete the withdrawal.

Each liquidation has a timestamp member variable `kickTime` which contains the time at which liquidation was commenced on the loan. When a loan is first made, it is automatically fully collateralized. Thus, it’s in a non-liquidating state, so kickTime is set to 0.

kick(borrower) / lenderKick(depositIndex)

Liquidation is begun by calling the `kick()` method. Anyone can send a transaction to call the `kick()` method on a loan if they post a liquidation bond. A liquidation bond can be posted using external quote token. There is a related function, lenderKick(), which allows a lender to kick loans that would be undercollateralized with respect to what the LUP would be if the deposit

were withdrawn. These methods will perform a check to see if the loan is eligible for liquidation (using criteria mentioned above), and that the loan is not already in liquidation. If both of these are satisfied, it will begin the liquidation process by:

- Setting `kickTime` to the current time
- Setting `neutralPrice` to the TP times the `NPtoTPratio` of the loan
- Remove the loan from the global loan heap
- Set the bondFactor used to determine the BPF (bond penalty factor)
- Add the loan to an auction queue ordered by `kickTime`

In order to execute a kick:

1. Call `kick()`, which checks if `TP > LUP` (this forces the taker to recheck that the loan is indeed eligible for liquidation, and reverts if not). In the case of lenderKick(), the LUP is computed using the quote token available to the lender to withdraw from the bucket – this computes what the LUP would drop to if that amount of quote token were withdrawn.
2. Check that current time is greater than an hour from `kickTime` and that `kickTime` is not 0. If not, revert.

When debt enters liquidation, debtInAuction is incremented by the amount of debt being liquidated.

However, if the loan is fully collateralized, the kick reverts.

While the loan is in auction, it is removed from the loans heap, not affecting the HTP. Any actor can repay the loan in whole or part, or post additional collateral, which might make the loan solvent.

The `take` method enables anyone to buy off portions of the collateral at an auction price that decreases over time. The collateral is bought using quote token, deposit, or a combination thereof. Details on this below.

The price of collateral in a liquidation is determined by the elapsed time since the start of the auction and the liquidation reference price by the following formula:

The **reference_price is max(HTP, NP)**.
The **auction starting price** is **256*reference_price**.
The auction **halves every 20 min for 6 halvings**, then **halves 120 min for 6 halvings**, and then goes to **60 min halvings for the rest of the auction**.

take(borrower, maxCollateral, who, data)

The *borrower* argument identifies the loan to take collateral from. The *maxCollateral* argument specifies an upper bound of the amount of collateral the taker wishes to purchase at the auction price. When *who* and *data* are specified, these arguments are used for atomically swapping collateral for quote token using external liquidity.

`take` operates as follows:

3. Let TPF be the Take Penalty Factor
4. TPF = $Take\ Penalty\ Factor\ = \left(\frac{5}{4} \cdot\ BondFactor\ - \frac{1}{4} \cdot\ BPF\right)$
5. Fungible case: The caller buys C = min(collateral_posted, qty /(min(1,1-BPF) * p), debt / (p*(1-TPF))). quantity of collateral from the loan. The loan's `collateral_posted` is decremented appropriately.
    a. If BPF > 0, then BPF*C*p units of quote token are awarded to the liquidation bond poster, and the balance of (1-BPF)*C*p is used to pay off the debt of the loan as usual
    b. If BPF < 0, then BPF*C*p units of quote token are forfeited by the liquidation bond to the contract balance ( ), and all of the the quote token used in the take is used to pay off the debt
6. NFT case: The caller must always purchase an integral number of NFTs. The purchaser may buy up to the smallest number of NFTs that would fully satisfy the debt and liquidation bond payment.
    a. The amount of collateral purchased is `C0:=min(collateral_posted, floor(qty/price), ceiling(debt/(1.0-TPF)*price))).`
    b. Let `C:=min(collateral_posted, floor(qty/price), debt/(1.0-TPF)*price)).` This is the amount of collateral that the bond payment is based on (this does not round up the debt to the nearest integer).
    c. If BPF > 0, then BPF*C*p units of quote token are awarded to the liquidation bond poster, and the balance is used to pay off the debt of the loan as usual
    d. If BPF < 0, then -BPF*C*p units of quote token are forfeited by the liquidation bond to the contract balance (hence increasing reserves), and all of the the quote token used in the take is used to pay off the debt
    e. There may be leftover quote token in this case after paying off the bond and the debt. This quote token is returned to the borrower
7. If the caller sent in excess quote token with the transaction (for example, if they tried to buy more collateral than was actually available), then only the quote token that could be supported by the collateral purchase of the requested size is withdrawn from their account

depositTake(loan, bucket)

The *borrower* argument identifies the loan to liquidate. *BucketIndex* identifies a price bucket above the current auction price with deposit. Note the caller of this method [depositTake] does not receive anything directly from the call. The caller may be someone with a deposit in the bucket (specified in the arguments) or may be the kicker who wants LPB in a bucket.

1. Check that loan is not being taken in the same block it was kicked.
2. Set `auction_price`
3. Check that qty:= min(deposit_in_bucket, auction_debt)
4. Check that the p:=price of the bucket is less than or equal to the liquidation price
5. Perform the take action:

    a. We compute the amount of profit that goes to liquidation bond per unit of debt (see BPF from the liquidation bond section)
    b. Set C = min(collateral_posted, qty /(min(1,1-BPF) * p), debt / (p*(1-TPF))).
    c. The bucket's deposit is decremented by the amount (1-BPF)*C*p
    d. C units of collateral are moved into the bucket's claimable collateral from the loan
    e. If BPF > 0, the liquidation bond is awarded LPB in the bucket worth BPF*C*p quote token
    f. If BPF < 0, BPF*C*p units of quote token are retained by the contract from the posted bond (which means, they increase the pool reserve)
    g. The debt of the loan is reduced by (1-TPF)*C*p
6. In the NFT case, after an auction is complete, the book will have an integral amount of collateral claimable. This is the same check run take(). Any user can then reconstruct the entire NFT by depositing enough quote token and using the MergeOrRemoveCollateral functionality described in Claiming NFT Collateral Across Multiple Buckets

arbTake(loan, bucket)

If the auction price is below the highest book price, a taker can "arbitrage" the top of the book out by buying the collateral in the auction and depositing the collateral to the top of the book. The taker would then be able to withdraw collateral with no net cost. This operation will reduce the debt by the amount of deposit utilized, and therefore will not move the LUP downward. The arbitrageur moves collateral into the bucket at a higher price that was paid. The original quote token could be withdrawn to offset the original quote token used to purchase the collateral, with the result being that the arb_taker receives "free" LP balance in the bucket. To ensure that this can be accomplished with no net quote token required, the bondholder also receives their payment in the form of LPB.

We provide an `arb_take` function that combines these three operations into one call. This provides a natural way to incentivize pushing auction collateral to the top of the book once the auction price falls below the top of the book. Although it usually makes sense to call this on the HPB, the contracts shall allow the caller to choose the bucket to arb out.

1. Call `liquidate(loan)` or check if `TP > LUP` to perform collateralization check (this forces the taker to recheck that the loan is indeed eligible for liquidation, and removes it from liquidation if not).
2. Set p = `auction_price`
3. Let q be the price of the bucket, and verify that p<=q
4. qty:= deposit in bucket q
5. Set C = min(collateral_posted, qty / (p*min(1,1-BPF)), debt / (p*(1-TPF))). This is the amount of collateral purchased.
6. Then:
    a. C units of collateral are moved from the loan to the bucket's claimable collateral

b. min(1,1-BPF)*C*p units of quote token are removed from the bucket's deposit. The loan's debt is reduced by this same amount
c. The taker is awarded C*(q-p) worth (in quote token terms) units of LPB in the bucket
d. If BPF >0, then the bondholder/kicker is awarded BPF*C*p worth of LPB in the bucket
e. If BPF<0, then the bond forfeits -BPF*C*p units of quote token to the pool

The following tables explain these three cases in more detail. They all assume that BPF>0, which is the more challenging case. If BPF<0, the appropriate quantity of quote token is forfeited from the bond, and the interactions between the borrower, taker and bondholder are more straightforward.

Method	Description	Q	C	Actor	Debt/Deposit/QT	Collateral	LPB (in QT)
Take	Buy Using QT	Quote Token	$\min\left\{\begin{matrix}C_{loan}\\C_{take}\\D_{loan}/(p\cdot(1-TPF))\end{matrix}\right\}$	Borrower	$+(1-TPF)\cdot C\cdot p$	$-C$	
				Taker	$-C\cdot p$	$+C$	
				Bond	$+BPF\cdot C\cdot p$		
Deposit Take	Buy Using Deposit	$Deposit(p)$	$\min\left\{\begin{matrix}C_{loan}\\Q/(p\cdot(1-BPF))\\D_{loan}/(p\cdot(1-TPF))\end{matrix}\right\}$	Borrower	$+(1-TPF)\cdot C\cdot p$	$-C$	
				Bucket p	$-(1-BPF)\cdot C\cdot p$	$+C$	Mint $BPF\cdot C\cdot p$
				Bond			$+BPF\cdot C\cdot p$
Arb Take	Arb Deposit in bucket $q>p$	$\frac{Deposit(q)}{1-BPF}$	$\min\left\{\begin{matrix}C_{loan}\\Q/(p\cdot(1-BPF))\\D_{loan}/(p\cdot(1-TPF))\end{matrix}\right\}$	Borrower	$+(1-TPF)\cdot C\cdot p$	$-C$	
				Taker			$+C\cdot(q-p)$
				Bond			$+BPF\cdot C\cdot p$
				Bucket q	$-(1-BPF)\cdot C\cdot p$	$+C$	Mint $C\cdot q-(1-BPF)\cdot C\cdot p$

settle(bucket_depth)

1. Auctions are enqueued upon liquidation, producing a collection naturally sorted by liquidation time.
2. Auctions are cleared from the auction queue, regardless of order, as they are completed. The liquidator is incentivized to settle their auction to release their liquidation bond from escrow. An auction is considered "completed" when either
    a. Collateral remaining to be auction is 0
    b. 72 hours have passed since the auction was kicked
3. When a loan is settled, the first step is to release the awards/remaining bond to the bondholder. This also means, in the case where the bond is posted in the form of deposit from buckets, adding the remaining deposit back into the relevant bucket(s), and crediting the bondholder the appropriate amount of LPB. Note: because LPB cannot be redeemed while there is a clearable auction, the bondholder would not be able to redeem that interest until clear has finished.
4. A loop iterates starting at HPB moving downward, bucket_depth times. Each iteration of this loop entails the following:

a. If loan.debt > 0 and loan.collateral > 0 and 72hrs has passed since auction was kicked:
    i. Finds HPB
    ii. Define clearable_debt := min (loan.debt,HPB.deposit, loan.collateral *HPB.price)
    iii. Decrement loan.debt and HPB.deposit clearable_debt
    iv. Decrement loan collateral by clearable_debt /HPB.price
    v. Increment HPB.claimable_collateral by clearable_debt/HPB.price
b. If loan.debt > 0 and loan.collateral == 0 and reserves > 0:
    i. Reduce loan.debt by min(loan.debt, reserves) (this implicitly also reduces reserves by the same amount)
    ii. Reserves which have been auctioned as part of a Claimable Reserve Auction, but were not taken in that auction, will no longer be available for covering liquidation debt.
c. If loan.debt > 0 and loan.collateral == 0 and reserve == 0:
    i. Finds HPB
    ii. Define forgive_amount := min(loan.debt,HPB.deposit)
    iii. Decrement loan.debt by forgive_amount (also global debt accumulators)
    iv. Decrement HPB.deposit by forgive_amount
    v. If HPB.deposit == 0 and HPB.claimable_collateral == 0, then existing LPB and LP tokens for the bucket shall become unclaimable.
d. If loan.debt == 0:
    i. Return any remaining collateral to borrower
    ii. Remove the auction/loan from the queue (i.e. finalize clearing the auction)
    iii. In the case of NFTs, the remaining collateral is returned to the borrower via calling “NFT Auction Remove” above, with price_bucket being the lowest priced bucket in the book.

5. If the auction at the head of the auction queue is “completed” but not yet “cleared”, no LPB can be redeemed by any lender. Likewise, the liquidation bond cannot be claimed by the kicker of the auction until the auction is cleared.

Auction Examples

Liquidation Bonds: Mechanics and Economics

Overview

Users that wish to initiate liquidations (“kickers”) must post a bond priced as a percentage of the debt of the loan that they are liquidating. If the liquidation auction yields a value that is over the “Neutral Price” (NP), the kicker forfeits a portion or all of their bond. If the liquidation auction yields a value below the NP, the kicker will have their bond returned plus a reward.

The goal of this mechanism is to disincentivize people for kicking off spurious auctions that involve gaming the system. The bonds should provide an economic guarantee to borrowers that they will only have their positions liquidated if the value of their collateral has *legitimately* fallen to a rate which endangers the ability of lenders to recover their funds.

Calculating the Threshold Price

Each loan in Ajna has a threshold price, which is defined as the loan's debt divided by the loan collateral.

$$Threshold\ Price\ = \frac{Debt_{Loan}}{Pledged\ Collateral_{Loan}}$$

In order to be eligible for liquidation, the *LUP must be below the loan's threshold price.*

Determining the Neutral Price

The Neutral Price (NP) of the auction is the price at which a kicker will have their bond returned to them without penalty or reward. If the collateral is purchased above the NP then some of the bond will be lost by the kicker, resulting in a penalty. Conversely, if collateral is purchased below the NP, then the kicker will receive an award on top of the bond.

When a loan is initiated, when collateral is removed from a loan, or when additional debt is drawn on a loan,, the loan is stamped with the following:

$$NP_t = (1.04 + \frac{\sqrt{rate_s}}{2}) \cdot TP_s \cdot \frac{BI_t}{BI_s}$$

where $rate_s$ is the annual interest rate of the pool. The s in the subscript denotes the time that the loan was initiated or last modified causing an update to the NP stamp. The actual neutral price for the loan is this price increased for the interest accumulated since the loan was stamped (implemented by dividing the stamp by the the inflator at the time of the the stamp, BI_s, then multiplying by the current borrower inflator BI_t when being used). Because the events that trigger a re-stamping of the loan's neutral price also require an accumulation of debt, we can use the existing borrower inflator to account for this increase due to interest. The loan can be re-stamped by the borrower any time they want.

We can define $1.04 + \frac{\sqrt{rate_s}}{2}$ as the *NP/TP Ratio*.

For each “take” function that is called in the auction above the neutral price, the kicker will lose a proportionate amount of their bond. For each “take” function that is called in the auction below the neutral price, the kicker will gain a proportionate reward on their bond.

Sizing the Bond

The size of the bond that must be posted by liquidation kickers is described below

$$BondFactor \;=\; min\{3\%,\; \frac{NP/TP\ Ratio - 1}{10}\}$$

$$Bond\ Size\ (value\ in\ QT) \;=\; BondFactor \times Debt_{Loan}$$

Determining the Reward (or Penalty)

The reward (or penalty) given to a kicker after the auction has yielded an amount below (or above) the clearing amount is proportionate to undercollateralization of the specific loan relative to the proceeds of the auction. The reward (or penalty), a percentage of the bond value, is calculated once again by taking the difference between the take price and the Neutral Price as a proportion of the difference between the Threshold Price price and the Neutral Price. The following equations govern the reward (or penalty) function:

$$BPF \;=\; BondFactor \cdot min\left\{1, max\left\{-\ 1, \frac{NP - Price_{take}}{NP - TP}\right\}\right\}$$

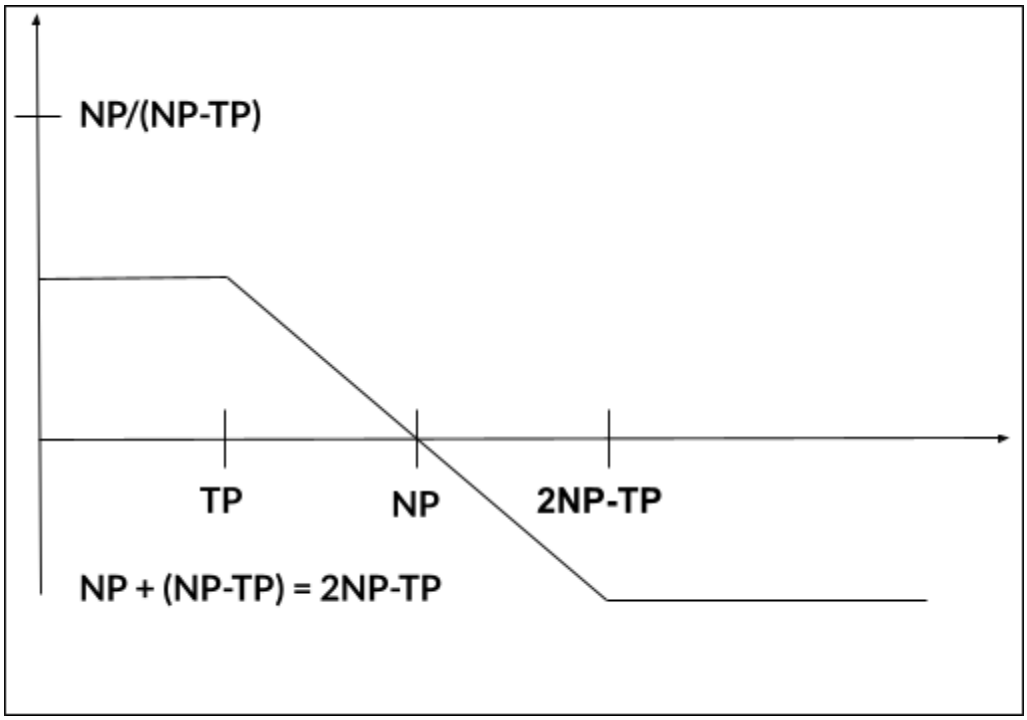
*Note: QT_Take is the same as (C * p) in the take() functions, there are many variables to consider when calculating this, the one used here is just an example*

During the auction, the *TP* is updated after each *take* to account for the new state of the debt and collateral.

More detailed example:

1. A loan with 50,000 QT in debt is eligible for liquidation (TP above LUP)
2. A “kicker” wishes to call liquidate and must calculate the appropriate liquidation bond
3. Threshold Price (loan) = $110,NP/TP Ratio = 1.136, Neutral Price (loan) = $125, Take Price = $130
4. The size of the bond that the kicker must post is the “Bond Factor” multiplied by the debt of the loan being liquidated

5. Bond Factor = Min(3%, (1.136-1)/10)) = 1.36%
6. Bond Size = 1.36% * Debt (50,000 QT) = 680 QT
7. The auction now begins and eventually clears for a price of $130 with the entire debt being extinguished (i.e. QT_{Take} is 50,000)
    - For purposes of example, a single buyer called take() inserting 50,000 QT in exchange for 384.61… units of collateral
    - They had posted 454.54… units of collateral for the loan
    - The remaining 69.93… units of collateral is returned to the borrower
8. Since the TakePrice of the auction is above the neutral price of the loan, the kicker must be penalized for kicking this liquidation
9. The BPF is Bond Factor [1.36%%] * min(1,(max(-1, ((NP [125]-TakePrice [130])/(NP [125]-TP [110])))) = -0.45%
10. Penalty = BPF [-.45%] * QT_{Take} [50,000] = -0.45% * 50,000 = -226.67 QT
    - This is a 33.33% loss on the kicker's bond



Bond Refund or Forfeiture

Kickers who improperly trigger a liquidation are penalized by forfeiting some or all of their liquidation bond.

Forfeited liquidation bonds accumulate in reserves.

Ajna Token Burn and Buy

Overview: Claimable excess reserves of Ajna pools are auctioned off using a reverse Dutch auction. In a Claimable Reserve Auction (CRA), bidders may purchase these excess reserves for Ajna tokens, which are burned.

- Claimable reserves of a pool are defined as (0.995 * Debt) + PoolBalanceQT - Deposit - (sum of liquidation bond escrows).
- Reserves accumulate through origination fees, deposit penalties, forfeited liquidation bonds and net interest margin.
- Any time at which there is no CRA active, any participant can kick off a new CRA with no other preconditions or requirements. The initial supply of Ajna tokens is to be 1,000,000,000
- Once the auction is kicked, the price to buy quote token per Ajna token is set to 1,000,000,000. The price decays exponentially with a half life of 1 hour.

  $price = 1,000,000,000 \times (\frac{1}{2})^{\#\ hours\ elapsed}$
- Ajna tokens used to purchase reserves in a CRA are burned.
- If 72 hours have passed, new claimable reserves are added to the amount in the expired auction, and start a new auction with both after the minimum period of time between auctions has elapsed.
- There is a minimum two week period between CRAs.

Grant Coordination

As a decentralized protocol with no external governance, Ajna requires a mechanism to coordinate ecosystem growth. Through this mechanism we seek to create a healthy cycle of protocol use and AJNA value accrual in a scalable and decentralized way. The growth coordination mechanism is premised upon the idea that the AJNA token has value; this is achieved through the buy and burn mechanism. Upon launch, the Ajna community has been given a treasury of AJNA tokens which represent 30% of the overall supply. This treasury will emit tokens through the Primary Funding Mechanism, *PFM*,. Through this mechanism, AJNA tokenholders can coordinate in a decentralized manner to fund useful activity for the ecosystem.

Additionally, Ajna provides voting delegation and delegate rewards. A token holder may delegate to themselves if they wish to capture this reward. In the primary funding mechanism 10% of the quarterly distribution is awarded to delegates. While delegation rewards are based on the total number of votes in the Funding Round, the delegate will have had to participate in both the Screening and Funding Rounds to be eligible for rewards. For example, if the quarterly distribution is 10M tokens, 1M tokens will be awarded to delegates and the remaining 9M will be distributed via the PFM. The award for a given delegate is proportional to the square root of the sum of the squares of their votes. If the delegate uses all of their voting power, then this will be the square root of their voting power.

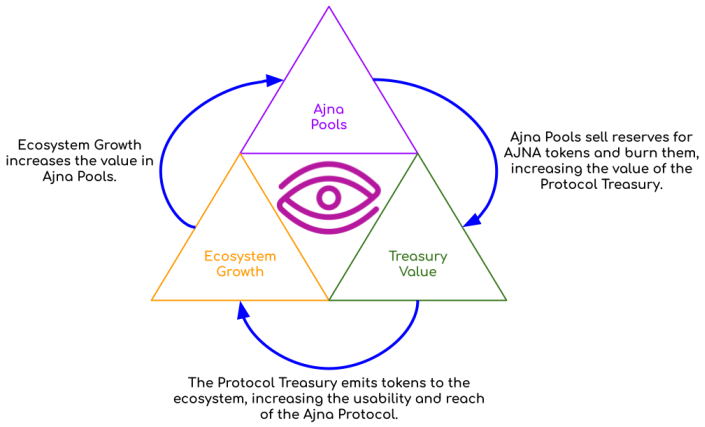


Image illustrating the intended virtuous cycle of Ajna's growth coordination mechanism

Funding Mechanism (FM)

On a quarterly basis, a portion of the treasury is disseminated to projects to facilitate growth of the Ajna system. Projects submit proposals (with a budget) and are voted on by Ajna token holders. The result of the voting is binary: either the proposal wins and is funded with the requested tokens, or fails and receives nothing. The overall set of approved proposals is decided upon by maximizing the number of votes for them, subject to an overall budgetary constraint. The voting system has two stages: a screening stage, in which the list of possible winning projects is culled down to 10 candidates using a simple 1-token-1-vote method, and a quadratic voting stage, in which the actual winning projects are determined. Voting power in each stage is determined by two snapshots, 34 blocks prior to stage start, and 1 block prior to stage start. Transfer of tokens out of the account (or delegation power transfers) at any point during the snapshot period results in the loss of that voting power.

The Grant Fund progresses through each stage of a quarterly distribution based upon the current block height in relation to the distribution period's start block. Once the hardcoded number of blocks has passed, it will automatically progress to the next stage. A new distribution period can only be started after the last period's end block has been reached. Each distribution period must be started manually.

The details are below

1. Each quarter (90 day cycle), up to 3% of the treasury can be distributed to projects that win a competitive bidding process. This is the global budgetary constraint, **GBC**.
2. Any project can submit a **proposal**. A proposal consists of a **budget** together with the address to send the tokens to. A budget is simply a quantity of Ajna tokens requested.
3. To avoid an overwhelming number of proposals, the slate of projects is filtered down to a manageable number during a *screening stage*. The screening stage lasts for the first 73 days of the cycle. Every address holding Ajna tokens can use them to support any

number of proposals, with each token supporting at most one project. At the end of the screening stage, the 10 proposals with the most support are deemed eligible to be funded.

4. These 10 projects are then voted upon in the *funding stage*. In the funding stage, Ajna token holders vote on the proposals using a quadratic system. Each address that holds Ajna tokens can vote as much as they want on every proposal, including negative votes, subject to the constraint that the sum of the squares of their votes cannot exceed the square of the number of Ajna tokens held. Note that this quadratic system is not vulnerable to sybil attacks, as splitting tokens across multiple addresses results in a corresponding decrease in the "vote budget" of the holder of the tokens.
5. Once the voting period ends, the winning proposals are decided. The slate of winning proposals maximizes the sum of the net number of votes cast in favor of the proposals, subject to the sum of the winning proposals' budgets must not exceed the GBC.
    a. Solving this optimization problem is not done on-chain. Instead, there is a one week period during which the contract considers a candidate slate of winners. The initial candidate slate of winners is empty: all proposals fail. During this week, anyone can submit a proposed better candidate slate. If the proposed slate has more net votes in favor of its members than the current best candidate, and still satisfies the global budgetary constraint, then this becomes the new candidate slate. The slate at the end of this one week period becomes the official slate of winners, and the tokens are distributed.

Non-Core Contracts

Position Manager Contract

This contract allows lenders to optionally mint and manage a single or multiple Position NFTs that represent either a portion of, or their entire position in a given pool. Lenders can transfer this NFT to other addresses, or stake it in contracts such as the Rewards Manager contract described below. Whoever owns a position NFT owns the underlying liquidity.

burn

Burn a Position NFT. It can only be called by the NFT's owner. All underlying liquidity must be removed from the represented positions before this method can be called.

memorializePositions

Given an array of bucket indexes and a NFT tokenId, this method will stamp the NFT with those underlying liquidity positions and transfer that LPB to the PositionManager contract. Prior to calling this method, the LPB owner must first call `approveLpOwnership` on the desired pool to enable the PositionManager to escrow the LPB. The NFT owner must own the underlying LP in the provided bucket indexes.

mint

Mint a new Position NFT. It can only be called for pools that have been deployed by an Ajna pool factory. A user can have multiple NFT positions in the same pool.

moveLiquidity

This method allows the owner of a Position NFT to move the underlying LPB between different buckets in the associated pool. It can only be called by the owner of the NFT, and LPB can only be moved within the same pool.

redeemPositions

This method will remove LPB from the PositionManager contract and transfer it to the NFT owner.

tokenURI

Part of the ERC721 spec, this view method returns a Base64 encoded JSON object that includes metadata about the NFT as well as SVG art that can be viewed in wallets or NFT marketplaces.

Rewards Manager Contract

Context: There is a desire to reward lender's in Ajna with $AJNA tokens. This contract allows lenders to be compensated based upon the relative amount of interest they accrued in the pool during the period of time between Collateral Reserve Auctions (an "epoch"). The amount of $AJNA awarded to the pool during these epochs is based upon the total amount of $AJNA that was burned in the Collateral Reserve Auction.

Reserve auctions and epochs are mapped one to one. Each time a new `startReserveAuction()` call occurs a new epoch is created (with the burnEvent.totalBurned initialized to zero). Each `takeReserves()` call sets burnEvent.totalBurned to the cumulative amount of Ajna token burned (including previous auctions).

NOTE: Epochs could have a totalBurned value of zero as a reserve auction could not receive any takes, thus not stamping the burnEvent.totalBurned value required to calculate rewards

earned for that epoch. Alternatively the auction price could be so low that 0 Ajna are provided in a valid take call.

Rewards are accrued over epoch periods proportional to a staker's interest earned in a period, normalized by the amount of Ajna tokens burned in the pool in a period. Anyone may update the bucket exchange rates used to determine interest earned in exchange for an update reward of 5% of the rewards earned in a bucket. This is done by calling the updateBucketExchangeRate method and specifying a list of buckets to update. If the bucket was already updated in an epoch, then no reward is provided to the updater. Furthermore, if an update is not performed within two weeks of the burn epoch for that bucket rewards will not be claimable for that epoch period for that specific bucket.

If epochs are properly associated with interest earned, and bucket exchange rates are updated, then a staker can return to the rewards contract and claim their reward for those buckets at any point in the future.

Functionality:

1. User deposits their LPB NFT into the contract.
2. Contract tracks interest earned by the bucket of the depositor.
3. When a reserve auction is triggered and the exchange rates have been updated for the respective staked buckets, AJNA becomes claimable from the staking contract proportionate to the amount of interest the lender's bucket accrued, with a cap of the minimum of x% of the amount of AJNA tokens burned in the reserve auction, or y% of the staking contract balance.
    a. For example, if a pool has burned 100 AJNA, it cannot receive more than 80 AJNA as rewards. Also, in an effort to taper off the distributions and not run out, there is a maximum amount of the rewards contract that any pool can take. For example, even if the pool would get 100 AJNA tokens, if that's over the predetermined threshold, they would be bumped down to a lower number.
4. Anyone who calculates the AJNA rewards owed to a bucket may automatically claim 5% of the rewards

Detail:

1. Notation:
    a. B(i) is the total Ajna burned in burn event i (i=0,1,2,…)
    b. I(i) is total pool interest earned between burn event i-1 and i (i=1,2..3..)
    c. R(i,k) = exchange rate in bucket k "at the time" of burn event i. (note, we can't always have access to this exact value, so below we give a method to get it approximately)
    d. C is the claim constant (e.g. 0.5)
    e. E is the exchange rate reward constant (e.g. 0.05)

2. Then: someone who has staked an NFT with L LPB in bucket k for the entire time between burns i-1 and i can claim C * L * B(i) * [R(i,k) - R(i-1,k)] / I(i)
3. If someone staked during the interval between burns i-1 and i, the equation is the same, except replace R(i-1,k) with the exchange rate of the bucket at the time of staking
4. Additional overlay: once 0.8 * B(i) has been claimed for the burn in interval i-1 to i, no more tokens can be rewarded for that interval
5. How E and R(i,k) work: for a period of up to two weeks after burn i, anyone can act as a "recorder" (staker or not) can send an exchange rate recording transaction to the contract, which then records the current value of the exchange rate for a set of buckets into R(i,k). This recorder is awarded E*B(i)*Deposit(i,k) * (1-R(i-1,k)/R(i,k)) / I(i) Ajna tokens
    a. If an update is missed, no rewards will be earned for recording the new exchange rate of a bucket for the subsequent epoch.
    b. If R(i-1,k) > R(i,k) (i.e. the exchange rate actually declined), then the reward is 0.
6. No more than 0.1*B(i) can be awarded to the "recorders" in burn i
7. If no one records both R(i-1,k) and R(i,k), then no one can claim any tokens. Note, however, that R(i) gets recorded whenever someone stakes an ajna token (if it has not already been recorded).
8. R(i,k) can only be recorded once per burn — subsequent attempts to re-record it fail
9. Every time someone stakes, unstakes, or claims rewards they will also update the exchange rate of the bucket they are interacting with, with the current exchange rate of the pool at the time. The update reward would also be provided for these actions.

Arbitrary Variables

Name: Bucket_Spacing
- Purpose: Provide the most gas-efficient price distribution without hurting the user experience
- Summary: The percentage spacing between price buckets
- Current: 0.5%

Name: Rate_Change_Amount
- Purpose: To create a reactive but stable rate change environment in coordination with the Rate_Change_Frequency, must allow for a sufficient feedback loop
- Summary: The amount by which the interest rate can be increased or decreased when the pool is out of equilibrium
- Current: 10%

Name: Rate_Change_Frequency
- Purpose: To create a reactive but stable rate change environment in coordination with the Rate_Change_Amount must allow for a sufficient feedback loop

- Summary: The frequency at which the interest rates can be changed for a pool that is out of equilibrium
- Current: 12-hours

Name: Auction_Starting_Price

- Purpose: To create an auction where the starting auction price is with extremely high probability above the market price
- Summary: The price at which the reverse dutch auction for collateral will begin 256*reference_price, where reference_price=max(HTP, NP)

Name: Auction_Half_Life

- Purpose: To ensure that the auction is long enough to find bidders but not so long as to risk full recovery (an auction that is too short will not attract sufficient liquidity and an auction that is too long risks having the market price of the asset go below the Threshold Price)
- Summary: How often the reverse dutch auction halves in price
- Current: The auction **halves every 20 min for 6 halvings**, then **halves 120 min for 6 halvings**, and then goes to **60 min halvings for the rest of the auction**.

Name: Bond_Factor_Minimum

- Purpose: To avoid spurious liquidations by imposing a cost
- Summary: The minimum percentage of the debt being liquidated that must be posted as a liquidation bond
- The current minimum based on r=0.1% is 0.6581% is

Name: Deposit_Penalty_Period

- Purpose: To disincentivize interest rate manipulation via utilization manipulation
- Summary: The period of time where a lender will be charged a penalty for removing or moving a deposit (timer reset each time deposit is moved)
- Current: 13-hours (needs to be longer than Rate_Change_Frequency)

Name: Deposit_Penatly_Minimum

- Purpose: To disincentivize interest rate manipulation via utilization manipulation
- Summary: The percentage of a lender’s deposit charged as a penalty for removing or moving a deposit in the Deposit_Penalty_Period – this quote token goes to the BIP
- Current: 5bps

Name: Deposit_Penatly

- Purpose: To disincentivize interest rate manipulation via utilization manipulation
- Summary: The percentage of a lender’s deposit charged as a penalty for removing or moving a deposit in the Deposit_Penalty_Period – this quote token goes to the BIP

- Current: Borrower_Rate_APY / 52

Origination_Fee_Minimum
- Summary: The minimum interest paid upfront as an origination fee
- Current: 5bps

Origination_Fee
- Summary: The maximum interest paid upfront as an origination fee
- Current: Borrower_Rate_APY / 52

Loan_Size_Minimum
- Avoid dust balances causing unnecessary iteration when borrowing, repaying, taking, or reallocating
- Summary: The minimum size of a loan in quote token
- Current: (Total_Debt / Total_Loans) * 0.1

AU_TU_Rate_Relationship
- Purpose: To provide an environment in which interest rates can reach equilibrium with minimal volatility
- Summary: The relationship which determines if rates can be raised, lowered or must remain constant
- Current:
  - Raise if AU < (TU + ((1 - TU) / 2)
  - Lower if AU > (TU + ((1 - TU) / 2)
  - Note: ½ is an additional embedded arbitrary variable

Debt_Discount_Factor_Claimable_Reserves
- Purpose: To delay the accumulation of claimable reserves by discounting debt in the reserve equation: $reserves = debt + balance_{contract} - deposit$
- Current: 0.995

Claimable_Reserve_AJNA_Halflife
- Purpose: To find the correct market price of AJNA based in pool QT in a reverse dutch auction
- Current: 1-hour
-

Known Attacks

Dust loan defense

- Desc: A borrower takes out several dust loans across price buckets (creating an artificially high HTP) to prevent lenders from moving the LUP to the correct price
- Mitigation: Minimum borrow size

Interest rate manipulation

- Desc: Take a large loan, make a large deposit, or deposit a large amount of collateral impacting utilization metrics in order to manipulate the interest rate
- Mitigation: EMA of MAU prevents large manipulative deposits, origination fees prevent large manipulative loans, EMA of TU prevents large manipulative collateral deposits

Borrower griefing, Spurious liquidation

- Desc: A user liquidates a loan that will easily recover its debt in auction
- Mitigation: Neutral Price and liquidation bonds – Kickers must post a liquidation bond in order to trigger a liquidation; they will lose money if the auction clears above the neutral price

High priced self loan quote token theft

- Desc: A lender comes to a book with an existing deposit but no loans and creates a very high priced deposit and borrows it themselves, they then deposit a large amount of collateral but do not borrow against it and withdraw their deposit pushing the loan to the lower priced deposit but with a fraction of the collateral that should be pledged
- Mitigation: Linked list of HTPs and a lender cannot move the LUP below the current HTP

Kicker griefing

- Desc: Borrower overbids on own collateral to disincentive auction kickers, who then forfeit bond
- Mitigation: Have borrower penalized by new take penalty
  take_amount * (bond_factor * 5/4 - ¼ * bond_payoff_factor)

Removal of MOMP and change in Auction Starting Price

- Desc: The MOMP can be manipulated, so we removed the MOMP and went with reference_price = max(HTP, NP), but then the issue was if you just made the auction starting price 32*reference_price, borrower who was over collateralized by more than 32 would get their debt liquidated if 32*NP was less than the external market price which

can happen in single lender, single borrower situations. That is why we raised the multiplier to 256*reference_price. Now in a worst case single lender/ single borrower scenario if your debt is overcollateralized by 256x, and 256*NP is less than the external market price, your debt could get auto-liquidated, but the pain is much smaller because you could potentially lose 1/256 of your collateral or 40 bps of your collateral. Basically don't have dust loans or overly collateralized loans on Ajna, if there are multiple borrowers, this attack is harder to execute.

- Mitigation:
    - Because the auction starting price is so high now, we have a fast halving of 6 twenty minute halvings, followed by 6 two hour halvings, followed by one hour halvings till the end of the 72 hour auction
    - Changed from x32 to x256
    - Replaced MOMP with reference_price = max(HTP, NP)
- Heuristics for borrowers
    - Simple rule is borrowers should always be aware of their NP, if the external market price is below the borrower's NP, the loan will be kicked into auction,
    - If 256*NP is less than external market price, the kicker will kick their loan into auction