SmthCurve Bonding Curve Specification

Something Labs

October 27, 2025

Contents

Chapter 1

Formal Rationale and Motivation

1.1 Purpose

SmthCurve is a primary-market mechanism that sells a fixed token supply along a deterministic pricing rule while guaranteeing that the terminal marginal price equals the price of a secondary CPMM (e.g., a Uniswap-style pool). The design delivers continuous liquidity, predictable slippage, explicit fee capture, and a seamless hand-off into the secondary market.

1.2 Notation

We use the following notation throughout the specification:

- Q – total token supply minted at launch.
- T – tokens reserved to seed the AMM.
- $R = Q - T$ – tokens allocated to the bonding curve (inventory).
- (v_S, v_T) – virtual reserves (quote/base) that determine the bonding-curve trajectory.
- (r_S, r_T) – real reserves (ETH and token) held by the factory.
- $b \in [0, D)$ – trade fee in basis points with denominator $D = 10{,}000$.
- $m \in [0, 1)$ – migration fee in WAD precision (10^{18}).
- S_s – target net quote asset raised on the curve (after trade fee).
- S – quote asset paired with T when seeding the AMM.

1.3 Constant-Product Curve

Bonding-curve trades preserve the invariant

$$K = v_S \cdot v_T, \tag{1.1}$$

with instantaneous price $p \approx \frac{v_S}{v_T}$. To (i) sell R tokens for S_s net quote and (ii) end at the AMM price $\frac{S}{T}$, the initial virtual reserves satisfy

$$\frac{v_S + S_s}{v_T - R} = \frac{S}{T}, \qquad v_S v_T = (v_S + S_s)(v_T - R). \tag{1.2}$$

A solution exists when $SR > S_s T$ and yields

$$\boxed{v_T = \frac{RS_R}{S_R - S_s T}, \qquad v_S = S_s \frac{v_T - R}{R}.}$$

The terminal price $\frac{v_S + S_s}{v_T - R}$ equals $\frac{S}{T}$, ensuring price continuity at migration.

Chapter 2

Asset Computation and Quoting

2.1 Gross and Net Conversions

With trade fee b and denominator D:

- **Buy (gross to net):** $net = gross \cdot \frac{D-b}{D}$, $fee = gross - net$.
- **Buy (net to gross):** $gross = \left\lceil net \cdot \frac{D}{D-b} \right\rceil$.
- **Sell:** $fee = \left\lfloor \Delta S_{gross} \cdot \frac{b}{D} \right\rfloor$, $net = \Delta S_{gross} - fee$.

2.2 Rounding Discipline

- Ceiling operations on inverse CPMM steps avoid value leakage to traders.
- Buys are slightly conservative in favour of the protocol; sells are slightly conservative in favour of the user.

2.3 Buy Execution

Given net ETH $S_{net} > 0$:

$$S' = v_S + S_{net}, \tag{2.1}$$

$$T' = \frac{v_S v_T}{S'}, \tag{2.2}$$

$$\Delta T_{out} = v_T - T'. \tag{2.3}$$

If $\Delta T_{out} \leq r_T$ the order is fully filled, otherwise the protocol switches to the "buy-all-remaining" routine to consume the final inventory.

2.4 Sell Execution

For token amount $\Delta T > 0$:

$$T' = v_T + \Delta T, \tag{2.4}$$

$$S' = \left\lceil \frac{v_S v_T}{T'} \right\rceil, \tag{2.5}$$

$$\Delta S_{gross} = v_S - S'. \tag{2.6}$$

Net proceeds equal $\Delta S_{net} = \Delta S_{gross} - \left\lfloor \Delta S_{gross} \cdot \frac{b}{D} \right\rfloor$, subject to the solvency guard $r_S \geq \Delta S_{gross}$.

2.5 Quoting Helpers

- **Buy quote:** convert gross to net, attempt the full fill, and if necessary compute the minimum gross required to take the remaining inventory and refund the surplus.
- **Sell quote:** derive gross and net payouts and revert if the real ETH reserve is insufficient.

2.6 Migration Mechanics

Migrating $tokenToLP$ tokens at price $\frac{v_S}{v_T}$ requires

$$eth_{needed} = tokenToLP \cdot \frac{v_S}{v_T}, \qquad fee_{mig} = eth_{needed} \cdot m. \tag{2.7}$$

The factory enforces $r_S \geq eth_{needed} + fee_{mig}$; otherwise both quantities are scaled proportionally before proceeding.

Chapter 3

Implementation Highlights

3.1 Configurable Supply Split

Creators configure φ such that $T = \varphi Q$ and $R = (1 - \varphi)Q$, controlling how much supply is sold on the curve versus seeded into the AMM.

3.2 Capital Target

The parameter S_s specifies the net quote asset raised before migration, shaping the steepness of the bonding curve and the primary-market capital intake.

3.3 Price Continuity

Virtual reserves are set so that the terminal bonding-curve price equals the AMM seed price, eliminating migration arbitrage.

Chapter 4

External Interface Summary

Key Events

- SmthTokenFactory__TokenLaunch
- SmthTokenFactory__Buy
- SmthTokenFactory__Sell
- SmthTokenFactory__LiquidityMigrated
- SmthTokenFactory__CurveCompleted
- SmthTokenFactory__LiquidityLocked

4.1 Interface Outline

Listing 4.1: ISmthTokenFactory overview

```
1  interface ISmthTokenFactory is ISmthTokenErrors {
2      function totalSupply() external view returns (uint256);
3      function defaultBpsDenominator() external view returns
           (uint256);
4      function tokenDecimals() external view returns (uint8);
5      function isInitialized() external view returns (uint8);
6
7      function tokenInfo(address token) external view returns
           (TokenInfo memory);
8      function router() external view returns (address);
9      function factory() external view returns (address);
10     function WETH() external view returns (address);
```

```
11      function lockedLiquidity() external view returns
            (uint256);
12
13      function launchToken(
14          string memory name_,
15          string memory symbol_,
16          string memory uri_,
17          uint256 initialAmmEthAmount_,
18          uint256 initialRatio_,
19          address quoteAsset_,
20          FeePercent feePercent_
21      ) external payable returns (address tokenAddress);
22
23      function finalizeAndMigrate(address token_) external;
24      function buyToken(address token, uint256 minTokensOut,
            uint256 deadline) external payable;
25      function sellToken(address token, uint256 tokenAmount,
            uint256 minEthOut, uint256 deadline) external;
26      function sellTokenWithPermit(address token, uint256
            tokenAmount, uint256 minEthOut, uint256 deadline,
            bytes calldata signature) external;
27
28      function setLiquidityLocker(address locker) external;
29      function setRouter(address router_) external;
30      function sweepNative(address payable to, uint256
            amount) external;
31
32      receive() external payable;
33  }
```