RadixPlaza/StablePlaza - Mathematical Verification

octo

November 24, 2025

Abstract

This report documents a **(limited) mathematical verification** of the RadixPlaza AMM implementation against the CALM (Concentration-Asymmetric Liquidity Model) whitepaper. Due to time constraints, this is a **fairly quick audit focusing mostly on critical mathematical correctness**: the dual-curve mechanism, state transitions, and price reference frame handling. Quite some aspects were disregarded, and even covered items did not receive an extensive review. This should not be considered a comprehensive audit.

Repository: `https://github.com/OmegaSyndicate/RadixPlaza`
Commit: `8902049b17cb99d2e1662be6c3ffce475cdb30a1`
Whitepaper: Included in repository at `whitepaper/whitepaper.pdf`

1 Introduction

RadixPlaza is built in Scrypto, and implements a novel AMM design called CALM that uses two different curves for trades:

- **Incoming curve** (k_{in}): Denser liquidity for IL-reducing trades

- **Outgoing curve** (k_{out}): Sparser liquidity for IL-increasing trades

This audit focused on verifying mathematical correctness. Comparing the smart contracts to the CALM whitepaper especially.

2 Methodology

2.1 Review Process

1. Read whitepaper

2. Analyzed core implementation files:

- `src/pair.rs`

- `src/curves.rs`

- `src/types.rs`

3. Verified maths against whitepaper formulas

4. Ran and analyzed existing test suite

5. Added one integration test as a sanity check for full state cycle behavior

3 Mathematical Verification

3.1 Core Functions Verified

All six critical curve functions from `src/curves.rs` were verified against the CALM whitepaper.

3.1.1 calc_target_ratio - Equation 6

Whitepaper formula (Equation 6, Section 2.4):

$$\frac{B_0}{B} = \frac{2k - 1 + \sqrt{1 + 4k\frac{\Delta Q}{p_0 B}}}{2k}$$

Code (`curves.rs:26-35`):

```
pub fn calc_target_ratio(p0: Decimal, actual: Decimal,
                         surplus: Decimal, k: Decimal) -> Decimal {
    let radicand = ONE + FOUR * k * surplus / p0 / actual;
    let num = TWO * k - ONE + radicand.checked_sqrt().unwrap();
    num / k / TWO
}
```

Verification:

- radicand $= 1 + 4k \cdot \text{surplus}/(p_0 \cdot \text{actual})$
- num $= 2k - 1 + \sqrt{\text{radicand}}$
- Result $= (2k - 1 + \sqrt{...})/(2k)$

Verdict: Correct

3.1.2 calc_spot - Equation 1

Whitepaper formula (Equation 1, Section 1.2):

$$p_{\text{margin}} = \left(1 + k\left(\left(\frac{B_0}{B}\right)^2 - 1\right)\right) p_0$$

Code (`curves.rs:56-63`):

```
pub fn calc_spot(p0: Decimal, target_ratio: Decimal,
                 k: Decimal) -> Decimal {
    let ratio2 = target_ratio * target_ratio;
    (ONE + k * (ratio2 - ONE)) * p0
}
```

Verification: Directly implements $(1 + k(\text{target\_ratio}^2 - 1)) \cdot p_0$ where $\text{target\_ratio} = B_0/B$

Verdict: Correct

3.1.3 calc_p0_from_spot - Inverse of Equation 1

Derivation: Rearranging Equation 1:

$$p_{\text{spot}} = (1 + k(r^2 - 1))p_0 \implies p_0 = \frac{p_{\text{spot}}}{1 + k(r^2 - 1)}$$

Code (`curves.rs:87-94`):

```
pub fn calc_p0_from_spot(p_spot: Decimal, target_ratio: Decimal,
                         k: Decimal) -> Decimal {
    let ratio2 = target_ratio * target_ratio;
    p_spot / (ONE + k * (ratio2 - ONE))
}
```

Verdict: Correct

3.1.4 calc_p0_from_curve - Inverse of Equation 5

Derivation: From Equation 5, solving for p_0:

$$\Delta Q = \frac{\Delta B(B + k\Delta B)}{B}p_0 \implies p_0 = \frac{\Delta Q}{\Delta B} \cdot \frac{1}{1 + k\frac{\Delta B}{B}}$$

With $\Delta B = \text{shortfall}$, $B = \text{actual}$, $\Delta Q = \text{surplus}$, and $\frac{\Delta B}{B} = \text{target\_ratio} - 1$:

Code (`curves.rs:117-125`):

```
pub fn calc_p0_from_curve(shortfall: Decimal, surplus: Decimal,
                          target_ratio: Decimal, k: Decimal) -> Decimal {
    surplus / shortfall / (ONE + k * (target_ratio - ONE))
}
```

Verdict: Correct

3.1.5 calc_incoming - Equation 5 Integration

Whitepaper formula (Equation 5, Section 2.2):

$$\Delta Q = \frac{(B_0 - B)(B + k_{in}(B_0 - B))}{B}p_0$$

This is integrated from initial state to final state.

Code (`curves.rs:151-173`):

```
pub fn calc_incoming(input_amount: Decimal, target: Decimal,
                     actual: Decimal, p0: Decimal,
                     k_in: Decimal) -> Decimal {
    let actual_after = actual + input_amount;
    let surplus_before = (target - actual) * p0
                       * (ONE + k_in * (target - actual) / actual);
    let surplus_after = (target - actual_after) * p0
                      * (ONE + k_in * (target - actual_after) / actual_after);
    surplus_before - surplus_after
}
```

Verification:

- surplus_before $= (\text{target} - \text{actual}) \cdot p_0 \cdot (1 + k_{in} \cdot \frac{\text{target} - \text{actual}}{\text{actual}})$

- This simplifies to: $\frac{(B_0 - B)(B + k_{in}(B_0 - B))}{B} p_0$

- Matches Equation 5 exactly

Verdict: Correct

3.1.6 calc_outgoing - Equations 3 & 4

Whitepaper formulas:

Equation 3 (general case):

$$\Delta B = \frac{B_0 + \Delta Q' - \sqrt{B_0^2 + (4k_{out} - 2)B_0 \Delta Q' + \Delta Q'^2}}{2(1 - k_{out})}$$

Equation 4 (special case $k_{out} = 1$):

$$\Delta B = \frac{B_0 \Delta Q'}{B_0 + \Delta Q'}$$

where $\Delta Q' = \Delta Q / p_{\text{ref}}$.

Code (`curves.rs:201-243`):

```
pub fn calc_outgoing(input_amount: Decimal, target: Decimal,
                     actual: Decimal, p_ref: Decimal,
                     k_out: Decimal) -> Decimal {
    let shortfall = target - actual;
    let surplus = shortfall / actual * (actual + k_out * shortfall) * p_ref;
    let scaled_new_surplus = (surplus + input_amount) / p_ref;

    if k_out == ONE {
        // Equation 4
        let new_shortfall = scaled_new_surplus * target
                          / (target + scaled_new_surplus);
        new_shortfall - shortfall
    } else {
        // Equation 3
```

```
15        let new_shortfall = (
16            target + scaled_new_surplus -
17            (target * target
18             + (FOUR * k_out - TWO) * target * scaled_new_surplus
19             + scaled_new_surplus * scaled_new_surplus
20            ).checked_sqrt().unwrap()
21        ) / TWO / (ONE - k_out);
22        new_shortfall - shortfall
23    }
24 }
```

Verification:

- Equation 3 implementation matches whitepaper exactly

- Equation 4 special case for $k_{out} = 1$ is correct

- The k_{out} range constraint (pair.rs:85) prevents division by zero

Verdict: Correct

3.2 State Machine Verification

The AMM has three states defined in `src/types.rs`:

- `Equilibrium`: Balanced reserves, $r_{\text{target}} = 1$

- `QuoteShortage`: Quote pool is the shortage pool, uses incoming curve for quote input

- `BaseShortage`: Base pool is the shortage pool, uses incoming curve for base input

3.2.1 State Transitions

The state machine supports transitions between all three states. Pool selection logic (`pair.rs:642-653`):
```
1  fn select_pool(&self, state: &PairState, input_is_quote: bool)
2      -> (&Global<TwoResourcePool>, Decimal, bool) {
3      let p_ref = state.p0;
4      let p_ref_inv = ONE / p_ref;
5      match (state.shortage, input_is_quote) {
6          (Shortage::BaseShortage, true) => (&self.base_pool, p_ref, false),
7          (Shortage::BaseShortage, false) => (&self.base_pool, p_ref, true),
8          (Shortage::Equilibrium, true) => (&self.base_pool, p_ref, false),
9          (Shortage::Equilibrium, false) => (&self.quote_pool, p_ref_inv, false),
10         (Shortage::QuoteShortage, true) => (&self.quote_pool, p_ref_inv, true),
11         (Shortage::QuoteShortage, false) => (&self.quote_pool, p_ref_inv, false),
12     }
13 }
```

Key transitions:

1. **Equilibrium → QuoteShortage/BaseShortage**: Large input depletes one pool

2. **QuoteShortage/BaseShortage → Equilibrium**: Shortage pool input restores balance

3. **QuoteShortage ↔ BaseShortage**: Excess input crosses equilibrium to opposite shortage

Verification approach: Analyzed existing test suite (particularly `tests/pair_swap_incoming.rs` and `tests/pair_swap_outgoing.rs`) which test individual transitions with exact mathematical assertions. Additionally added an integration test (see Section 4.3) that verifies a complete bidirectional cycle through all states.

Verdict: Correct

3.3   Price Reference Frame Handling

The system uses two pools and switches between them during equilibrium crossings. `state.p0` is ALWAYS stored in base_pool frame (B[Q]), regardless of which pool is active.

select_pool handles this (pair.rs:642-653):

- base_pool: uses $p_{\mathrm{ref}} = $ state.p0 directly (B[Q] frame)

- quote_pool: uses $p_{\mathrm{ref}} = 1/$state.p0 (inverts to Q[B] frame)

Transition pattern (quote_pool → base_pool, pair.rs:476-489):

```
false => {
    new_state.last_out_spot = ONE / p_ref;
    new_state.p0 = ONE / p_ref;  // Temporarily inverted
    &self.base_pool
},
// ...
// Update running parameters for possible outgoing leg
p_ref = ONE / p_ref;  // Inverted again!
p_ref_ss = p_ref;
```

Verdict: Correct (maintains p0 in consistent reference frame throughout)

3.4   Rounding Strategy

The implementation uses asymmetric rounding to protect liquidity providers where necessary (`e.g.` `pair.rs:496-518`):

```
// Incoming phase
quote_base = output_amount.checked_round(
    self.base_divisibility,
    RoundingMode::ToZero        // Output rounded DOWN
).unwrap();
quote_quote = amount_traded.checked_round(
    self.quote_divisibility,
    RoundingMode::AwayFromZero  // Input rounded UP
).unwrap();

// Outgoing phase
base_base = amount_traded.checked_round(
    self.base_divisibility,
    RoundingMode::AwayFromZero  // Input rounded UP
```

```
15  ).unwrap();
16  base_quote = output_amount.checked_round(
17      self.quote_divisibility,
18      RoundingMode::ToZero        // Output rounded DOWN
19  ).unwrap();
```

- **Output amounts**: Rounded `ToZero` (down) - users receive slightly less

- **Input amounts**: Rounded `AwayFromZero` (up) - users must provide slightly more

Verdict: Correct, favors the pool over the user and prevents precision-based arbitrage exploits.

4 Testing

4.1 Existing Test Suite

The codebase includes a comprehensive test suite with exact mathematical verification. All existing tests pass.

4.2 Test Quality

The existing test suite is **excellent**. Tests use carefully chosen parameters ($k_{in} = 0.5$, $k_{out} = 1.0$) that produce round numbers, enabling exact equality assertions rather than approximate bounds. This provides strong mathematical verification.

All state transitions (Equilibrium $\leftrightarrow$ QuoteShortage $\leftrightarrow$ BaseShortage) are tested with exact mathematical verification.

4.3 State machine integration test

Added one integration test as a sanity check:
`tests/pair_state_cycle.rs::full_state_cycle_with_exact_math`

This test verifies a complete bidirectional cycle through all three states:

$$\text{Equilibrium} \rightarrow \text{QuoteShortage} \rightarrow \text{Equilibrium} \rightarrow \text{BaseShortage} \rightarrow \text{Equilibrium}$$

After four consecutive state transitions, the system returns to exactly the initial state with zero accumulated rounding errors. The existing test suite covers individual transitions well, but this integration test confirms they compose correctly over a full cycle.

5 Conclusion

The RadixPlaza AMM implementation is mathematically sound and correctly implements the CALM algorithm from the whitepaper. All core formulas are accurately transcribed and state transitions work correctly.

The code seems production-ready from a mathematical correctness standpoint.

Note: review was done with limited time. I would have loved doing a longer one, and a longer one would be necessary to be 100 percent sure no vulnurabilities are present, but I am at least confident no glaring issues exist.