Algo_LatencyArb

Sunday, April 27, 2025 10:32 AM

Parameter Adjustment Algorithm

You want to **adjust X** based on the **magnitude of volumeImbalance**. Here's a flexible, production-style approach:

Adjustment Function

```
def adjust_parameter_X(volume_imbalance, base_X, k):
    """
    Adjusts X based on volume imbalance.
    - base_X: the baseline parameter value
    - k: sensitivity factor
    """
    # e.g., X increases linearly with the absolute imbalance
    # You can use other functions (exponential, piecewise, etc.) as needed
    return base_X + k * abs(volume_imbalance)
```

In practice, you might want to cap X within bounds.

```
def adjust_parameter_X(self, product_id, counterparty_type):
        imbalance = self.get_volume_imbalance(product_id, counterparty_type)
        # Example adjustment: linear with cap
        X = self.base_X + self.k * abs(imbalance)
        return min(max(X, 0.5), 5.0)  # Clamp X between 0.5 and 5.0
```

$EMA_{new} = \alpha \times value_{new} + (1-\alpha) \times EMA_{old}$

α (smoothing factor) defines how quickly old values lose influence (typical: 0.01–0.2).
In your context, each trade updates the EMA of volumeImbalance.

This is often called a **continuous-time EMA** and is **ideal for irregularly-timed events** (like trades that don't arrive exactly every second). Here's how and why it works, plus sample code.

Why Time-Based EMA?

- In financial trading, trades can occur at any moment.
- If you use a fixed alpha (as in regular EMA), the "decay" rate is tied to trade frequency, not wall-clock time.
- **Time-based EMA** ensures that the impact of a trade decays at a consistent rate over real time, regardless of how often trades arrive.

$\alpha = 1 - \exp(-\Delta t / \tau)$

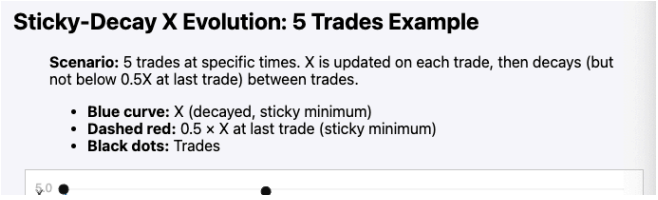
$EMA_{new} = \alpha \times x_{new} + (1-\alpha) \times EMA_{old}$

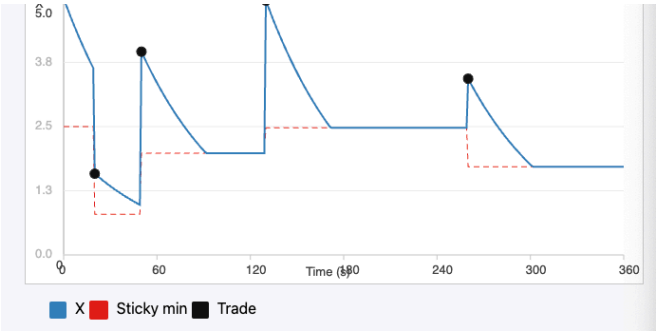
Key Points

- **(\tau)**: Higher τ means slower decay (old trades matter longer); lower τ means faster decay (recent trades matter more).
- **Handles irregular intervals**: If many seconds pass between trades, alpha is higher (old EMA "forgets" faster).

Retail Market-making Algorithm

- On each trade
  - update imbalance (via time-based EMA or running sum)
  - Compute a new X:
    - X new = k * imbalance
- Between trades
  - If there's no new activity, X should decay over time
  - However, X cannot decay below a "sticky minimum", which is 0.5 x the last nonzero X
- Implementation plan
  - Track last trade time and last adjusted X
  - On each trade
    - Update the running sum of imbalance
    - Compute new X
    - Set the new sticky minimum: Xmin = 0.5 * Xnew
  - On each X query (even without new trades):
    - Decay X using the time-based EMA (with no new input, so input = 0)
    - Clamp X to Xmin





step_num = floor(EMA imbalance / threshold).

- If step_num increases, increment X by k for each step up.
- If step_num decreases, decrement X by k for each step down.
- sticky_min = 0.5 * abs(X) * Math.sign(X)
- Between trades, **decay X exponentially towards zero but do not cross the sticky minimum** (sticky min is always half the magnitude of the last X after a step).

1. Initialization

- Set parameters:
  - k: the increment/decrement per threshold crossed (e.g., 1.0)
  - threshold: the imbalance threshold for each step (e.g., 50)
  - tau: EMA decay time constant (e.g., 60 seconds)
  - sticky_factor: fraction for sticky minimum (e.g., 0.5)

  Initialize for each trading pair (product_id, counterparty_type):
  - ema_imbalance = 0
  - X = 0
  - last_step_num = 0
  - sticky_min = 0
  - last_update_time = None

2. On Each Trade

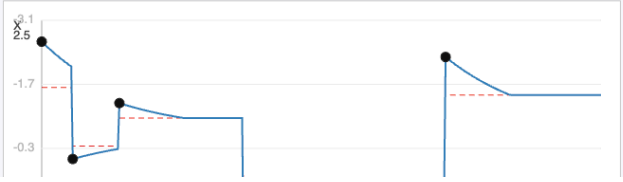
- **2.1. Decay EMA and X to the current trade's timestamp** (if any time has passed):
  - $\text{dt} = \text{trade.time} - \text{last\_update\_time}$
  - $\text{ema\_imbalance} \leftarrow \text{ema\_imbalance} \times e^{-\text{dt}/\tau}$
  - $X \leftarrow X \times e^{-\text{dt}/\tau}$
  - Clamp X to not cross the sticky minimum:
    - If $X > 0$: $X \leftarrow \max(X, \text{sticky\_min})$
    - If $X < 0$: $X \leftarrow \min(X, \text{sticky\_min})$
- **2.2. Update EMA imbalance with the new trade:**
  - $\text{ema\_imbalance} \leftarrow \text{ema\_imbalance} + \text{trade.qty}$
- **2.3. Calculate the new step number (can be negative):**
  - $\text{step\_num} = \text{floor}(\text{ema\_imbalance}/\text{threshold})$
- **2.4. If the step number changed (up or down):**
  - $\text{steps\_change} = \text{step\_num} - \text{last\_step\_num}$
  - $X \leftarrow X + \text{steps\_change} \times k$
  - $\text{sticky\_min} \leftarrow \text{sticky\_factor} \times |X| \times \text{sign}(X)$
  - $\text{last\_step\_num} \leftarrow \text{step\_num}$
- **2.5. Update last update time:**
  - $\text{last\_update\_time} = \text{trade.time}$

3. Between Trades (when you want to check X at any time)

- Decay EMA and X to the current time as in step 2.1.
- Clamp X to sticky minimum as above.

Scenario: X increases or decreases by **k** every time the signed EMA imbalance crosses a multiple of the threshold (50).

- **Blue curve:** X (stepwise, sticky minimum)
- **Dashed red:** Sticky min (always 0.5 × |X| × sign(X) after last step)
- **Black dots:** Trades
- X can go negative if imbalance reverses direction!




1.1
-3.1
0
60
120
Time (s)
180
240
300
360
X
Sticky min
Trade