MACD Meets Market Microstructure

Edward Atkinson

University Of Bristol

Abstract. PT2 uses a Mean Average Convergence Divergence cross over strategy combined with level-2 Limit Order Book Data signals which detects shifts in the market based on volume imbalance and price volatility.

Keywords: Limit Order Book · MACD · Bristol Stock Exchange · Cryptocurrency · Proprietary Trading

1 Introduction

In the last 25 years, algorithmic trading has grown from little involvement on financial markets, to accounting for 60-75% of trading volume across European and Asian markets. The advent of high volume and high frequency trading, birthed a microstructural aspect into modern markets, transforming them with new dynamics and complexities and presenting profit-making opportunities. Driven by the potential for large profit margins, innovative trading algorithms are being developed at a faster rate than ever.

The Mean-Average-Convergence-Divergence (MACD) crossover strategy is a momentum indicator which provides signals for entering and exiting a broad range markets. However, it overlooks a key aspect of modern intra-day trading: market microstructure. We present our proprietary trader PT2, an autonomous trading agent which augments a MACD crossover strategy with volume-based Limit Order Book (LOB) features derived from the Bristol Stock Exchange [3]. We investigate if a microstructure aware MACD strategy, can out-perform another proprietary trader, PT1.

2 Trading Strategy

The MACD crossover strategy has three components, fast and slow Exponential Moving Averages (EMA), the MACD and the Signal. EMA's are defined as:

$$\text{EMA}_t(L) = p_t \cdot \alpha + \text{EMA}(L)_{t-1} \cdot (1 - \alpha) \tag{1}$$

$$\alpha = \frac{\mu}{1 + L} \tag{2}$$

where,

- $\text{EMA}_{t-1}(L)$ is the value of the $\text{EMA}(L)$ at the previous trade price,

- p_t is the last purchase price of the asset,
- μ is the smoothing coefficient,
- L is the trade window size: L_{fast}, L_{slow}

$\text{EMA}_0(L)$ is initialised to the Simple Moving Average of the asset prices after L_{slow} trades have taken place. The smaller the window size, L, of an EMA the more reactive it is to recent price changes. This makes it a good indicator for high frequency trading algorithms, since they make trades based on recent price movements. Choosing our fast $\text{EMA}_t(12)$ and slow $\text{EMA}_t(24)$, we define our MACD_t as $\text{EMA}_t(12) - \text{EMA}_t(24)$. This measures the gap between the short term faster moving prices and the longer term slower moving prices. Finally the Signal_t, defined by the $\text{EMA}_t(9)$ of the MACD_t. Buy and sell signals are triggered by *breakouts* — when MACD_t was previously lower (higher) than Signal_t, but is now greater (lower), representing a bullish (bearish) market outlook, triggers a bid (ask) order. Crucially, before this order is sent to the exchange, the trader uses the LOB to gain an extra signal – the volume balance between bids and asks coupled with price volatility.

2.1 LOB Inference

The LOB provides information about the various quantities and prices market participants are willing to trade with. To quantify the market's direction we use the volume balance (3) on the LOB. If $V_{balance}$ is $\approx \pm 1$, the market direction is considered heavily bullish (+1) because there is bid-side dominance, and vice versa for ask-side dominance (-1). A linear weighted combination of the full depth and *levelled* depth of the LOB captures a balanced indication of imminent and underlying market forces. Levelled depth is set to 5, encompassing the top 5 best bids and asks and their respective quantities.

This measure works under the assumption that, if at any moment a market is bid-side (ask-side) dominant we expect the market price to rise (fall), meaning participants will be willing to pay (accept) more (less). So, we preemptively adjust our order prices to maximise the possible profit. This is referred to as the *spread*, and is calculated by (4), where κ and β are constant coefficients which scale the effect volatility (5) and volume balance (3) have on the *spread* pricing. κ and β can be calibrated through gridsearch optimisation methods, however they are largely exclusive to market conditions. We set κ and β to 1 and 0.6, emphasising LOB features more than volatility.

Once the *spread* is calculated for the latest price on the tape, `PT2` applies order specific adjustments. For ask orders when the supply is high, the *spread* is widened by 10% to capture a more profitable outcome if the ask is lifted. For bid orders when demand is strong, the *spread* is tightened by 10% to be more aggressive, raising the chance of filling.

In the case when MACD_t and Signal_t haven't crossed, `profit_perc` (1.05) and `stop_loss` (0.9) parameters define when the model can sell an asset, if there is an asset in the `inventory`. The inventory can grow to a size < `liquidate` (0), otherwise it must sell if it can. When `liquidate` is set to 0, `PT2` alternates between buy and sell jobs similarly to `PT1`. `PT2`'s trading frequency can become very high if `liquidate` is set to 0, so a throttle is introduced to curb exorbitant trading activity.

$$V_{balance}(d) = \frac{V_{bids} - V_{asks}}{V_{bids} + V_{asks}} \tag{3}$$

$$spread = (1 + \kappa \cdot V_{balance}) \cdot (1 + \beta \cdot v) \tag{4}$$

$$v = \frac{\sqrt{\sum_{i=0}^{n} (p_i - \mu)^2}}{N} \tag{5}$$

3 Setting up the tests

To benchmark `PT2` against `PT1`, we set up a suite of market *scenarios*, where key parameters in the BSE framework are changed to exhibit different market dynamics. We analyse and compare the performance of `PT2` and `PT1` for each scenario. Our procedure consists of three distinct phases that together address a broad spectrum of market dynamics. For all phases, the order schedule uses the *poisson process* for customer order arrivals, where the inter-arrival time is set to 10 seconds. The poisson process's Markovian nature is the most realistic mode of order arrival.

3.1 Phase One

Six intraday BTC-USD 5-minute interval data files are collected from yahoo finance [5], and scaled for use in BSE (Fig. 1). The market dynamics (shifts, volatility, open-close) across these 6 time series vary enough to test the trader's ability to profit in any market condition. Although BSE natively injects stochasticity into market simulations by trader activity, a breadth of different underlying trends must be considered for rigorous evaluation. For example, `PT1` may struggle to execute trades during surges in prices whereas `PT2` could be more opportunistic in highly volatile environments.

When conducting rigorous statistical analysis, it is important to gather observations from each scenario a sufficient number of times. This is because any independent scenario depends on the random nature of robot trader activity inside each simulation. Therefore we opt to run each scenario for 500 independent and identical one-day market simulations. BSE is populated with 20 of each `GVWY`, `SHVR`, `ZIC` and `ZIP` robot traders. The supply and demand schedules are setup to be symmetric at each equilibrium point in the offset file with step-mode set to `random`.

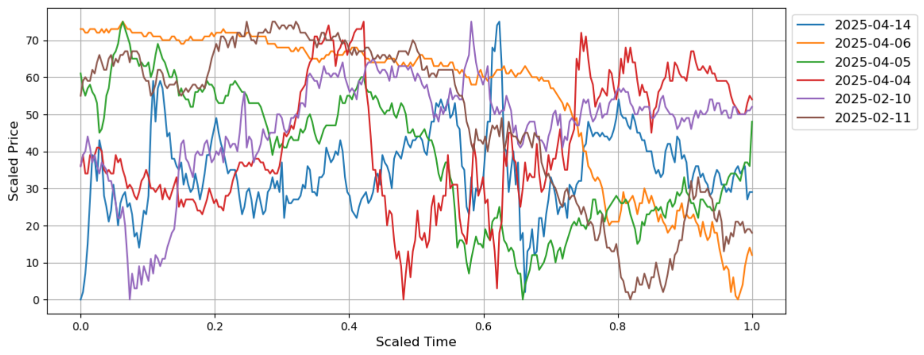

Fig. 1. BSE scaled BTC-USD offset intraday 5m interval market dynamics scenarios for testing the performance of PT1 and PT2. Time is scaled down from 86,400 seconds (1 day) to between 0 and 1.

3.2 Phase Two

The types of robot sales traders populating the market in BSE have varying levels of aptitude. Populating the market with different ratios of robot traders, provides insightful analyses into the sophistication of PT1 vs PT2. For example, in a market dominated by ZIP traders, the limit order prices will quickly converge towards the equilibrium price as they are more efficient traders who demonstrably trade as well, if not better than humans [2], reducing market inefficiencies. Whereas if the market consists of mainly GVWY traders (short for “giveaway” traders that only submit *market orders* — orders that immediately execute by hitting/lifting the best bid/ask), there are more market inefficiencies for PT1 and PT2 to exploit, so we expect them both to be more active and profitable. It is important that PT2 consistently outperforms PT1 over a broad range of market populations, proving it’s robustness against competitive players in the market. Therefore market populations are split into 3 types of ratios, 2:1:1:1 (50), 3:1:1:1 (84), 2:2:1:1 (60), where the numbers in brackets denote the total population. All permutations of these ratio’s are considered, totaling 14 market scenarios. For each population, buyers and sellers are equally represented, and $>$50 traders are used in each scenario to maintain high market activity– akin to real markets. We run 200 identical independent market sessions per scenario to produce enough samples to construct statistically robust conclusions.

3.3 Phase Three

Phase 3 sees the effect of elastic demand and supply schedules on the traders’ performances. In the previous phases, supply and demand has been symmetric, meaning the gradient of the demand schedule is the reciprocal to the gradient of the supply schedule. Therefore, their intersection is at right angles, given some stochasticity introduced by the `random stepmode` parameter inside BSE.

Elasticity refers to the responsiveness of supply or demand to changes in price governed by the ranges set in BSE's schedules. A narrow range results in a flatter slope, indicating greater elasticity because quantity changes more with small changes in price. Conversely, a wider range means a steeper slope, indicating inelasticity. This setup mirrors real-world market dynamics. Elastic supply mimics a market where changes in prices cause large quantity shifts, an example of high liquidity. Elastic demand reflects price-sensitive buyer behaviour, which is apparent in volatile and illiquid markets. These scenarios help to evaluate the risk associated with using `PT2` in illiquid and liquid markets, ensuring usability as the market continuously evolves. Phase three recycles the 10-02-2025 scenario from Phase one, with a relatively elastic demand, configured by widening the demand schedule's range from `[90,125]` to `[70,135]` and keeping the supply schedule at `[75,120]`. This is repeated for elastic supply by widening the supply schedule range from `[75,125]` to `[60,140]`. A comparison of the "jittered" and "fixed" BSE supply and demand scheduling `stepmode`'s is also included to investigate their impacts on trader behaviour.

4 Statistical Analysis

Over 25 different market scenarios, 6,800 market days are simulated, where each day has 86,400 seconds. This scale of testing required serious planning and compute resources, facilitated by the UOB's High Performance Computing infrastructure Blue Pebble. The large number of independent trials ($>$200 per scenario) ensures sufficient statistical power to detect small differences in trader performance. Details of the setup and computation of statistics can be found [1]. We use box and whisker plots to comparatively visualise the performance of `PT1` and `PT2`. These plots efficiently display key observations for each scenario. The median and inter-quartile range show trader's central tendency, while whiskers represent outliers across all sessions.

In Phase One of testing, `PT2` consistently out-performs `PT1`. The non-overlapping box-plots in Figure 2 for both total returns and number of trades strongly suggests `PT2` is more profitable than `PT1`. It is clear `PT2` is more opportunistic since it places more profitable trades throughout the market sessions, illustrating a robust strategy which can handle the broad market conditions imposed by each offset. To investigate further, we introduce Phase 2 of testing which examines how `PT2` performs in market's populated with different compositions of robot traders.

Figure 3 immediately suggests that `PT1`'s trading patterns change in markets populated with more `GVWY` traders. The large variability in the number of trades executed by `PT1` across scenarios suggests it struggles to make profits in more competitive markets. Whereas `PT2` consistently performs better than `PT1`. There is a clear correlation between the number of trades and the total profit accumulated by `PT1`, where the pearsons correlation coefficient is $0.99 (p = 2.63 \times 10^{-15})$. It is a different story for `PT2`, where profits are less correlated with the number

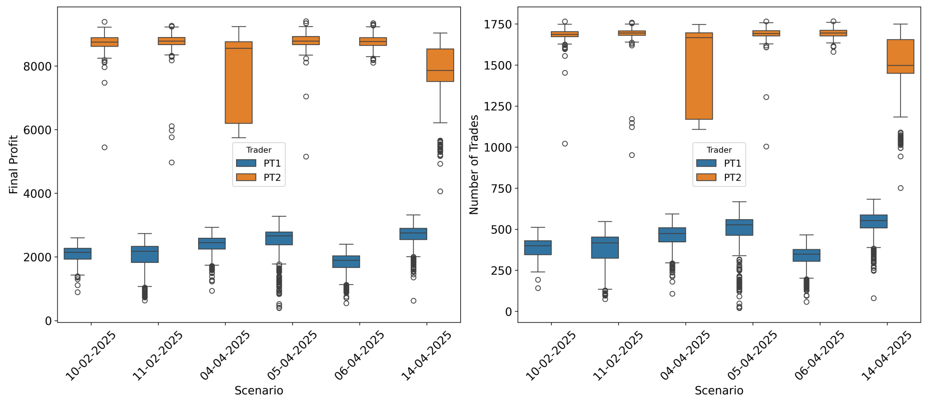

Fig. 2. Box plots for the Phase 1 market scenarios. **Left** visualises the returns for each scenario. **Right** visualises the number of trades made by each trader for each scenario.

of trades, reporting a pearsons correlation coefficient of $0.519 (p = 0.041)$. This hints that PT2's profit is not entirely dependent on trade volume.

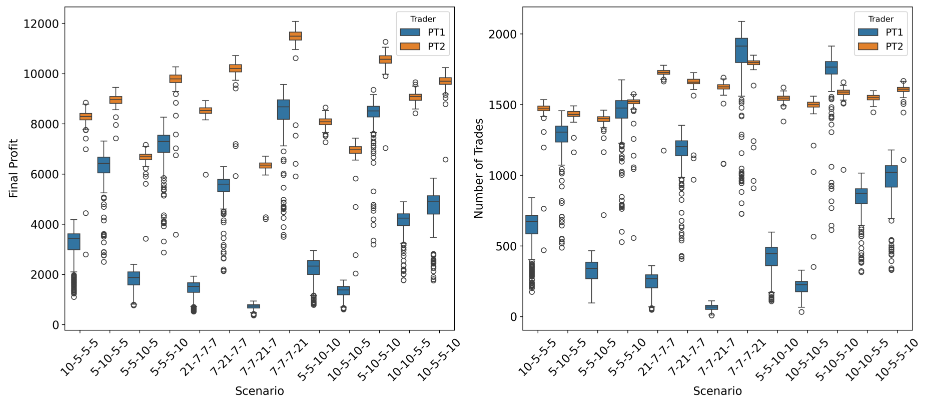

Fig. 3. Box plots for the Phase 2 market scenarios. **Left** visualises the returns for each scenario. **Right** visualises the number of trades made by each trader for each scenario. The compositions of robot traders are ordered by `ZIP`, `ZIC`, `SHVR`, `GVWY`

In Phase Three, `PT2`'s median return breaks into 5 figures at 10,097, up from 8,750, while maintaining a similarly low standard deviation of 211 across 200 trials. `PT1`'s return varies considerably, it's median increases from 2,143 to 6,085, the extremely high median absolute deviation (MAD) of 1,041 proves `PT1` struggles to consistently exploit market inefficiencies. MAD is a more robust approach to observing variance when the distribution has heavy tails. On the flip side, relatively elastic supply conditions see only marginal improvements in both traders performances. When `stepmode` is set to "fixed" and "jittered" the variance of `PT2`'s profitability in elastic supply and demand scenarios increases from ≈ 220 (random) to 800–900.

Table 1. Profit Per Trade (PPT) comparison between PT1 and PT2 across different scenarios. The single-tailed Wilcoxon Rank Sum Test W-stat along with its p-value are provided to show statistical difference of `PT2` > `PT1`.

	Scenario	PT1			PT2			W-Stat	p-value	trials
#		Sharpe	Median	Std	Sharpe	Median	Std			
0	2025-04-04	17.22	10	0.61	43.96	10	0.24	66245	0.13	500
1	2025-04-14	20.92	10	0.48	56.69	10	0.18	113055	0.00	500
2	2025-02-11	10.99	11	0.99	61.91	10	0.17	30690	1.00	500
3	2025-02-10	24.17	11	0.45	62.07	10	0.17	8189	1.00	500
4	2025-04-05	3.94	10	2.72	60.75	10	0.17	83711	0.00	500
5	2025-04-06	14.11	11	0.78	63.85	10	0.16	3952	1.00	500
6	2025-02-11	21.50	10	0.48	55.34	11	0.20	121967	0.00	200
7	7-7-20-7	2.17	22	11.78	55.74	8	0.14	0	1.00	200
8	5-10-5-10	54.67	10	0.18	59.78	13	0.22	20100	0.00	200
9	5-5-10-10	11.85	11	0.92	60.03	10	0.17	4994	1.00	200
10	5-5-5-10	46.66	10	0.21	60.13	13	0.21	20100	0.00	200
11	20-7-7-7	6.50	12	1.88	63.40	10	0.16	0	1.00	200
12	10-10-5-5	32.51	10	0.30	60.65	11	0.19	20100	0.00	200
13	10-5-5-10	30.69	9	0.32	58.48	12	0.21	20100	0.00	200
14	7-7-7-20	52.29	10	0.17	66.58	12	0.19	20100	0.00	200
15	5-10-5-5	49.83	9	0.20	59.72	13	0.21	20100	0.00	200
16	5-5-10-5	9.80	10	1.19	54.23	13	0.18	0	1.00	200
17	10-5-10-5	5.12	12	2.56	38.61	10	0.24	0	1.00	200
18	7-20-7-7	42.90	9	0.22	64.62	9	0.19	20100	0.00	200
19	ES-fixed	0.66	58	186.07	11.48	12	0.71	0	1.00	200
20	ED-fixed	0.62	30	80.63	9.03	8	0.92	2	1.00	200
21	ES-jit	0.67	71	233.01	10.34	8	0.78	0	1.00	200
22	ES-rand	24.86	11	0.44	63.52	11	0.18	18866	0.00	200
23	ED-jit	0.66	31	96.72	10.16	8	0.81	5	1.00	200
24	ED-rand	34.68	9	0.27	62.59	12	0.19	0	0.00	200

4.1 Phase-wide overview

We can confirm that `PT2` consistently makes more profit, but the common denominator across most scenarios is that `PT2` makes **significantly** more trades than `PT1`. This prompts the question: is `PT2` more profitable on a **trade-by-trade** basis? To investigate this, the profit-per-trade (PPT) for both traders is calculated, along with Sharpe ratios (using risk-free return = 0 due to intra-day environment). To determine if `PT2`'s PPT is greater for each scenario, we use the single tailed Wilcoxon Rank Sum test [4]. This tests for a shift in distributions of PPT between the 2 traders by comparing the medians, crucially without assuming normality because each scenario's distribution is likely different. Setting the null hypothesis (H0) to `PT2 = PT1`, and the alternate hypothesis (H1) to `PT2 > PT1`. We reject H0 when p-value < 0.05 and deem `PT2` to have greater PPT. Table 1 displays each test statistic and corresponding p-value, accompanied by the sharpe ratio, median and standard deviation for each trader. Overall, `PT2` has greater PPT on half of the scenarios, with the most success during Phase 2 scenarios. However, `PT1` seems to have a much higher intra and inter scenario variance. This means that `PT2` is a more robust trader; regardless of trade volume it consistently makes returns. This is also proven by `PT2`'s high Sharpe ratios. Typically "good" Sharpe ratios are within the 1 - 5 range for realistic market dynamics, however since BSE simulations are not as competitive as real markets, ratios span up to as high as 67. `PT2`'s high sharpe ratios across trials is attributed to it's consistently low variance of returns.

5 Conclusion

This study demonstrates that augmenting a traditional MACD crossover strategy with limit order book microstructure features significantly improves trading performance across a variety of market conditions. The PT2 trader consistently outperforms PT1 in profitability and robustness as shown by higher Sharpe ratios and lower return variances. Statistical analyses, including the Wilcoxon Rank Sum Test, confirm that PT2's advantage is significant across most scenarios. The Bristol Stock Exchange environment, cannot fully capture the adversarial dynamics of real-world high-frequency trading. Future work could explore PT2's performance in live environments or more adversarial simulations, for example include trading fees and order latencies. Optimization of the LOB weighting coefficients κ and β under different volatility regimes could also yield improvements. Overall, the work highlights the importance of market microstructure awareness in modern algorithmic trading strategies, and offers a strong foundation for future development of hybrid technical-microstructure trading agents.

References

1. Algorithmic trading coursework. https://github.com/edatkinson/AlgoTrading/tree/main/Cousework (2025)
2. Cliff, D.: Minimal intelligence agents for bargaining behaviours in market-based environments. Technical Report HPL-97-91, Hewlett-Packard Labs (1997)
3. Cliff, D.: Bse: A minimal simulation of a limit-order-book stock exchange (2018), https://arxiv.org/abs/1809.06027
4. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**(6), 80–83 (1945), http://www.jstor.org/stable/3001968
5. Yahoo Finance: BTC-USD. https://uk.finance.yahoo.com/quote/BTC-USD/, accessed: 2025-04-26

A Appendix

A.1 TraderPT2

```
class TraderPT2(Trader):

    def __init__(self, ttype, tid, balance, params, time):
        """
        New parameters:
            - portfolio value (net worth)
            - inventory (the number of stocks we have at any
              time)
            - bid_order & ask_order store the orders
            - job decides which orders we are working
            - last_update time and last_purchase price store
              exactly what they say, this is to manage the
              amount of trades we place.
        """
        super().__init__(ttype, tid, balance, params, time)

        self.portfolio = 0
        self.inventory = 0
        self.bid_order = None #the bid order object
        self.ask_order = None #the ask order object

        self.job = 'Bid' #switches between bid and ask.

        self.last_update_time = 0
        self.last_purchase_price = None
        self.last_sell_price = None
        self.latest_price = 0
        self.ema = [0, 0]            # Short EMA, Long EMA
        self.macd = [0, 9]           # MACD value, with fixed
            period for signal
        self.signal = 0
        self.sma = 0
        self.volatility = 0
        self.smooth = 2              # Smoothing factor for
            the EMA

        # Set default parameters (if they exist) and validate
             inputs
        if isinstance(params, dict):
            self.n_past_trades = int(round(params.get('
                n_past_trades', 10)))
            if self.n_past_trades < 1:
                sys.exit('Fail: PT2 n_past trades must be 1
                    or more')
            self.EMA_long = int(round(params.get('EMA_long',
               24))) #
            if self.EMA_long < 1:
                sys.exit('Fail: PT2 EMA_long must be 1 or
                   more')

            self.EMA_short = int(round(params.get('EMA_short'
               , 12)))
            if self.EMA_short < 1:
                sys.exit('Fail: PT2 EMA_short must be 1 or
                   more')

            self.spread = int(round(params.get('spread', 2)))
                #this is dynamic, so doesn't really matter
               but we will keep it, why not
            if self.spread < 1:
                sys.exit('Fail: PT2 spread must be 1 or more'
                   )

            # This is the defualt, and is sufficient. You can
                specify them in args
            self.max_inv = int(round(params.get('max_inv', 5)
               ))
            self.liquidate = int(round(params.get('liquidate'
               , 2)))
            self.profit_perc = params.get('profit_perc',
               1.05)
            self.stop_loss = params.get('stop_loss', 0.95)
            self.depth = params.get("lob_depth", 5)
            self.min_order_interval = params.get("
               min_order_interval", 30) # default is 30 to
               reduce HFT. But reducing this to 1 makes more
                profit but more trades.

        # Tracks the current bids and asks on the lob down to
             self.depth level.
        self.track_lob = {'asks': [], 'bids': [], 'time': [],
             'imbalance': []}

    def getorder(self, time, countdown, lob):
        """
        Return this trader's order when polled by the market
            session.
        """
        if countdown < 0:
            sys.exit('Negative countdown')

        if len(self.orders) < 1 or time < 5 * 60: # No orders
             to work
            order = None
        else:
            quoteprice = self.orders[0].price
            order = Order(self.tid,
                          self.orders[0].otype,
                          quoteprice,
                          self.orders[0].qty,
                          time, lob['QID'])
            self.lastquote = order
        return order

    def inspect_lob(self,lob, time):
        """
        Essentially, we don't want to pay a constant spread
            each time as the market changes and we need to
            change with it.
        We need to look at LOB prices to see where the supply
            &demand is.
        """
        depth_asks = self.depth if len(lob['asks']['lob']) >
            self.depth else len(lob['asks']['lob'])
        depth_bids = self.depth if len(lob['bids']['lob']) >
            self.depth else len(lob['bids']['lob'])

        asks = lob['asks']['lob'][:depth_asks] # list of best
             asks
        bids = lob['bids']['lob'][:depth_bids] # list of best
             bids

        vol_asks = lob['asks']['n'] # all levels
        vol_bids = lob['bids']['n'] # all levels

        """
        LOB balance = (V_bid - V_ask) / (V_bid + V_ask)
        """
        if vol_bids > 0 or vol_asks > 0:
            full_lob_balance = (vol_bids-vol_asks) / (
                vol_bids+vol_asks) # 1 is bidside dominance,
                -1 is askside dominance. 0 is balance.
            full_bid_ratio = (vol_bids) / (vol_bids+vol_asks)
                 # >0.5 = bid dominance, <0.5 = ask dominance
        else:
            full_lob_balance = 0
            full_bid_ratio = 0

        #To inspect this further, we look into volume of top
            self.depth levels
        ask_prices = [item[0] for item in asks]
        ask_quantities = [item[1] for item in asks]

        bid_prices = [item[0] for item in bids]
        bid_quantities = [item[1] for item in bids]

        # Update the track lob with ask and bid prices. Could
             get very big, so be careful here
        self.track_lob['asks'].append(ask_prices)
        self.track_lob['bids'].append(bid_prices)
        self.track_lob['time'].append(time)
        self.track_lob['imbalance'].append(full_lob_balance)
        levelled_vol_asks = sum(ask_quantities)
        levelled_vol_bid = sum(bid_quantities)

        if levelled_vol_bid > 0 or levelled_vol_asks > 0:
            levelled_lob_balance = (levelled_vol_bid -
                levelled_vol_asks) / (levelled_vol_asks+
                levelled_vol_bid)
            levelled_bid_ratio = (levelled_vol_bid) / (
                levelled_vol_bid+levelled_vol_asks)
        else:
            levelled_lob_balance = 0
            levelled_bid_ratio = 0

        levelled_lob = [levelled_lob_balance,
            levelled_bid_ratio] # balance, ratio
        full_lob = [full_lob_balance, full_bid_ratio]

        snapshot = {'leveld_lob': levelled_lob, 'full_lob':
            full_lob} # This is the lob at the current time
            point. What about previous time points?

        return snapshot



    def adjust_ticksize(self, lob, time):
        # First, extract the LOB signals.
        # Get a snapshot of levelled and full LOB measures.
        lob_dict = self.inspect_lob(lob, time)
        full_lob_balance, full_bid_ratio = lob_dict['full_lob
            '][0], lob_dict['full_lob'][1]
        leveld_lob_balance, leveld_bid_ratio = lob_dict['
            leveld_lob'][0], lob_dict['leveld_lob'][1]

        # Compute a weighted imbalance.
        # weigh full LOB imbalance at 60% and levelled
            imbalance at 40%.
        weighted_imbalance = 0.6 * full_lob_balance + 0.4 *
            leveld_lob_balance

        vol_factor = getattr(self, 'volatility', 1)
```

```
base_spread = getattr(self, 'base_spread', 1)

k = 1
v = 0.6

# Compute an adjusted spread, this will also be
    adjusted again depending on the current self.job.
# if weighted_imbalance > 0, there is upward pressure
     (bid dominance).
# If volatility is high, we may wish to widen the
    spread to avoid overpaying (for Bid orders) or
    leave more room (for Ask orders).
adjusted_spread = base_spread * (1 + k *
    weighted_imbalance) * (1 + v * vol_factor)

# Apply job-specific adjustments.
# For Bid orders, be slightly more aggressive because
     demand is high, so reduce the spread.
# For an Ask order, widen the spread if supply is
    high.
if self.job == 'Bid':
    final_spread = max(1, int(round(adjusted_spread *
         0.9)))
elif self.job == 'Ask':
    final_spread = int(round(adjusted_spread * 1.1))
else:
    final_spread = int(round(adjusted_spread))

# We need boundaries to make sure the spread isnt
    ridiculous
min_spread = 1
max_spread = 10
final_spread = max(min_spread, min(final_spread,
    max_spread))
self.spread = final_spread  # update the trader's
    spread

# Compute a candidate order price based on the best
    LOB price.
if self.job == 'Bid' and lob['bids']['best'] is not
    None:
    # For a bid order, subtract the spread from the
        best bid, since we want to pay less
    candidate_price = lob['bids']['best'] -
        final_spread

elif self.job == 'Ask' and lob['asks']['best'] is not
     None:
    # For ask orders, add the spread to the best ask.
```

```
            candidate_price = lob['asks']['best'] +
                final_spread
        else:
            candidate_price = None

        return candidate_price, final_spread

    def respond(self, time, lob, trade, vrbs):
        vstr = f"t={time} PT2 respond: "

        # Decide if order placement is allowed.
        can_post_ask = self.inventory > 0
        can_post_bid = (lob['bids']['best'] is not None and
            self.balance > lob['bids']['best'])

        # Quickly collect the most recent n_past_trades "
            Trade" entries by iterating backwards.
        recent_trades = []
        # self.n_past_trades needs to be > EMA_long
        for entry in reversed(lob['tape']):
            if entry['type'] == 'Trade':
                recent_trades.append(entry)
                if len(recent_trades) >= self.n_past_trades:
                    break

        if len(recent_trades) < self.n_past_trades:
            if vrbs:
                print(f"{vstr}Not enough tape trade data (
                    have {len(recent_trades)}, need {self.
                    n_past_trades}).")
            return


        ### -------------------- Indicators
            --------------------- ###
        # --- Compute SMA and Standard Deviation ---
        prices = [t['price'] for t in recent_trades]
        self.sma = sum(prices) / len(prices)
        std_recent = (sum((p - self.sma) ** 2 for p in prices
            ) / (len(prices) + 1)) ** 0.5
        self.volatility = std_recent

        # check best bid/ask: if they deviate too much from
            the SMA, default to SMA. This is to avoid
            considering bids and asks which are far away from
             the true market price, and accidentally placing
            silly orders and bids.
```

```
best_bid = lob['bids']['best'] if (lob['bids']['best'
    ] is not None and abs(lob['bids']['best'] - self.
    sma) <= 2 * std_recent) else self.sma
best_ask = lob['asks']['best'] if (lob['asks']['best'
    ] is not None and abs(lob['asks']['best'] - self.
    sma) <= 2 * std_recent) else self.sma

# Define a fair price as the midpoint between the
    best bid and ask.
fair_price = (best_bid + best_ask) // 2

# Adjust the tick size depending on the LOB outlook.
candidate_price, tick_size = self.adjust_ticksize(lob
    , time)


# --- Update Latest Trade Price ---
# Recent trades is reversed
self.latest_price = recent_trades[0]['price']

# --- EMA, MACD, and Signal Calculation ---
if self.ema == [0, 0]:
    # this is the first update so init both EMAs to
        the SMA.
    ema_short = self.sma
    ema_long = self.sma
    macd = 0
    signal = macd
else:
    alpha_short = self.smooth / (1 + self.EMA_short)
    alpha_long = self.smooth / (1 + self.EMA_long)
    ema_short = self.latest_price * alpha_short +
        self.ema[0] * (1 - alpha_short)
    ema_long = self.latest_price * alpha_long + self.
        ema[1] * (1 - alpha_long)
    macd = ema_short - ema_long

    #Signal line as an EMA of the MACD.
    alpha_signal = self.smooth / (1 + (self.macd[1]
        if self.macd and self.macd[1] else 9))
    signal = macd * alpha_signal + self.signal * (1 -
         alpha_signal)

# Update stored EMA values.
self.ema = [ema_short, ema_long]

prev_macd = getattr(self, 'prev_macd', macd)
prev_sig = getattr(self, 'prev_sig', signal)
self.prev_macd, self.prev_sig = macd, signal # Stores
     the macd and signal for the next run. getattr
```

```
       retrieves them and means we don't need to define
       in __init__

    # --- Throttle Order Frequency ---
    # Use the minimum order interval to reduce high-
       frequency updates.
    min_order_interval = getattr(self, '
       min_order_interval', 30)  # default 30 seconds.
    last_order_time = getattr(self, 'last_order_time', 0)
    if time - last_order_time < min_order_interval:
        if vrbs:
            elapsed = time - last_order_time
            print(f"{vstr}Skipping order update: only {
                elapsed:.2f} sec elapsed (min interval: {
                min_order_interval}s).")
        return

    # --- Check for Crossovers and Determine Orders ---
    # Bullish crossover: MACD rises above Signal (and
       previous MACD was below Signal)
    # and the latest price is below fair value to ensure
       a favorable buy.

    if self.job == 'Bid':
        if macd > signal and prev_macd <= prev_sig and
           can_post_bid and self.latest_price <
           fair_price:
            candidate_bid = candidate_price if
                candidate_price is not None else best_bid
            # Avoid immediately repurchasing at the price
                 just sold.
            if self.last_sell_price is not None and
                candidate_bid == self.last_sell_price:
                 candidate_bid -= tick_size -1 # this is
                     just best bid
            # Final safety check.
            if candidate_bid > 0 and self.balance -
                candidate_bid >= 0:
                 self.bid_order = Order(self.tid, 'Bid',
                     int(round(candidate_bid)), 1, time,
                     lob['QID'])
                 self.orders = [self.bid_order]

                 self.last_order_time = time
                 if vrbs:
                     print(f"{vstr}Bullish crossover:
                         Posting Buy Order at {
                         candidate_bid:.2f} | SMA={self.
                         sma:.2f}, Latest={self.
```

```
                        latest_price:.2f}, Fair={
                        fair_price:.2f}")
                else:
                    if vrbs:
                        print(f"{vstr}Bullish signal but
                            candidate bid ({candidate_bid})
                            is invalid or exceeds balance.")
            else:
                self.job = 'Ask'

        elif self.job == 'Ask':

            # Bearish crossover: MACD falls below Signal (and
                previous MACD was above Signal)
            # and the latest price has reached a profit
                threshold relative to our last purchase.
            if macd < signal and prev_macd >= prev_sig and
                can_post_ask and self.last_purchase_price is
                not None and self.latest_price >= self.
                last_purchase_price * self.profit_perc:
                candidate_ask = candidate_price if
                    candidate_price is not None else best_ask
                     + self.spread
                # Ensure we do not sell below our last
                    purchase price.
                if candidate_ask < self.last_purchase_price:
                    candidate_ask = self.last_purchase_price
                        + tick_size
                self.ask_order = Order(self.tid, 'Ask', int(
                    round(candidate_ask)), 1, time, lob['QID'
                    ])

                self.orders = [self.ask_order]
                self.last_order_time = time
                if vrbs:
                    print(f"{vstr}Bearish crossover: Posting
                        Sell Order at {candidate_ask:.2f} |
                        SMA={self.sma:.2f}, Latest={self.
                        latest_price:.2f}, Purchase={self.
                        last_purchase_price}")

        # If no clear crossover but we hold > threshold to
            liquidate inventory, default to a conservative
            selling order.
            elif self.inventory > self.liquidate:
                candidate_ask = candidate_price if
                    candidate_price is not None else best_ask
                     + self.spread
                if self.last_purchase_price and candidate_ask
                     < self.last_purchase_price:
```

```
                    candidate_ask = self.last_purchase_price
                        + tick_size
                self.ask_order = Order(self.tid, 'Ask', int(
                    round(candidate_ask)), 1, time, lob['QID'
                    ])

                self.orders = [self.ask_order]
                self.last_order_time = time
                if vrbs:
                    print(f"{vstr}Holding inventory without
                        crossover. Default Sell Order posted
                        at {candidate_ask:.2f}")
            else:
                self.job = 'Bid' # need more inventory to in
                    order to ask



        # --- Update Performance Metrics ---
        self.portfolio = self.balance + (self.
            last_purchase_price or 0) * self.inventory

        self.profitpertime = self.profitpertime_update(time,
            self.birthtime, self.balance if self.liquidate >
            0 else self.portfolio)
        if vrbs:
            print(f"{vstr}Indicators -> SMA={self.sma:.2f},
                Latest={self.latest_price:.2f}, EMA_short={
                self.ema[0]:.2f}, "
                f"EMA_long={self.ema[1]:.2f}, MACD={macd:.2f
                    }, Signal={signal:.2f}, BestBid={best_bid
                    }, "
                f"BestAsk={best_ask}, Fair={fair_price:.2f}")



    def bookkeep(self, time, trade, order, vrbs):
        """
        Update trader's records of its bank balance, current
            orders, and current job
        :param trade: the current time
        :param order: this trader's successful order
        :param vrbs: if True then print a running commentary,
             otherwise stay silent.
        :param time: the current time.
        :return: <nothing>
        """

        # output string outstr is printed if vrbs==True
```

```
mins = int(time//60)
secs = time - 60 * mins
hrs = int(mins//60)
mins = mins - 60 * hrs
outstr = 't=%f (%dh%02dm%02ds) %s (%s) bookkeep:
    orders=' % (time, hrs, mins, secs, self.tid, self
    .ttype)

for order in self.orders:
    outstr = outstr + str(order)

self.blotter.append(trade)  # add trade record to
    trader's blotter

transactionprice = trade['price']
# Determine which order was filled and update
    accordingly.
if order.otype == 'Bid':
    # A filled bid means we bought one unit.
    self.balance -= transactionprice
    self.inventory += 1
    self.last_purchase_price = transactionprice
    self.job = 'Ask'
elif order.otype == 'Ask':
    # A filled ask means we sold one unit.
    self.balance += transactionprice
    self.inventory -= 1
    self.last_sell_price = transactionprice
    self.job = 'Bid'

else:
    sys.exit(f"FATAL: Unknown order type {order.otype
        }")

self.portfolio = self.balance + (self.inventory *
    transactionprice)


# Remove the executed order from our list.
if order in self.orders:
    self.del_order(order)


net_worth = self.balance + (self.inventory * self.
    last_purchase_price)
print('%s q=%d Balance=%d NetWorth=%d, spread=%d,
    price=%d' % (outstr, self.inventory, self.balance
    , net_worth, self.spread, order.price))
    def del_order(self, order):
        try:
            self.orders.remove(order)
        except ValueError:
            pass
```