Gravity Team – Project Report

In this project, we assume to be buying and selling a theoretical Token using USDT capital, on the exchange with the provided order book characteristics. The goal is to design a directional trading strategy that is profitable in this environment.

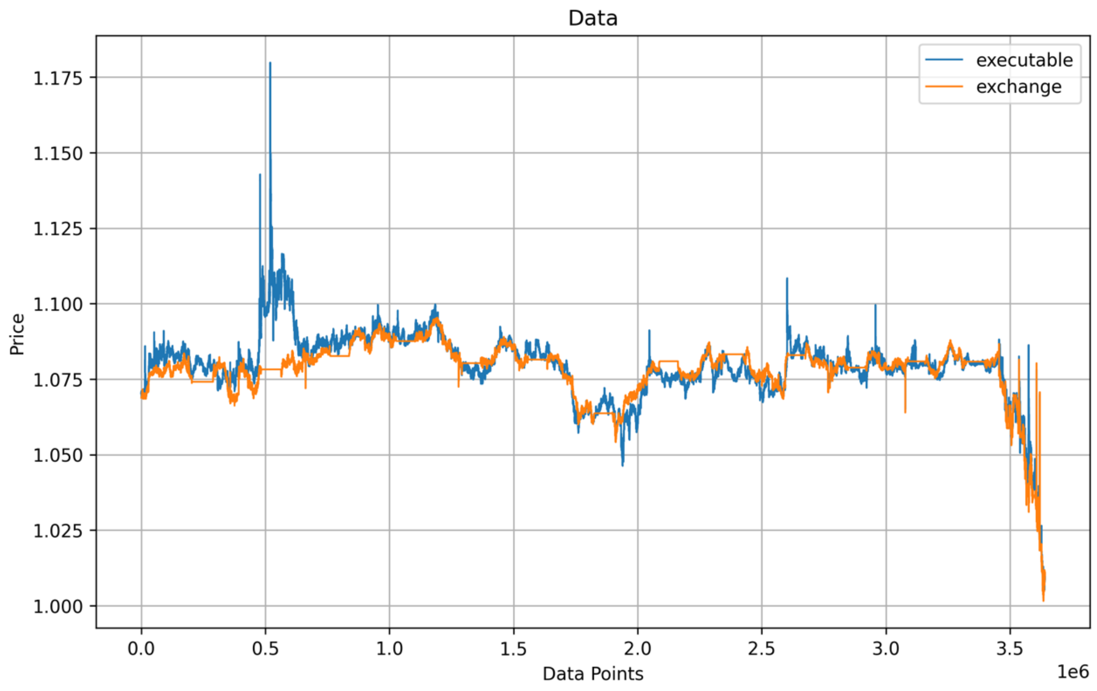
Contents:

Assumptions

I have made several assumptions while interpreting the provided brief of the project. They are listed below with rational and their implications (in no particular order).

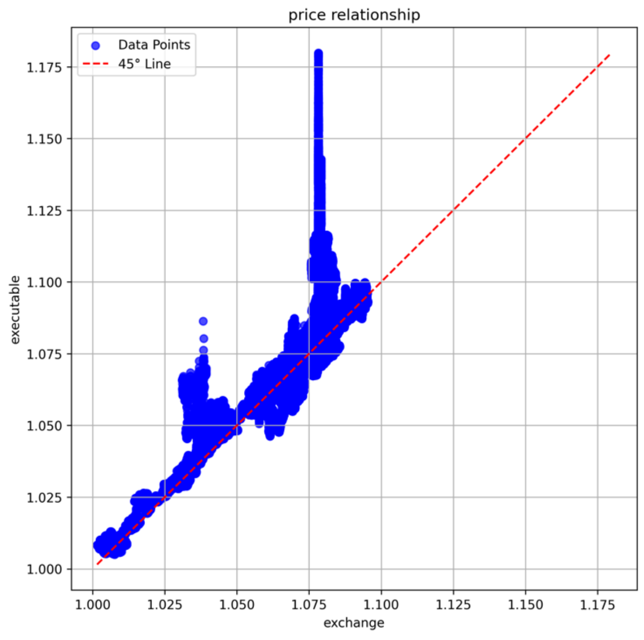
1. Both 'exchange_rate' and 'executable_rate' are interpreted to be representing an exchange rate of 1 Token to USDT. So a value of 1.3 means that 1 Token is worth 1.3 USDT.
2. 'executable_rate's is the mid-price of the exchange.
3. Spread is symmetrical around mid-price
4. Only one order can be submitted at any point in time. For a maker order, this means we must wait for all fills to be completed before another order (maker or taker) can be placed.
5. Short-selling is not permitted, positions to be long-only in [0, 100%].
6. USDT balance does not earn any yield.
7. Maker order is always placed at the top of the book. By this we mean that maker order's price limit is best bid + 1bp if it is a BUY order (or best ask – 1bp if it is a SELL order).
8. Taker order assumed to be executed immediately.
9. Maker/Taker order fees are assumed as a one-off fixed adjustments to the price.
10. For maker orders, execution price (after adjustments for costs) is kept fixed for the duration of the order fill, meaning each 1% increment is assumed to be executed with the same price.

Data



Some observations on the provided data:

- The provided file with the data has been converted from '.csv' to '.parquet' format to allow for faster loading into Pandas DataFrame.
- As provided, data was not in chronological order (based on the 'gt_timestamp' column), so it was sorted accordingly during processing.
- Time difference between data points is variable:
  Mean (seconds): 3.41 seconds
  Median (seconds): 2.30 seconds
  Standard Deviation (seconds): 85.52 seconds
  Minimum (seconds): 1.51 seconds
  Maximum (seconds): 160581.47 seconds
  25th Percentile (seconds): 2.13 seconds
  75th Percentile (seconds): 2.47 seconds
- The 'executable' rate clearly exhibits mean-reverting behaviour towards 'exchange' rate, however:
  - The spread between 'executable_rate' and 'exchange_rate' exhibits significant positive bias
  - 'exchange_rate' goes through prolonged periods of being unchanged



Project structure

The code base for the project is structured as follows:

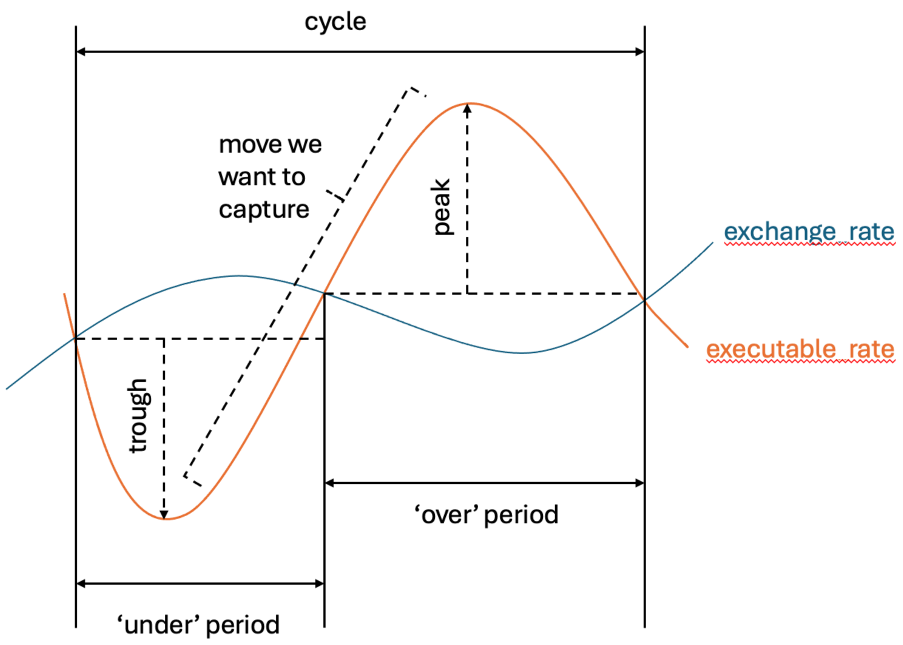
- /data
    - data.parquet – provided data file
- /results
- /src
    - ***concepts.py*** – various descriptive data classes that represent standardised bits of information involved in the strategy lifecycle
    - ***configs.py*** – parameters of the order book, trading strategy and data objects
    - ***data_processing.py*** – data loading and processing methods
    - ***data_analysis.py*** – data exploration and cycle analysis
    - ***execution.py*** – all aspects of execution, order type selection, position management and trade execution
    - ***period_cycle.py*** – definitions and related methods for detecting periods and cycles in the data
    - ***signals.py*** – trading strategies
    - ***simulator***.py – methods for trading simulator
    - ***time_manager.py*** – creation and management of time increments
    - ***reporting.py*** – performance methodology for reports and charts
    - ***utils.py*** – helper methods
    - ***run_simulation.py*** – end-to-end process for simulating the trading strategy

Approach

The main challenge with this project has been finding a trading strategy that is capable of balancing high cost of execution (e.g. up to 60bp price impact for a 100% Taker order) with relative lack of lucrative opportunities to exploit. The latter is a function of the data provided where the moves in ‘executable_rate’ tend to be very small, while being noisy with occasional large spikes, as discussed in the Data section above. I have tried several approaches to designing a trading strategy, with varying degrees of complexity. You can look at *signals.py* module to get an idea of some of the ideas that I have tested. Fundamentally, all were trying to leverage a hypothetical causal relationship between moves in ‘executable_rate’ and the relative spread dynamics between the two time-series. Most suffered from over-trading given the noisiness of the data and failed to achieve desirable results because of conservative order book mechanics.

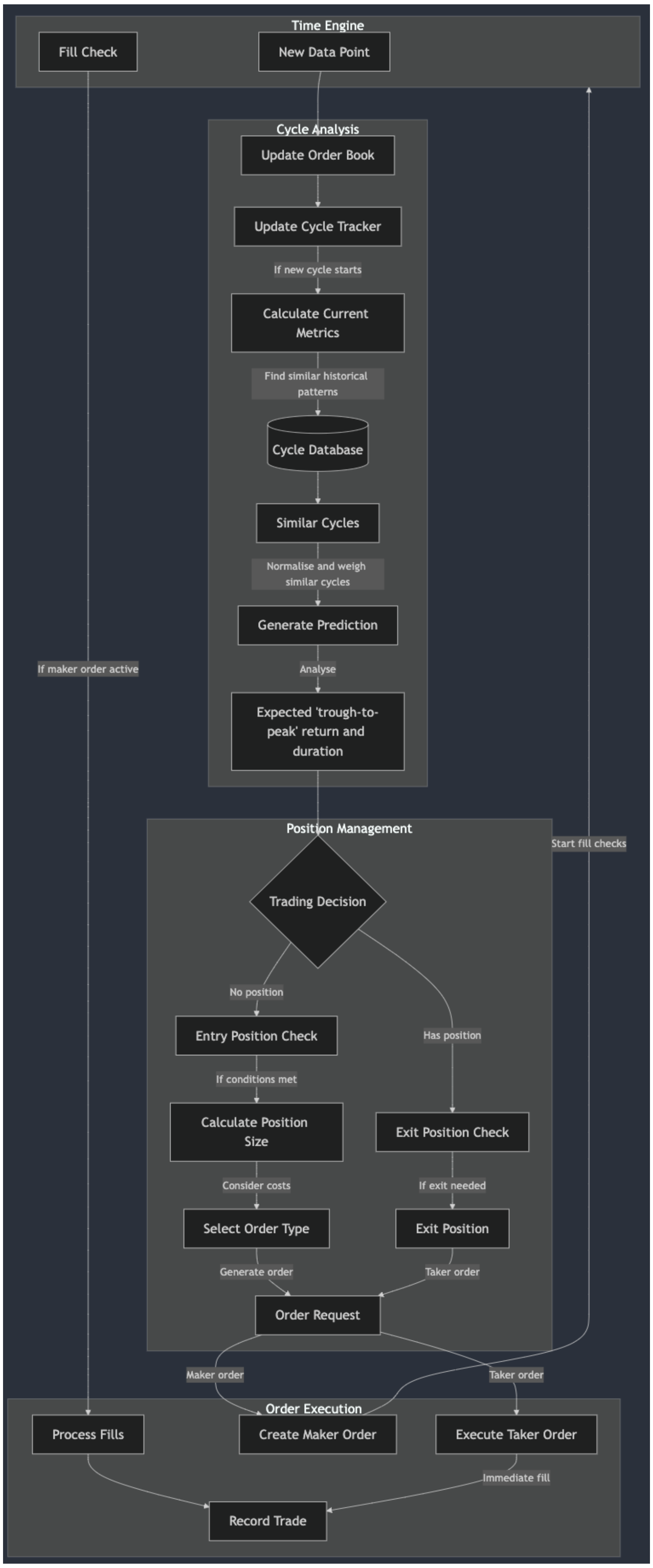
The approach I settled on (given time constraints) also makes assumptions about mean-reversion nature of relationship between ‘executable_rate’ and ‘exchange_rate’. But I tried to make trade timing and sizing much more targeted to avoid expensive whipsaws, meaning the algorithm attempts to trade only when there is a prediction of a move sufficiently large enough to offset execution-related costs.

I decompose the data into ‘over’ and ‘under’ **periods**, based on whether ‘executable_rate’ is higher or lower than then ‘exchange_rate’, respectively. Each period can then be assessed based on several metrics, like how many datapoint it lasted for, what was the maximum move achieved by two time-series, when did the peak occur, what was the maximum spread and when it occurred etc. You can see an illustration of this in the *Period* class of *period_cycle.py* module. The way I decompose data series into periods, ensures that I always get an alternating 'under'->'over'->'under'->... sequence of periods. This is intentional, because the next step is to go one level up in abstraction and consider a pair of consecutive 'under' and 'over' periods as a **cycle**. A cycle is important to the approach in that it defines a trading opportunity. By construction, a cycle will have a trough (lowest price in the 'under' period) and a peak (highest price in the 'over' period). This gives us an idea of a potential move (trough-to-peak) in the 'executable' price that we can exploit with a long position in the token. The key assertion here is that the key forces that govern how 'executable' rate moves around 'exchange' rate are persistent throughout the data history. The diagram below demonstrates the concept of a period and a cycle.



I start by allocating a section of the data (training section) to build up a database of cycles with their respective metrics. This section is used solely for this purpose. I use the rest of the data as part of the trading section. There, I implemented an 'online learning' approach - as new data points are experienced in the trading section, we gradually discover new cycles which get added to the cycle database, thus hopefully making the predictions more accurate. It is highly important that we avoid any potential 'look-aheads' in the trading section of the data.

More specifically, the process is as follows. Algorithm identifies whether we are in an 'under' period. It then continues to collect a certain amount of data about the current period. After enough data is collected, various metrics about this period are calculated (e.g. duration, price change and vol, spread change etc). Based on these metrics, similar historical cycles are searched for in the cycle database. Specific information that we are after is extracted from this potentially similar historical cycles, normalised weighted and averaged. This constitutes a prediction about the current cycle. This prediction is used to decide on an appropriate size and type of order that can be placed in such a way that the incurred costs are more than offset by the expected appreciation in the 'executable_rate' by some margin. Once the position is established, it is monitored and is closed when the predicted peak is reached or if cycle ends prematurely (e.g. prediction was inaccurate) or a stop-loss is hit. Algorithm then waits for the next cycle (i.e. a new 'under' period) to start.



This diagram provides a further high-level overview of the trading process and the flow of data.

Some of the key components are:

- **Time Engine**: this class is responsible for serving Time Slices that contain a timestamp and a datapoint. This component also makes it possible to account for partial fills of Marker orders that occur in between data points.
- **Cycle Analysis**: this section performs pattern matching with the cycle database, from which a prediction is constructed. Matching is currently implemented based on a Euclidean distance measure constructed using normalised averages of several early period metrics.
- **Position Management**: Prediction contains expected trough-to-peak move as well as when is it likely to start and how long it could last. This data is used to decide what order type is most appropriate and what position can be taken. The former is dictated by 1) how long it would take to build up a position via maker order VS how long the move will last for, 2) what size-driven costs are acceptable in the context of the predicted upside. This only happens if we don't already have an active position.
- **Order Execution**: This section deals with execution and recording of trades. There is a special logic for completing maker-type partial fills.

Implementation aspects

Given the project assumptions listed above, I tried to adopt the most realistic methodologies for:

a) incorporating execution costs

	Maker order	Taker order
1) Base price	mid	
2) Spread	-5bp (to get to best bid) +1bp (to be at the top of the book)	+5bp (to be matched with best ask)
3) Fee	0bp	+5bp
4) Slippage	0bp	(Order size / 2%) * 1bp
5) Execution price = base price * (1 +/- [spread + fee + slippage]) for BUY/SELL		

b) deciding on the order type

There are two factors contributing to a decision on what order type to chose: price impact and fill time. Since we have a prediction of time from current data point to trough in the cycle and then from there to the peak, we can check if the slow filling of a maker order is acceptable as we want to have most of the position in place before the move. Then we compare the relative price impact each order implies and determine the maximum size so that the expected move is enough to cover the costs + a margin.

c) accounting for partial fills

When a maker order is selected, a list of trades is created to mimic partial fills. The length of the list depends on the size of the order. Since we are told that trades filled every 3 seconds, while the prices are updated about every 2 seconds, we need to account for partial fills that would have occurred in between two datapoints when calculating Token position and portfolio P&L. This is where TimeEngine class (in time_manager.py) comes in, as it creates additional slices in time when only partial fills are processed.

d) keeping track of P&L

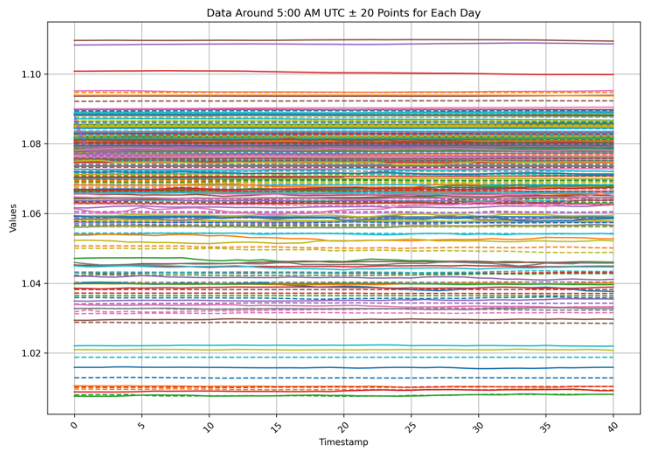
All of the performance metrics are calculated after the simulation has finished using the provided list of trades. This is done using methods in performance.py. At each time point, if any trade needs to be executed, we calculate the new amount of Tokens that needs to be held based on the trade size (as a percentage of previous portfolio USDT value) converted at the execution price. Portfolio is then marked-to-market using prevailing Bid price (since we can only be long Token) back to USDT. P&L is then calculated as the delta in portfolio value between time points.

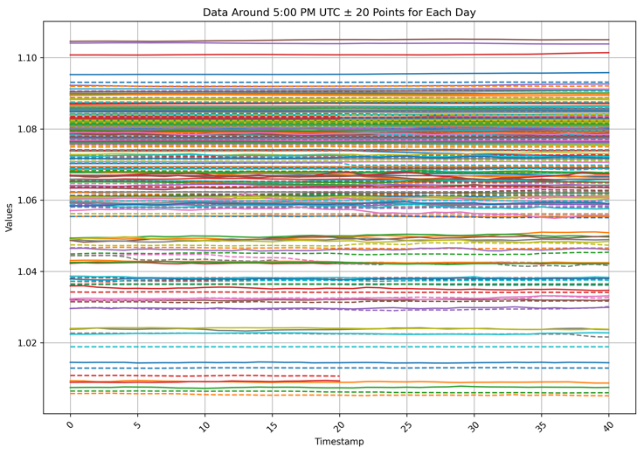
Simulation results

Coming soon – simulation over the full 3.6 million data points is taking a long time to run. I am currently in the process of re-factoring the codebase to allow for batch processing.

Additional project questions

1. It would be useful to have external information of several kinds. Firstly, the order book dynamics is very simplistic, so having more visibility into the order book depth, trade flows and inventory levels would be help in getting a deeper understanding of some of the driving forces behind the ‘executable’ rate. Secondly, if ‘executable’ rate is representative of a particular centralised exchange, same token is probably traded on other exchanges. Thirdly, derivative markets could offer some insightful front-running information, like perps funding rate or option implied volatilities. Finally, other data like market sentiment indicators and macroeconomic data could help provide a broader market context.
2. I am not sure what is the special meaning of 5am (or 5pm) UTC. Data indicates no special price dynamics around those times, e.g. price convergence. (assuming provided timestamps are in UTC format). Charts below show ‘executable’ (solid) and ‘exchange’ (dashed) rates.





However, if I assume that I can now execute at a reference rate (e.g. similar to 4pm London fixing for FX), then this has significant ramifications for the strategy. Firstly, we can worry much less about the order book dynamics, since I can assume no slippage at the reference fix rate. Secondly, the time scale would change as I can now target daily moves, thus shifting focus to much larger cycles. This in turn opens more opportunities to capture larger moves (with larger positions going into the fix). Thirdly, the lower execution costs would improve overall performance. On the flipside, the positions would need to be held for longer periods.

3. With the current setup, maker orders are preferred if time is not of the essence due to their lower price impact. Change the maker/taker fees from 0/5 to -5/10, would make maker orders even more attractive. With the new 5bp rebate, cycles with smaller expected trough-to-peak moves might become profitable enough with maker orders. The increased emphasis on longer fill time might require looking at cycles over longer-term scale and with higher levels of confidence.

Further research & potential improvements

I would like to mention some other ideas and considerations I had while working on this project, which, given more time, could be worth exploring.

- Strategy-related:

    - I have played with an idea of assigning confidence levels to predictions, but this requires further thought. This confidence level can impact position sizing and urgency. It can also decay with time, requiring a regular review/regeneration of the prediction.
    - We can explore using more sophisticated approaches for cycle pattern matching (e.g. some supervised ML techniques like Random Forest or unsupervised clustering).
    - Metrics used for pattern matching can also be broadened out to include other external data.
    - Accuracy of the predictions needs to be tracked with a feedback mechanism to impact pattern matching and confidence levels. This would allow the algorithm to more efficiently capture the shifts in market regimes.
- System-related:
    - As algorithms get more complex, they take more time to update. When working in a high-frequency setting, this needs to be factored in when building any backtesting engine. For example, if at each increment algorithm's logic (e.g. optimisation or inference of some sort) requires N milliseconds to compute then this needs to be modelled in the simulation to mimic realistically what output the strategy can act on in the markets.
    - Running time at the moment is too long - a more efficient memory usage should help when dealing large quantities of data, together with some optimisation of commonly used methods.
- Code-related:
    - Of course, comprehensive tests need to be written for the trading system
    - More detailed docstrings should be added too