CITS4404 Team Project

Building AI Trading Bots

Context and Aims

Automated algorithms, or *bots*, now perform a substantial percentage of the world's trading. It is difficult to know the exact numbers (and depends how broadly you define algorithmic trading) with some estimates suggesting around 70% of share trading is performed by bots, with digital assets perhaps higher at around 80%. Trading bots can work 24/7, react quickly to changes in the markets, and are not subject to human emotions that some suggest can lead to poor trades. Trading bots can be sold or rented on subscription. There is no doubt that trading bots are big business!

In this assignment we will use adaptive AI techniques to build trading bots. We'll do this for the digital asset Bitcoin — the world's hardest currency — but the same bots could be used for other currencies, for crypto tokens, for stocks, commodities and the like. The bot itself is "agnostic" to what the underlying data represents, although it would be optimised differently for different domains.

In this assignment you will be given the building blocks of a bot that utilises the most widely adopted type of trading signal. We will see how building the bot is a multi-dimensional optimisation problem.

Nature-inspired algorithms (or *adaptive algorithms*) are designed to seek good solutions to difficult, high-dimensional optimisation problems. They typically use a population of candidate solutions that are adapted according to some *heuristics* that seek to find improved solutions.

The practical aim of this project is to apply one or more nature-inspired algorithms to develop an effective trading bot. In doing so, we seek to learn more about problem representation and its impact on hypothesis space size and difficulty, as well as nature-inspired algorithms and how they compare with each other and with more traditional approaches.

The structure of the assignment follows a typical research project, albeit in a shorter timeframe — review of relevant literature, implementation and experimentation, and evaluation. For simplicity is it divided into two parts:

- Investigate and compare a number of nature-inspired algorithms, focussing on their distinguishing characteristics that will determine their promise for the task.
- Compare the performance of selected algorithms by developing and testing your trading bot.

Part 1 includes a milestone deliverable. It is expected that work on Part 2 can begin in parallel to Part 1 however. It is up to the team to apportion and manage the tasks.

Part I

Research Paper: Nature-inspired Algorithms

The first part of the project takes the form of a small *literature review*. It involves researching nature-inspired optimisation algorithms, one or more of which can be selected to implement and experiment with in Part 2. You should read through the specification for Part 2 prior to conducting the review to help you think about which algorithms may be most relevant and ultimately inform your algorithm choice. To accelerate the process a palette of suggested algorithms is provided.

The Mendeley Data repository includes a dataset uploaded by Alexandros Tzanetos that seeks to provide a comprehensive collection of all "nature-Inspired algorithms" (sometimes referred to as metaheuristic algorithms, population-based stochastic algorithms or generically as "evolutionary" algorithms) that have been published to-date as at January 2021. While it is almost impossible to produce a truly comprehensive list, it is an impressive attempt that includes almost 300 algorithms or variants, categorised according to natural phenomena they seek to mimic or use as an explanatory metaphor.

Getting Started

- Read through the Tzanetos *et al.*[1] and the repository's `Description.txt`[2] .
- Download the dataset and import it into a spreadsheet. Notice that the list is sorted by Category, Subcategory then name. Opening it as a spreadsheet will allow you to sort on other criteria, such as year, that may help you establish the chronology of the algorithms.

Some additional lists of algorithms can be found in Bayzidi *et al.*[3] (see Table 1) and Price *et al.*[4] (see Table 1).

A constraint on the algorithms you choose to investigate is that they must be *optimisation algorithms* (not for example, neural networks or machine learning algorithms). At least one (but likely all) will also be "population-based" algorithms, meaning they maintain a collection of candidate solutions and adapt these in some way to find better solutions.

The goal of Part 1 is to familiarise yourself with a number of algorithms and provide a synopsis on their distinguishing features. The number of algorithms summarised should correspond to the number of members of your team. You might ask each member to the team to research one algorithm and discuss it with the group prior to writing the report — if it can be communicated in an understandable way to the group then it is more likely to be understandable in the report.

[1] A. Tzanetos, I. Fister and G. Dounias, "A comprehensive database of Nature-Inspired Algorithms", *Data in Brief*, vol. 31, 2020, https://doi.org/10.1016/j.dib.2020.105792

[2] A. Tzanetos, "Nature-Inspired Algorithm", Mendeley Data, V2, January 14, 2021, doi: 10.17632/xfnzd2c8v7.2

[3] Hadi Bayzidi, Siamak Talatahari, Meysam Saraee, Charles-Philippe Lamarche, "Social Network Search for Solving Engineering Optimization Problems", *Computational Intelligence and Neuroscience*, Wiley, September 2021.

[4] K. V. Price,, N. H. Awad, M. Z. Ali and P. N. Suganthan, "The 2019 100-Digit Challenge on Real-Parameter, Single Objective Optimization: Analysis of Results", available on GitHub.

In the second part of the project you will select one or more of these algorithms to test and compare. It may be worth considering, in this context, whether you wish to explore "variants on a theme" in one Category, or explore more than one Category.

Deliverable 1

Synopses

For each paper you should provide a 1–1$\frac{1}{2}$ page synopsis.

As you work through the selected paper (and any supporting literature) you should seek to address the following questions, and reflect this in your own document.

(These are the kinds of questions you would be asking yourself as a Reviewer in deciding whether you would recommend the paper be included for presentation and publication. They are also questions you should ask yourself if submitting your own research as a dissertation or for publication.)

1. **What problem with existing algorithms is the new algorithm attempting to solve?**

   The algorithm may be directly presented in relation to previous algorithms, or it may be presented as a better solution for a problem in some application domain. In the latter case you may find it helpful to your exposition to give some brief context of the problem or application domain. The main focus, however, should be on the implications for the algorithm.

2. **Why, or in what respect, have previous attempts failed?**

   The work will have been done because of a perceived deficiency in previous approaches, or lack of solution to a problem. Your job is to understand and convey the motivation for the work.

3. **What is the new idea presented in this paper?**

   How does this paper seek to improve on previous work? What is its novelty?

4. **How is the new approach demonstrated?**

   This is the "method" of the paper. Is it demonstrated, for example, by an implementation and experiments, or perhaps by a mathematical proof, or human feedback/surveys from use cases? Does it build on previous work? Is enough information (or references, links) provided that you are confident the results could be replicated?

5. **What are the results or outcomes and how are they validated?**

   How are the results validated? Was the approach compared/benchmarked against previous results? Were the comparisons against other "evolutionary" approaches or more "traditional" approaches? Did the results show substantive differences? Were they improvements, or perhaps improvements on some metrics and not others?

6. **What is your assessment of the conclusions?**

   Did the authors claim positive results? (Note that negative results can also be useful results.) Do you think they were justified in any claims from the content of the paper? (This may have some bearing on whether you are likely to choose this algorithm for your own experiments.)

Conclusions

Having provided a synopsis of each paper individually, provide a brief account (not more than 1–2 pages) comparing the selected algorithms. What algorithms did you choose to review and why? Are they variants on a theme, or quite different in nature? Is there a relevant chronology or taxonomy? (A diagram may prove useful in some cases.)

Part II

Experiment: Design and Optimisation of an AI Trading Bot

1 Introduction to Technical Analysis

Technical analysis (TA) (sometimes called *quantitative analysis*) refers to trading strategies that focus on numerical measures, in particular price and volume history, to try to predict future prices — or at least the direction of the market. This contrasts with *fundamentals* trading that might focus more on broader factors (such as geopolitics and macro factors) and inputs and outputs (such as employment and earnings reports), that are perceived to have an impact on price.

Many (human) traders of stocks, currencies and commodities trade primarily on technical analysis, and are known as *chartists* (or sometimes *quants*). They are also often *momentum traders* who rely on trends in the data. Fundamentals traders, who often consider themselves to be *value traders*, on the other hand, sometimes pejoratively liken TA to horoscopes! We are certainly not advocating either approach! Nevertheless, TA forms the basis for not only billions in trade, but also for most trading bots.

The key aspects of technical analysis needed for this assignment follow. You may find it useful to follow the examples at the (free) information website CoinGecko, where you can play around with the technical indicators discussed.

OHLCV data

Figure 1 shows a *candlestick chart*, in this case for Bitcoin in US dollars (technically a USD stablecoin), known as a BTC/USDT *pair*. This plot shows one candle for each day. The same can be done for different time periods, for example hours or weeks.

Each candle shows the open and close prices (thick line) and the intra-day high and low (thin line). The histogram at the bottom shows the volume of transactions in that time period. This standard representation of the historical data is also known as an *OHLCV (Open-High-Low-Close-Volume)* data.

You can see the OHLCV data for Bitcoin at CoinGecko by clicking on "TradingView". You will see that by default it gives you 1-hour candles. You can change this by selecting different time periods at the top of the graph window.

Figure 1: Candles for BTC/USDT

TA Indicators

TA seeks to identify trends, patterns or distinctive features in the historical data that may indicate future price movement. The data generated to represent these patterns are called *(techincal) indicators*.

For example, one of the simplest indicators that many would be familiar with is a *simple moving average (SMA)*. This is a lagging indicator, because it averages data over a window of time that has passed.

You can add an SMA to the current data at CoinGecko in the TradingView window by clicking on "Indicators" and choosing "Moving Average Simple" from the dropdown menu. You will see that a new line is added that is smoother and slightly delayed from the price data. In the top left of the graph it will say something like "SMA 9 close". This tells you that the SMA is calculated from the closing price of the last 9 candles. By double clicking on "9 close" a menu will appear where you can change the duration (or "window" size) of the moving average and see the resulting changes on the SMA line. Notice that a shorter window will track the price signal more closely but be more erratic, while a longer window will smooth out more of the small deviations.

An *exponential moving average (EMA)* also summarises data over a period of elapsed time, but gives greater weight to more recent values. Again, you can use the TradingView to compare these yourself.

Figure 2 shows two moving average indicators: the green line has a greater delay and more smoothing (lower frequency), while the orange line reacts more quickly to changes in the data (higher frequency).

Figure 2: A chart showing two technical indicators with buy/sell triggers.

A simple trading *strategy* is to buy when the higher frequency indicator crosses above the lower frequency indicator and sell when the reverse happens. The blue crosses in Figure 2 indicate where the trades would take place using this strategy. We can say that these *trigger* a buy or sell.

The idea is to catch the uptrends and skip the downtrends. You can see that it works quite well for this example, although its less effective in the middle period where the market "chops sideways", especially if there is a cost involved with each transaction. It's a compelling story, but if it were that easy, everyone would be rich![5]

One more thing to notice is that these indicators do not effectively pick up the price surge on the 14th of February because the time delay is too long. A shorter window size may pick this up. On the other hand, a shorter window might do a lot of other unnecessary trading, at increased cost, due to market fluctuations such as sideways chop.

We can see therefore that the trading *strategy* is parameterised by the window sizes. Human traders have to "guestimate" the best values to use, but we will be a little more scientific and use optimisation methods.

There are plenty of other indicators — there is even a Bart Simpson Trading Pattern!

Of course, we are not advocating trading your hard-earned cash with any of these. Its always safer to get a job where you make money trading with OPM (other people's money)! Or perhaps sell you bots.

[5] You may like to consider this in light of the "No Free Lunch Theorem" that will be mentioned in lectures.

We will not need CoinGecko again because we are going to build and optimise our own TA indicators for our bot. Rather than arbitrarily choosing parameters such as the SMA window size discussed above, we will use our adaptive algorithms to seek optimal values based on historical data. We can also use algorithms to choose what our buy and sell triggers are constructed from.

2 The Bot Building Blocks

Our bots will be constructed from filter units that implement various weighted moving averages (WMAs), along with simple arithmetic and comparison units.

2.1 Weighted Moving Averages

To understand the WMA units, we consider a time series of data points $P = p_0, p_1, p_2, \ldots$

The time series could represent, for example, the weekly, daily or hourly closing prices of an asset. In the following we will assume daily prices just as an example.

Simple Moving Average (SMA)

The *simple moving average* at any point p_n is simply the average of the N data points preceding (and including) p_n:

$$\text{SMA}_n = \frac{1}{N} \sum_{k=0}^{N-1} p_{n-k} \tag{1}$$

We can think of this as a weighted sum of all the data points in a moving "window" of size N, with all points given equal weight of 1, normalised by the sum of all the weights, N. (Equivalently we can think of a window of N weights each of magnitude $\frac{1}{N}$.)

If we calculate the SMA for each point p_n, we generate a new series, or "signal". Figure 3 shows two SMA signals using two different window sizes — 10 time units (in this case days) and 40 time units.

Notice that the SMA provides a *smoothing* effect. It can also be understood as a way of filtering out the small fluctuations and revealing the "big picture" trends (more on this below). Furthermore, we can use the window size N as a *tuning parameter* to trade off responsiveness to the signal against the degree of smoothing.

Secondly, notice that the SMAs are *lagging indicators* — they change after the changes in the (price) data.This is because in Equation (1) the "right hand side" (RHS) of the data is aligned with the RHS of the window (as opposed to, say, the centre of the window). This is necessary because we would like to make a decision on the next day's trade, not wait $N/2$ days to fill the window. As can be seen in the figure, therefore, the size of the lag increases with the window size.

In practice it is inefficient to generate the SMA for each data point using Equation (1). One approach that is more efficient, and as we shall see, more versatile, is to use *convolution*.

Consider a second time series (known as a "boxcar function") defined by:

$$K = \frac{1}{N} \cdot \begin{cases} 1 & \text{if } 0 \leq k < N \\ 0 & k \geq N \end{cases} \tag{2}$$

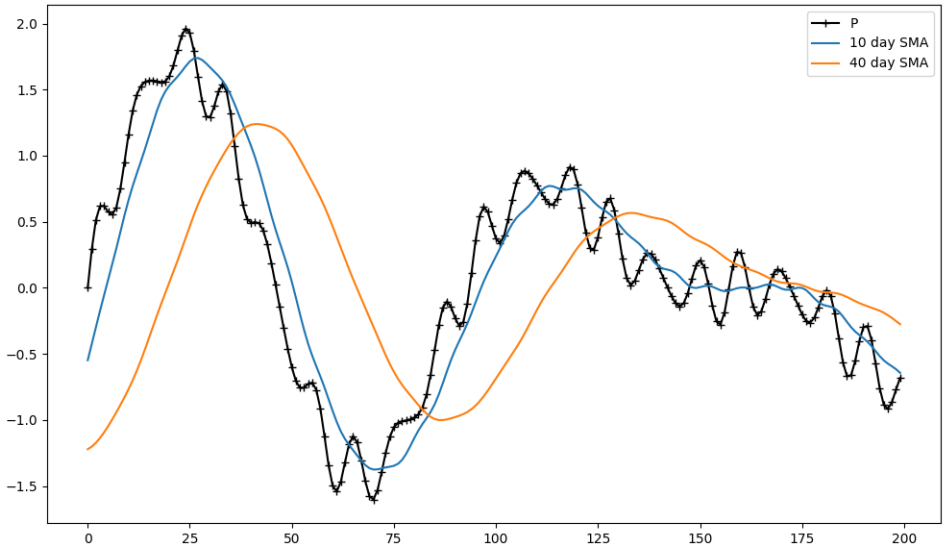

Figure 3: Smoothed signals using 10 and 40 day SMAs

Then the SMA is the convolution of the two series:

$$\text{SMA} = P * K \tag{3}$$

Technically K is a *filter* (sometimes called a *kernel*). In this case K is known as a *low-pass filter* because it averages out higher frequency changes in the signal P while revealing the lower frequency trends.

Finally, there is one edge-condition that we have so far brushed over. It is convenient to start all our moving averages at time zero (as shown in the figure), however P is undefined prior to $n = 0$ so the window is not filled. We can get around this by "padding" the signal up to the window width. There is no single "right" answer for how to do this — we will "flip" the first part of the sequence over (rotate it 180°), so that if for example the series is rising from p_0, it will also be rising into p_0.

We can achieve all of this with a small amount of numpy code.

```
def pad(P,N):
    padding = -np.flip(P[1:N])
    return = np.append(padding, P)

def sma_filter(N):
    return np.ones(N)/N

def wma(P,N,kernel):
    return np.convolve(pad(P,N), kernel, 'valid')
```

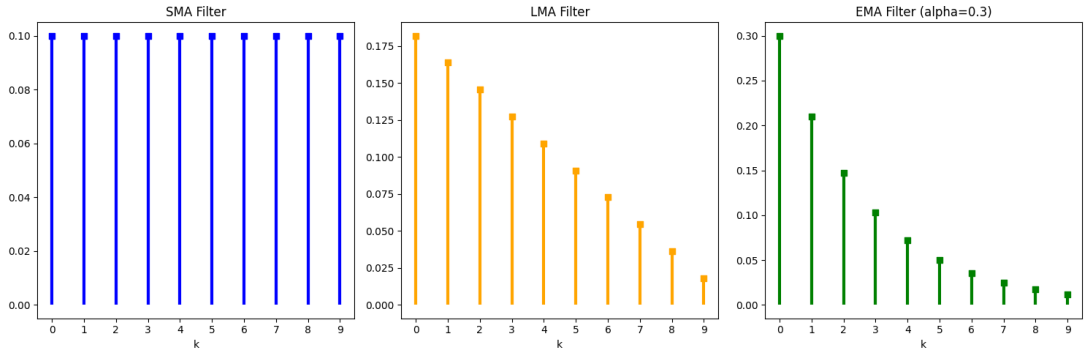

Figure 4: Various weighted moving average filters

To generate an N-day SMA we can simply call:

`wma(P,N,sma_filter(N))`.

Linear-Weighted Moving Average (LMA)

The SMA gives equal weighting to all time points back to the window size, then suddenly drops off. It may seem more intuitive to give a higher weighting to more recent points, and have the weighting reduce as the proximity to the current point reduces. One way to do this is to use a *linear-weighted moving average*.

Following common use in finance, we will have the weights reduce linearly to zero over the width of the window (although this is not the only option).

While we can easily write a summation equation similar to Equation (1) for the LMA, we'll instead jump straight to a convolution. This is where we see the beauty of this approach. By treating our WMAs as a convolution, we can get different moving averages simply by swapping out the filter!

For our LMA, we simply use a "triangular" filter:

$$K = \frac{2}{N+1} \cdot \begin{cases} 1 - k\frac{1}{N} & \text{if } 0 \leq k < N \\ 0 & k \geq N \end{cases} \tag{4}$$

The filter is shown in the centre of Figure 4, compared to the SMA filter on the left. The term $\frac{2}{N+1}$ again normalises the filter so that the weights sum to 1.

Equation (3) doesn't change!

In our code above, we mirrored this by also separating out the filter definition (reflecting Equation (2)) from the convolution step in Equation (3). So all we need to do to implement an LWA is write code for `lma_filter(N)` defining the filter according to Equation (4).

Exponential Moving Average (EMA)

We can take the idea of the LMA one step further. While the SMA magnitude changes abruptly after N "impulses", the LMS *slope* changes abruptly after N impulses (we could think of as like

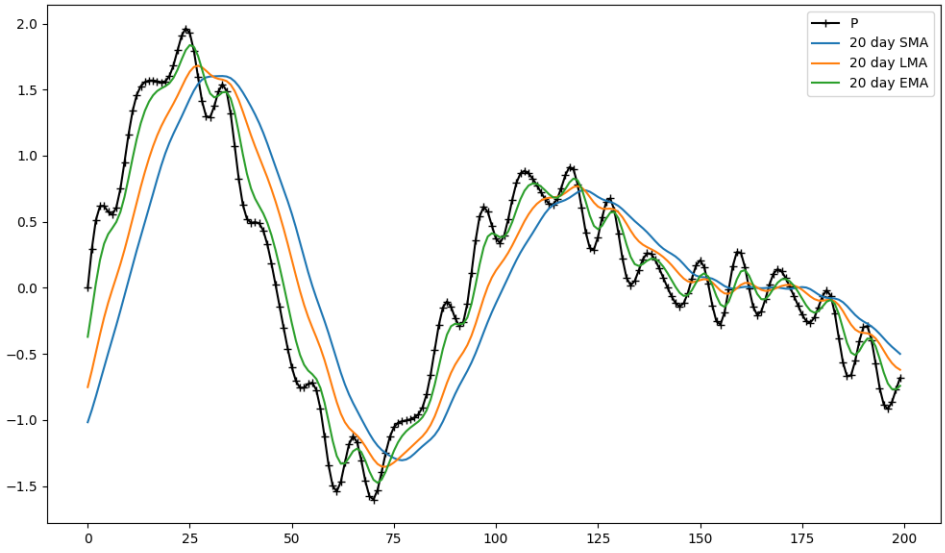

Figure 5: Comparison of SMA, LMA and EMA

the SMA as having a discontinuity in the magnitudes and the LMA as a discontinuity in the first "derivative" of the magnitudes).

An alternative is to use an exponential rather than a linear decay. This gives even greater priority to more recent data points, and tails off more gently to zero.

Strictly speaking an exponential moving average would use a filter as long as the data series itself, however we can approximate it over a fixed-length window. We define our filter by:

$$K = \alpha \cdot \begin{cases} (1-\alpha)^k & \text{if } 0 \leq k < N \\ 0 & k \geq N \end{cases} \tag{5}$$

Notice that we have two"tuning parameters" for the EMA — the window size N, and smoothing factor α which determines the rate of decay. A filter with $\alpha = 0.3$ is shown in Figure 4. You may wish to try different values of α in your own code to see the effect it has.

Your Own Filters

Each of the filters has their own characteristics. Figure 5 shows a comparison of a 20-day SMA, LMA, and EMA. It can be seen that the LMA responds more quickly than the SMA to changes in the signal p, and similarly the EMA responds more quickly than the LMA. On the other hand, the SMA provides the most smoothing of short-term fluctuations.

The response of each filter can be tuned by the window length N. The EMA has an additional parameter α. When it comes to analysing price signals, we could also consider the timeframe (eg. hours, days, weeks, months, years) as an additional parameter that can be optimised.

The way the averages are applied as convolution filters makes it very easy to try other profiles of your own, with their own parameters, and explore their response if you wish.

It is worth noting that one could even use a more general filter in which the weight given to each of the impulses in the window is a separate parameter. This would subsume all of the above filters since each could be described as an instance of this general case. The down side of this is that it would introduce many more parameters. This is a classic case of a trade-off between the expressiveness of the hypothesis space and the size (and in this case the dimension) of the hypothesis space!

2.2 Crossover Points as Buy and Sell Signals

There are innumerable conjectures, and their implementation in bots, for how weighted averages can be used to pick trades. The vast majority of these use some variation of the crossover between a short-term signal and a longer-term signal. The idea is that when the short-term signal crosses the long-term signal it suggests the breaking of a longer term trend. When the short-term signal crosses below the long-term signal it is sometimes called a *death cross* (a sell signal), while the short-term signal crossing above the long-term signal is sometimes called a *golden cross* (a buy signal).

There are two most common approaches to choosing the two signals. The first is to choose two moving averages of the same type but different durations (or window sizes N), with the smaller duration being the more responsive. We have already seen from Figure 3, for example, that a shorter term (smaller N) SMA reacts more quickly to higher frequency changes in the input than a longer term SMA.

The second is to choose a different type of WMA that is more responsive. The most common would be to choose an EMA for the more responsive signal, and an SMA for the smoothed trend signal. We have also seen an example of this in Figure 5. In both cases we can see intuitively that the crossover points would seem to be reasonable places to buy and sell.

Compound Filters

The WMAs provide building blocks for designing functions from the price signal to generate buy or sell signals. We can add other small "units" or "blocks", for example a subtraction block to obtain a difference signal for identifying crosses. The difference signal will cross the x-axis from positive to negative when the short-term average crosses below the long-term average.

Another useful filter, therefore, is one that recognises a change in sign of the difference signal. We can achieve this by convolving with a filter such as:

$$K = \frac{1}{2} \cdot \begin{cases} 1 & \text{if } k = 0 \\ -1 & \text{if } k = 1 \\ 0 & k > 1 \end{cases} \tag{6}$$

We'll look at how this is used in a concrete example.

2.3 Building a Bot

Figure 6 shows a simple example of a trading bot. It subtracts the 20 day SMA series from the 10 day SMA series, generates a new series by taking the sign of the result (1, -1, or 0), and convolves this with a filter that responds to a change from negative to positive (Equation (6)).

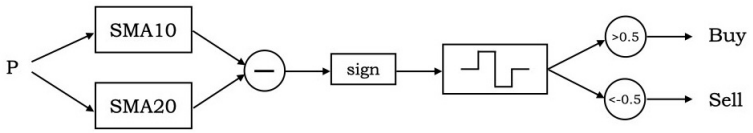

Figure 6: A simple trading bot

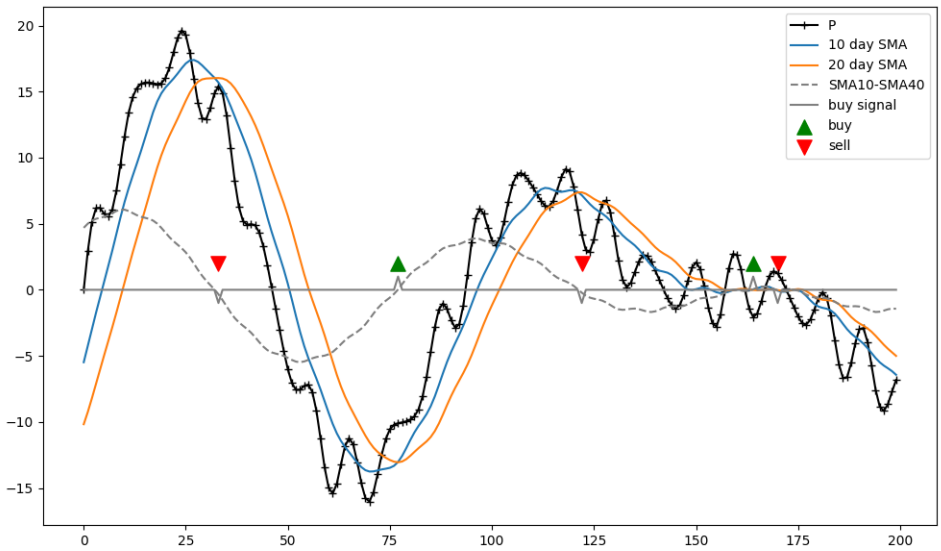

Figure 7: Trades made by the simple trading bot

Figure 7 shows the simple bot operating on the signal from Figure 3.

A more elaborate and very widely used trading signal is the MACD. This technique creates a MACD series from the difference of two EMAs (for example, in the common case of MACD(12,26,9), from the difference of the 12-day and 26-day EMA). The "MACD Signal" or "average" signal is in turn the created from the (9-day) EMA of the MACD series[6]. When these two lines cross (or the difference changes sign) is is considered a bullish (buy) or bearish (sell) signal. Alternatively the sign change (zero crossing) of the MACD line itself may simply be used.

It is easy to see how any of these bots can now be implemented with the tools we have.

[6]The language is a little confusing but the important thing is that it is the EMA of the difference of two EMAs.

3 The Experiments

Generalisation and Design

There are a number of ways that you can generalise (and hence optimise) the basic bot components in your own design.

In Figure 7 the two signals that we use to find the crossover points are simple SMAs. There is no particular reason to believe that an SMA, or for that matter an LMA or EMA, with a given window size, will perform the best. The conventions used are most commonly people's best "guesstimates" at what works for them (unless they are a professional operation doing what we are doing). Similarly, for the use of the 12, 26 and 9 day windows in the MACD.

We have already seen a number of tuning parameters for individual units that we can seek to improve. We can generalise this more by combining different types of unit and allowing optimisation to choose how much of each one to incorporate. For example, we could replace both the "high-frequency" component SMA10 a with weighted sum:

$$\text{HIGH} = \frac{w_1\text{SMA}(d_1) + w_2\text{LMA}(d_2) + w_3\text{EMA}(d_3, \alpha_3)}{\sum_i w_i} \tag{7}$$

where d_i are the durations, α is the EMA decay, and w_i are weights that balance how much attention to give to each type of filter. By allowing the w_i to "float" freely, we let the optimisation algorithm choose how much attention to give each kind of filter based on historical data, rather than guess them ourselves.

It will be interesting to look after optimising to see how much weight the bot gives to each component — for example, does it consistently favour one or continue to draw from all three.

To optimise our bot in this case we need to optimise a 7-dimensional vector:

$$[w_1, \ldots, w_3, d_1, \ldots, d_3, \alpha_3]$$

Doing the same for the low-frequency component will give us a 14-dimensional vector to optimise.

For the MACD we have a third component that smooths the difference signal, leading to a 21-dimensional optimisation problem.

We can take this further. For example, the SMA, LMA and EMA actually impose somewhat arbitrary constraints on the weights of a WMA (that is, the profile of the filter in Figure 4). If we choose a WMA with a given duration, we could let each weight float freely and therefore optimise the profile of the individual WMA. Notice that this would *subsume* the SMA, LMA and EMA which would just be instances of a more general WMA. The downside is of course that this introduced more degrees of freedom and adds further dimensions to the optimisation problem. It is classic example of the trade-off between descriptiveness of our language and size of the hypothesis space that we talked about in lectures!

Another way that we could introduce more scope to our bot is to use optimisation techniques to adapt the configuration of our bot — which components we use and how they are connected. This is an example of using optimisation for the design itself, not just to tailor the weights within a design. This will also add complexity to our hypothesis space in the form of discontinuities — adding a component may cause a large "jump" in the quality of the agents' predictions (that is, the evaluation).

Really the sky is the limit — we are constrained only by our imagination and computational cost!

Choosing Algorithms

Once you have chosen the scope of the design (and hence the hypothesis space) we want to find the best performing model that we can within that space. To achieve this we need to choose the algorithm(s) and an evaluation function.

The algorithms were primarily addressed in Part 1. The intention is to use one or more algorithms from those you researched. Trying more than one algorithm allows you to make an informed comparison between them, which may help elucidate important differences. You may if you wish add additional nature-inspired algorithms if the experimentation suggests that the original algorithms are lacking. They should again be population-based algorithms — remember the primary goal is to learn about the algorithms, not just to obtain a good bot.

A more complete comparison could be achieved by also trying one of the single-state algorithms covered in lectures (eg. direct methods/stochastic/single global optimisation algorithms). In this case you would need to compare the outcome on a fixed number of evaluations (as opposed to, say, generations).

Evaluation of Bots

Bots can be evaluated on historical data. Evaluation experiments should be repeatable as follows. Assume that the bot is to be evaluated over a period of price data $P_E = p_0, \ldots, p_m$. We assume that:

- the bot begins with $1000 USD (and zero bitcoin)
- each time the bot is holding cash and generates a *buy* signal, it trades all the cash (minus fees) for bitcoin at the current price
- each time the bot is holding bitcoin and generates a *sell* signal, it trades all its bitcoin for cash (minus fees) at the current price
- each transaction attracts a fee of 3%
- at the end of the sequence, the bot sells its remaining bitcoin at the final price

The *fitness* of the bot is the cash it is holding at the end of the evaluation.

We can think of bots as competing to implement the best strategy. This approach to evaluating a strategy or bot is known as *back-testing*. Of course, success on past sequences of data does not guarantee a strategy will be successful on future sequences!

You may if you wish use a set of sequences P_E for your evaluation. In that case the fitness would be the average or total score over the whole set — of course it must be repeatable and any two parameterisations (bots) must be compared on the same set. (For those who have studied deep learning, we are not for example using batch processing to generate some sort of probabilistic average over the sequences — we are seeking to find an optimum on a deterministic evaluation function.)

The time scale of your evaluation sequence(s) (for example, minutes, hours, days) is up to you. Indeed, components of your bot may use different timescales, and the timescale may be a parameter you optimise over.

Data

Real-time data can be accessed from most cypto exchanges, for example the Kraken exchange. Most offer some period of data for free without having to sign up for an account. A library such as ccxt can help with this.

To simplify matters, however, we will use as the base case a dataset from the kaggle repository (you may add to this if you wish). The Bitcoin Historical Dataset provides historical data between (variously) 2014 and 2022 daily, hourly and by the minute. It ranges from sequences of 2652 data points for daily data to around 600,000 data points for each yearly set of minute data.

You should optimise your agent on data prior to 2020, and preserve the data from 2020 onwards for final testing of your optimised bot. This will allow you to see how well your bot performs on *unseen* data. (Normally we could see how it performs over time on future data (or "forward test" it), but as its a limited time project, we'll act as if it is the start of 2020.) Typically this testing on unseen data could be used to compare your final bot against bots from other developers.

"Rules of Engagement"

While it is satisfying to produce a good trading bot, it is not the primary purpose of the assignment. Rather it is to:

- Understand the "pluggable" components as a *language* for designing bots.
- Use nature-inspired algorithms to generate instances (*hypotheses*, models) within that language.
- Use the algorithms along with an evaluation function, over real-world data, to adaptively improve those models.
- Understand and have practical experience with the trade-off between degrees of freedom (or richness of the models) and hence existence of better solutions, and the size of the search space consequent difficulty of finding them, and finding the right balance through experimentation.
- Be able to use an experimental design to assess (through fair comparisons) the algorithms.

To achieve this we have the following "ground rules":

- You may *not use trading bots or code for bots from any other source* — all coding for the bot(s) and their evaluation must be written by the team members as an implementation of the building blocks described in this specification.
- You may not use general optimisation libraries or optimisation code from other sources for this assignment.
- You may adapt for your use code provided in conjunction with a *specific research paper for a nature-inspired algorithm*, providing you acknowledge the code source in both your written and verbal deliverables.

4 Deliverables

Part 2 of the project has three deliverables: a video presentation, a report, and a code repository.

Presentation

Each team will be required to provide a video presentation, of up to 25 minutes, on their project. The presentation should include the following:

Algorithms investigated
A short overview of the algorithms you examined in Part 1, along with any other optimisation algorithms you tested in your practical work. You won't have time to go into each one in detail, but you should attempt to categorise them if relevant, and point out their distinguishing features. (Visual aids such as a table or graph may be useful.)

Bot design and parameterisation
A discussion of the considerations that went into choosing your bot configuration (and hence hypothesis space) and the parameterisation that results from it.

It is anticipated that there will be some iteration here when experimenting with different configurations, the degrees of freedom (dimensionality of the search space), cost of evaluation and practicality of search. You may wish to talk about what you tried and what you learnt from your experimentation.

Algorithm selection
Your algorithm selection for the experiments should be motivated by your design and parameterisation. Why did you choose to test the algorithms that you did? Here you can go into a little more detail. A high level explanation of the optimisation algorithm(s) chosen and how they work. The explanation may make use of diagrams and/ or pseudocode rather than actual code. Issues in the implementation may be discussed, but a full methodology is not required.

Again, it is anticipated there may be some iteration. For example different meta-parameters of the algorithms (such as population size) will prove more viable than others. What did you learn from the experiments?

Experiments and evaluation
Your evaluation/testing regime. Again there may be trade-offs, for example using more data may, at least theoretically, lead to better results, but may be prohibitively expensive for the number of evaluations required.

Results
Presentation of results. Visual representations are preferred (images, tables, animations, simulations).

Conclusions
Conclusions from your work.

Remember, you have been presented with an *open-ended real-world problem* (that many people are seeking to "solve"). There will always be more that could be done. It is not ultimately about whether you "solved" the problem, but what you learnt in the process.

Report

The report should complement your video, including extra details that could distract from the flow of the presentation. This includes references and links. It may include additional results that didn't "make the cut" in the video (which should focus on the key themes), diagrams or pseudo-code. It should not include code, but may refer to code in the repository.

The report does not need to repeat information from the project specification — you can take this document "as read". You may assume the reader is already familiar with algorithmic trading and TA, and with the terminology and concepts in the unit's lecture notes. In other words, you should include enough introduction for the document to flow, but you don't need the detailed introduction you would normally require for a "stand-alone" paper.

You may also refer back to the deliverable for Part 1, you do not need to repeat it.

Code

The code, and any results files, should be provided in digital form. Depending on the size, this could be a directory uploaded to the LMS, as part of a project in CoCalc, or linked as a private repository on GitHub or similar. The code should allow reproduction of the results. Brief instructions for running the code can be included in a README file or notebook.

The report should provide a pointer to where the code, and instructions, can be found.

The coding will not be marked *per se*, but the results should be repeatable if required.

5 Practicalities

Working as a Team

- This is a group project with allocated teams of 4–6 members (much like a workplace). You may wish to choose a team name to help distinguish the team in queries, marking and so on.

  How well the team works together can be as much a factor in the success of the project as the technical aspects. A good mindset is to seek to ensure everyone will learn from the project.

- *Planning*: This is a large project and to get the most from it in the time available, it is recommended the team meet early and plan out tasks, time allocation for tasks, and initial allocation of activities or responsibilities (these may be revisited as the difficulty of various tasks becomes better understood).

- *Communications*: It is important that the team regularly communicates so that everyone is "on the same page" and any emerging difficulties can be addressed. You should decide upfront how the team will communicate both in-person (eg. how frequently will there be in-person meetings) and electronically.

Submissions

- The report is limited to a maximum of 3000 words excluding diagrams and references. Text beyond the maximum word count will not be marked. A word count must be included on the title page, along with the team name or number and the *names and student numbers of the team members*.

- The report must be submitted as a pdf file. If submitted as a file, the video should be in mp4 format unless otherwise arranged. Alternatively the video may be submitted as a link to a streaming service such as YouTube.

- Referencing should follow the IEEE style. As usual, all materials obtained from other sources should be referenced.
- This assignment is group work. Submitting work is considered a declaration that the work submitted is entirely the work of the team members except as referenced.
- A Teams channel will be provided for queries relating to the assignment.

Deadlines

Deliverable 1: 11:59pm Friday 11th April

Deliverable 2: 11:59pm Friday 9th May

Late assignments will penalised according to the University's standard rules for late submissions described in the Unit Outline.