Forex Arbitrage with Bellman-Ford

Github link

1. Background

Foreign Exchange Overview

The foreign exchange (FX or forex) market is a decentralized network of banks and financial institutions trading pairs of currencies. This market trades 24 hours per day and five days per week across four sessions, New York City, London, Tokyo and Sydney. Rather than trading assets at one centralized exchange, each member of this network offers their own bid and ask spreads for various currency pairs.

Currency pairs are listed as [base currency]/[quote currency]. Bid and ask prices are listed in terms of the quote currency. Buying a currency pair converts the quote currency to the base currency; selling is the inverse transaction. For example, selling EUR/USD at a price of $0.95 means converting 0.95 Euros to 1 US Dollar. This means we can define the exchange rate from EUR to USD as the market bid price since that is the price at which we can sell. The opposite exchange rate equals either the bid price of USD/EUR or the multiplicative inverse of the ask price of EUR/USD.

The goal of the project is to find and analyze cyclic arbitrage opportunities in the forex market. An arbitrage opportunity is a cycle of currency pairs that we can trade back to the initial currency and make a per-unit profit. For example, suppose we can sell USD/EUR for 0.9, EUR/GBP for 0.72, and GBP/USD for 1.4. Then, we could make a small profit as illustrated in Figure 1.

Figure 1. Example Arbitrage Cycle between USD, Euros and Pounds

Bellman-Ford

We sought to analyze the arbitrage opportunities that may arise when provided with quotes for several currency pairs over time. To do this, we construct a weighted, directed graph with each currency as a node. The edges represent the exchange rates, or in our case, the bid prices for each currency pair. Then, if we have a cycle in the graph, we can compute the product of the cycle's edge weights to find the profit of that cycle on a per unit basis. If this profit exceeds one, then the cycle is considered a potential arbitrage and in an ideal world we would fill several exchanges to capture the profit. This is also why we only consider selling each currency pair, so that in a successful arbitrage, a trader would end up completely neutral except for their base currency. That being said, in an arbitrage cycle, the base currency can be any currency involved in the cycle as the profit is independent of the entry and exit. Our initial attempt to find these arbitrage

opportunities used the Bellman-Ford algorithm for detecting negative cycles.
Bellman-Ford is a single-source-shortest-path algorithm that not only detects negative cycles, but also works with negative edge weights, unlike Dijkstra's, another shortest path algorithm. Negative cycles are important in the context of forex arbitrage as we can reduce our arbitrage condition to finding a negative cycle (Figure 2).

$$
\begin{aligned}
e_1 \cdot e_2 \cdots e_n &> 1 \\
\frac{1}{e_1} \cdot \frac{1}{e_2} \cdots \frac{1}{e_n} &< 1 \\
-\log(e_1) - \log(e_2) - \cdots - \log(e_n) &< 0
\end{aligned}
$$

Figure 2. Reduction of Arbitrage Condition to Finding Negative Cycle.

Our implementation relies on a graph of V currency nodes and their respective exchange rates, however we add a source node with edges of weight zero connecting it to each currency. This allows us to be non-deterministic in selecting a starting node when running our implementation of the algorithm. We keep two arrays, distance and predecessor, which store the distance from our source to each node and the nearest predecessor to each node, respectively. An example of the Bellman-Ford algorithm can be seen in Figure 3.

The algorithm then executes the following steps:

1. Iterate over all edges between currencies V - 1 times. An edge corresponds to a currency pair (c1, c2).

   a. If distance[c1] + weight(c1, c2) < distance[c2] then we update distance[c2] and predecessor[c2].
   b. Performing this V - 1 times ensures that all edges are fully relaxed.

2. Perform the prior step with one more iteration and if any edge relaxes, there must be a negative cycle, which we can reconstruct using the predecessor array

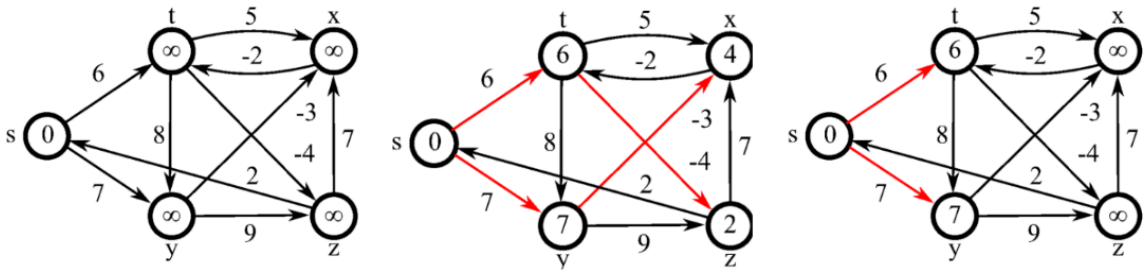
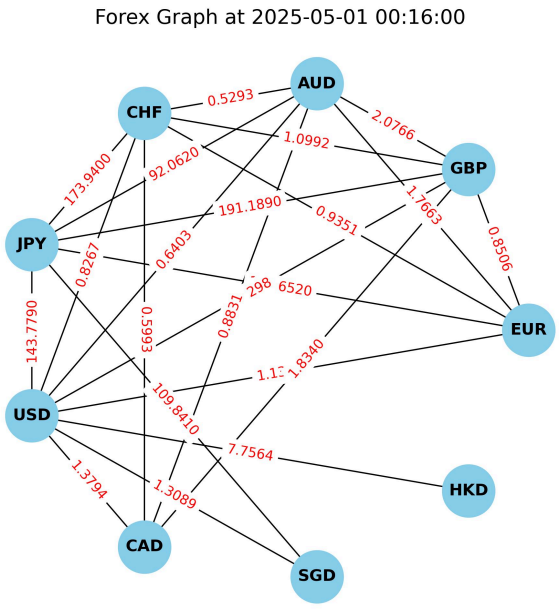


Figure 3. Example Steps of the Bellman-Ford Algorithm

2. Dataset

Histdata.com generously provided free access to their historical FX tick data, sourced from multiple financial institutions. This process consisted of individually downloading forex data of 23 currency pairs during the month of May 2025. In total, I ended up with roughly one billion individual quotes and 10 unique currencies with a one-second timestamp resolution. Unfortunately with our low resolution data, it is impossible to determine how long any one individual arbitrage opportunity lasts and thus we cannot say how quickly we would need to trade in order to take advantage of such an opportunity.

To keep our analysis tractable, I retained only the ten most liquid currencies: USD, EUR, JPY, GBP, CNH, AUD, CAD, CHF, HKD, and SGD. Upon isolating these ten currencies, I sorted the data by timestamp to provide a time series of the quotes. A sample view of our currency graph at a single timestamp can be seen in Figure.



3. Trade Implementation

Bellman-Ford

Code

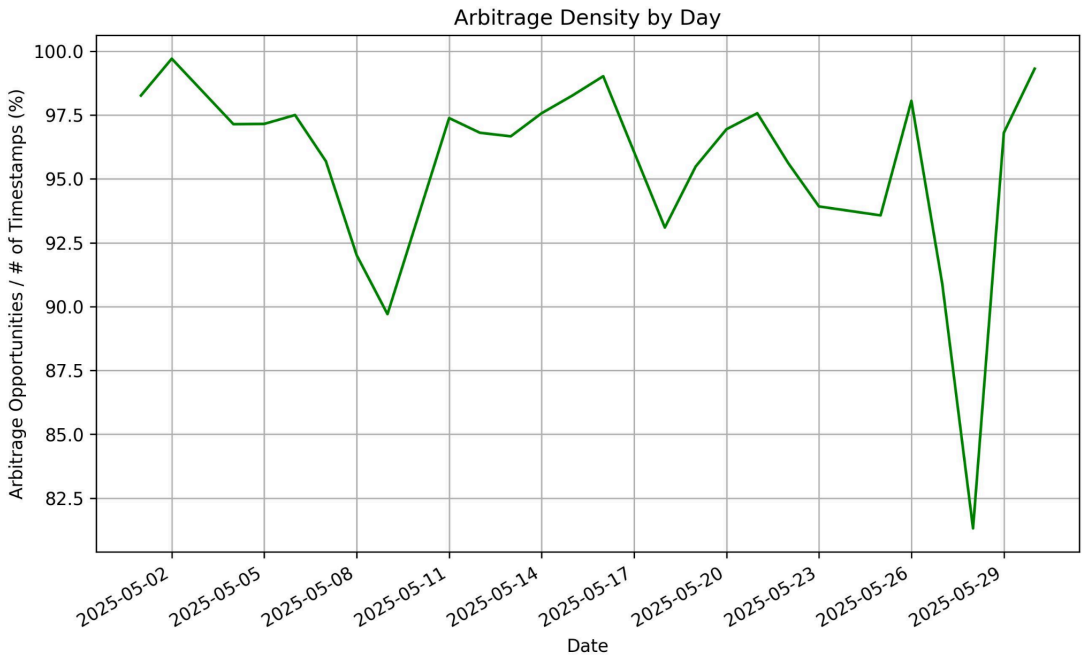
The Bellman-Ford implementation involves a class named ForexArbitrage with 3 primary methods: Update, IsArbitragePossible, FindArbitrageOpportunity. The ForexArbitrage object is initialized with a vector of currencies represented as strings. These string currencies are mapped to numeric IDs. The class stores a double **graph_ which stores the graph connecting

currencies with weights equal to the negative logarithm of the corresponding exchange rates. The Update method simply updates the graph using a new exchange rate for a given currency Pair.

The IsArbitragePossible method contains the Bellman-Ford implementation. Since the Bellman-Ford algorithm must begin with a source node, a sentinel node with weight 0 to every other node (currency) is used as the source node. First, the distance array is initialized with values of infinity. Then, the edge relaxation process updates these distances. The edges are relaxed C times where C equals the number of currencies, aligning with the Bellman-Ford algorithm. If no edge relaxes in any of the C relaxation attempts, the function immediately returns that there is no arbitrage opportunity available. Then, the function attempts to relax the edges one more time. If an edge relaxes, there must be a negative cycle, so the function immediately returns true; otherwise, the function returns false.

The FindArbitrageOpportunity method contains the same Bellman-Ford implementation extended to reconstruct the negative cycle (the arbitrage cycle). If a negative cycle is detected, the function iterates backwards using the predecessor array to find a node inside the negative cycle. Then, the function iterates forward again to reconstruct the cycle in the correct order. When the function completes iterating through the entire cycle, the function returns the detected arbitrage cycle and profit. Note that this function will always return the first negative cycle that it finds. This means that if there exists more than one arbitrage cycle, it will ignore all the others, whether or not they are more profitable. Further, this means the found arbitrage cycles are inadvertently tied to the order in which the currencies are passed to the ForexArbitrage object

I wanted to figure out how many arbitrage opportunities I found in relation to the amount of data, so we computed the ratio of the number of opportunities per unique timestamp for each day, giving us an indicator of what proportion of each day we would be able to arbitrage Figure. This indicates that there are in fact a reasonable amount of arbitrage opportunities to take advantage of given our dataset, though not all might be profitable after accounting for fees and order book volume.



Upon finding over thousands of total arbitrage opportunities in our dataset with the selected ten currencies and cycles of length five or smaller, I plotted the average net profit on a logarithmic scale with respect to the date.