Bitcoin Arbitrage Loop Opportunity

Kindly note, to understand how to run the python program, go to section 4 below

1.1 Problem Evaluation

Given the currency conversion rates, I have converted these to a table for ease of explanation

	BTC	**EUR**	**JPY**	**USD**
BTC	1.0	101.16588	14039.60311	137.15075
EUR	0.00969	1.0	113.35804	1.10639
JPY	0.0000824	0.00772	1.0	0.00992
USD	0.00816	0.76448	100.69101	1.0

Given these pairs,

1. **For a constant conversion (ie no loss or gain):**

$$(\boldsymbol{a} \rightarrow \boldsymbol{b}) * (\boldsymbol{b} \rightarrow \boldsymbol{a}) = \boldsymbol{1}$$

Where a, b = currency notations

2. **For Arbitrage Gain opportunity:**

$$(\boldsymbol{a} \rightarrow \boldsymbol{b}) * (\boldsymbol{b} \rightarrow \boldsymbol{a}) > \boldsymbol{1}$$

E.g: $(BTC \rightarrow JPY) * (JPY \rightarrow BTC) > 1$

14039.60311 * 0.0000824 = **1.1568633**

3. **For Arbitrage Loss:**

$$(\boldsymbol{a} \rightarrow \boldsymbol{b}) * (\boldsymbol{b} \rightarrow \boldsymbol{a}) < \boldsymbol{1}$$

E.g: $(BTC \rightarrow EUR) * (EUR \rightarrow BTC) < 1$ = 101.16588 * 0.00969 = 0.9803

1.2 Arbitrage Gain Opportunity: (A->B) * (B-> A) > 1

Mathematically, log (C * D) = log C + log D

Thus, taking log of both ends of: $(a \rightarrow b) * (b \rightarrow a) > 1$

$$\log\big((a \rightarrow b) * (b \rightarrow a)\big) > \log 1$$

$$\log(a \rightarrow b) + \log(b \rightarrow a) > 0$$

Taking the negative of both sides

$$-\boldsymbol{\log(a \rightarrow b) - \log(b \rightarrow a) < 0}$$ **Negative Cycle Eqn.**

The sum of the (-log of the) component paths resulted to a negative value; thus we have a **NEGATIVE CYCLE.**

So, an arbitrage opportunity is only available when we have a negative cycle within the directed graph.

1.3 Negative Cycle Determination: Bellman-Ford's Algorithm

How can you check if a graph has a negative cycle?

Solution Logic:

Representing the system as a Directed Graph, we assume all the currencies are nodes, conversion rates are weight of edges, and currency pairs are edges.

The Bellman-Ford's algorithm for a Directed Graph serves to find the shortest path between a source currency and all other currencies(vertices) in a weighted graph (of conversion rates). Fundamentally, for Bellman-Ford, there can be at most V-1 edges in a path from a starting node to any other node in the graph. If there are V or more edges in such path, then (there is a repeated vertex, leading to a cycle. It can handle negative edge weights, and detect negative cycles. A negative cycle is one in which the overall sum of the cycle becomes negative.

Since we have a negative cycle, we can utilize Bellman-Ford's Algorithm.

Converting 1 BTC for instance:

1. We have 4 currencies = 4 vertices (V), = 4 nodes

   BTC, EUR, JPY, USD

2. We assume that each vertex (other currencies, EUR, JPY, USD) are at infinity from BTC.
3. Note a simple shortest path from source currency to **any other** vertex can have at-most |V| - 1 edges. Example

   BTC→EUR

   EUR→JPY

   JPY→**USD** **ie** a total of 3 edges for 4 vertices

   However, to end at BTC (ie with additional edge of USD→**BTC**), implies a cycle.

   Starting with BTC and ending with BTC translates to 4 conversions(edges) ie V edges

4. Relax the edges V-1 times to find the shortest path from any of the sources (one currency) to any other vertices (other currencies) and track this distances and each currency's predecessor.
5. The above step guarantees shortest distances if graph doesn't contain negative weight cycle. If we can still relax edges after **step 4 above**, and get a shorter path, then we have a negative cycle.
6. We add the source currency and destination currency within an array of called print-cycle.
7. Using the predecessor tracker and distance tracker, we trace the source predecessor, and predecessor's predecessor until we find a predecessor within the print-cycle array. This forms a closed loop. Meaning I can start from BTC and end at BTC

The mathematics and logic are clarifying, the code walks the talk!

2 Complexity Analysis for four (4) currencies (vertices):

Note: We are leveraging the ability (a loophole) of Bellman-Ford's to **detect** negative cycle, which is the worst-case scenario of the algorithm.

Time Complexity: Being a complete graph with edges between every pair of vertices (currency pairs), and having to relax V-1 times (4-1 = 3), ie 3 nested loops, we have a complexity of a negative cycle.

$O(V^3) = O(4^3)$

Where V = number of vertices

3. What is CHSB, and what are its key features

CHSB is an ERC20 utility token that permits the SwissBorg community participants to invest, support and drive the SwissBorg ecosystem.

Six Important Fact about CHSB

1. **Staking**: Allows users to stake their tokens and cryptos for reward
2. **Yielding** (Yield 2.0): Weighted reward tiering that reward both large and small holders. Ensures the small holders earn higher percentage and large holder also gain from the higher percentage that small holder enjoy lower CHSB and earn lower percentage as quantity increases
3. Referendum Participation:
4. Community Index Growth:
5. Premium Benefits

4. How to Run the Python Program:

Requirement:

Python Interpreter needs to be installed

To Run:

1. Unzip and copy the two files to the same path/ folder on your system:

   currArbitrage and **ReadAPI**

   The **readAPI** is called within the **currArbitrage**. And as the name implies, its function is to read the API weblink and feed the output to the **currArbitrage** inputs
2. Using CMD or bash navigate to the path where the two files are stored:
3. Run the command:

   Python currArbitrage.py

4. This will read the API link and return the results