FX Arbitrage – From a Search Problem Perspective

Nikolaos Skiadaressis

Athens University of Economics and Business

Table of Contents

Abstract

This term project explores the implementation of FX arbitrage on a MATLAB environment by formulating the discovery of the arbitrage opportunities as a search problem. In this way the problem can be solved by any search algorithm and the expansion of the search depth limit is possible.

FX Arbitrage – From a Search Problem Perspective

Problem Description

Given the bid forex quotes of 7 currencies at a 7x7 table format [Table 1] we are expected to:

1. Create a MATLAB program that seeks all possible triangular arbitrage opportunities, performs the corresponding trades and records the cumulative result.
2. Create a MATLAB program that seeks all one-way-arbitrage opportunities, given the starting and ending currencies. Afterwards, the program should execute the trades and record the results.

Problem Pre-formulation

Triangular Arbitrage

By applying the theoretical notion of searching for an FX arbitrage opportunity, we could approach question 1 by extracting the ask quotes from the quotation matrix, create all possible cross rates and check if the bid-ask rates of the cross rates are overlapped from the corresponding bid-ask rates of the quotation matrix. If not, we would perform a round trip trade whose starting and ending currencies would depend on the comparison of the quoted and cross rates.

An alternative approach would be to test the profitability of all possible round-trip trades. Under the notion that an existing arbitrage opportunity would produce an exploitable round-trip trade, if a round-trip trade produces a profit, this means that its currencies can compose a triangular arbitrage opportunity.

Proof

Let’s say we want to check if a triangular opportunity exists at the z/x quotation via the y/x and z/y and we are given only the bid ask pairs of the quotations z/x, y/x. and z/y. Suppose the quoted rate z/x is more expensive than the cross rate ($\frac{y}{x}\frac{z}{y}$) and the bid ask intervals are not overlapped. At this point we can buy z at the cross rate and sell at the more expensive quoted rate. The profit per unit of x that we would generate would be:

$$profit_x^1 = \left(bid\ \frac{y}{x}\right)\left(bid\frac{z}{y}\right)\left(ask\frac{z}{y}\right)^{-1} - 1$$

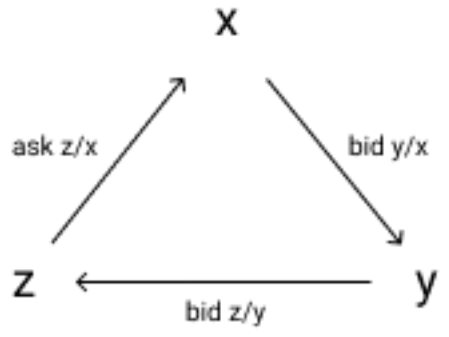

Figure 1

Now suppose the quoted rate z/x is cheaper than the cross rate ($\frac{y}{x}\frac{z}{y}$) and the bid ask intervals are not overlapped. This means that, we can buy z at the quoted rate and sell it at the more expensive cross rate. The profit per unit of x that we would generate would be:

$$profit_x^2 = \left(bid\frac{z}{x}\right)\left(ask\frac{z}{y}\right)^{-1}\left(ask\frac{y}{x}\right)^{-1} - 1$$

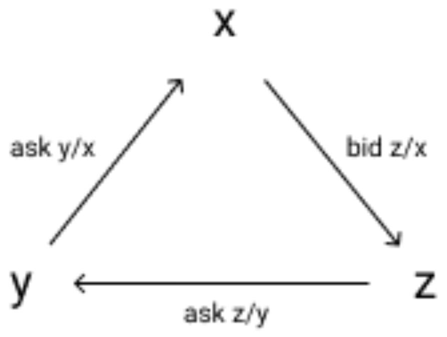

Figure 2

We notice two things:

1. Both $profit_x^1$ and $profit_x^2$ can be written in terms of bid quotes. That is:

$$profit_x^1 = \left(bid\ \frac{y}{x}\right)\left(bid\frac{z}{y}\right)\left(ask\frac{z}{y}\right)^{-1} - 1 = \left(bid\ \frac{y}{x}\right)\left(bid\frac{z}{y}\right)\left(bid\frac{x}{z}\right) - 1$$

$$profit_x^2 = \left(bid\frac{z}{x}\right)\left(ask\frac{z}{y}\right)^{-1}\left(ask\frac{y}{x}\right)^{-1} - 1 = \left(bid\frac{z}{x}\right)\left(bid\frac{y}{z}\right)\left(bid\frac{x}{y}\right) - 1$$

   On the above example where we are given only the bid-ask pairs of y/x, z/y and z/y this would not be ideal. However, on our problem, we are given only the bid quotes and the revision of the profit equations means that there is no need to calculate the ask quotes to perform the profit calculation.

2. Notice that although the above hypothetical arbitrage opportunities refer to the same cross rate, the round-trip-trades that we produce are different so as the illustrated triangles, under the assumption that the sequence of the currencies at the triangle matters. That is, the triangle (x, y, z) [Figure 1] is different than (x, z, y) [Figure 2]. These two triangles are mutually exclusive as the quoted rate can either be greater or less than the cross rate but not both. So, by checking if these round-trip-trades are profitable, we check if there is an arbitrage opportunity on these set of currencies.

Having said these, we can repeat our example by simply checking the following conditions:

$$if\ profit_x^1 > 0,\ \ there\ is\ an\ arbitrage\ opportunity\ else\ none.$$

$$if\ profit_x^2 > 0,\ \ there\ is\ an\ arbitrage\ opportunity\ esle\ none.$$

Hence, we can approach question 1 as simply as iterating over all the possible (x, y, z) permutations[1] that our currencies can produce and checking if the generated profit is greater than one. The profit is expressed in terms of the bid quotes. This formulation creates two implications that we have to resolve in order to continue with our analysis.

Implication 1:

First, we have to address the case where a currency might appear more than one time inside at a tringle, for example (x, y, x). I let this issue be self-resolved by MATLAB. When using MATLAB's readtable function to import the Quotation Matrix, x/x quotes are imported as Nan. The result of the profit calculation is Nan (any operation with Nan results to Nan) and the result of the test condition is again Nan, which is similar to logical 0. Therefore, triangles that are not properly defined do not pass the test condition and they are self-rejected.

Implication 2:

The second implication addresses the issue of repetition of the profitable triangles on the final results. Imagine that the triangle (x, y, z) generates a profit when testing the round-trip-trade with the bid quotes (as shown in the above example). Now, the triangle (y, z, x) would also generate the same profit as the former but in a different currency. That can be easily shown using the equation 1.

$$profit_x^1 = \left(bid\ \frac{y}{x}\right)\left(bid\frac{z}{y}\right)\left(bid\frac{x}{z}\right) - 1$$

$$profit_y^3 = \left(bid\frac{z}{y}\right)\left(bid\ \frac{y}{x}\right)\left(bid\frac{x}{z}\right) - 1 = profit_x^1$$

The only difference between $profit_x^1$ and $profit_y^3$ is that they are expressed in different currencies. Consequently, we can conclude that the order of the triangle does matter, but the starting currency is irrelevant. For this reason, after having identified all profitable round-trip-trades, we group the same triangles and we consider them one opportunity that can be exploited with 3 different ways[2]. This implication appears only in the case of triangular arbitrage and is irrelevant to one-way.

At last, the user can define up to five currencies on which he wants to sum the cumulative profits of all the profitable trades. Five is the theoretical limit of the currency "buckets" that can exist in our example. That is if all the triangular arbitrage opportunities start from 5 unique different currencies, but the intermediate currencies are the same at all triangles and different from the predefined 5.

One-Way-Arbitrage

The approach is similar to the second question with the exception of the profit calculation. Because we test for one-way-trip opportunities our goal should be to beat the quoted rate by buying the wanted currency through the cross rate. Hence, if the starting and ending currencies are x and y respectively and z is the intermediate currency, the profit equation would be:

$$profit_y = \left(bid\ \frac{z}{x}\right)\left(bid\frac{y}{z}\right) - (bid\frac{y}{x})$$

Note that profit is expressed in terms of the y currency which is predefined from the user. Consequently, after having all possible currency combinations enumerated and tested (through variable z), we add up the profits of all the one-way-arbitrage opportunities. The rest of the process is exactly the same as at the triangular arbitrage case.

Implementation

Implementations of these two approaches can be found at file arbitrage_classic.mlx.

Problem Formulation

Framework

Before formulating the above questions as search problems, we have to create the default problem and data-structure classes. This implementation follows the approach of Artificial Intelligence: A Modern Approach by P. Norvig and S. Russel and the code is an implementation of the book's code repository to MATLAB code[3]. I will not dive deep into the classes' structure. An understanding of search problems is assumed, so the reader is encouraged to read the code and no in-depth analysis will follow. Their approach is based on a Problem class and a Node class. The Problem class will define the structure that will be inherited to all search problem definitions and the Node class is a relational data structure similar to the ones found on a tree or graph structure. In these terms, we have to transcribe the pre-defined problem to one that matches our specifications and then apply any general-purpose search algorithm. The file tutorial_framework.mlx contains a brief example that emphasizes on the capabilities of this framework.

Triangular arbitrage formulation

Formulating a problem means that we have to specify five different aspects of the parent Problem class. These are:

- the state description for each node and the initial state of the problem

- the actions that make the transition from a state to an another,
- the result of each action given the current state,
- the goal-test condition
- and the value of a node relative to the current problem.

Problem Constructor

The Problem Constructor does not belong formally in the formulation section. It is part of the problem definition. However, its definition is essential in order to proceed with our analysis. We will start by formulating the problem of seeking all possible arbitrage opportunities, as described in the pre-formulation section. Consequently, the problem constructor takes as an input a starting currency, on which we want to generate all profitable round-trip-trades, and the quotation matrix that is a representation of our environment.

State Description

Then we have to define the state description of our problem. A state is a representation of the environment where the search is taking place and it can be viewed as a snapshot of the position of the searching agent at any given time. In our case, after each trade we are interested knowing the received amount of money and the currency that they are expressed to. In this notion, I represented the state as a cell array of the following format:

$$state = \{amount, currency\}$$

Initial state

Having specified the state description, it is important to define the starting state of the problem given the constructor inputs. By following the same conversion, the initial state will be

$$initial\ state = \{1, starting\ currency\}.$$

Actions and Result

Now that we know which is our starting state, we have to define the rules of traversing from one state to another. We can traverse to another state by performing an action. The action should belong to the possible action space given the current state. In our case, the action space is all the bid quotes on which we can perform a trade given an amount and a currency pair. From this space we should except the quotes that have the starting currency[4] as term also. After performing an action, the new state will be

$$new\ state = \{current\_state\{1,1\} * quote, quote.bid\_currency\}.$$

Goal Test

To check if a current state satisfies our goal, which is to generate a profit by converting to the starting currency, we perform the previously excepted action and subtract 1. Assuming x is our starting currency and y is the currency of the current state, the test condition will be

$$current\_state\{1,1\} * quotes(term = x, base = y) - 1 > 0$$

Value of Nodes

Finally, a problem can attach a value to each node which is unique to the problem we are trying to solve. On our case, the profit (or loss) that would be generated if we converted the current state (node) to the goal currency could represent the value of our problem. Notice that in this case the goal currency is the starting currency and our objective is to extract the profitable round-trip-trades. The bigger the value, the better is the solution.

Implementation

The RoundTripTrade.m is a class .m file that contains the problem formulation of the triangular arbitrage problem. That is all the above modifications relative to the parent class.

One-Way-Arbitrage formulation

In spite of re-formulating the one-way-trip arbitrage problem form a blank canvas (the parent class), we can inherit the triangular arbitrage problem formulation and just change the goal state in the constructor method. So, the file OneWayTripTrade.m is a class .m file that inherits from RoundTripTrade.m and simply changes the goal state inside the constructor method.

Searching algorithms

The main advantage of properly formulating the problem is that we can apply general any purpose searching algorithms by plugging them into our code. In an attempt to simulate a brute force enumeration for solving our problem, we will use a modified version of depth-limited-search algorithm. The algorithm in spite of stopping when a solution is found, it saves the solution-node and continues until the depth limit is exhausted. Afterwards we can extract the trade sequences by extracting the parents of the solution-nodes recursively[5].

If we assume that on our initial state the depth is 0, the classic triangular arbitrage opportunity is of depth 2. That is because it trades in 2 different currencies before the test condition occurs to check the profitability of the round-trip-trade. In a similar fashion, the one-way arbitrage is of depth 1.

Expanding the Search Limit

Up until now, we have specified the problem of seeking for arbitrage opportunities by following the guidelines of search problem theory and we have assigned a depth limit that simulates our conventional solution. From here, we could expand the search limit in order for us to find solutions of higher depth. On our implementation this is as simple as specifying the preferred limit as an input on the depth-limited-search algorithm. However, this implies some further filtering on the solutions that we are going to find.

Filtering higher depth solutions.

Upon finding a solution of higher than standard depth, this solution could belong in one of the following cases:

- it could be a unique solution that is composed from a unique sequence of currencies
- or it could be a solution that contains a lower depth solution inside.

The implication exists if the solution falls into the second case. In that case we can only accept the solution if it adds up to the profit of the lower one. In any other case the extra trades that happed absorb the profit of the low depth profitable sub-solution and we cannot accept it as a unique solution.

The filtering process is performed on an ungrouped solution set as it is described in Implication 2 of the Pre-Formulation section. This essentially means that each solution can exist as many times as the currencies that composes it. Given the ungrouped solutions of the triangular arbitrage[6] the filtering process is essentially composed of two parts:

- Firstly, for every solution lower than max depth we check if its currencies are contained in a higher depth solution sequentially. If they do, the solution of higher depth is accepted only if it adds up to the profit of the lower one. Else, it is discarded.

- Remember from implication 2 of the pre-formulation section that each arbitrage opportunity will appear as much times as the number of currencies that composes it. This number is equal to depth plus 1. At this notion, if a version of a triangular arbitrage opportunity has been rejected by the previous step, the round-trip-trades that exploit this opportunity would be less than the solution depth plus 1. Hence, this opportunity is ineffective, because it has a version that absorbs some profit of a lower depth one and should be discarded.

We should recognize that this process applies quadratic time complexity to our execution time. Nevertheless, a vectorized approach should resolve this issue[7].

The second step of the filtering process is redundant for the one-way-arbitrage, as every opportunity can be exploited with exactly one way That is because the starting and ending currencies are predefined.

Implementation

The file arbitrage_search_problem.mlx contains the solution to our problem based on the above analysis. A search limit of 3 and 2 have been used for the triangular arbitrage and the one-way arbitrage respectively in order to demonstrate the capabilities of our approach. Tables 2-5 depict the currency combinations that can be exploited and their explicit and cumulative profit for both problems. Moreover, the transaction logging can be found on the files “roundtrip logging.txt” and “oneway logging.txt”.

Comment

We have to admit that this approach has a significant implementation overhead when comparing it with the solutions of erotima_1.m and erotima_2.m. However, it entitles some benefits that we have to address.

Readers that are accustomed with search problem theory will notice that we did not implement a path cost function into our problem definition. That is because we limited ourselves to uninformed search. The real advantage of this approach is in the use of more sophisticated informed search algorithms. These will upward bound our time and space complexity and will allow us to explore higher depth solutions without the execution time trade off.

Furthermore, we have to note that this project is far from being ready for production. Utility functions should be vectorized and the whole code should be rewritten in a compiled language where we will have at least a x100 improvements.

Conclusion

In this term paper we explored the problem of seeking normal and one-way arbitrage opportunities. Initially, we reduced the problem to a simple enumeration with one condition testing. Then we briefly described the framework based on which we analyzed the problem as a search problem according to search problem theory. Finally, we applied general purpose search algorithms to solve our problem and filtered out ineffective solutions.

References

Norvig P. and Russel S. (2010). Artificial Intelligence: A Modern Approach – Third Edition

Footnotes

[1] These are 7^3 possible triangle permutation if we do not exclude repetitive occurrences.

[2] See utils.py/profits_group_by for grouping utility function.

[3] Files: Problem.m and Node.m.

[4] Starting currency is the currency that is passed to problem constructor.

[5] Also, from Artificial Intelligence: A Modern Approach's repository. See depth_limited_search.m for the original version and depth_limited_search_all.m for the twisted. For extracting see utils/extract_profits_all

[6] The format of the solutions is a {n :} cell array where n is the number of all the exploitable round-trip-trades.

[7] See utils/profits_depth_filter and utils/filter_oneway_profit utility function.

Tables

Table 1

Quotation Matrix

	USD	EUR	JPY	GBP	CHF	CAD	AUD
AUD	1.2440	1.7441	0.0115	2.1829	1.0959	1.1634	
CAD	1.0693	1.4992	0.0099	1.8763	0.9420		0.8584
CHF	1.1352	1.5915	0.0105	1.9920		1.0602	0.9111
GBP	0.5699	0.7990	0.0053		0.5013	0.5325	0.4575
JPY	107.8600	151.2200		188.5473	95.1715	100.9495	86.8523
EUR	0.7133		0.0066	1.2506	0.6275	0.6670	0.5732
USD		1.4001	0.0093	1.7533	0.8802	0.9338	0.8034

Note: The quotation matrix that describes the environment on which we perform the search.

Table 2

Triangular arbitrage opportunities

Profit	Starting Currency	Currency 1	Currency 2	Currency 3
0.001207721	USD	AUD	JPY	
0.000291008	USD	CAD	JPY	
0.001155816	USD	CHF	JPY	
0.002282925	USD	JPY	GBP	
0.000213043	EUR	AUD	JPY	
0.000122636	EUR	CHF	JPY	
0.00228453	EUR	JPY	GBP	
0.00033525	JPY	CAD	AUD	
0.004826382	JPY	GBP	AUD	
0.003881509	JPY	GBP	CAD	
0.004782367	JPY	GBP	CHF	
0.001226731	USD	CAD	AUD	JPY
0.001260582	USD	CHF	AUD	JPY
0.001261266	USD	EUR	AUD	JPY
0.000365246	USD	EUR	CAD	JPY
0.001170765	USD	EUR	CHF	JPY
0.002331488	USD	EUR	JPY	GBP
0.000227294	EUR	CHF	AUD	JPY
0.004887512	JPY	GBP	CHF	AUD

Note: The currency sequences that can produce profitable and effective round-trip-trades based on Table 1 and with a search limit of three. We observe that the expansion of the search limit contributes with eight more effective solutions that the conventional approach would not have discovered. Table has been generated through the file arbitrage_search_problem.mlx.

Table 3

Cumulative profits of triangular arbitrage

USD	AUD	JPY
0.012553549	0.010489481	0.011071042

Note: The cumulative profits of all profitable triangular arbitrage opportunities that were found given the quotation matrix of Table 1 and with a search limit of three. Table has been generated through the file arbitrage_search_problem.mlx.

Table 4

One-way arbitrage opportunities

Profit	Starting Currency	Currency 1	Currency 2
0.259094652	EUR	AUD	
0.123537312	EUR	CAD	
0.245402789	EUR	CHF	
0.265111779	EUR	CAD	AUD
0.261252931	EUR	CHF	AUD
0.262316872	EUR	GBP	AUD
0.119872844	EUR	GBP	CAD
0.255681309	EUR	GBP	CHF
0.054337237	EUR	USD	AUD
0.046494806	EUR	USD	CHF

Note: The currency sequences that can produce profitable and effective one-way-trip trades based on Table 1 and with a search limit of two. We can observe that the expansion of the search limit contributes with seven new solutions that in any other case would not have been discovered. Table has been generated through the file arbitrage_search_problem.mlx.

Table 5

Cumulative profits of one-way arbitrage

JPY
1.893102531

Note: The cumulative profits of all profitable arbitrage opportunities that were found given the quotation matrix of Table 1 and with a search limit of two. Table has been generated through the file arbitrage_search_problem.mlx.