Austin Thiele

Professor Arias

Final Write-up

12/11/17

Triangular Arbitrage Detector

The project I intend to complete is a system that scans for and analyzes potential triangular arbitrage opportunities that may arise between three respective currencies. To do so, I intend to extract data from an online source, organize it, and then analyze the data to determine if an arbitrage opportunity exists. The program achieves this goal through the use of three classes: main, fixer, and fixerModel. The fixer class retrieves the foreign exchange data from the online source directly, without having to access your internet browser. It models this data into useable variables through the fixerModel class. The data is finally organized into a method also named fixerModel. The main class retrieves the information from the fixerModel class and method. The data is analyzed in every possible combination for the three currencies to evaluate if an arbitrage opportunity exists. If an opportunity exists, then the system will print "Arbitrage Opportunity" to the user. Otherwise it will print "Equation Works", indicating the system completed the analysis and found not indication of an arbitrage opportunity.

For my research project for CMPT 220 I completed an application project that would detect triangular arbitrage opportunities within the foreign exchange market as they occur. I became interested in completing this project when learning that triangular arbitrage is of mathematical significance, and that the math can be trusted to execute trades between these

currencies risk-free. The way this happens is a simultaneous buying and selling of currencies at different offering prices. The markets have always been of interest to me, so the opportunity to complete a research project in this field could not be wasted. In the paper, I first discuss all of the classes, how they interact, and the functions they perform. I then describe the specific problems that my program seeks to accomplish. Finally, we look at how others have attacked the same problem, how to operate the program, and what the program accomplishes.

The first public class “Fixer” deals with retrieving the data from the API. There were various choices online to choose an API for foreign exchange prices. The platforms that you needed to pay for delivered more up to date information, but as far as my model was built it can be adapted to any API. For this reason, I chose a free API Fixer. To retrieve the information from the API, I had to download a few external libraries. One was Gson, which helps serialize and un-serialize objects in java. What Gson does is convert the data into its Json equivalent, which was then used in another class to more easily access the information by storing the rate data in a hash map. A hash map is just another kind of data structure that, for my program, was easier to organize the data into. The code first accesses the webpage, it then retrieves the data and stores it in the String “ApiResponse” which is continually updated as more information is stored. The returned value is the final value of data requested.

The next public class “FixerModel” does exactly as the name intends and models the data into a more useable fashion. First, the private variables are defined. Getters are generated to retrieve information needed from the API data. There is no need for setters because the user

is not inputting any information regarding base, date, or the rates retrieved. Rates are stored in a hashmap so we can store the prefix of the currency (ex. JPY, USD, EUR) as a string, and the subsequent value as a double. Avoiding taking the whole value as a string, we can work with each part of the hash map. This means we can perform arithmetic on the double variables as we should be able to when comparing currencies. The final part to public class FixerModel is the method FixerModel. Arguments in this method are the three previous data fields (base,date,rates) that we had retrieved and organized. FixerModel finalizes the base date and rates of the information requested using the "this" keyword, referring to the specific instance that the method was called.

The final part to my project is the "Main" java file, which executes the code and performs the application. Each block of code contained in the try catch clauses analyze a different relationship between the three preset currencies to analyze if an arbitrage condition exists in any direction or relationship between them. In each block, the code operates in the same fashion. Firstly, the rates are retrieved through the Fixer class, and the FixerModel method sets these to the currentRates. The remainder of the code operates simple algebraic equations to evaluate if an irregularity exists between the three currencies. If an opportunity exists, the system will print "Arbitrage Opportunity". If there is no opportunity, the system will print "Equation Works", indicating no irregularities between the currencies.

The program deals with a few problems. Firstly and most importantly, the program is designed to analyze the relationship between three currencies to decide if an arbitrage position

is present. It gets the values for specific dates and what the closing price of those currencies are denoted in the base currency of euros. In future iterations of the program, you will be able to change the base currency that others are valued in relation to. It solves the problem of organizing these rates by putting them into a hash map where rates can be found based off of double value of the rate or the string value of the currency prefix. With the data poised in this manner, the program could be manipulated to view currencies between different mathematical spreads, or by keyword entering of the currency prefix. With the value of currencies constantly changing, this system seeks to execute trades on the rare opportunities for arbitrage that occur. Arbitrage in a basic sense is taking advantage of the inefficiencies of the market.

Where my program is limited is in the fact that the API only discloses end day information. Ideally, foreign exchange arbitrage opportunities occur in mere second-to-second windows. Considering it is a rare occurrence to make a risk-free profit from such a situation, many banks, financial institutions, and intermediaries have poured money into computing systems and applications that can recognize these situations as soon as they arise. With their increased capital, they are able to run larger faster computers that can analyze data across the entire market and make trading decisions instantaneously off of that data. Also, considering the price differences between the currencies may be mere pennies worth, you have to trade a large propensity of money to realize any significant gain. This makes programs such as these much more attractive to large businesses who can afford to move cash in large quantities. The idea of arbitrage is not a new one, so this particular problem has been solved in many different iterations of applications and software. The money-making software, however, is not publicly

available. Those with the computing power and capital to identify these opportunities know well to keep their system well-guarded.

To use the program it is very simple, just click run. The interaction between the main, fixer, and fixerModel classes automates the retrieval, organization, analyzation, and display of the information requested.

In conclusion, the current iteration of my program operates in a limited space. It is limited to analyzing a one three-way relationship between three predetermined currencies. However, the program does succeed in analyzing this data properly and giving the correct indication based off of end-day currency rates. On a larger scope, the program could accomplish analysis between more currencies and at a faster rate if the API accessed was more advanced.

*UML DIAGRAMS FOR ALL CLASSES ON NEXT PAGE

Fixer
- https://api.fixer.io/latest String
+ApiResponse String
retrieveCurrentRates()

FixerModel
- base String
- date String
- rates HashMap
+ getBase String
+ getDate String
+getRates HashMap
FixerModel()

Main
Main()