ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ
SCHOOL OF BUSINESS

ΤΜΗΜΑ ΛΟΓΙΣΤΙΚΗΣ & ΧΡΗΜΑΤΟ-ΟΙΚΟΝΟΜΙΚΗΣ
DEPARTMENT OF ACCOUNTING & FINANCE

Exercise 3: Advanced Computational Finance

Topic: “Computation of arbitrage opportunities in a quotation matrix using Python”

8th Semester 2020-2021

Xagoraris Emmanouil (Ξαγοράρης Εμμανουήλ), 7170123

Abstract

The aim of this assignment is to analyze the arbitrage opportunities -if existing- from a quotation matrix using python and its modules, in this assignment we are using Pandas and Numpy. The following report analyzes the steps taken and the results of the code.

Contents

General

Both modules are reading an excel sheet using python's module Pandas. In our case this table is called quotation matrix and it contains the exchange rates between 7 currencies. Therefore, the matrix is seven by seven. Where the rows are the terms and columns are the base currencies. We can identify that using Japanese Yen as index due to the fact that 1 USD equals to more than 100 JPY.

In the beginning of the code, we predefine variables, tables and DataFrames that will be used afterwards.

```
FXMatrix = pd.read_excel("Quotation Matrix.xlsx",header=0,index_col=0)
k = len(FXMatrix)

currencies = np.array(FXMatrix.columns,dtype=str)

pd.options.display.float_format = "{:,.7f}".format

ask = np.zeros([k,k],dtype=float)
bid = np.zeros([k,k],dtype=float)
mid = np.zeros([k,k],dtype=float)
implied = np.ones([k,k],dtype=float)
quoted = np.ones([k,k],dtype=float)
arbitrage = np.ones([k,k],dtype=float)
differences = np.ones([k,k],dtype=float)
triangular_earnings = pd.DataFrame(np.zeros([k,k],dtype=float), columns = currencies,
                                   index=np.flip(currencies,0))
strategy = pd.DataFrame(np.zeros([k*3,5],dtype = float), columns = ["Buy","Crosscurrency","Sell","Result","Terms"],
                        index = np.arange(1,k*3 + 1,1))
implied_frame = pd.DataFrame(np.zeros([k,k],dtype=float), columns = currencies,
                                   index=np.flip(currencies,0))
Crosscurrency = " "
flipped_cur = np.flip(currencies)
```

Briefly,

- The DataFrame *FXmatrix* contains the exchange rates as read by the excel file.
- The variable *k* contains the length of the DataFrame FXMatrix (k = 7 in this case).
- The array *currencies* contain the tickers, the names, of each currency.
- The array *flipped_cur* contains the tickers, the names, of each currency inverted. We did that because the rows and the columns are not consistent.
- The rest of the variables or tables or DataFrames are created in order to save the values of the following steps.

We assume that within the quotation matrix bid and ask prices are mixed. Therefore, we should identify which are which. In this case, we are comparing one exchange rate with its identical reverse divided.

For example, let us assume that we want to find the ask/bid price for the AUD/USD.

We take from the quotation matrix:

- 1.244 AUD/USD
- 0.802815 USD/AUD

Then, we divide the USD/AUD, and we have $\frac{1}{0.802815}\ \frac{AUD}{USD} = 1.2456$ AUD/USD

We compare the two number and the larger one is assigned as Ask price and the smaller one as Bid.

	A	B	C	D	E	F	G	H
1		USD	EUR	JPY	GBP	CHF	CAD	AUD
2	AUD	1.244	1.7441	0.0115	2.1829	1.0959	1.1634	
3	CAD	1.0693	1.4992	0.0099	1.8763	0.942		0.859378
4	CHF	1.1352	1.5915	0.0105	1.992		1.060617	0.911489
5	GBP	0.5699	0.799	0.0053		0.501256	0.532538	0.457557
6	JPY	107.86	151.22		188.6415	95.16197	101	86.87833
7	EUR	0.7133		0.006603	1.250064	0.62815	0.666023	0.572846
8	USD		1.400115	0.009267	1.753642	0.879934	0.934724	0.802815

Using python we are computing that for every combination of currencies and we create three DataFrames (ask,bid,mid).

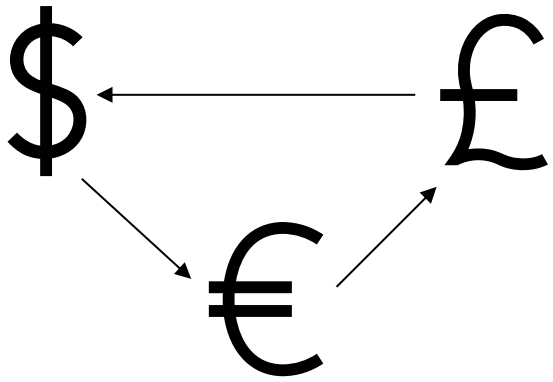
```
for i in range(k):
    for j in range(k):
        if FXMatrix.iloc[i,j] == np.nan:
            ask[i,j] = np.nan
            bid[i,j] = np.nan
        else:
            fx1 = FXMatrix.iloc[j,i]
            fx2 = 1/FXMatrix.iloc[k-1-i,k-1-j]
            if fx1 > fx2:
                ask[i,j] = fx1
                bid[i,j] = fx2
            else:
                ask[i,j] = fx2
                bid[i,j] = fx1

        mid[i,j] = (ask[i,j]+bid[i,j])/2
```

Question 1

In this question we will try to identify the arbitrage opportunities using the method of triangular arbitrage.

In this method the investor/arbitrageur tries to find an opportunity of arbitrage so as to make profit.



For example, let us assume that we have the following exchange rates:

- 1.753642 USD/GBP
- 0.799 GBP/EUR

- 0.7133 EUR/USD

The market USD/GBP = 1.753642. However, if we calculate the implied rate using a cross-currency we will take a different result.

$$\frac{USD}{GBP} = \frac{USD}{EUR} \cdot \frac{EUR}{GBP} = \frac{1}{\frac{EUR}{USD}} \cdot \frac{1}{\frac{GBP}{EUR}} = \frac{1}{0.7133} \cdot \frac{1}{0.799} = 1.754611 \frac{USD}{GBP}$$

The implied rate is higher than the quoted rate therefore GBP is undervalued in USD terms. Which means that there is an arbitrage opportunity.

Using the triangular arbitrage strategy, we will execute the following transactions:

- Firstly, we will buy the undervalued GBP using USD which is the overvalued
- Afterwards, we will exchange our GBP for EUR
- Finally, we will exchange our EUR for USD

Our profit will be the initial amount multiplied by the difference between the quoted and the implied rate.

In the bibliography (Tuckman & Serrat, 2011), this method usually brings profit when we do not take into consideration transaction costs (ask-bid spread). That is the reason why we created the table mid, so as to check for every combination of currency; if there is an indication of arbitrage.

```
constant2 = mid_frame.loc["EUR","USD"] #we define this variable in order to calculate
#USD implied rate using EUR
for i in range(k):
    constant1 = mid_frame.loc[flipped_cur[i],"USD"] #for every other case we use USD
    quoted[i,:]= mid_frame.iloc[i,:] #we create an array called quoted which
    # the "quoted mid prices"
    implied_frame.loc[flipped_cur[i],"USD"] = constant2*mid_frame.loc[flipped_cur[i],"EUR"] #for the USD we use euro base
    #and variable terms
    for j in range(1,k,1):
        implied_frame.loc[flipped_cur[i],currencies[j]] = constant1*mid_frame.loc["USD",currencies[j]]

    implied_frame.loc["EUR","USD"] = mid_frame.loc["EUR","JPY"]*mid_frame.loc["JPY","USD"]
    for j in range(k):
        if implied_frame.iloc[i,j] == quoted[i,j]:
            arbitrage[i,j] = 0

        implied_frame.iloc[-1,j]=1/implied_frame.iloc[k-j-1,0]
```

In the code above, we define two variables here called constant1 and constant2 accordingly. We do that because the cross-currencies used should be predefined. For every couple of currencies, we are using dollar as cross-currency with 2 exceptions. The first one is when we are trying to identify the existence of arbitrage in a couple of currencies where one of them is the USD, in that case we are using EUR as cross-currency. However, it is reasonable to introduce a third cross-currency when examining an arbitrage opportunity in the exchange rate of EUR/USD, in this case we choose JPY.

Therefore, the DataFrame implied_frame contains the implied rates of the mid values so as to examine with which currencies we should proceed. The table arbitrage is predefined as a matrix with ones. In case where the implied rate is equal with the quoted rate it takes the value zero.

In the next step we calculate the difference between the implied rate and the quoted rate so as to understand which currency is overvalued/undervalued. However, we need to find just k*3 differences because if we calculate all of them we will repeat some of the transactions. To give some context, let us assume that we have the exchange rates of USD/EUR and EUR/USD. If we calculate the difference of USD/EUR we can identify which one is overvalued/undervalued therefore we will perform this transaction only once despite the fact that we have two exchange rates.

```
for i in range(k-1,-1,-1):
    for j in range(0,k-i,1):
      differences[i,j] = implied_frame.iloc[i,j] - quoted[i,j]


differences_frame = pd.DataFrame(differences, columns = currencies
                        ,index=np.flip(currencies,0))
```

Index	USD	EUR	JPY	GBP	CHF	CAD	AUD
AUD	0.000502013	-0.000703702	2.42648e-05	-0.00147813	-0.00071534	-8.02545e-05	nan
CAD	0.000453688	-0.000635963	3.87575e-06	-0.00234982	-0.000663683	nan	1
CHF	1.69647e-05	-2.37652e-05	2.00274e-05	-0.00166374	nan	1	1
GBP	-0.000115747	0.000162262	-1.70987e-05	nan	1	1	1
JPY	0.0321995	-0.0451356	nan	1	1	1	1
EUR	-0.000212647	nan	1	1	1	1	1
USD	nan	1	1	1	1	1	1

Implementing the above code, we calculate the differences only for the inverse lower triangular matrix.

Our final step is to calculate the result of the arbitrage opportunities taking into consideration transaction costs.

```
sum0 = 0
for i in range(k):
    terms = flipped_cur[i]
    for j in range(k-1-i):
      base = currencies[j]
      if arbitrage[i,j] ==1:
         if base == "USD":
            if terms == "EUR":
               Crosscurrency = "JPY"
            else:
                  Crosscurrency = "EUR"
         elif terms == "USD" and base == "EUR":
            Crosscurrency = "JPY"
         else:
            Crosscurrency = "USD"

      (success,results,currency1,currency2) = TriangularArbitrage(terms, base, Crosscurrency, ask_frame, bid_frame, differences[i,j])
```

Here, we are using as a condition the matrix *arbitrage,* therefore if *arbitrage == 1 (it is true)* then we proceed with computation of the arbitrage result. As in the previous part, look at page 5, we predefine the *Crosscurrency* according to a specific set of conditions. After that we pass the data to the custom function *TriangularArbitrage* which take as inputs the terms, the base and the cross-currency currencies, the Ask and Bid DataFrames and the difference. This function returns the following variables:

- success: Boolean that returns with True when the triangular arbitrage is successful and False when it is not.
- result: It returns the Gain or loss from arbitrage procedure.
- currency1: It returns the currency that will be used as line index. Also, it indicates the overvalued currency.
- currency2: It returns the currency that will be used as column index. Also, it indicates the undervalued currency.

```
def TriangularArbitrage (Terms,Base,Crosscurrency,ask,bid,difference):

    initial = 10**6 #assuming that we have a million of the currency that
                    #we are selling
    if difference > 0: #Terms are undervalued
        arbitrage_results = bid.loc[Base,Terms]*initial * bid.loc[Crosscurrency,Base] * bid.loc[Terms,Crosscurrency]
        currency1 = Base
        currency2 = Terms
    else:
        arbitrage_results = bid.loc[Terms,Base]*initial *bid.loc[Crosscurrency,Terms] * bid.loc[Base,Crosscurrency]
        currency1 = Terms
        currency2 = Base

    if arbitrage_results == np.nan:
        results = 0
    else:
        results = arbitrage_results - initial
    if  results > 0:
        success = True

    else:
        success = False

    return (success, results, currency1,currency2)
```

We define an initial amount of 1 million of the currency that will be sold. On the next step, we examine the two different scenarios. Firstly, we check which currency is over/under valued execute transactions as proper and after that we define currency1 and currency2. Finally, we return the values to be used on the next step.

```
      (success,results,currency1,currency2) = TriangularArbitrage(terms, base, Crosscurrency, ask_frame, bid_frame, differences[i,j])
      triangular_earnings.loc[currency1,currency2]= results
      strategy.iloc[sum0,0] = currency2
      strategy.iloc[sum0,1] = Crosscurrency
      strategy.iloc[sum0,2] = currency1
      strategy.iloc[sum0,3] = results
      strategy.iloc[sum0,4] = currency2
      sum0 = sum0 +1

print(strategy)
strategy.to_excel("Triangular Arbitrage.xlsx",sheet_name ="Sheet1")
```

The final step is to save the values into DataFrames and to present them it properly. Additionally, we save the results in an excel file so as to be easily accessible.

Index	Buy	Crosscurrency	Sell	Result	Terms
1	AUD	EUR	USD	-246.445	AUD
2	EUR	USD	AUD	-246.445	EUR
3	AUD	USD	JPY	1007.62	AUD
4	GBP	USD	AUD	-272.016	GBP
5	CHF	USD	AUD	-247.128	CHF
6	CAD	USD	AUD	-280.929	CAD
7	CAD	EUR	USD	-125.758	CAD
8	EUR	USD	CAD	-125.758	EUR
9	CAD	USD	JPY	-807.644	CAD
10	GBP	USD	CAD	-196.809	GBP
11	CHF	USD	CAD	-145.356	CHF
12	CHF	EUR	USD	-384.915	CHF
13	EUR	USD	CHF	-384.915	EUR
14	CHF	USD	JPY	1055.76	CHF
15	GBP	USD	CHF	-363.914	GBP
16	USD	EUR	GBP	-846.291	USD
17	GBP	USD	EUR	-846.291	GBP
18	JPY	USD	GBP	1782.43	JPY
19	JPY	EUR	USD	-651.093	JPY
20	EUR	USD	JPY	-651.093	EUR
21	USD	JPY	EUR	-651.093	USD

This form of presentation indicates the transactions that should be executed. We can identify the strategies that are profitable. However, as expected the majority of them are at a loss due to transaction costs, whereas only 3 transactions are profitable. One thing that we should mention here is the fact that only transactions that involve JPY are profitable in this case. This finding is not statistically important, but it is something worth mentioning.

These arbitrage opportunities exist in the market for fragments of seconds, as they are corrected automatically from arbitrageurs who use algorithmic programmes (practically bots) that buy/sell when an opportunity arises.

Question 2

In this question we will try to identify the arbitrage opportunities using the method of one-way arbitrage.

In this method the investor/arbitrageur wants to execute a specific transaction between two currencies, but he/she tries to find if there is a cheaper way to do it. Therefore, it requires specific currencies.

```
flagbuy =0
while flagbuy == 0:
    print("Valid Currencies: ",np.transpose(currencies))
    buy = input("Please indicate the currency that you want to buy: ")
    buy = buy.upper()
    for i in range(k):
        if buy == currencies[i]:
            flagbuy =1
            break
        else:
            continue

flagsell =0
while flagsell == 0:
    print("Valid Currencies: ",np.transpose(currencies))
    sell = input("Please indicate the currency that you want to sell: ")
    sell = sell.upper()
    for i in range(k):
        if sell == currencies[i]:
            flagsell =1
            break
        else:
            continue
```

The above implementation reads the currencies from the user; however, it checks them to see if they are valid. In case that one of them is not, it asks the user to insert the currency properly.

Following that we calculate the arbitrage opportunities.

```
sum0 = 0
for i in range(k):
    crosscurrency = currencies[i]
    if buy != crosscurrency and sell !=crosscurrency:
        result=OnewayArbitrage(buy,sell,crosscurrency,initial_amount,bid_frame)
        strategy.iloc[sum0,0] = sell
        strategy.iloc[sum0,1] = buy
        strategy.iloc[sum0,2] = crosscurrency
        strategy.iloc[sum0,3] = result
        sum0 = sum0 + 1

print(strategy)
```

The condition inside the loop is used because we do not want to initiate a calculation if the crosscurrency is equal to either the buy or sell currency. Afterwards, we pass the data into the custom function *OnewayArbitrage* which take as imput the buy, sell, crosscurrency currencies

and the initial amount which is defined as the ask price of sell/buy (sell is used as terms and buy as base) in the start of the module and it returns the result of the arbitrage opportunity.

```
def OnewayArbitrage (buy_currency,sell_currency,crosscurrency,initial_amount,bid):
    """
    Parameters
    ----------
    buy_currency : str
        Inserted by the user.
    sell_currency : str
        Inserted by the user.
    crosscurrency : str
        Crosscurrency.
    initial_amount : float
        amount of selling currency.
    bid : DataFrame
        Bid DataFrame.

    Returns
    -------
    result: float
        result of arbitrage execution.

    """
    result = initial_amount * bid.loc[crosscurrency,sell] * bid.loc[buy,crosscurrency]
    return result
```

Finally, we present the results properly as to be readable by the user.

```
Valid Currencies:  ['USD' 'EUR' 'JPY' 'GBP' 'CHF' 'CAD' 'AUD']

Please indicate the currency that you want to buy: USD
Valid Currencies:  ['USD' 'EUR' 'JPY' 'GBP' 'CHF' 'CAD' 'AUD']

Please indicate the currency that you want to sell: EUR
  Sell  Buy  Via   Results
1  EUR  USD  JPY 0.9995488
2  EUR  USD  GBP 0.9989482
3  EUR  USD  CHF 0.9998150
4  EUR  USD  CAD 1.0000742
5  EUR  USD  AUD 0.9999535
```

The results that are greater than one indicate the existence of arbitrage opportunities. Typically, in this method we expect to find more arbitrage opportunities in contrast with the other one as we have fewer transaction and in extend fewer transaction costs.

Bibliography

Tuckman, B., & Serrat, A. (2011). *Fixed Income Securities.* John Wiley and Sons.