Introduction to Quantitative Finance and Financial Risk

Assignment 1: Triangular and One-way Arbitrage

Ioannis Michalopoulos

p3352314

Introduction

In international financial markets, currency arbitrage is a practice used to exploit price discrepancies between different exchange rates to secure a risk-free profit. This project focuses on two key types of arbitrage strategies: **triangular arbitrage** and **one-way arbitrage**.

Triangular arbitrage involves a sequence of three currency exchanges that starts and ends with the same currency. This strategy takes advantage of inconsistent cross-currency rates in the foreign exchange market and can yield profit if the product of the involved exchange rates deviates from unity. On the other hand, one-way arbitrage identifies favorable indirect paths between a pair of currencies that offer a better effective exchange rate than the direct conversion, usually involving one or two intermediate currencies.

The objective of this assignment is to analyze a customized quotation matrix and develop Python programs to:

- Detect and report all possible profitable triangular arbitrage opportunities.
- Evaluate indirect conversion paths for a user-defined one-way currency trade and detect any profitable alternatives to the direct exchange.

This report documents the methodology, implementation, and results obtained from analyzing the quotation matrix. All computations were conducted using Python within a Jupyter Notebook environment.

Triangular Arbitrage

The first part systematically explores all possible combinations of three different currencies to detect **triangular arbitrage** opportunities. For each cycle (e.g., USD → EUR → GBP → USD), it calculates the product of the involved exchange rates. If the result exceeds 1 (with a small margin to filter out noise from floating-point precision), a potential arbitrage opportunity is recorded.

The user is prompted to specify a base currency in which the profit should be expressed. The program outputs:

- The currency cycle.
- The product of the exchange rates.
- The profit percentage and profit in both the cycle's starting currency and the selected base currency.

Example Result:
For the base currency USD, one detected cycle was USD → AUD → JPY → USD with a profit of approximately **0.22%**, indicating a potential arbitrage gain from the discrepancy in rates.

```
import pandas as pd
from itertools import permutations

def load_matrix(file_path):
    # Load matrix from Excel; assumes first row and first column are currency
codes
    df = pd.read_excel(file_path, index_col=0).T
    return df

def triangular_arbitrage(matrix, base_currency):
    currencies = list(matrix.index)
    opportunities = []

    # Generate all permutations of 3 distinct currencies for triangular
arbitrage
    for cycle in permutations(currencies, 3):
        start, mid, end = cycle

        # Calculate product of rates: start->mid, mid->end, end->start
        rate1 = matrix.loc[start, mid]
        rate2 = matrix.loc[mid, end]
        rate3 = matrix.loc[end, start]

        if pd.isna(rate1) or pd.isna(rate2) or pd.isna(rate3):
            continue  # skip if any rate is missing
```

```
        result = rate1 * rate2 * rate3

        # Only consider arbitrage if result > 1 (profit)
        if result > 1.0001:  # add small threshold to avoid floating error
noise
            # Calculate profit starting with 1 unit of base currency
            if start == base_currency:
                profit_in_base = result - 1
            else:
                # Convert the profit back to base_currency using direct rate
if available
                if start == base_currency:
                    profit_in_base = result - 1
                else:
                    try:
                        conv_rate = matrix.loc[start, base_currency]
                        profit_in_base = (result - 1) * conv_rate
                    except KeyError:
                        # If no direct conversion, skip conversion
                        profit_in_base = result - 1

            opportunities.append({
                'Cycle': f"{start} → {mid} → {end} → {start}",
                'Product_of_Rates': result,
                'Profit_in_Starting_Currency': result - 1,
                f'Profit_in_{base_currency}': profit_in_base
            })

    return opportunities

def main():
    file_path = 'Quotation Matrix.xlsx'
    matrix = load_matrix(file_path)

    print("Currencies detected:", list(matrix.columns))

    base_currency = input("Enter the currency code to express profit in (e.g.,
USD): ").strip().upper()
    if base_currency not in matrix.columns:
        print(f"Currency '{base_currency}' not found in matrix. Please check
the input.")
        return

    opportunities = triangular_arbitrage(matrix, base_currency)

    if not opportunities:
        print("No triangular arbitrage opportunities found.")
```

```
    else:
        print("\nTriangular Arbitrage Opportunities:")
        for opp in opportunities:
            print(f"Cycle: {opp['Cycle']}")
            print(f"  Product of rates: {opp['Product_of_Rates']:.6f}")
            print(f"  Percentage profit:
{opp['Profit_in_Starting_Currency']:.6%}")
            print(f"  Profit in starting currency:
{opp['Profit_in_Starting_Currency']:.6f}")
            print(f"  Profit in {base_currency}:
{opp[f'Profit_in_{base_currency}']:.6f}")
            print("-" * 40)

if __name__ == "__main__":
    main()
```

```
Triangular Arbitrage Opportunities:
Cycle: USD → JPY → GBP → USD
  Product of rates: 1.001782
  Percentage profit: 0.178243%
  Profit in starting currency: 0.001782
  Profit in USD: 0.001782
----------------------------------------
Cycle: USD → CHF → JPY → USD
  Product of rates: 1.000756
  Percentage profit: 0.075579%
  Profit in starting currency: 0.000756
  Profit in USD: 0.000756
----------------------------------------
Cycle: USD → AUD → JPY → USD
  Product of rates: 1.002209
  Percentage profit: 0.220873%
  Profit in starting currency: 0.002209
  Profit in USD: 0.002209
----------------------------------------
Cycle: EUR → JPY → GBP → EUR
  Product of rates: 1.002285
  Percentage profit: 0.228453%
  Profit in starting currency: 0.002285
  Profit in USD: 0.003198
----------------------------------------
Cycle: EUR → AUD → JPY → EUR
  Product of rates: 1.001413
  Percentage profit: 0.141304%
  Profit in starting currency: 0.001413
  Profit in USD: 0.001978
----------------------------------------
Cycle: JPY → USD → CHF → JPY
  Product of rates: 1.000756
  Percentage profit: 0.075579%
  Profit in starting currency: 0.000756
  Profit in USD: 0.000007
```

```
---------------------------------------
Cycle: JPY → USD → AUD → JPY
  Product of rates: 1.002209
  Percentage profit: 0.220873%
  Profit in starting currency: 0.002209
  Profit in USD: 0.000020
---------------------------------------
Cycle: JPY → EUR → AUD → JPY
  Product of rates: 1.001413
  Percentage profit: 0.141304%
  Profit in starting currency: 0.001413
  Profit in USD: 0.000013
---------------------------------------
Cycle: JPY → GBP → USD → JPY
  Product of rates: 1.001782
  Percentage profit: 0.178243%
  Profit in starting currency: 0.001782
  Profit in USD: 0.000017
---------------------------------------
Cycle: JPY → GBP → EUR → JPY
  Product of rates: 1.002285
  Percentage profit: 0.228453%
  Profit in starting currency: 0.002285
  Profit in USD: 0.000021
---------------------------------------
Cycle: JPY → GBP → CHF → JPY
  Product of rates: 1.004381
  Percentage profit: 0.438090%
  Profit in starting currency: 0.004381
  Profit in USD: 0.000041
---------------------------------------
Cycle: JPY → GBP → CAD → JPY
  Product of rates: 1.003581
  Percentage profit: 0.358062%
  Profit in starting currency: 0.003581
  Profit in USD: 0.000033
---------------------------------------
Cycle: JPY → GBP → AUD → JPY
  Product of rates: 1.005831
  Percentage profit: 0.583101%
  Profit in starting currency: 0.005831
  Profit in USD: 0.000054
---------------------------------------
Cycle: JPY → CHF → AUD → JPY
  Product of rates: 1.000404
  Percentage profit: 0.040427%
  Profit in starting currency: 0.000404
  Profit in USD: 0.000004
---------------------------------------
Cycle: JPY → CAD → AUD → JPY
  Product of rates: 1.001335
  Percentage profit: 0.133539%
  Profit in starting currency: 0.001335
  Profit in USD: 0.000012
---------------------------------------
Cycle: GBP → USD → JPY → GBP
```

```
  Product of rates: 1.001782
  Percentage profit: 0.178243%
  Profit in starting currency: 0.001782
  Profit in USD: 0.003124
----------------------------------------
Cycle: GBP → EUR → JPY → GBP
  Product of rates: 1.002285
  Percentage profit: 0.228453%
  Profit in starting currency: 0.002285
  Profit in USD: 0.004003
----------------------------------------
Cycle: GBP → CHF → JPY → GBP
  Product of rates: 1.004381
  Percentage profit: 0.438090%
  Profit in starting currency: 0.004381
  Profit in USD: 0.007677
----------------------------------------
Cycle: GBP → CAD → JPY → GBP
  Product of rates: 1.003581
  Percentage profit: 0.358062%
  Profit in starting currency: 0.003581
  Profit in USD: 0.006275
----------------------------------------
Cycle: GBP → AUD → JPY → GBP
  Product of rates: 1.005831
  Percentage profit: 0.583101%
  Profit in starting currency: 0.005831
  Profit in USD: 0.010218
----------------------------------------
Cycle: CHF → JPY → USD → CHF
  Product of rates: 1.000756
  Percentage profit: 0.075579%
  Profit in starting currency: 0.000756
  Profit in USD: 0.000666
----------------------------------------
Cycle: CHF → JPY → GBP → CHF
  Product of rates: 1.004381
  Percentage profit: 0.438090%
  Profit in starting currency: 0.004381
  Profit in USD: 0.003858
----------------------------------------
Cycle: CHF → AUD → JPY → CHF
  Product of rates: 1.000404
  Percentage profit: 0.040427%
  Profit in starting currency: 0.000404
  Profit in USD: 0.000356
----------------------------------------
Cycle: CAD → JPY → GBP → CAD
  Product of rates: 1.003581
  Percentage profit: 0.358062%
  Profit in starting currency: 0.003581
  Profit in USD: 0.003347
----------------------------------------
Cycle: CAD → AUD → JPY → CAD
  Product of rates: 1.001335
  Percentage profit: 0.133539%
```

```
  Profit in starting currency: 0.001335
  Profit in USD: 0.001248
---------------------------------------
Cycle: AUD → JPY → USD → AUD
  Product of rates: 1.002209
  Percentage profit: 0.220873%
  Profit in starting currency: 0.002209
  Profit in USD: 0.001773
---------------------------------------
Cycle: AUD → JPY → EUR → AUD
  Product of rates: 1.001413
  Percentage profit: 0.141304%
  Profit in starting currency: 0.001413
  Profit in USD: 0.001134
---------------------------------------
Cycle: AUD → JPY → GBP → AUD
  Product of rates: 1.005831
  Percentage profit: 0.583101%
  Profit in starting currency: 0.005831
  Profit in USD: 0.004681
---------------------------------------
Cycle: AUD → JPY → CHF → AUD
  Product of rates: 1.000404
  Percentage profit: 0.040427%
  Profit in starting currency: 0.000404
  Profit in USD: 0.000325
---------------------------------------
Cycle: AUD → JPY → CAD → AUD
  Product of rates: 1.001335
  Percentage profit: 0.133539%
  Profit in starting currency: 0.001335
  Profit in USD: 0.001072
---------------------------------------
```

One-Way Arbitrage

The second part addresses **one-way arbitrage**. The user inputs the currency they wish to sell and the currency they want to buy. The program then evaluates all possible two-step indirect paths (e.g., EUR → CHF → USD) and compares their effective rates against the direct conversion rate.

If a path yields a better exchange rate than the direct conversion, the program highlights it as a potential arbitrage opportunity.

Example Result:
When selling EUR to buy USD, the best indirect route found was **EUR → CHF → USD**, offering a better rate than the direct EUR → USD exchange, resulting in a potential gain of approximately **0.10%**.

```
import pandas as pd
from itertools import permutations

def load_matrix(file_path):
    return pd.read_excel(file_path, index_col=0).T

def get_effective_rate(path, matrix):
    rate = 1.0
    for i in range(len(path) - 1):
        from_cur = path[i]
        to_cur = path[i + 1]
        r = matrix.loc[from_cur, to_cur]
        if pd.isna(r):
            return None  # Invalid path
        rate *= r
    return rate

def find_one_way_arbitrage(matrix, sell_currency, buy_currency):
    currencies = list(matrix.columns)
    if sell_currency not in currencies or buy_currency not in currencies:
        print("Invalid currency codes.")
        return

    paths = []

    # Allow up to 3-step paths (i.e., 1 or 2 intermediate currencies)
    for intermediate in currencies:
        if intermediate not in [sell_currency, buy_currency]:
            # 2-hop: sell → intermediate → buy
            path = [sell_currency, intermediate, buy_currency]
            rate = get_effective_rate(path, matrix)
```

```
            if rate:
                paths.append((path, rate))

    # Also include direct path
    direct_rate = matrix.loc[sell_currency, buy_currency]
    if not pd.isna(direct_rate):
        paths.append(([sell_currency, buy_currency], direct_rate))

    # Sort paths by effective rate (most favorable last currency amount per 1
unit sold)
    paths.sort(key=lambda x: x[1], reverse=True)

    print(f"\nAll possible paths from {sell_currency} to {buy_currency}:")
    for path, rate in paths:
        print(f"  Path: {' → '.join(path)} | Effective Rate: {rate:.6f}")

    if len(paths) > 1:
        best_path = paths[0]
        direct = next(((p, r) for p, r in paths if p == [sell_currency,
buy_currency]), None)
        if direct and best_path[1] > direct[1]: #* 1.0001:
            print("\n Arbitrage Opportunity Detected!")
            print(f"  Best Path: {' → '.join(best_path[0])} yields more than
direct route")
            print(f"  Gain: {(best_path[1] / direct[1] - 1) * 100:.4f}%")

def main():
    file_path = 'Quotation Matrix.xlsx'
    matrix = load_matrix(file_path)

    print("Available currencies:", list(matrix.columns))

    sell_currency = input("Enter the currency you want to SELL:
").strip().upper()
    buy_currency = input("Enter the currency you want to BUY:
").strip().upper()

    if sell_currency == buy_currency:
        print("Sell and buy currencies must differ.")
        return

    find_one_way_arbitrage(matrix, sell_currency, buy_currency)

if __name__ == "__main__":
    main()
```

```
All possible paths from EUR to USD:
  Path: EUR → CHF → USD | Effective Rate: 1.401395
  Path: EUR → JPY → USD | Effective Rate: 1.401302
  Path: EUR → CAD → USD | Effective Rate: 1.401198
  Path: EUR → GBP → USD | Effective Rate: 1.400180
  Path: EUR → AUD → USD | Effective Rate: 1.400050
  Path: EUR → USD | Effective Rate: 1.399975

 Arbitrage Opportunity Detected!
  Best Path: EUR → CHF → USD yields more than direct route
  Gain: 0.1015%
```