Currency Exchange Problem

Problem Definition

We are given 10 currencies in total, say `C1`, `C2`, ..., `C10`

Initially, we have c_init[n] Finally, we are supposed to have

There is an exchange matrix F (n * n)

- `Fij`: Number of units of j it takes to buy 1 unit of currency i
- `Example`: Use `F21` amount of currency 2 to buy 1 USD
- `1/Fij`: **Bid Price** of Cj in terms of Ci
- `Fji`: **Ask Price** of Cj in terms of Ci

Value of currency j in terms of USD = sqrt (Fj1 / Fij)

Find a matrix X (n * n) (Exchange Matrix)

- `Xij`: The amount of Cj converted into Ci
- We obtain `Xij/Fij` of Ci after spending 1 Cj
- The total of Ci spent can not exceed the initial ci
- In the end after the conversion, we are left with at least c_req

Solution

Value

Say we have `c = (c1, c2, ..., c10)` units of (`C1`, `C2`, ..., `C10`)

c is a (n * 1) Matrix

Define vals (1 * n), a column matrix where ith entry is the value of Ci

```
vals = [sqrt(Fi1/F1i) for all i]
```

```
Value(c) = vals * c
```

```
Dim(Value(c)) = 1
```

Objective Function

The Objective Function is the function we're trying to minimize

Here, we're trying to minimize the loss of value in total currencies we will have

Value Lost = Value(c_init) - Value(c_final)

Therefore, Our objective here is

`Value(c_init) - Value(c_final)`

c_final

This is the final list of currencies we'll have

`c_final = c_init + c_gained_due_to_exchnage - c_lost_due_to_exchange`

`c_gained_due_to_exchange`:

- This is the list of currency that is exchanged and gained
- For Ci, This is equal to sum of `Xij/Fij` over all j
- So, this is equal to `X/F @ [1, 1, ..., 1]`

`c_lost_due_to_exchange`:

- This is the list of the currency that is used for exchange and is hence lost
- For Ci, This is equal to sum of `Xji` over all j
- So, this is equal to `Transpose(X) @ [1, 1, ..., 1]`

Constraints

- `Xij` >= 0 for all i, j
- `Xii` = 0 for all i
- `c_lost_due_to_exchange` <= c_init
- `c_final` >= c_req

Implementation

```
import numpy as np
import cvxpy as cp

# Exchange rate data.
tickers = ["USD", "EUR", "GBP", "CAD", "JPY", "CNY", "RUB", "MXN",
"INR", "BRL"]
n = len(tickers)
F = np.zeros((n, n))

data = (
    # USD
    [1.0, 0.87, 0.76, 1.31, 108.90, 6.72, 65.45, 19.11, 71.13, 3.69],
    # EUR
    [1.0, 0.88, 1.51, 125.15, 7.72, 75.23, 21.96, 81.85, 4.24],
    # GBP
    [1.0, 1.72, 142.94, 8.82, 85.90, 25.08, 93.50, 4.84],
    # CAD
    [1.0, 82.93, 5.11, 49.82, 14.54, 54.23, 2.81],
    # JPY
```

```
    [1.0, 0.062, 0.60, 0.18, 0.65, 0.034],
    # CNY
    [1.0, 9.74, 2.85, 10.61, 0.55],
    # RUB
    [1.0, 0.29, 1.09, 0.056],
    # MXN
    [1.0, 3.73, 0.19],
    # INR
    [1.0, 0.052],
    # BRL
    [1.0]
)

for i in range(n):
    F[i,i:] = data[i]

for j in range(n):
    for i in range(j+1,n):
        F[i,j] = 1.035/F[j,i]

# Initial and final portfolios.
c_req = np.arange(1,n+1)
c_req = 1e4*c_req/c_req.sum()
c_init = c_req[::-1]

X = cp.Variable((n, n))

from math import sqrt
# vals (1 * n)
vals = np.array([[sqrt(F[i, 0]/F[0, i])] for i in range(0, n)])
# vals = np.sqrt(F[:,0]/F[0,:])

def value(c):
    return c @ vals

def value_lost(c_final):
    return value(c_init - c_final)

c_final = c_init + (X/F)@np.ones(n) - X.T@np.ones(n)

objective = cp.Minimize(value_lost(c_final))

constraints = [
               0 <= X,
               0 == cp.diag(X),
               X.T@np.ones(n) <= c_init,
               c_final >= c_req
]
```

```
prob = cp.Problem(objective, constraints)
prob.solve()

print("Minimal Value Lost:", value_lost(c_final.value), "USD")
```

```
Minimal Value Lost: [7.72005934] USD
```