Shipping pair trading code from Matlab to Python

Imports

Importig Two helper functions from `./helper_functions`:

- `stock_list`: This functions gets an index name (e.g. 'Dow Jones', 'CAC 40', 'DAX', 'Teh50') and returns the list of stocks in that index.
- `stock_prices`: This functions recieves a list of tickers and returns a pandas dataframe containing prices of the corresponding tickers.

[11]:
```
from helper_functions import stock_list, stock_prices
```

[12]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Constants and Fucntions

We will work with the 521 trailing days of prices series. 126 last days will be the test period.

[13]:
```
testsmpl=126

interval = 521
```

Cointegration function

We explain the code in 7 steps. The starting point of each step is commented in the code by the corresponding number:

1. Importing required functions and classes.
2. We define a function to test the cointegration relation and return the parameters if the relation holds. The function gets a two-column dataframe consisting of prices series for two tickers and a `verbose` flag that will make the function to print extra information if set to True.
3. The best lag order for the Cointegration test is determined using `select_order` function.
4. We try to select the best cointegratino rank for our two series. The `select_coint_rank` functions uses the Johansen Test internally.
5. If the stat of the test is greater than the critical value, We can conclude that the two series are cointegrated.
6. If two series are not cointegrated, we return False as the test result and NaN as the parameters of VECM model.
7. To find the VECM parameters, We use the `VECM` class.

[14]:
```
# 1
from statsmodels.tsa.vector_ar.vecm import VECM, select_coint_rank, select_order

# 2
def get_cointegration_params(df, verbose=False):
    # 3
    lag_order = select_order(df, maxlags=10, deterministic="ci")
    lag_order = lag_order.aic
    # 4
    rank_test = select_coint_rank(df, 0, lag_order, method="trace",
                                  signif=0.05)

    # 5
    is_cointegrated = rank_test.test_stats[0] > rank_test.crit_vals[0]
    if verbose:
        print(rank_test.summary())

    # 6
    if not is_cointegrated:
        return False, np.NaN, np.NAN

    # 7
    model = VECM(df, deterministic="ci",
             k_ar_diff=lag_order,
             coint_rank=rank_test.rank)
    vecm_res = model.fit()

    return True, vecm_res.beta, vecm_res.const_coint
```

This helper function Will convert the arabic glyphs to standard farsi glyphs. This will be helpful while looking Tehran50 tickers up:

[15]:
```
def groom(s):
    s = s.replace(' ', ' '(
    s = s.replace(' ', ' '(
    return s
```

Suppressing all the warnings in order to have a clear output:

[16]:
```
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

Pair Trading

We explain the code in 7 steps. The starting point of each step is commented in the code by the corresponding number:

1. We Will save all the plots in the `./plots/{index_name}/` directory. We will remove and recreate the directory each time we run the code.
2. We write a for loop that itterates over four indices: 'Dow Jones', 'CAC 40', 'Dax', 'Teh50'
3. We get the list of tickers in the given indice and then get the price of all the tickers in that index. We split the data into train and test.
4. For each pair of tickers, we test for cointegration using `get_cointegration_params` function and if They are in fact cointegrated, we save them in the `pairs` list.
5. We compute the ECM of the cointegrated pairs and try to extract Long/Short signals from them.
6. We plot 4 graphs: prices of the tickers, The cointegratino Relatino, Out of Sample Cumulative Return and In Sample Cumulative Return
7. We save the cointegrates pairs in an excel file to use it in other part of the project.

```
[17]: # 1
import shutil
PATH = r'./plots/'
if os.path.exists(PATH):
    shutil.rmtree(PATH)
os.makedirs(PATH)

pairs = []

# 2
for indice in ['Dow Jones', 'CAC 40', 'Dax', 'Teh50']:
    print(indice, sep=' ', end='', flush=True)

    # Creating the required directories to save the plots
    PATH = rf'./plots/{indice}/'
    if not os.path.exists(PATH):
        os.makedirs(PATH)

    # 3
    tickers = stock_list.get_stock_list(index=indice)
    symbolsnum = len(tickers)
    isTSE = (indice == 'Teh50')
    if isTSE:
        tickers = [groom(x) for x in tickers]
    data_historical = stock_prices.get_prices(tickers, is_tse=isTSE)
    data_historical = data_historical.dropna(how='all')
    data = data_historical[-interval:]
    limitPer = len(data) * .85
    data = data.dropna(thresh=limitPer, axis=1)
    data = np.log(data)
    data_train = data[:-testsmpl]
    data_test = data[-testsmpl:]

    # 4
    cols = data_train.columns
    for i in range(len(cols)-1):
        for j in range(i+1, len(cols)):
            df_train = data_train[[cols[i], cols[j]]].copy()
            df_test = data_test[[cols[i], cols[j]]].copy()
            try:
                is_cointegrated, BJ2n, COJ2n = get_cointegration_params(df_train.
 ↪dropna(how='any'))
            except:
                continue
            if not is_cointegrated:
                continue
            pairs.append({
                'sym1': cols[i],
                'sym2': cols[j],
                'indice': indice
            })
        # 5
        cointRinsmpl = np.matmul(df_train, BJ2n) + C0J2n
        cointRtest = np.matmul(df_test, BJ2n) + C0J2n

        scointR = np.std(cointRinsmpl)[0]
        mcointR = np.mean(cointRinsmpl)[0]

        cointR = cointRinsmpl.append(cointRtest)
        longs = cointR<=mcointR-2*scointR
        shorts=cointR>=mcointR+2*scointR;
        exitLongs=cointR>=mcointR-1*scointR;
        exitShorts=cointR<=mcointR+1*scointR;

        positionsL = np.zeros((cointR.shape[0], 2))
        positionsS = np.zeros((cointR.shape[0], 2))

        positionsL = pd.DataFrame(positionsL)
        positionsS = pd.DataFrame(positionsS)


        positionsL.iloc[positionsL[longs.values].index, 0] = 1
        positionsL.iloc[positionsL[longs.values].index, 1] = -1
        positionsL.iloc[positionsL[exitLongs.values].index, 0] = 0
        positionsL.iloc[positionsL[exitLongs.values].index, 1] = 0

        positionsS.iloc[positionsS[shorts.values].index, 0] = -1
        positionsS.iloc[positionsS[shorts.values].index, 1] = 1
        positionsS.iloc[positionsS[exitShorts.values].index, 0] = 0
        positionsS.iloc[positionsS[exitShorts.values].index, 1] = 0

        positions = positionsL + positionsS

        yret = np.log(df_train.append(df_test)).diff()
        yret = yret[1:]

        pnl=(
        positions[0:-1][0] * yret[yret.columns[0]].values
        - BJ2n[1][0]*positions[0:-1][1]*yret[yret.columns[1]].values
        )

        rsuminsmpl = np.cumsum(pnl[:-df_test.shape[0]])
        rsumtest = np.cumsum(pnl[-df_test.shape[0]:])

        ShrpRatinsmpl = np.sqrt(252)*np.mean(pnl[:-df_test.shape[0]])/np.std(pnl[:
↪-df_test.shape[0]])
        ShrpRatiTest = np.sqrt(252)*np.mean(pnl[-df_test.shape[0]:])/np.
↪std(pnl[-df_test.shape[0]:])

        # 6
        ticker1, ticker2 = df_train.columns
```

```
            fig, axs = plt.subplots(2, 2, figsize=(20, 10))
            axs[0, 0].plot(df_train[ticker1])
            axs[0, 0].plot(df_test[ticker1])
            axs[0, 0].plot(df_train[ticker2])
            axs[0, 0].plot(df_test[ticker2])
            axs[0, 0].set_title(f'Pair Prices for {ticker1} and {ticker2}')
            axs[0, 0].tick_params(axis='x', rotation=15)

            axs[0, 1].plot(cointR[:df_train.shape[0]])
            axs[0, 1].plot(cointR[-df_test.shape[0]:])
            axs[0, 1].set_title(f'Cointegrating Relations for {ticker1} and {ticker2}')
            axs[0, 1].plot(cointR.index, [mcointR - 2*scointR]*cointR.shape[0])
            axs[0, 1].plot(cointR.index, [mcointR + 2*scointR]*cointR.shape[0])
            axs[0, 1].tick_params(axis='x', rotation=15)

            axs[1, 0].plot(df_test.index, rsumtest)
            axs[1, 0].set_title(f'Out of Sample Cumulative Return for Pair {ticker1}␣
 ↪and {ticker2}')

            axs[1, 1].plot(df_train.index[1:], rsuminsmpl)
            axs[1, 1].set_title(f'In Sample Cumulative Return for Pair {ticker1} and␣
 ↪{ticker2}');
            axs[1, 1].tick_params(axis='x', rotation=15);

            fig.subplots_adjust(hspace=.3);

            fig.savefig(rf'./plots/{indice}/cointr_{ticker1}_{ticker2}');
            plt.close()

# 7
import datetime
filename = rf'./pairs_{str(datetime.datetime.now().date())}.xlsx'
writer = pd.ExcelWriter(filename, engine='xlsxwriter')
df_pairs = pd.DataFrame(pairs)
for index, group_df in df_pairs.groupby("indice"):
    group_df.to_excel(writer, sheet_name=str(index),index=False)
writer.save()
```

```
[*********************100%***********************]  30 of 30 completed
[*********************100%***********************]  40 of 40 completed

1 Failed download:
- OCBI: No data found, symbol may be delisted
[*********************100%***********************]  40 of 40 completed

1 Failed download:
- AZSEY: No data found, symbol may be delisted
Teh50
```