IDSP Program Homework，劉高甸，R12945070

前置作業、import、讀取檔案：

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

# Q2~Q7 synthetic_data
synthetic_data = pd.read_csv('synthetic_time_series.csv')
node_labels_synthetic = synthetic_data.columns.values

# Q2~Q7  原始 stock_data
stock_data = pd.read_csv('stock_price_globe.csv')
#  計算 Q1
stock_data['Log_Return'] = np.log(stock_data['Adj Close'] / stock_data['Adj Close'].shift(1))
#stock_data['Adj Close'] = stock_data['Adj Close'].ffill() #  好像用不到，Fill the first row with itself to avoid NaN
# Set the first row's Log_Return to natural log (1)
stock_data.loc[0, 'Log_Return'] = 0.0001
#  讀取新的 stock_data
stock_data.to_csv('log_returns_R12945070.csv', index=False)
stock_data = pd.read_csv('log_returns_R12945070.csv')

# Extract column names for node labeling
num_countries = 18
data_length = 814
node_labels = [stock_data.iloc[i, 0] for i in range(0, len(stock_data), data_length)]
# Extract 'Log_Return' column
log_returns = stock_data['Log_Return'].values

#  檢查 log_returns_R12945070.csv
#print("Shape of log_returns:", log_returns.shape)
#print(len(node_labels))
#print(node_labels)
```

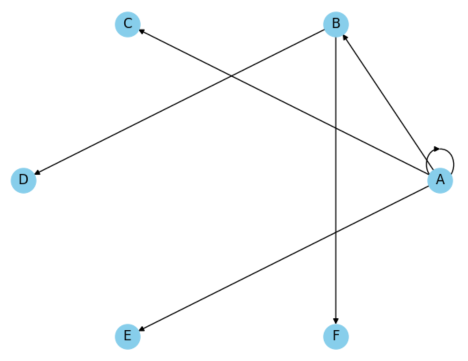
Q1：

```
Log Returns of Last 5 Dates in Taiwan:
[-0.00041077  0.00389542 -0.00842499  0.00267906  0.00185052]
```

===

```
data = pd.read_csv('stock_price_globe.csv')
#data = pd.read_csv('stock_price_globe.csv',index_col = [0,1],skipinitialspace=True)
data.head()
# 選擇最後 5 個日期的資料
last_5_data = data.loc[11390:11395] #把第二列當 0，實際範圍[0:14651]

# 計算對數收益率
adj_close = last_5_data['Adj Close']
adj_close_shifted = adj_close.shift(1)   # 取得上一期的 adj_close

'''
# 顯示每一列的 adj_close 和 adj_close.shift(1)
for idx, (close, close_shifted) in enumerate(zip(adj_close, adj_close_shifted)):
    print(f"Row {idx + 1}: adj_close = {close}, adj_close_shifted = {close_shifted}")
'''

# 計算對數收益率
log_returns_Q1 = np.log(adj_close / adj_close_shifted)

#Q1 顯示最後 5 個日期的對數收益率
print("\nLog Returns of Last 5 Dates in Taiwan:")
print(log_returns_Q1[1:].values)   # 不印出索引為 11390 的結果
#print("Shape of log returns:", log_returns.shape)
```

Q2：



===

```
def omp(X, y, sparsity):
    n_features = X.shape[1]
    idxs = []
    res = y.copy()
    for _ in range(sparsity):
        corr = np.abs(X.T @ res)
        max_idx = np.argmax(corr)
        idxs.append(max_idx)
        X_idx = X[:, idxs]
        beta = np.linalg.lstsq(X_idx, y, rcond=None)[0]
        #beta = np.linalg.pinv(X_idx) @ y
        res = y - X_idx @ beta
    return idxs

# Q2: OMP for synthetic_time_series.csv
def synthetic_omp(synthetic_data, lag, sparsity):
    synthetic_data = synthetic_data.values
    synthetic_returns = np.log(synthetic_data[1:] / synthetic_data[:-1])
    idxs = []

    for i in range(synthetic_returns.shape[1]):
        X = np.column_stack(
            [synthetic_returns[:, j].reshape(-1, 1) for j in range(synthetic_returns.shape[1]) if j != i])
        y = synthetic_returns[:, i]
        idxs.append(omp(X, y, sparsity))

    G = nx.DiGraph()
    for i, idx in enumerate(idxs):
        for j in idx:
            G.add_edge(node_labels_synthetic[j], node_labels_synthetic[i])    # Use node_labels_synthetic for
node names
    pos = nx.circular_layout(G)
```

```
    nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500)
    plt.title('OMP for synthetic_time_series.csv')
    plt.show()

# Q2: OMP for synthetic_time_series.csv
lag = 5
sparsity = 1
synthetic_omp(synthetic_data, lag, sparsity)
```

Q3：

===

```
# Q3: OMP for log_returns.csv
def stock_omp(stock_data, lag, sparsity):
    # Reshape the data
    log_returns = stock_data['Log_Return'].values
    idxs = []
    for i in range(0, len(log_returns), data_length):
        country_returns = log_returns[i:i+data_length]
        X = np.column_stack([log_returns[j:j+data_length] for j in range(0, len(log_returns), data_length) if j != i])
        y = country_returns
        idxs.append(omp(X, y, sparsity))

    G = nx.DiGraph()
    for i, idx in enumerate(idxs):
        for j in idx:
            G.add_edge(node_labels[j], node_labels[i])

    pos = nx.circular_layout(G)
    nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500)
    plt.title('OMP for log_returns.csv')
    plt.show()

# Q3: OMP for log_returns.csv
lag = 20
sparsity = 1
stock_omp(stock_data, lag, sparsity)
```

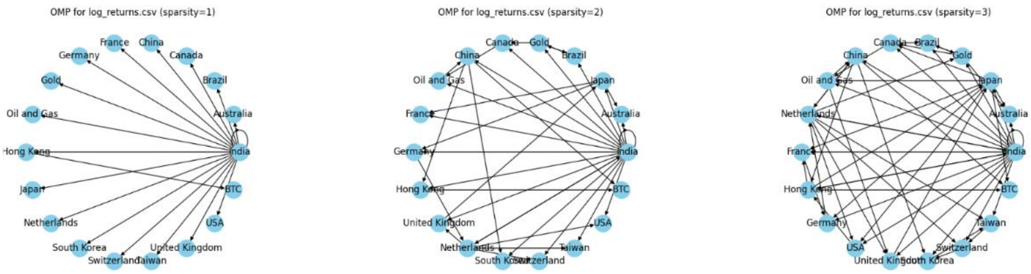
Q4：

Discuss the situations when L=20 and sparsity=1,2,3. How sparsity affects the result.

當 L=20，而 sparsity=1,2,3 則表示我們將選擇 1、2 或 3 個最重要的特徵。

Sparsity=1：只選擇了一個最重要的特徵。這意味著模型非常簡化，僅使用了最重要的因素來預測目標變量，可能會導致模型的預測能力不足。
Sparsity=2：選擇了兩個最重要的特徵。相對於 Sparsity=1，模型複雜度略微增加，可以更好地捕捉特徵之間的關係，預測能力可能會有所提高，但仍保持了一定的解釋性。
Sparsity=3：選擇了三個最重要的特徵。模型的複雜度進一步增加，可以更好地捕捉特徵之間的複雜關係，預測能力也可能進一步提高，但同時可能會失去一些解釋性。

隨著 sparsity 的增加，模型的複雜度和預測能力通常會增加，但同時可能會減少模型的解釋性。

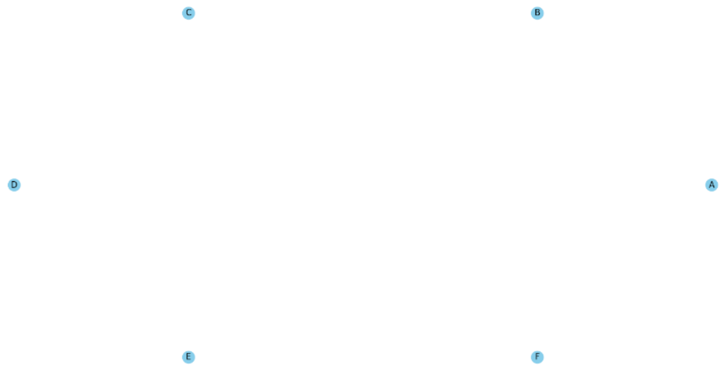


＝＝＝

```
def stock_omp_multiple_sparsity(stock_data, lag, sparsity_list):
    fig = plt.figure(figsize=(22, 7))    # Increase the width of the figure
    grid = plt.GridSpec(1, len(sparsity_list), wspace=0.5)    # Adjust the horizontal space between subplots

    for i, sparsity in enumerate(sparsity_list):
        ax = fig.add_subplot(grid[0, i])    # Use a single subplot for each sparsity value

        idxs = []
        for j in range(0, len(log_returns), data_length):
            country_returns = log_returns[j:j+data_length]
            X = np.column_stack([log_returns[k:k+data_length] for k in range(0, len(log_returns), data_length) if 
k != j])
            y = country_returns
            idxs.append(omp(X, y, sparsity))

        G = nx.DiGraph()
        for j, idx in enumerate(idxs):
            for k in idx:
                G.add_edge(node_labels[k], node_labels[j])

        pos = nx.circular_layout(G)
```

```python
        nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500, ax=ax)
        ax.set_title(f'OMP for log_returns.csv (sparsity={sparsity})')

    plt.subplots_adjust(left=0.05, right=0.95, top=0.85, bottom=0.15)    # Manually adjust subplot spacing
    #plt.savefig('omp_log_returns.png', dpi=300, bbox_inches='tight')
    plt.show()

# Q4: OMP for log_returns.csv with multiple sparsity values
lag = 20
sparsity_list = [1, 2, 3]
stock_omp_multiple_sparsity(stock_data, lag, sparsity_list)
```

Q5：

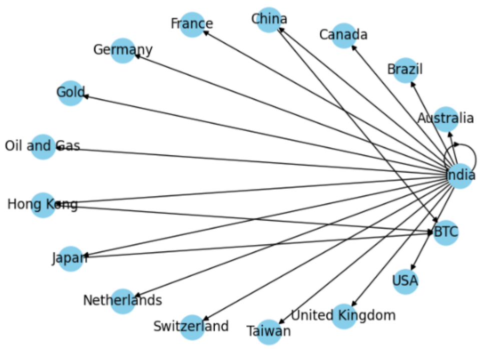


＝＝＝

```
def coordinate_descent_lasso(X, y, lambda_val):
    n_samples, n_features = X.shape
    B = np.zeros(n_features)
    alpha = 1 / (2 * n_samples)    # Learning rate
    max_iter = 1000
    for _ in range(max_iter):
        for j in range(n_features):
            X_j = X[:, j]
            R = y.flatten() - X @ B.flatten() + X_j * B[j]    # Flatten B to match the shape of X
            B[j] = np.sign(np.dot(X_j, R)) * max(0, abs(np.dot(X_j, R)) - alpha * lambda_val) / np.dot(X_j, X_j)
            print(B)
    return B

def coordinate_descent_lasso(X, y, lambda_val):
    n_samples, n_features = X.shape
    B = np.zeros(n_features)
    alpha = 1 / (2 * n_samples)    # Learning rate
    max_iter = 1000
    for _ in range(max_iter):
        for j in range(n_features):
            X_j = X[:, j]
            R = y.flatten() - X @ B.flatten() + X_j * B[j]    # Flatten B to match the shape of X
            B[j] = np.sign(np.dot(X_j, R)) * max(0, abs(np.dot(X_j, R)) - alpha * lambda_val) / np.dot(X_j, X_j)
    return B

def lasso_granger_graph(synthetic_data, lag, lambda_val):
    synthetic_data = synthetic_data.values
    synthetic_returns = np.log(synthetic_data[1:] / synthetic_data[:-1])
    idxs = []
    isolated_nodes = []

    for i in range(synthetic_returns.shape[1]):
```

```
        X = np.column_stack(
            [synthetic_returns[:, j].reshape(-1, 1) for j in range(synthetic_returns.shape[1]) if j != i])
        y = synthetic_returns[:, i]
        beta = coordinate_descent_lasso(X, y, lambda_val)
        if (beta == 0).all():   # Check if all elements in beta are zero
            isolated_nodes.append(i)   # Store the index of the isolated node
        else:
            idxs.append(np.where(beta != 0)[0])   # Append the non-zero indices

    G = nx.DiGraph()
    for i, idx in enumerate(idxs):
        for j in idx:
            G.add_edge(node_labels_synthetic[j], node_labels_synthetic[i])   # Use node_labels_synthetic for
node names

    # Add isolated nodes
    for node in isolated_nodes:
        G.add_node(node_labels_synthetic[node])

    pos = nx.circular_layout(G)
    plt.figure(figsize=(10, 8))   # Adjust figure size
    nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=300)
    plt.title(f'Granger Graph for lag L={lag}, λ={lambda_val}')
    plt.axis('off')
    plt.show()

# Q5
lag_Q5 = 5
lambda_Q5 = 1e7
lasso_granger_graph(synthetic_data, lag_Q5, lambda_Q5)
```

Q6：



===

```
def lasso_coordinate_descent(X, y, lamb, max_iter=1000):
    n_samples, n_features = X.shape
    beta = np.zeros(n_features)

    for _ in range(max_iter):
        for j in range(n_features):
            X_j = X[:, j]
            r_j = y - X @ beta + beta[j] * X_j
            beta[j] = np.sign(X_j @ r_j) * max(abs(X_j @ r_j) - lamb, 0) / (X_j @ X_j)

    return beta

# Q6: Lasso for log_returns.csv
def lasso_granger(stock_data, lag, lamb):
    log_returns = stock_data['Log_Return'].values
    idxs = []
    for i in range(0, len(log_returns), data_length):
        country_returns = log_returns[i:i + data_length]
        X = np.column_stack(
            [log_returns[j:j + data_length - lag] for j in range(0, len(log_returns) - lag, data_length) if j != i])
        y = country_returns[lag:]
        beta = lasso_coordinate_descent(X, y, lamb)
        idxs.append(np.where(beta != 0)[0])

    G = nx.DiGraph()
    for i, idx in enumerate(idxs):
        for j in idx:
            G.add_edge(node_labels[j], node_labels[i])

    pos = nx.circular_layout(G)
    nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500)
    plt.title(f'Lasso for log_returns.csv (lag={lag}, λ={lamb})')
```

```
    plt.show()

# Q6: Lasso for log_returns.csv
lag = 20
lamb = 0.03
lasso_granger(stock_data, lag, lamb)
```

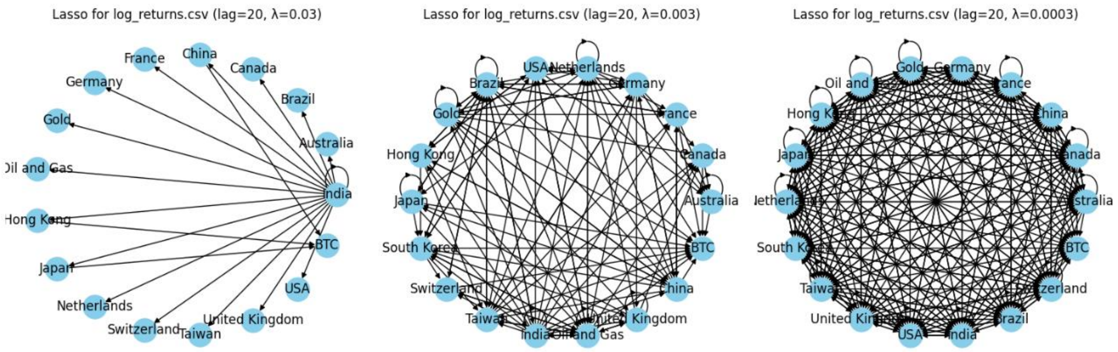
Q7：

Discuss the situations when the parameter λ of ℓ1 regularization term changed. (try different λs)

在 Coordinate Descent for Lasso 中，λ（lambda）是 ℓ1 正則化項的參數，
控制著特徵的選擇和模型的擬合程度。

較小的 λ 值：當 λ 值很小時，正則化的影響減弱，模型更傾向於擬合訓練數據，可能導致過度擬合。
適度的 λ 值：適度的 λ 值可以平衡模型的擬合和特徵的選擇，通常可以得到較好的泛化能力。
較大的 λ 值：當 λ 值增加時，正則化的作用更明顯，模型更傾向於選擇少量重要特徵，可能會提高模型的泛化能力，降低過度擬合。

因此，適當調整 λ 值可以幫助我們在模型的擬合和泛化能力之間取得平衡。



＝＝＝

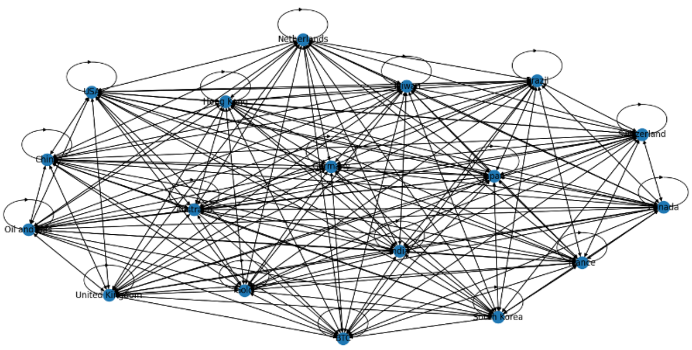
```
# Q7: Lasso for each lamb in the same figure
def lasso_granger_multiple(stock_data, lag, lamb_values):
    fig, axes = plt.subplots(1, len(lamb_values), figsize=(15, 5))    # create 1 row, with a number of subplots for each lambda value

    # go through and model each lambda value on a subplot
    for ix, lamb in enumerate(lamb_values):
        log_returns = stock_data['Log_Return'].values
        idxs = []
        for i in range(0, len(log_returns), data_length):
            country_returns = log_returns[i:i + data_length]
            X = np.column_stack(
                [log_returns[j:j + data_length - lag] for j in range(0, len(log_returns) - lag, data_length) if j != i])
            y = country_returns[lag:]
            beta = lasso_coordinate_descent(X, y, lamb)
            idxs.append(np.where(beta != 0)[0])

        G = nx.DiGraph()
        for i, idx in enumerate(idxs):
            for j in idx:
```

```
                G.add_edge(node_labels[j], node_labels[i])

        pos = nx.circular_layout(G)
        nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500, ax=axes[ix])   # Use current axis
        axes[ix].set_title(f'Lasso for log_returns.csv (lag={lag}, λ={lamb})')   # Add title for each axis

    plt.tight_layout()   # fits plots nicely in figure
    plt.show()   # Shows figure with all subplots

# Q7: Lasso for log_returns.csv with multiple lambda values
lag = 20
lamb_values = [0.03, 0.003, 0.0003]   # Adjust the lambda values as needed
lasso_granger_multiple(stock_data, lag, lamb_values)
```

加分題：

Considering the limitations of OMP and Lasso in precisely identifying the ground truth within synthetic datasets, propose an algorithm or explore existing alternatives that could potentially overcome these shortcomings. Describe your proposed solution in detail, explaining how it would outperform OMP and Lasso. Additionally, discuss any potential challenges in implementing your proposed solution and how they could be addressed.

一種可能的優化方法是 Elastic Net（彈性網）演算法。Elastic Net 是 Lasso 和 Ridge 回歸（一種使用 L2 規範進行最小化的 ReLU 演算法）的一種結合。
它適用於不同的問題，提供更好的準確度，特別是在處理具有許多相關特徵的數據集時。
Elastic Net 使用了一個混合參數 alpha 來平衡 L1 和 L2 的使用。當 alpha 設置為 1，它相當於 Lasso，而 alpha 為 0 時，它就是 Ridge。
透過這種方式，可以在保留 Lasso 稀疏性的同時，避免 OMP 在處理高度相關特徵時的過度擬合問題。



＝＝＝

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

# 讀取及整理資料
stock_data = pd.read_csv('stock_price_globe.csv')
stock_data['Log_Return'] = np.log(stock_data['Adj Close'] / stock_data['Adj Close'].shift(1))
stock_data.loc[0, 'Log_Return'] = 0.0001
stock_data.to_csv('log_returns_R12945070.csv', index=False)
stock_data = pd.read_csv('log_returns_R12945070.csv')

num_countries = 18
data_length = 814
node_labels = [stock_data.iloc[i, 0] for i in range(0, len(stock_data), data_length)]
log_returns = stock_data['Log_Return'].values

# 實現 Elastic Net
def elastic_net(X, y, alpha, l1_ratio):
    num_iters = 1000
```

```
    learning_rate = 0.1
    beta = np.zeros(X.shape[1])

    m = len(y)

    for _ in range(num_iters):
        gradients = 2 / m * X.T.dot(X.dot(beta) - y) + alpha * l1_ratio * np.sign(beta) + alpha * (1 - l1_ratio) * beta
        beta -= learning_rate * gradients

    return beta

def elastic_net_granger(stock_data, lag, alpha, l1_ratio):
    idxs = []
    #  逐個計算國家的因果關係
    for i in range(num_countries):
        country_returns = log_returns[i:i + data_length][lag:]
        X = np.column_stack(
            [log_returns[j:j + data_length - lag] for j in range(0, len(log_returns) - lag, data_length) if j != i])

        beta = elastic_net(X, country_returns, alpha, l1_ratio)
        idxs.append(np.where(abs(beta) > 1e-5)[0])

    #  建立有向圖
    G = nx.DiGraph()
    for i, idx in enumerate(idxs):
        for j in idx:
            G.add_edge(node_labels[j], node_labels[i])

    nx.draw(G, with_labels=True)
    plt.title('Granger causality using Elastic Net')
    plt.show()

#加分題
lag = 20
alpha = 0.5
l1_ratio = 0.5
elastic_net_granger(stock_data, lag, alpha, l1_ratio)
```