**1-1.ipynb**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tools.eval\_measures import bic

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from statsmodels.stats.stattools import durbin\_watson

from statsmodels.graphics.api import qqplot

from scipy.special import inv\_boxcox

from scipy.stats import boxcox

from pandas.tseries.offsets import DateOffset

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

df = pd.read\_excel("数据.xlsx")

# 将时间列转换为时间格式，并将其设置为索引

df.iloc[:, 0] = pd.to\_datetime(df.iloc[:, 0])

df.set\_index(df.iloc[:, 0], inplace=True)

x = df.iloc[:, 0]

y = df.iloc[:, 1]

plt.figure(figsize=(10, 6))  # 设置图像大小

plt.plot(x, y/1e13, label='数据变化')  # 绘制线图

plt.xlabel('时间 (单位：季度)')

plt.ylabel('美国未偿还公共债务总额 (单位：十万亿美元)')

plt.title('1992-2023美国未偿还公共债务总额')

plt.savefig('结果1-1/1992-2023美国未偿还公共债务总额.png')

plt.show()

df\_daily = df.resample('D').mean().interpolate()

df\_mon = df\_daily.resample('ME').mean()

df\_quar = df\_daily.resample('QE').mean()

df\_year = df\_daily.resample('YE').mean()

dfs = [df\_daily, df\_mon, df\_quar, df\_year]

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

for i, ax in enumerate(axes.flat):

    ax.plot(dfs[i].iloc[:, 0], dfs[i].iloc[:, 1])

    ax.set\_title(['按日重采样', '按月重采样', '按季度重采样', '按年重采样'][i])

plt.tight\_layout()

plt.savefig('结果1-1/不同重采样.png')

plt.show()

fig, axes = plt.subplots(4, 2, figsize=(15, 20))

for i, ax in enumerate(axes.flat):

    if i % 2 == 0:

        plot\_acf(dfs[i // 2].iloc[:, 1], ax=ax, lags=15)

    else:

        plot\_pacf(dfs[i // 2].iloc[:, 1], ax=ax, lags=15)

    ax.set\_title(['按日重采样后的', '按月重采样后的', '按季度重采样后的', '按年重采样后的'][i // 2] + ['ACF', 'PACF'][i % 2])

plt.tight\_layout()

plt.savefig('结果1-1/不同重采样的ACF和PACF.png')

plt.show()

df\_quar\_d = df\_quar.diff().dropna()

df\_quar\_dd = df\_quar\_d.diff().dropna()

dfds = [df\_quar\_d, df\_quar\_dd]

# 画出两种差分后的自相关和偏自相关图

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

for i, ax in enumerate(axes.flat):

    if i % 2 == 0:

        plot\_acf(dfds[i // 2].iloc[:, 1], ax=ax, lags=30)

    else:

        plot\_pacf(dfds[i // 2].iloc[:, 1], ax=ax, lags=30)

    ax.set\_title(['一阶差分', '二阶差分'][i // 2] + ['ACF', 'PACF'][i % 2])

plt.tight\_layout

plt.savefig('结果1-1/一阶差分和二阶差分的ACF和PACF.png')

plt.show()

ts = df\_quar.iloc[:, 1]

# Box-Cox变换

ts\_boxcox, lambda\_ = boxcox(ts)

# ts\_boxcox = pd.Series(ts\_boxcox, index=ts.index).diff().dropna()

bcdf\_quar\_d = pd.Series(ts\_boxcox, index=ts.index).diff().dropna()

# bcdf\_quar\_dd = bcdf\_quar\_d.diff().dropna()

# bcdfds = [bcdf\_quar\_d, bcdf\_quar\_dd]

# 画出两种差分后的自相关和偏自相关图

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

for i, ax in enumerate(axes.flat):

    if i % 2 == 0:

        plot\_acf(bcdf\_quar\_d, ax=ax, lags=30)

    else:

        plot\_pacf(bcdf\_quar\_d, ax=ax, lags=30)

    ax.set\_title(['Box-cox后一阶差分'][i // 2] + ['ACF', 'PACF'][i % 2])

plt.tight\_layout()

plt.savefig('结果1-1/Box-cox后一阶差分的ACF和PACF.png')

plt.show()

import warnings

from statsmodels.tools.sm\_exceptions import ConvergenceWarning

# 忽略ConvergenceWarning警告

warnings.simplefilter('ignore', ConvergenceWarning)

p\_range = range(5)

q\_range = range(5)

bic\_values = pd.DataFrame(index=p\_range, columns=q\_range)

# 遍历所有的p, q组合

for p in p\_range:

    for q in q\_range:

        try:

            # 拟合ARIMA模型

            model = ARIMA(ts\_boxcox, order=(p, 1, q))

            model\_fit = model.fit()

            # 存储BIC值

            bic\_values.loc[p, q] = model\_fit.bic

        except:

            # 如果模型拟合失败，则存储一个大数

            bic\_values.loc[p, q] = np.inf

# 将BIC值转换为float类型，以便于绘图

bic\_values = bic\_values.astype(float)

# 绘制热图

plt.figure(figsize=(10, 8))

sns.heatmap(bic\_values, annot=True, fmt=".2f", cmap='coolwarm')

plt.title('d=1时不同p, q组合的BIC值')

plt.xlabel('q')

plt.ylabel('p')

plt.savefig('结果1-1/d=1时不同pq组合的BIC值.png')

plt.show()

# 1. 模型拟合

model = ARIMA(ts\_boxcox, order=(1, 1, 1))

model\_fit = model.fit()

# 2. 模型诊断

model\_fit.plot\_diagnostics(figsize=(10, 8))

plt.savefig('结果1-1/ARIMA模型诊断.png')

plt.show()

# 3. 预测

steps = 7

forecast = model\_fit.forecast(steps=steps)

# 将预测结果转换回原始尺度

forecast\_original\_scale = inv\_boxcox(forecast, lambda\_)

# 获取最后一个时间点

last\_date = ts.index[-1]

# 创建未来6个季度的时间点

future\_dates = [last\_date + DateOffset(months=x) for x in range(0, 3\*steps, 3)]

forecast\_matrix = forecast\_original\_scale.reshape(-1, 1)

dates\_matrix = np.array(future\_dates).reshape(-1, 1)

result\_matrix = np.hstack((dates\_matrix, forecast\_matrix))

result\_df = pd.DataFrame(result\_matrix, columns=['Date', 'Forecast'])

print(forecast\_original\_scale)

print(ts[-5:])

print(result\_df)

# 绘制原始数据

plt.figure(figsize=(12, 6))

plt.plot(ts/1e13, label='Original Data')

# 绘制预测结果

plt.plot(pd.to\_datetime(result\_df['Date']), result\_df['Forecast']/1e13, label='Forecast', color='red')

# 特定日期

dates\_to\_highlight = ['2024-12-30', '2025-12-30']

# 在原始数据和预测结果中查找这些日期的值

for date in dates\_to\_highlight:

    # 转换日期字符串为pandas的日期时间格式

    date\_pd = pd.to\_datetime(date)

    # 检查这个日期是否在预测结果中

    if date\_pd in pd.to\_datetime(result\_df['Date']).values:

        forecast\_value = result\_df.loc[pd.to\_datetime(result\_df['Date']) == date\_pd, 'Forecast'].values[0]

        plt.scatter(date\_pd, forecast\_value/1e13, color='red', s=50)

# 显示图表

plt.title('ARIMA模型预测结果')

plt.xlabel('时间 (单位：年)')

plt.ylabel('美国未偿还公共债务总额 (单位：十万亿美元)')

plt.savefig('结果1-1/ARIMA模型预测结果.png')

plt.show()

plt.figure(figsize=(10, 6))

plt.bar(ts.index[-7:-1], ts[-7:-1]/1e13, label='原始数据', width=40)

plt.bar(pd.to\_datetime(result\_df['Date']), result\_df['Forecast']/1e13, label='预测数据', width=40)

plt.xticks(rotation=45)

plt.xlabel('时间 (单位：季度)')

plt.ylabel('美国未偿还公共债务总额 (单位：十万亿美元)')

plt.ylim(3.0, 3.8)

plt.title('ARIMA模型预测结果')

plt.legend()

plt.savefig('结果1-1/ARIMA模型预测结果柱状图.png')

plt.show()

**1-2.ipynb**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

from statsmodels.tsa.api import VAR

from statsmodels.tsa.stattools import adfuller

from statsmodels.tools.eval\_measures import rmse, aic

from statsmodels.tsa.stattools import grangercausalitytests

import statsmodels.api as sm

from statsmodels.stats.stattools import durbin\_watson

from openpyxl import load\_workbook

from openpyxl.styles import Font

df = pd.read\_excel('数据2.xlsx')

# 定义转换函数

def quarter\_to\_date(quarter\_str):

    year, quarter = quarter\_str[:-4], quarter\_str[-3:]

    quarter\_dict = {"季度一": "03-31", "季度二": "06-30", "季度三": "09-30", "季度四": "12-31"}

    return pd.to\_datetime(f"{year}-{quarter\_dict[quarter]}")

# 将表头转换为日期格式

new\_index = [quarter\_to\_date(q) for q in df.columns]

df.columns = new\_index

df = df.T

indexs = ['GDP', '居民消费水平', '社会消费品零售', '固定资产投资', '出口', '进口', '财政支出']

df.columns = indexs

plt.plot(df.iloc[:, 0]/1e7)

plt.title(df.columns[0])

plt.xlabel('时间 (单位：季度)')

plt.ylabel('GDP（单位：十万亿美元）')

plt.savefig('结果1-2/GDP.png')

plt.show()

# 创建一个3\*2的子图，绘制df的后六列数据

fig, axes = plt.subplots(3, 2, figsize=(15, 10))

for i, ax in enumerate(axes.flatten()):

    data = df[df.columns[i+1]]

    ax.plot(data)

    ax.set\_title(df.columns[i+1])

plt.tight\_layout()

plt.savefig('结果1-2/六项.png')

plt.show()

# 初始化一个空的DataFrame来存储结果，使用列名作为索引和列

results\_matrix = pd.DataFrame(index=df.columns, columns=df.columns, dtype=float)

# 双重循环遍历所有列的组合

for col1 in df.columns:

    for col2 in df.columns:

        # 进行格兰杰因果测试，maxlag设置为检测的最大滞后期数，这里假设为2

        test\_result = grangercausalitytests(df[[col1, col2]], maxlag=2)

        # 提取p值，这里假设使用第一个滞后期的结果

        p\_value = test\_result[1][0]['ssr\_chi2test'][1]

        # 将p值存储在结果矩阵中的相应位置

        results\_matrix.loc[col1, col2] = p\_value

# 输出结果

print(results\_matrix)

results\_matrix.to\_excel('结果1-2/格兰杰因果检验.xlsx')

plt.bar(df.columns, results\_matrix.iloc[0])

plt.title('GDP与其他指标的格兰杰因果检验结果')

plt.ylabel('p值')

plt.xticks(rotation=45)

plt.show()

train = df[['社会消费品零售', '出口', '进口', '财政支出', 'GDP']]

# 假设train是已经定义好的DataFrame

columns = ['原始数据', '一阶差分', '二阶差分']

# 初始化DataFrame来保存p值

p\_values = pd.DataFrame(index=train.columns, columns=columns, dtype=float)

# 循环进行0次、1次、2次差分

for diff\_level in range(3):

    if diff\_level == 0:

        data\_diff = train

        column\_name = '原始数据'

    else:

        data\_diff = data\_diff.diff().dropna()

        column\_name = columns[diff\_level]

    # 对每一列进行ADF测试

    for col in data\_diff.columns:

        result = adfuller(data\_diff[col])

        p\_values.loc[col, column\_name] = result[1]

# 保存结果到Excel

excel\_path = '结果1-2/ADF检验.xlsx'

p\_values.to\_excel(excel\_path)

# 使用openpyxl加载工作簿

wb = load\_workbook(excel\_path)

ws = wb.active

# 遍历p值DataFrame，将p值小于0.05的单元格标红

for row in range(2, len(p\_values) + 2):  # Excel行列索引从1开始，且跳过标题行

    for col in range(2, len(p\_values.columns) + 2):  # 跳过索引列

        \_value = ws.cell(row=row, column=col).value

        if \_value < 0.05:

            ws.cell(row=row, column=col).font = Font(color="FF0000")

# 保存修改后的工作簿

wb.save(excel\_path)

train\_diff = train.diff().dropna()

train\_diff\_diff = train\_diff.diff().dropna()

fig, axes = plt.subplots(2, 2, figsize=(10, 6))

for i, ax in enumerate(axes.flatten()):

    data = train\_diff\_diff[train\_diff\_diff.columns[i]]

    ax.plot(data)

    ax.set\_title(train\_diff\_diff.columns[i])

plt.tight\_layout()

plt.show()

# 假设train\_diff\_diff是你的训练数据集

model = VAR(train\_diff\_diff)

# 初始化存储结果的字典

criteria = {'BIC': []}

lags = range(1, 15)

# 计算每个滞后阶数下的BIC值

for i in lags:

    result = model.fit(i)

    criteria['BIC'].append(result.bic)

# 绘制BIC图

plt.figure(figsize=(6, 4))

plt.plot(lags, criteria['BIC'], marker='o')

plt.title('BIC值随滞后阶数变化图')

plt.xlabel('滞后阶数')

# plt.ylabel('值')

plt.tight\_layout()

plt.savefig('结果1-2/BIC.png')

plt.show()

model\_fitted = model.fit(4)

# model\_fitted.summary()

# 假设model\_fitted是已经拟合好的VAR模型对象

residuals = model\_fitted.resid

# print(residuals.loc[:, 'GDP'].head())

residuals = residuals.loc[:, 'GDP']

# 残差的时间序列图

plt.figure(figsize=(10, 6))

plt.plot(residuals)

plt.title('Residuals Time Series')

plt.show()

# 残差的ACF图和PACF图

fig, axes = plt.subplots(1, 2, figsize=(15, 6))

sm.graphics.tsa.plot\_acf(residuals.values.squeeze(), lags=40, ax=axes[0], title='残差ACF图')

sm.graphics.tsa.plot\_pacf(residuals.values.squeeze(), lags=40, ax=axes[1], title='残差PACF图')

plt.savefig('结果1-2/残差ACFPACF图.png')

plt.show()

# Q-Q图

plt.figure(figsize=(10, 6))

sm.qqplot(residuals.values.ravel(), line='s')

plt.title('残差Q-Q图')

plt.savefig('结果1-2/残差QQ图.png')

plt.xlabel('理论分位数')

plt.ylabel('样本分位数')

plt.show()

out = durbin\_watson(model\_fitted.resid)

for col, val in zip(df.columns, out):

    print(col, ':', round(val, 2))

model = VAR(train\_diff\_diff)

model\_fitted = model.fit(4)

# 预测未来7个季度

forecasted\_values\_diff = model\_fitted.forecast(train\_diff\_diff.values[-model\_fitted.k\_ar:], steps=7)

# 首先，将二阶差分的预测值还原为一阶差分的尺度

last\_observation\_diff = train\_diff.iloc[-1].values

forecasted\_values\_diff\_cumsum = forecasted\_values\_diff.cumsum(axis=0)

forecasted\_values\_diff\_cumsum += last\_observation\_diff

# 然后，将一阶差分的预测值还原为原始尺度的值

last\_original\_observation = train.iloc[-1].values

forecasted\_values\_original = forecasted\_values\_diff\_cumsum.cumsum(axis=0)

forecasted\_values\_original += last\_original\_observation

# 步骤1: 将预测值转换为DataFrame

forecasted\_df = pd.DataFrame(forecasted\_values\_original, columns=train.columns)

# 步骤2: 创建新的时间索引

last\_date = train.index[-1]

new\_dates = pd.date\_range(start=last\_date, periods=8, freq='Q')[1:]  # 创建7个季度的新日期，从最后一个日期之后开始

# 将新的时间索引赋值给预测的DataFrame

forecasted\_df.index = new\_dates

# 步骤3: 合并原始数据和预测数据

train\_forecasted = pd.concat([train, forecasted\_df])

print(train\_forecasted.tail(10))  # 打印合并后的新数据集的最后10行，以检查追加的预测数据

# 确保 train\_forecasted.index 是 DatetimeIndex 类型

train\_forecasted.index = pd.to\_datetime(train\_forecasted.index)

plt.figure(figsize=(12, 6))

plt.plot(train\_forecasted['GDP'].iloc[:-7]/1e7, label='历史数据')

plt.plot(train\_forecasted['GDP'].iloc[-7:]/1e7, label='预测数据', color='red')

plt.title('VAR模型预测结果')

plt.xlabel('时间 (单位：年)')

plt.ylabel('美国未偿还公共债务总额 (单位：十万亿美元)')

# 特定日期

dates\_to\_highlight = ['2024-12-31', '2025-12-31']

# 在图上用红色圆点标记特定日期

for date in dates\_to\_highlight:

    date\_time = pd.to\_datetime(date)  # 将字符串日期转换为日期时间对象

    if date\_time in train\_forecasted.index:

        plt.scatter(date\_time, train\_forecasted.loc[date\_time, 'GDP']/1e7, color='red', s=50)  # s是点的大小

plt.savefig('结果1-2/VAR模型预测结果.png')

plt.show()

# 绘制原始GDP数据

plt.figure(figsize=(10, 6))

plt.bar(df.index[-7:], df['GDP'][-7:]/1e7, width=40, label='原始数据')

# 绘制预测GDP数据

plt.bar(forecasted\_df.index, forecasted\_df['GDP']/1e7, width=40, label='预测数据')

plt.legend()

plt.title('利用VAR模型对美国未来两年GDP的预测结果')

plt.xlabel('时间 (单位：季度)')

plt.ylabel('美国未偿还公共债务总额 (单位：十万亿美元)')

plt.ylim(2.4, 3.03)

plt.savefig('结果1-2/预测.png')

plt.show()

**2-1.ipynb**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

from sklearn.preprocessing import PolynomialFeatures

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

df = pd.read\_excel('数据/GDP、GDP增长率、美债.xlsx')

# 定义转换函数

def quarter\_to\_date(quarter\_str):

    year, quarter = quarter\_str.split('年')

    quarter\_dict = {"第一季度": "03-31", "第二季度": "06-30", "第三季度": "09-30", "第四季度": "12-31"}

    return pd.to\_datetime(f"{year}-{quarter\_dict[quarter]}")

columns = df.columns

new\_columns = [quarter\_to\_date(col) for col in columns]

df.columns = new\_columns

df = df.T

df.columns = ['GDP增长率', 'GDP', '美债']

print(df.head())

fig, axs = plt.subplots(1, 3, figsize=(15, 6))  # 创建3个子图

plots = ['GDP增长率', 'GDP', '美债']  # 3个变量

axs[0].plot(df.index, df['GDP增长率'])

axs[0].set\_ylabel('GDP增长率（百分比）')

axs[1].plot(df.index, df['GDP']/1e13)

axs[1].set\_ylabel('GDP(单位：十万亿美元)')

axs[2].plot(df.index, df['美债']/1e13)

axs[2].set\_ylabel('美债(单位：十万亿美元)')

for i, plot in enumerate(plots):

    axs[i].set\_title(plot)

    # axs[i].legend()

plt.tight\_layout()  # 调整每个子图之间的间距

plt.savefig('结果2-1/GDP、GDP增长率、美债时间序列图.png')

plt.show()

fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # 创建2个子图

plots = ['GDP增长率', 'GDP']  # 2个变量

axs[0].scatter(df['GDP增长率'], df['美债']/1e13, label=plot, s=10)

axs[1].scatter(df['GDP']/1e13, df['美债']/1e13, label=plot, s=10)

axs[0].set\_xlabel('GDP增长率（百分比）')

axs[1].set\_xlabel('GDP(单位：十万亿美元)')

for i, plot in enumerate(plots):

    axs[i].set\_title(f'{plot}与美债散点图')

    axs[i].set\_ylabel('美债(单位：十万亿美元)')

    # axs[i].legend(loc='upper left')

plt.tight\_layout()  # 调整每个子图之间的间距

plt.savefig('结果2-1/GDP、GDP增长率与美债散点图.png')

plt.show()

# 分割数据集

df\_1 = df[df.index <= '2008-09-30']

df\_2 = df[(df.index >= '2009-03-31') & (df.index <= '2019-12-31')]

df\_3 = df[df.index >= '2020-12-31']

dfs = [df\_1, df\_2, df\_3]

# 设置绘图大小

plt.figure(figsize=(15, 5))

for i, dfi in enumerate(dfs, 1):

    # 定义自变量和因变量

    X = dfi[['GDP', 'GDP增长率']]

    y = dfi['美债']

    # 添加常数项到自变量中

    X = sm.add\_constant(X)

    # 构建线性回归模型

    model = sm.OLS(y, X).fit()

    # 获取模型的预测值

    predictions = model.predict(X)

    # 计算残差

    residuals = y - predictions

    # 绘制残差图

    plt.subplot(1, 3, i)

    sns.residplot(x=predictions, y=residuals, lowess=True, line\_kws={'color': 'red', 'lw': 1})

    plt.ylim(-1e12, 1e12)

    plt.title(f'数据集{i} 残差图\nR-squared: {model.rsquared:.2f}')

    plt.xlabel('预测值')

    plt.ylabel('残差')

    plt.axhline(0, color='red', linewidth=1)

plt.tight\_layout()

plt.savefig('结果2-1/线性回归残差图.png')

plt.show()

# 梯度下降法更新系数

def gradient\_descent(X, y, theta, learning\_rate=0.001):

    iterations=10000 \* (100//y.shape[0])

    m = len(y)

    cost\_history = np.zeros(iterations)

    for it in range(iterations):

        prediction = np.dot(X, theta)

        theta = theta - (1/m) \* learning\_rate \* (X.T.dot((prediction - y)))

        cost\_history[it] = (1/(2\*m)) \* np.sum(np.square(prediction - y))

    return theta, cost\_history

# 在应用梯度下降之前对特征进行缩放

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# 修改初始化的三维数组大小为5x5x3

r\_squared\_values = np.zeros((4, 4, 3))  # 包括GDP和GDP增长率的1到5次项，3个数据集

for i, dfi in enumerate(dfs):

    X = dfi[['GDP', 'GDP增长率']]

    y = dfi['美债'].values

    # 应用特征缩放

    X\_scaled = scaler.fit\_transform(X)

    for degree\_gdp in range(1, 5):  # 包括GDP的1到5次项

        for degree\_growth in range(1, 5):  # 包括GDP增长率的1到5次项

            # 为GDP和GDP增长率添加指定次数的多项式特征

            poly\_gdp = PolynomialFeatures(degree=degree\_gdp, include\_bias=False)

            poly\_growth = PolynomialFeatures(degree=degree\_growth, include\_bias=False)

            X\_gdp\_poly = poly\_gdp.fit\_transform(X\_scaled[:, 0].reshape(-1, 1))

            X\_growth\_poly = poly\_growth.fit\_transform(X\_scaled[:, 1].reshape(-1, 1))

            # 合并多项式特征

            X\_poly = np.concatenate((X\_gdp\_poly, X\_growth\_poly), axis=1)

            # 添加常数项

            X\_poly = sm.add\_constant(X\_poly)

            # 初始化系数

            theta = np.random.randn(X\_poly.shape[1], 1)

            # 使用梯度下降法求解系数

            theta, \_ = gradient\_descent(X\_poly, y.reshape(-1,1), theta)

            # 使用求解的系数计算R方值

            prediction = X\_poly.dot(theta)

            ss\_res = np.sum(np.square(prediction - y.reshape(-1,1)))

            ss\_tot = np.sum(np.square(y - np.mean(y)))

            r\_squared = 1 - (ss\_res / ss\_tot)

            # 存储R方值

            r\_squared\_values[degree\_gdp-1, degree\_growth-1, i] = r\_squared

# 使用pandas的ExcelWriter保存多个sheet

with pd.ExcelWriter('数据/R方矩阵.xlsx') as writer:

    for i in range(3):

        df\_r\_squared = pd.DataFrame(r\_squared\_values[:, :, i],

                                    index=[f'GDP {degree}次项' for degree in range(1, 5)],

                                    columns=[f'GDP增长率 {degree}次项' for degree in range(1, 5)])

        print(f"数据集 {i+1} 的R方矩阵:")

        print(df\_r\_squared)

        print("\n")

        # 使用ExcelWriter对象保存DataFrame到不同的sheet中

        df\_r\_squared.to\_excel(writer, sheet\_name=f'数据集{i+1}')

r\_squared\_values\_np = np.array(r\_squared\_values)

np.save('数据/r\_squared\_values.npy', r\_squared\_values\_np)

fig, axs = plt.subplots(1, 3, figsize=(18, 6))

for i, dfi in enumerate(dfs):

    X = dfi[['GDP', 'GDP增长率']]

    y = dfi['美债'].values

    X\_scaled = scaler.fit\_transform(X) # 输入的是标准化后的值

    # 找到R方值最大的组合

    max\_r\_squared\_index = np.unravel\_index(np.argmax(r\_squared\_values[:, :, i]), r\_squared\_values[:, :, i].shape)

    degree\_gdp\_best, degree\_growth\_best = max\_r\_squared\_index

    # 重新构建模型

    poly\_gdp\_best = PolynomialFeatures(degree=degree\_gdp\_best+1, include\_bias=False)

    poly\_growth\_best = PolynomialFeatures(degree=degree\_growth\_best+1, include\_bias=False)

    X\_gdp\_poly\_best = poly\_gdp\_best.fit\_transform(X\_scaled[:, 0].reshape(-1, 1))

    X\_growth\_poly\_best = poly\_growth\_best.fit\_transform(X\_scaled[:, 1].reshape(-1, 1))

    X\_poly\_best = np.concatenate((X\_gdp\_poly\_best, X\_growth\_poly\_best), axis=1)

    X\_poly\_best = sm.add\_constant(X\_poly\_best)  # 添加常数项

    # 初始化系数

    theta\_best = np.random.randn(X\_poly\_best.shape[1], 1)

    # 使用梯度下降法求解系数

    theta\_best, \_ = gradient\_descent(X\_poly\_best, y.reshape(-1,1), theta\_best)

    # 使用求解的系数计算预测值

    predictions = X\_poly\_best.dot(theta\_best)

    # 计算残差

    residuals = y.reshape(-1,1) - predictions

    # 绘制残差图

    axs[i].scatter(predictions, residuals)

    axs[i].set\_title(f'数据集 {i+1} 残差图：GDP {degree\_gdp\_best + 1}次, GDP增长率 {degree\_growth\_best + 1}次,\nR-squared: {r\_squared\_values[degree\_gdp\_best, degree\_growth\_best, i]:.2f}')

    axs[i].set\_xlabel('美债预测值')

    axs[i].set\_ylabel('残差')

    axs[i].axhline(y=0, color='red', linewidth=1, linestyle='--')

    axs[i].set\_ylim(-1e12, 1e12)

    # 在原有代码基础上增加以下部分

    # 为了生成函数表达式，我们需要特征的名称

    feature\_names\_gdp = poly\_gdp\_best.get\_feature\_names\_out(['g'])

    feature\_names\_growth = poly\_growth\_best.get\_feature\_names\_out(['ggr'])

    feature\_names = np.concatenate((feature\_names\_gdp, feature\_names\_growth))

    # 构建函数表达式

    expression = f"美债预测值 = {theta\_best.flatten()[0]/1e12:.3f}"

    for coef, name in zip(theta\_best.flatten()[1:]/1e12, feature\_names):

        if coef >= 0:

            expression += f"+ {coef:.3f}\*{name} "

        else:

            expression += f"- {-coef:.3f}\*{name} "

    print(f"数据集 {i+1} 的函数表达式:\n{expression}\n")

plt.tight\_layout()

plt.savefig('结果2-1/最佳的多项式回归残差图.png')

plt.show()

# 创建一个新的图形

fig = plt.figure(figsize=(18, 6))

ax = fig.add\_subplot(111, projection='3d')  # 使用111表示在一个1x1网格的第一个位置创建子图

for i, dfi in enumerate(dfs):

    X = dfi[['GDP', 'GDP增长率']]

    y = dfi['美债'].values

    X\_scaled = scaler.fit\_transform(X)  # 输入的是标准化后的值

    # 找到R方值最大的组合

    max\_r\_squared\_index = np.unravel\_index(np.argmax(r\_squared\_values[:, :, i]), r\_squared\_values[:, :, i].shape)

    degree\_gdp\_best, degree\_growth\_best = max\_r\_squared\_index

    # 重新构建模型

    poly\_gdp\_best = PolynomialFeatures(degree=degree\_gdp\_best+1, include\_bias=False)

    poly\_growth\_best = PolynomialFeatures(degree=degree\_growth\_best+1, include\_bias=False)

    X\_gdp\_poly\_best = poly\_gdp\_best.fit\_transform(X\_scaled[:, 0].reshape(-1, 1))

    X\_growth\_poly\_best = poly\_growth\_best.fit\_transform(X\_scaled[:, 1].reshape(-1, 1))

    X\_poly\_best = np.concatenate((X\_gdp\_poly\_best, X\_growth\_poly\_best), axis=1)

    X\_poly\_best = sm.add\_constant(X\_poly\_best)  # 添加常数项

    # 初始化系数

    theta\_best = np.random.randn(X\_poly\_best.shape[1], 1)

    # 使用梯度下降法求解系数

    theta\_best, \_ = gradient\_descent(X\_poly\_best, y.reshape(-1,1), theta\_best)

    # 创建网格数据

    gdp\_range = np.linspace(X\_scaled[:, 0].min(), X\_scaled[:, 0].max(), 20)

    growth\_range = np.linspace(X\_scaled[:, 1].min(), X\_scaled[:, 1].max(), 20)

    gdp\_grid, growth\_grid = np.meshgrid(gdp\_range, growth\_range)

    # 计算网格上每一点的预测值

    X\_grid\_poly\_gdp = poly\_gdp\_best.transform(gdp\_grid.reshape(-1, 1))

    X\_grid\_poly\_growth = poly\_growth\_best.transform(growth\_grid.reshape(-1, 1))

    X\_grid\_poly = np.concatenate((X\_grid\_poly\_gdp, X\_grid\_poly\_growth), axis=1)

    X\_grid\_poly = sm.add\_constant(X\_grid\_poly)  # 添加常数项

    predictions\_grid = X\_grid\_poly.dot(theta\_best)

    # 将gdp\_grid和growth\_grid转换回原始尺度

    gdp\_growth\_grid\_scaled = np.vstack((gdp\_grid.flatten(), growth\_grid.flatten())).T

    gdp\_growth\_grid\_original = scaler.inverse\_transform(gdp\_growth\_grid\_scaled)

    gdp\_grid\_original, growth\_grid\_original = gdp\_growth\_grid\_original[:, 0].reshape(gdp\_grid.shape), gdp\_growth\_grid\_original[:, 1].reshape(growth\_grid.shape)

    # 绘制预测曲面

    ax.plot\_surface(gdp\_grid\_original, growth\_grid\_original, predictions\_grid.reshape(gdp\_grid.shape), alpha=0.3)

    # 绘制原始数据的三维散点图

    X\_original = scaler.inverse\_transform(X\_scaled)

    ax.scatter(X\_original[:, 0], X\_original[:, 1], y, color='blue', s=10)

ax.set\_xlabel('GDP（单位：十万亿美元）')

ax.set\_ylabel('GDP增长率（百分比）')

ax.set\_zlabel('美债（单位：十万亿美元）')

ax.set\_title('原始数据三维散点图和 BGD 预测曲面')

plt.savefig('结果2-1/原始数据三维散点图和 BGD 预测曲面.png')

plt.show()

**2-1 不同国家对比.py**

import pandas as pd

import pandas as pd

import matplotlib.pyplot as plt

import os

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# 获取当前文件的完整路径

current\_file\_path = \_\_file\_\_

# 获取当前文件所在目录的路径

current\_dir = os.path.dirname(current\_file\_path)

# 改变当前工作目录

os.chdir(current\_dir)

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

# 读取当前目录下的222.xlsx文件中的"英国"和"日本"这两个Sheet的数据

data\_uk = pd.read\_excel('数据/222.xlsx', sheet\_name='英国1993').T

data\_uk.columns = ['GDP', 'GDP增长率', '债务']

data\_japan = pd.read\_excel('数据/222.xlsx', sheet\_name='日本1993').T

data\_japan.columns = ['GDP', 'GDP增长率', '债务']

# print(data\_uk.head())

df = pd.read\_excel('数据/GDP、GDP增长率、美债.xlsx')

# 定义转换函数

def quarter\_to\_date(quarter\_str):

    year, quarter = quarter\_str.split('年')

    quarter\_dict = {"第一季度": "03-31", "第二季度": "06-30", "第三季度": "09-30", "第四季度": "12-31"}

    return pd.to\_datetime(f"{year}-{quarter\_dict[quarter]}")

columns = df.columns

new\_columns = [quarter\_to\_date(col) for col in columns]

df.columns = new\_columns

df = df.T

df.columns = ['GDP增长率', 'GDP', '美债']

# print(df.head())

# 将df重采样到年

data\_us = df.resample('YE').mean().dropna()

# print(df\_yearly.head())

countries\_data = {

    '英国': data\_uk,

    '日本': data\_japan,

    '美国': data\_us

}

fig = plt.figure(figsize=(18, 6))

for i, (country, data) in enumerate(countries\_data.items(), start=1):

    ax = fig.add\_subplot(1, 3, i, projection='3d')

    ax.scatter(data['GDP'], data['GDP增长率'], data['美债'] if country == '美国' else data['债务'])

    ax.set\_title(country)

    ax.set\_xlabel('GDP')

    ax.set\_ylabel('GDP增长率')

    ax.set\_zlabel('债务')

plt.tight\_layout()

plt.show()

**2-1 预测曲面绘制.py**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

import os

from sklearn.preprocessing import PolynomialFeatures

# 获取当前文件的完整路径

current\_file\_path = \_\_file\_\_

# 获取当前文件所在目录的路径

current\_dir = os.path.dirname(current\_file\_path)

# 改变当前工作目录

os.chdir(current\_dir)

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

df = pd.read\_excel('GDP、GDP增长率、美债.xlsx')

# 定义转换函数

def quarter\_to\_date(quarter\_str):

    year, quarter = quarter\_str.split('年')

    quarter\_dict = {"第一季度": "03-31", "第二季度": "06-30", "第三季度": "09-30", "第四季度": "12-31"}

    return pd.to\_datetime(f"{year}-{quarter\_dict[quarter]}")

columns = df.columns

new\_columns = [quarter\_to\_date(col) for col in columns]

df.columns = new\_columns

df = df.T

df.columns = ['GDP增长率', 'GDP', '美债']

df\_1 = df[df.index <= '2008-09-30']

df\_2 = df[(df.index >= '2009-03-31') & (df.index <= '2019-12-31')]

df\_3 = df[df.index >= '2020-12-31']

dfs = [df\_1, df\_2, df\_3]

# 梯度下降法更新系数

def gradient\_descent(X, y, theta, learning\_rate=0.001):

    iterations=10000 \* (100//y.shape[0])

    m = len(y)

    cost\_history = np.zeros(iterations)

    for it in range(iterations):

        prediction = np.dot(X, theta)

        theta = theta - (1/m) \* learning\_rate \* (X.T.dot((prediction - y)))

        cost\_history[it] = (1/(2\*m)) \* np.sum(np.square(prediction - y))

    return theta, cost\_history

# 在应用梯度下降之前对特征进行缩放

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

r\_squared\_values = np.load('r\_squared\_values.npy')

for i, dfi in enumerate(dfs):

    X = dfi[['GDP', 'GDP增长率']]

    y = dfi['美债'].values

    X\_scaled = scaler.fit\_transform(X) # 输入的是标准化后的值

    # 找到R方值最大的组合

    max\_r\_squared\_index = np.unravel\_index(np.argmax(r\_squared\_values[:, :, i]), r\_squared\_values[:, :, i].shape)

    degree\_gdp\_best, degree\_growth\_best = max\_r\_squared\_index

    # 重新构建模型

    poly\_gdp\_best = PolynomialFeatures(degree=degree\_gdp\_best+1, include\_bias=False)

    poly\_growth\_best = PolynomialFeatures(degree=degree\_growth\_best+1, include\_bias=False)

    X\_gdp\_poly\_best = poly\_gdp\_best.fit\_transform(X\_scaled[:, 0].reshape(-1, 1))

    X\_growth\_poly\_best = poly\_growth\_best.fit\_transform(X\_scaled[:, 1].reshape(-1, 1))

    X\_poly\_best = np.concatenate((X\_gdp\_poly\_best, X\_growth\_poly\_best), axis=1)

    X\_poly\_best = sm.add\_constant(X\_poly\_best)  # 添加常数项

    # 初始化系数

    theta\_best = np.random.randn(X\_poly\_best.shape[1], 1)

    # 使用梯度下降法求解系数

    theta\_best, \_ = gradient\_descent(X\_poly\_best, y.reshape(-1,1), theta\_best)

    # 使用求解的系数计算预测值

    predictions = X\_poly\_best.dot(theta\_best)

    # 计算残差

    residuals = y.reshape(-1,1) - predictions

    # 在原有代码基础上增加以下部分

    # 为了生成函数表达式，我们需要特征的名称

    feature\_names\_gdp = poly\_gdp\_best.get\_feature\_names\_out(['g'])

    feature\_names\_growth = poly\_growth\_best.get\_feature\_names\_out(['ggr'])

    feature\_names = np.concatenate((feature\_names\_gdp, feature\_names\_growth))

    # 构建函数表达式

    expression = f"美债预测值 = {theta\_best.flatten()[0]:.3f}"

    for coef, name in zip(theta\_best.flatten()[1:], feature\_names):

        if coef >= 0:

            expression += f"+ {coef:.3f}\*{name} "

        else:

            expression += f"- {-coef:.3f}\*{name} "

    print(f"数据集 {i+1} 的函数表达式:\n{expression}\n")

# 创建一个新的图形

fig = plt.figure(figsize=(10, 6))

ax = fig.add\_subplot(111, projection='3d')  # 使用111表示在一个1x1网格的第一个位置创建子图

for i, dfi in enumerate(dfs):

    X = dfi[['GDP', 'GDP增长率']]

    y = dfi['美债'].values

    X\_scaled = scaler.fit\_transform(X)  # 输入的是标准化后的值

    # 找到R方值最大的组合

    max\_r\_squared\_index = np.unravel\_index(np.argmax(r\_squared\_values[:, :, i]), r\_squared\_values[:, :, i].shape)

    degree\_gdp\_best, degree\_growth\_best = max\_r\_squared\_index

    # 重新构建模型

    poly\_gdp\_best = PolynomialFeatures(degree=degree\_gdp\_best+1, include\_bias=False)

    poly\_growth\_best = PolynomialFeatures(degree=degree\_growth\_best+1, include\_bias=False)

    X\_gdp\_poly\_best = poly\_gdp\_best.fit\_transform(X\_scaled[:, 0].reshape(-1, 1))

    X\_growth\_poly\_best = poly\_growth\_best.fit\_transform(X\_scaled[:, 1].reshape(-1, 1))

    X\_poly\_best = np.concatenate((X\_gdp\_poly\_best, X\_growth\_poly\_best), axis=1)

    X\_poly\_best = sm.add\_constant(X\_poly\_best)  # 添加常数项

    # 初始化系数

    theta\_best = np.random.randn(X\_poly\_best.shape[1], 1)

    # 使用梯度下降法求解系数

    theta\_best, \_ = gradient\_descent(X\_poly\_best, y.reshape(-1,1), theta\_best)

    # 创建网格数据

    gdp\_range = np.linspace(X\_scaled[:, 0].min(), X\_scaled[:, 0].max(), 20)

    growth\_range = np.linspace(X\_scaled[:, 1].min(), X\_scaled[:, 1].max(), 20)

    gdp\_grid, growth\_grid = np.meshgrid(gdp\_range, growth\_range)

    # 计算网格上每一点的预测值

    X\_grid\_poly\_gdp = poly\_gdp\_best.transform(gdp\_grid.reshape(-1, 1))

    X\_grid\_poly\_growth = poly\_growth\_best.transform(growth\_grid.reshape(-1, 1))

    X\_grid\_poly = np.concatenate((X\_grid\_poly\_gdp, X\_grid\_poly\_growth), axis=1)

    X\_grid\_poly = sm.add\_constant(X\_grid\_poly)  # 添加常数项

    predictions\_grid = X\_grid\_poly.dot(theta\_best)

    # 将gdp\_grid和growth\_grid转换回原始尺度

    gdp\_growth\_grid\_scaled = np.vstack((gdp\_grid.flatten(), growth\_grid.flatten())).T

    gdp\_growth\_grid\_original = scaler.inverse\_transform(gdp\_growth\_grid\_scaled)

    gdp\_grid\_original, growth\_grid\_original = gdp\_growth\_grid\_original[:, 0].reshape(gdp\_grid.shape), gdp\_growth\_grid\_original[:, 1].reshape(growth\_grid.shape)

    # 绘制预测曲面

    ax.plot\_surface(gdp\_grid\_original, growth\_grid\_original, predictions\_grid.reshape(gdp\_grid.shape), alpha=0.3)

    # 绘制原始数据的三维散点图

    X\_original = scaler.inverse\_transform(X\_scaled)

    ax.scatter(X\_original[:, 0], X\_original[:, 1], y, color='blue')

ax.set\_xlabel('GDP（单位：十万亿美元）')

ax.set\_ylabel('GDP增长率（百分比）')

ax.set\_zlabel('美债（单位：十万亿美元）')

ax.set\_title('原始数据三维散点图和 BGD 预测曲面')

plt.show()