**ARIMA模型构建**

# 重采样

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

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

# 差分

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

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

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

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

# Box-Cox变换

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

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

# 模型拟合

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

model\_fit = model.fit()

# 预测

steps = 7

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

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

last\_date = ts.index[-1]

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)

**VAR模型构建**

# 格兰杰因果检验

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

for col1 in df.columns:

    for col2 in df.columns:

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

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

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

print(results\_matrix)

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

# ADF检验

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

columns = ['原始数据', '一阶差分', '二阶差分']

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

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]

    for col in data\_diff.columns:

        result = adfuller(data\_diff[col])

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

excel\_path = '结果1-2/ADF检验.xlsx'

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

# 阶数确定

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

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

model = VAR(train\_diff\_diff)

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.tight\_layout()

plt.savefig('结果1-2/BIC.png')

plt.show()

# 预测

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

# 执行预测

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

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

**多项式回归模型构建**

# 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]

# 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

scaler = StandardScaler()

r\_squared\_values = np.zeros((4, 4, 3))

# 求R方矩阵

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):

        for degree\_growth in range(1, 5):

            # 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

# 求最佳函数表达式

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)

    # 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")