目录

统计套利及策略实现

小组成员：赵呈亮、许函嘉、盛靖斐、闫文超

1 定义及概念辨析

1.1 平稳性

如果数据产生过程的参数（例如均值和方差）不随着时间变化，那么数据平稳。在时间序列中，若其均值和方差是常数，与时间无关，那么此时间序列具有平稳性。

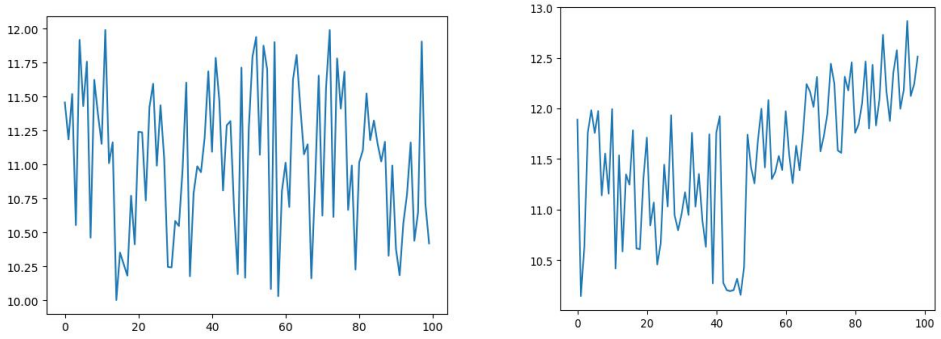

图 1 平稳序列与非平稳序列

例如在图 1 中，左侧的序列始终围绕着一个长期均值在波动，是一个平稳的序列；右侧的序列长期均值是变动的，是一个非平稳序列。

1.2 相关性

指两个序列 X 与 Y 之间的线性依存关系，常用相关系数 $\mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma(X)\sigma(Y)}$ 来检测。X、Y 相关系数 corr(X,Y)=0，未必独立（独立定义 P(XY)=P(X)P(Y)），因为 X、Y 可能服从非线性的相关关系；而独立一定不相关。corr(X,Y)=0，是指 X 的线性组合无法解释 Y。许多统计检验要求被检验数据是平稳的，因为非平稳数据可能会导致不好的结果。

1.3 协整关系

如果所考虑的时间序列具有相同的单整阶数，且某种线性组合（协整向量）使得组合时间序列的单整阶数降低，则称这些时间序列之间存在显著的协整关系。在现实中，绝大多数的股票都是非平稳的，如果两组序列是非平稳的，但它们的线性组合可以得到一个平稳序列，那么我们就说这两组时间序列数据具有协整的

性质，我们同样可以把平稳性的统计性质用到这个组合的序列上来。

1.4 概念辨析

1.4.1 协整不一定平稳

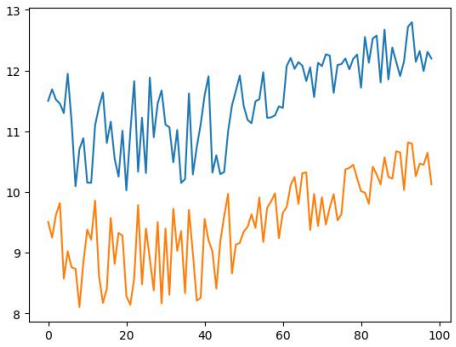

图 2 两个时间序列

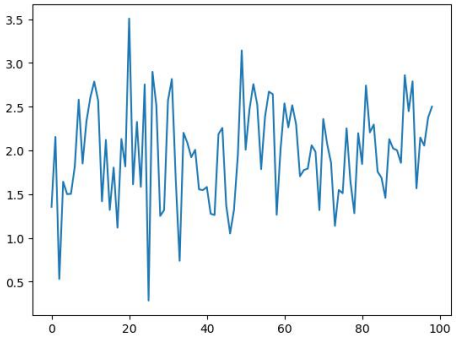

图 3 两时间序列之差

如上图，图 2 中两个时间序列为非平稳序列，因为其均值随着时间变化而变化，但图 3 中两时间序列之差为平稳序列，故两序列是协整的，即协整不一定平稳。

1.4.2 相关不一定协整

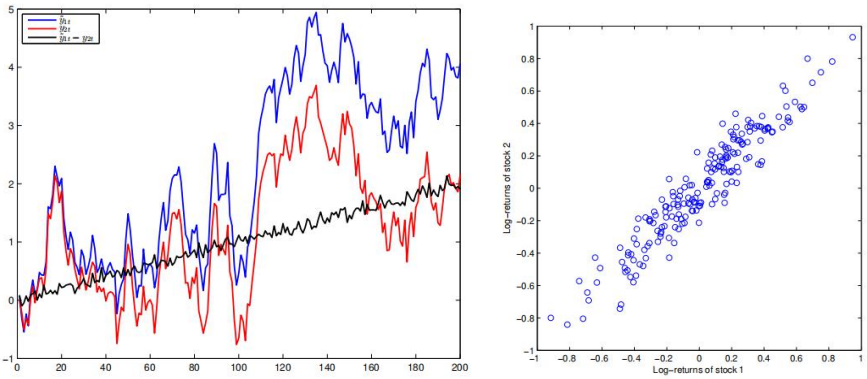

图 4 两序列及对数散点图

图 4 中 X、Y 两序列图像趋势及形状相似，且根据对数散点图可知两序列具有高度相关性，但其差的均值随着时间变化而变化，是非平稳序列，故两序列不是协整的，即相关不一定协整。

1.4.3 协整不一定相关

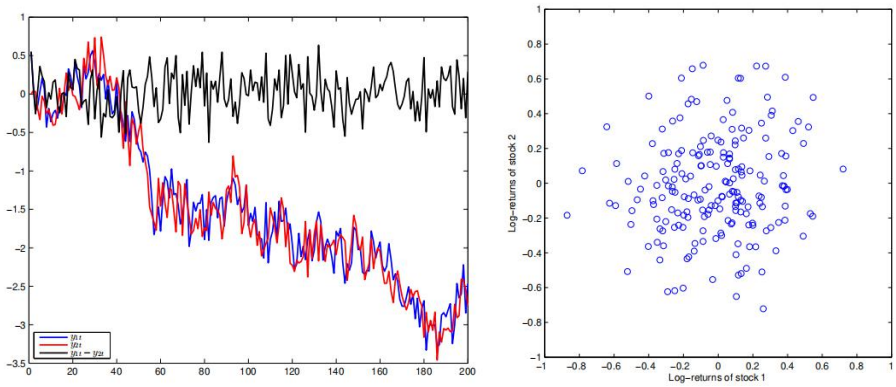

图 5 两序列及对数散点图

由图 5 中对数散点图可知，两序列不具有相关性，但两序列差是平稳序列，故是协整的，即协整不一定相关。

2 统计套利

2.1 概念

如果两个时间序列是协整的，那么从长期来看，它们仍然接近于彼此接近。换句话说，价差$z_t = y_{1t} - \gamma y_{2t}$是均值回归的。差价的这种均值回复特性可以被用于交易，它通常被称为通常被称为 "配对交易 "或 "统计套利"。成对交易背后的想法是卖空相对被高估的股票，买入相对被低估的股票。当它们的价值相对公平时再平仓。

2.2 步骤

我们以两只股票之间的统计套利作为例子来介绍统计套利的主要步骤。在实践中，配对交易包含三个主要步骤。

1、交易对选取：确定有可能被协整的股票对。

2、协整测试：测试所确定的股票对是否确实是协整的。

3、交易策略设计：研究价差动态，设计适当的交易规则。

3 策略实现

3.1 交易对选取

3.1.1 获取股票池数据

首先，考虑到服务器性能与运行时间，我们选取了 400 只股票作为股票池，并获取了其 2020 年全部数据。

```
import pandas as pd
import quantmind as qm
import numpy as np
import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint, adfuller
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(style="whitegrid")

#数据选取2020年一整年的数据
trading_date=qm.get_trade_date("20200101","20201231")
stock_list=qm.get_stock_list()['code'][:200]
# 选取400只股票作为股票池
total_data=pd.DataFrame([])
for stock_code in stock_list:
    total_data[stock_code]=qm.get_bar(stock_code, start_date='20200101', end_date='20201231', freq='60m')['close']
```

最终得到的数据形式如下：

表 1 股票池数据

	000001.SZ	000002.SZ	000004.SZ	000005.SZ	...	000620.SZ	000628.SZ	000629.SZ	000630.SZ
0	16.69	33.38	22.70	3.13	...	7.12	9.97	2.96	2.37
1	16.92	33.00	22.64	3.14	...	7.14	9.90	2.95	2.36
2	16.88	32.82	22.51	3.14	...	7.11	9.92	2.94	2.35
3	16.89	32.92	22.41	3.13	...	7.09	9.97	2.94	2.35
...	...	...	...	...	...	...	...	...	...
1211	19.35	28.48	20.68	2.50	...	4.89	8.32	2.18	2.58
1212	19.16	28.43	20.61	2.51	...	4.86	8.28	2.17	2.58
1213	19.11	28.53	20.66	2.51	...	4.87	8.51	2.16	2.55
1214	19.34	28.70	20.70	2.53	...	4.87	8.45	2.17	2.57

3.1.2 平稳性检验

接下来，在寻找交易对之前，要验证每一只股票数据的平稳性，采用 ADF 检验。

```
# 由于实际数据波动性较大，这里P指的阈值放宽一些
def stationarity_test(data, cutoff=0.1):
    # H_0 in adfuller is unit root exists (non-stationary)
    # We must observe significant p-value to convince ourselves that the series is stationary
    pvalue = adfuller(data)[1]
    if pvalue < cutoff:
```

```
            print('p-value = ' + str(pvalue) + ' The series ' + data.name +' is likely stationary.')
            return True
        else:
            print('p-value = ' + str(pvalue) + ' The series ' + data.name +' is likely non-stationary.')
            return False

stationarity_data=pd.DataFrame([])
for stock_code in stock_list:
    data=total_data[stock_code]
    if stationarity_test(data):
        stationarity_data[stock_code]=data
```

最终得到了 51 只具有稳定性的股票数据：

表 2 平稳性股票数据

	000002.SZ	000005.SZ	000007.SZ	000025.SZ	...	000622.SZ	000623.SZ	000628.SZ	000629.SZ
0	33.38	3.13	9.53	21.43	...	2.47	4.82	16.79	9.97
1	33.00	3.14	9.55	21.30	...	2.46	4.84	17.06	9.90
2	32.82	3.14	9.54	21.30	...	2.46	4.86	17.09	9.92
3	32.92	3.13	9.55	21.37	...	2.46	4.87	17.07	9.97
...	...	...	...	...	...	...	...	...	...
1211	28.48	2.50	6.97	16.92	...	2.42	3.95	16.44	8.32
1212	28.43	2.51	6.97	16.89	...	2.40	3.91	16.45	8.28
1213	28.53	2.51	6.97	16.91	...	2.40	3.92	16.45	8.51
1214	28.70	2.53	6.97	16.90	...	2.40	3.93	16.47	8.45

3.1.3 协整性检验

下一步进行协整性测试，找出具有协整性的股票。

```
def find_cointegrated_pairs(data):
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.keys()
    pairs = []
    for i in range(n):
        for j in range(i+1, n):
```

```
                S1 = data[keys[i]]
                S2 = data[keys[j]]
                result = coint(S1, S2)
                score = result[0]
                pvalue = result[1]
                score_matrix[i, j] = score
                pvalue_matrix[i, j] = pvalue
                if pvalue < 0.05:
                    pairs.append((keys[i], keys[j]))
        return score_matrix, pvalue_matrix, pairs

scores, pvalues, pairs = find_cointegrated_pairs(stationarity_data)
tickers=stationarity_data.keys()
import seaborn
fig, ax = plt.subplots(figsize=(51,51))
seaborn.heatmap(pvalues, xticklabels=tickers, yticklabels=tickers, cmap='RdYlGn_r' ,mask = (pvalues >= 0.05))
plt.show()
```

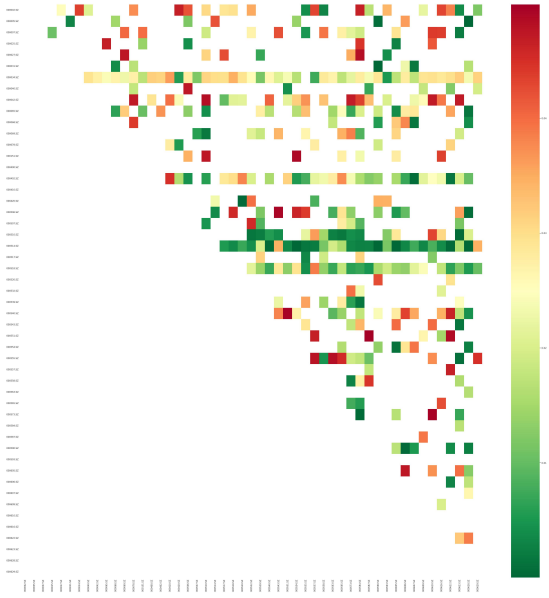

图 6 股票间 p 值热图

我们发现有 401 对股票数据存在协整关系。

3.1.4 相关性检验

```
corr_dict={}
for pair in pairs:
    stock1_data=stationarity_data[pair[0]]
    stock2_data=stationarity_data[pair[1]]
    corr_dict[pair]=stock1_data.corr(stock2_data)
```

因此我们得到了所有股票间两两相关系数。

3.1.5 交易对确定

我们策略需要的交易对要同时满足平稳、协整、相关三个条件，因此在平稳、协整的股票中选出相关性最大的一组，就是交易对。

```
print(max(corr_dict.values()))
max(corr_dict,key=lambda x:corr_dict[x])
```

最终得到的股票是'000031.SZ'，'000606.SZ'。接下来我们画图来直观看一下选取的交易对。

```
stock1=stationarity_data['000031.SZ']
stock2=stationarity_data['000606.SZ']
plt.plot(stock1)
plt.plot(stock2)
plt.show()

stock1 = sm.add_constant(stock1)
results = sm.OLS(stock2, stock1).fit()
stock1=stock1['000031.SZ']
b = results.params['000031.SZ']

spread = stock2 - b * stock1
spread.plot()
plt.axhline(spread.mean(), color='black')
plt.legend(['Spread'])
plt.show()
```

运行结果为：

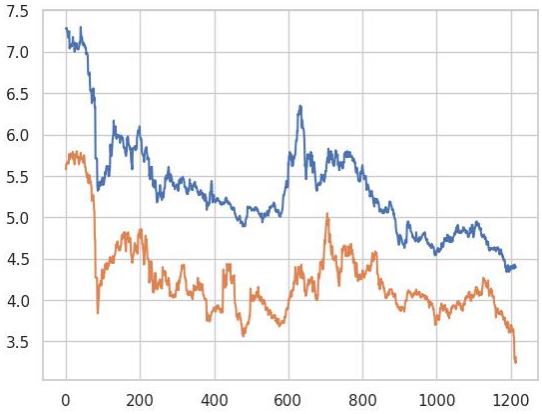

图 7 股票走势图

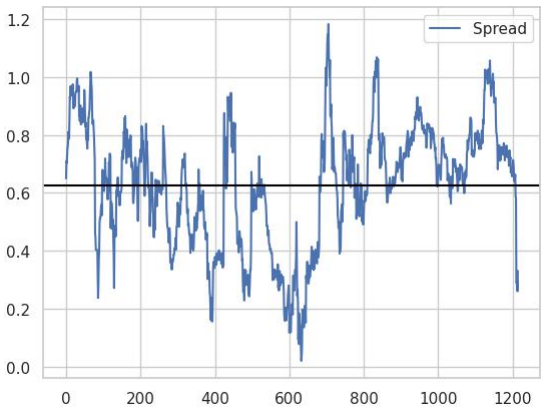

图 8 股票价差图

可见，两只股票走势十分接近，价差的均值也十分稳定，适合作为统计套利的交易对。

3.2 交易阈值确定

确定交易对后，人们仍然需要决定分别打开和平仓的进入和退出阈值。我们重点研究一下进入门槛：当价差偏离其长期均值s_0时开仓，当它恢复到其均值时平仓。因此，现在的关键问题是如何设计s_0的值，使得总利润为最大化。

假设价差遵循标准正态分布。点差偏离均值s_0或更多的概率为 $1-\Phi(s_0)$其中 $\Phi(.)$是标准正态分布。对于 T 天的路径，可交易事件数为 $T(1-\Phi(s_0))$。对于每笔交易，利润为s_0，那么总利润为$s_0T(1-\Phi(s_0))$ 。那么最佳阈值是$s_0 = argmax_{s_0}\{T(1-\Phi(s_0)\}$。我们无法知道真实的分布，但可以估计分布参数，然后根据估计的分布计算总利润。

```
# 用数学的方法确定阈
from scipy.stats import norm
from scipy.signal import find_peaks
def profit(threshold,T=len(trading_date)):
    return T*threshold*(1-norm.cdf(threshold))

from scipy import optimize
maximum = optimize.fminbound(profit, -0.5, 1.5)

# 网格搜索 确定最大值点 买入价差
X=np.linspace(-0.5,1.5,1000)
Y=list(map(profit,X))

buy_threhold1=X[np.argmax(Y)]
buy_threhold2=2*spread_mean-buy_threhold1
sell_threhold=(spread_mean-0.3*(spread_mean-buy_threhold2)
  ,spread_mean+0.3*(buy_threhold1-spread_mean))
```

根据结果我们制定的策略如下：

卖出阈值区间为 (0.5896400445211823, 0.6642298322427526)

价差计算公式为 price['000606.SZ']-price['000031.SZ']*0.6771655901542087

策略逻辑为：

当价差 > 0.7512512512512513:

买入 '000031.SZ'

当价差 < 0.5026186255126837:

买入 '000606.SZ'

当价差在卖出阈值间:

卖出 '000031.SZ','000606.SZ'

3.3 交易策略执行

最后，我们把上述策略在数以万贝平台上运行，选取了 2015-01-01 至 2015-03-29 三个月的数据进行回测，结果如下：

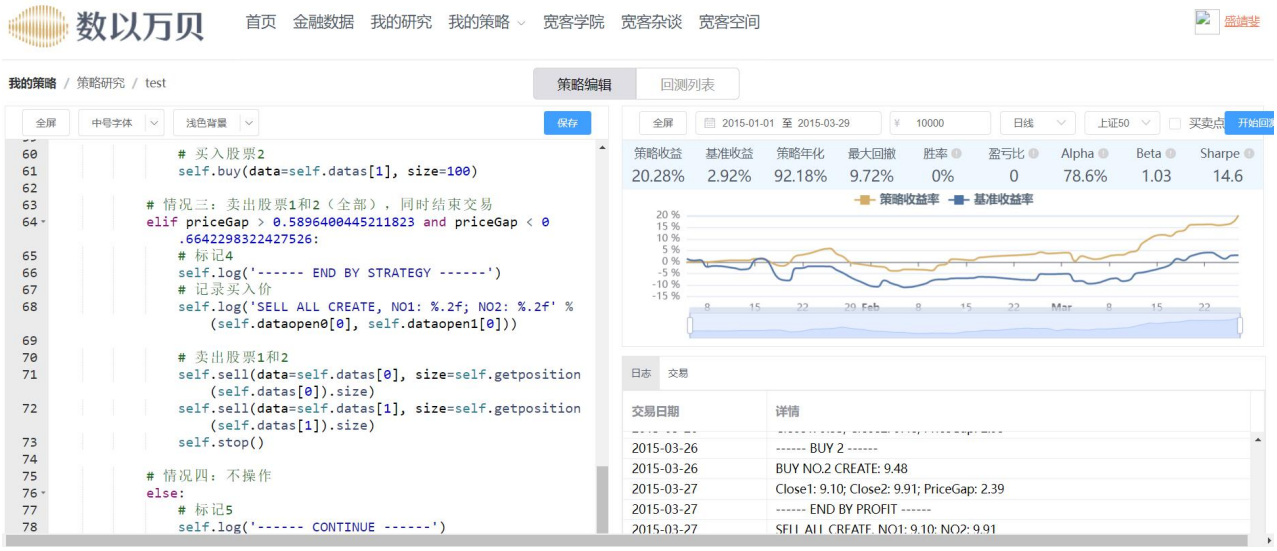

图 9 策略回测图

可见，根据此策略，三个月间收益率达到 20.28%，年化收益率为 92.18%，效果较好，也证明了此策略的成功。

附录

回测代码：

```
import backtrader as bt
import pandas as pd
import time

set_target(['000606.SZ', '000031.SZ'])

# 初始资金
g_init_cash = 10000

class TestStrategy(bt.Strategy):
    def __init__(self):

        # 获取每一只股票的开盘价（open），datas[0]代表000606.SZ，datas[1]代表000031.SZ
        self.dataopen0 = self.datas[0].open
        self.dataopen1 = self.datas[1].open

    def log(self, arg):
        # 打印日志函数，要有这个其他的self.log()才能正常显示
        print('{} {}'.format(self.datetime.date(), arg))

    def next(self):
        # 计算两只股票的差价
        priceGap = self.dataopen0[0] - self.dataopen1[0] * 0.6771655901542087
        # 记录当前价格和差价
        self.log('Close1: %.2f; Close2: %.2f; PriceGap: %.2f' % (self.dataopen0[0], self.dataopen1[0], priceGap))
        # 当前资金
        totalFund = self.broker.getvalue()

        # 判断止盈止亏条件（到达20%收益 / 10%亏损时全部卖出），若达到则结束交易
        if totalFund >= 1.2 * g_init_cash or totalFund <= 0.9 * g_init_cash:
            # 标记1
            self.log('------ END BY PROFIT ------')
            # 记录买入价
            self.log('SELL ALL CREATE, NO1: %.2f; NO2: %.2f' % (self.dataopen0[0], self.dataopen1[0]))

```

```
36.                 # 卖出股票1 和2
37.                 self.sell(data=self.datas[0], size=self.getpo
   sition(self.datas[0]).size)
38.                 self.sell(data=self.datas[1], size=self.getpo
   sition(self.datas[1]).size)
39.                 self.stop()
40.
41.             # 未达到盈亏条件，则进入统计套利策略
42.             else:
43.                 # 情况一：买入股票1
44.                 if priceGap < 0.5026186255126837:
45.                     # 标记2
46.                     self.log('------ BUY 1 ------')
47.                     # 记录买入价
48.                     self.log('BUY NO.1 CREATE: %.2f' % self.d
   ataopen0[0])
49.
50.                     # 买入股票1
51.                     self.buy(data=self.datas[0], size=100)
52.
53.                 # 情况二：买入股票2
54.                 elif priceGap > 0.7512512512512513:
55.                     # 标记3
56.                     self.log('------ BUY 2 ------')
57.                     # 记录买入价
58.                     self.log('BUY NO.2 CREATE: %.2f' % self.d
   ataopen1[0])
59.
60.                     # 买入股票2
61.                     self.buy(data=self.datas[1], size=100)
62.
63.                 # 情况三：卖出股票1 和2（全部），同时结束交易
64.                 elif priceGap > 0.5896400445211823 and priceG
   ap < 0.6642298322427526:
65.                     # 标记4
66.                     self.log('------ END BY STRATEGY ------')
67.                     # 记录买入价
68.                     self.log('SELL ALL CREATE, NO1: %.2f; NO2:
    %.2f' % (self.dataopen0[0], self.dataopen1[0]))
69.
70.                     # 卖出股票1 和2
71.                     self.sell(data=self.datas[0], size=self.g
   etposition(self.datas[0]).size)
```

```
                    self.sell(data=self.datas[1], size=self.getposition(self.datas[1]).size)
                    self.stop()

                # 情况四：不操作
                else:
                    # 标记5
                    self.log('------ CONTINUE ------')
                    return

    def stop(self):
        self.log('------ STOP ------')
        self.log('最终总资产:%.2f 元, 盈利:%.2f 元' % (
            self.broker.getvalue(), self.broker.getvalue() - g_init_cash))
```