1. 摘要

本文档参考东方证券《衍生品系列研究之五：商品期货套利策略实证》，对其中跨商品套利策略进行了复现。该策略通过产业链上原料和产品的固定配比实现套利，并采用钢厂利润、甲醇制PP利润以及炼焦利润进行验证。笔者对原文中开仓和平仓的条件进行改进，对部分参数进行调整。本文第二部分介绍跨商品套利策略的通用思想和逻辑，第三部分具体到产业链中介绍套利的操作流程，第四部分评估三种情景下的套利表现。

2. 策略主要思想

国内商品期货套利类策略主要包括跨商品套利、跨期套利、跨市场套利、期现套利等。本策略采取跨商品套利，通过选取部分稳定的商品投入产出配比，借助产业链上下游利润进行套利。这里采用三种套利模式，分别为钢厂利润套利、甲醇制PP利润套利和炼焦套利。

策略的核心思想是在生产过程中，原料和产品的用量会形成一个固定的配比，按这个配比对原料和产品的价格进行加权，计算出的价差“产品-原料”可以表示下游企业的利润。企业利润应维持在一定范围，当利润过高时，产品供给增加，从而原料价格上涨、利润回落；当利润过低时，产品供给减少，从而产品价格上涨、利润回升。跟随企业利润的变化，通过对价差“产品-原料”进行套利可以获得稳定的收益。

下面以钢铁厂利润套利为例，阐释策略思想。

标记	含义
s_t	t时刻价差
$\bar{s}_t$	t时刻过去21天的价差均值
σ_t	t时刻过去21天的价差标准差
RB	螺纹钢期货
I	铁矿石期货
J	焦炭期货
V_t	t时刻的总资金
α_t	t时刻的仓位

具体的策略设置如下：

1. 开仓条件
    1. 做空RB，做多I、J：
    $s_t \in [\bar{s}_t + 0.8\sigma_t, \bar{s}_t + \sigma_t]$，且$s_t < s_{t-1}$
    2. 做多RB，做空I、J：
    $s_t \in [\bar{s}_t - \sigma_t, \bar{s}_t - 0.8\sigma_t]$，且$s_t > s_{t-1}$
    3. 当$s_t > \bar{s}_t + \sigma_t$或$s_t < \bar{s}_t - \sigma_t$时，认定期货价格出现异动，不适合开仓
2. 平仓条件

$(s_t - \bar{s}_t)(s_{t-1} - \bar{s}_{t-1}) < 0$

3. 止损

   $V_t/V_{t-1} - 1 < -0.02$，则立即平仓，且其后10日都不开仓

4. 仓位

   若开仓：$\alpha_t = 0.3 \times V_t$

   若平仓：$\alpha_t = 0$

3. 配对品种测试

3.1 钢厂利润套利

3.1.1 套利逻辑

根据生产公式

$$\text{吨钢成本} = 1.6\text{吨铁精粉} + 0.5\text{吨焦炭} + \text{生铁费} + \text{钢坯费} + \text{轧材费} + \text{其他费用} \tag{1}$$

我们推导出以下期货价格对应关系

$$\text{螺纹钢期货价格} = 1.6 * \text{铁矿石期货价格} + 0.5 * \text{焦炭期货价格} \tag{2}$$

当上述等式发生显著偏离时，出现套利机会。具体而言，当螺纹钢价格偏高时，钢厂利润空间过大，钢厂会提高开工率，导致铁矿石和焦炭需求增大、价格上涨，从而挤压钢厂利润。此时应做空螺纹钢期货，按上式配比做多铁矿石和焦炭期货。反之，若螺纹钢价格偏低，钢厂将选择减产，从而螺纹钢供给减少、价格上涨，带来螺纹钢价格回归。此时应做多螺纹钢期货，按比例做空铁矿石和焦炭期货。

注意，考虑到每手期货对应的商品吨数不同，在构建价差公式时要有所调整。由于rb为10吨/手，i和j均为100吨/手，因此价差为

$$10 \times rb - 1.6 \times i - 0.5 \times j \tag{3}$$

3.1.2 交易方式

交易要素	内容
交易标的	rb螺纹钢，i铁矿石，j焦炭
交易合约	连续合约0004
交易频率	日频，每日调仓
交易成交时间	当日收盘成交
信号计算方法	价差：10rb-1.6i-0.5j
信号计算使用的价格类型	日线Open
信号交易规则	信号1: 价差在21日均值加 0.8 倍标准差和 1 倍标准差之间，且有回归趋势。 信号2: 穿越21日均值线则进行平仓。
模拟成交使用的价格类型	当日日线收盘价格
交易成本估计	2tick
是否考虑了换月	未考虑换月
收益计算类型	绝对收益
收益计算使用价格类型	真实价格

3.2 甲醇制 PP 利润套利

3.2.1 套利逻辑

甲醇既可自用又可外售的特征是甲醇制 PP 套利可行的重要原因。根据理论生产成本，3吨甲醇另加 800 元加工费用可制得 1 吨聚丙烯。具体到期货价格的层面，在无套利机会的情况下可以得到如下等式

$$\text{聚丙烯期货价格} = 3 * \text{甲醇期货价格} + 800 \tag{4}$$

由于聚丙烯期货和甲醇期货每手吨数相差两倍，价差公式应为

$$\text{价差} = 2 \times MA - 3 \times pp \tag{5}$$

3.2.2 交易方式

交易要素	内容
交易标的	pp聚丙烯，MA甲醇
交易合约	连续合约0004
交易频率	日频，每日调仓
交易成交时间	当日收盘成交
信号计算方法	价差：2pp-3MA
信号计算使用的价格类型	日线Open
信号交易规则	信号1: 价差在21日均值加 0.8 倍标准差和 1 倍标准差之间，且有回归趋势。 信号2: 穿越21日均值线则进行平仓。
模拟成交使用的价格类型	当日日线收盘价格
交易成本估计	2tick
是否考虑了换月	未考虑换月
收益计算类型	绝对收益
收益计算使用价格类型	真实价格

3.3 炼焦套利

煤焦加工套利包括三种模式，分别为独立焦化企业模式、煤矿企业模式以及自有焦化厂的钢 铁企业模式。一级冶炼焦的配煤比例是主焦煤占比 35%、1/3 焦煤占比 25%、气煤 占比 12%、肥煤占比 18%、瘦煤占比 10%。

参考一般的炼焦工艺，平均 1.3 吨炼焦煤加工产生 1 吨焦炭和若干副产品。自焦煤期货上市 以来，期货焦炭指数/焦煤指数的比价均值却高达 1.37，同时较长时间维在 1.38 以上，指数最高比 价曾达到 1.45。因此，我们最终确定炼焦利润的公式为

$$\text{炼焦利润} = \text{焦炭期货价格} - 1.4 * \text{焦煤期货价格} - \text{其他成本} \tag{6}$$

由于焦炭期货为100吨/手，焦煤期货为60吨/手，因此需要调整权重。

$$\text{价差} = 0.6 \times \text{焦炭} + 1.4 \times \text{焦煤} \tag{7}$$

4. 策略表现

4.1 钢厂利润套利表现

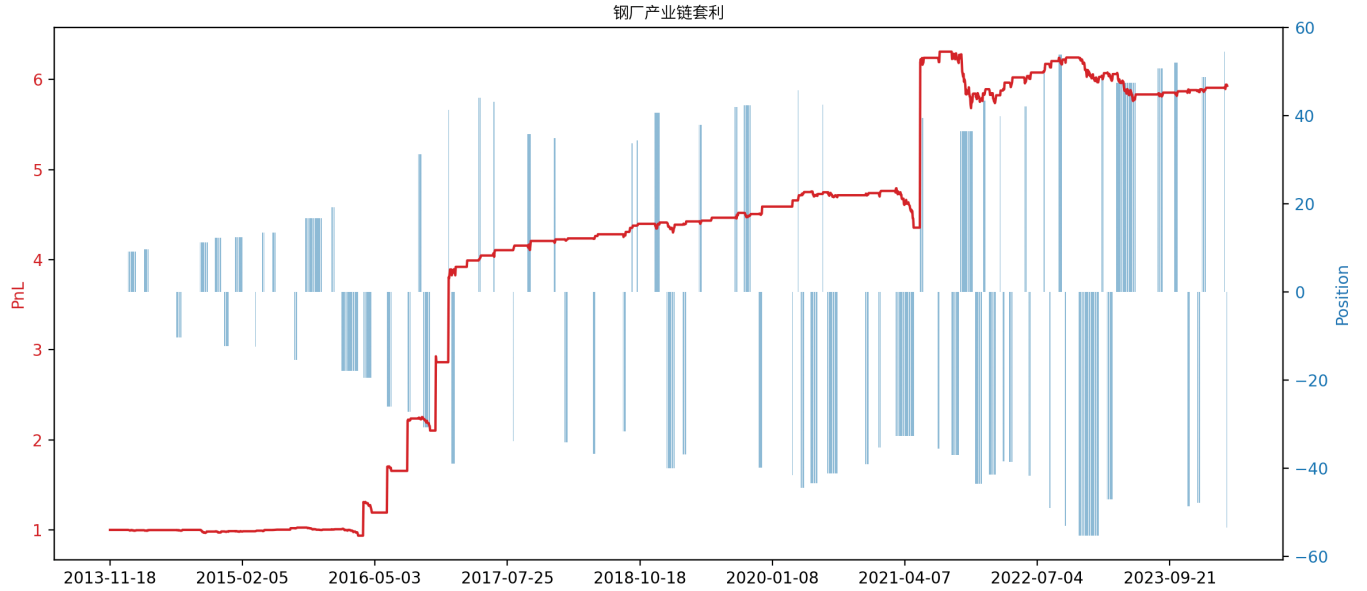


图1: 钢厂产业链套利Pnl曲线和仓位柱状图

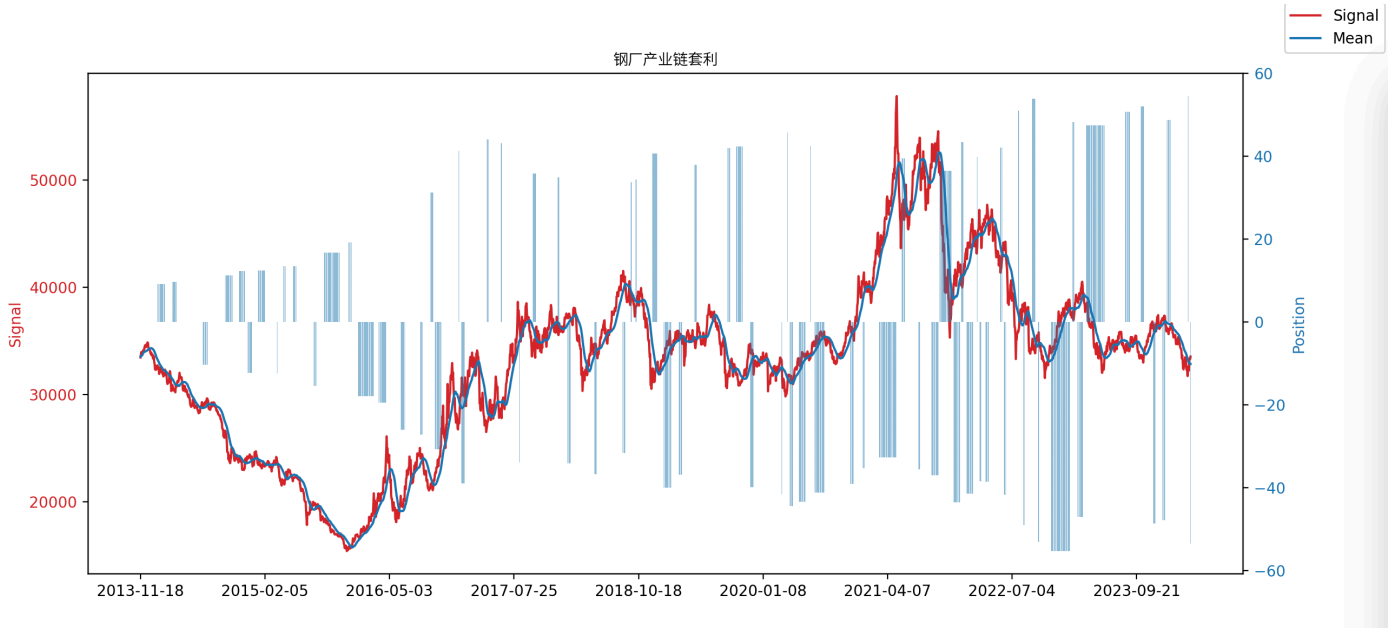


图2: 钢厂产业链套利信号曲线

指标	数值	指标	数值
年化收益率	0.1939	最大回撤率	0.1103
平均持仓周期	10.2625	Calmar	1.7575
胜率	0.5102	盈亏比	1.7463

4.2 甲醇制 PP 利润套利表现

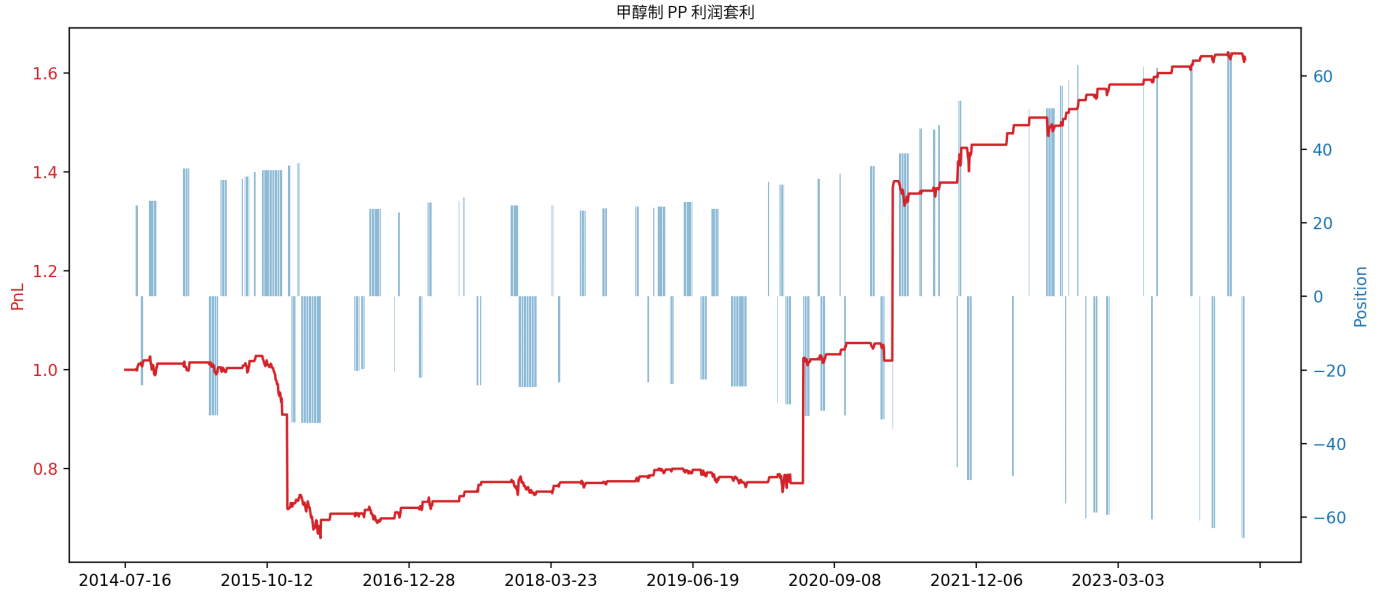


图3: 甲醇制PP利润套利Pnl曲线和仓位柱状图

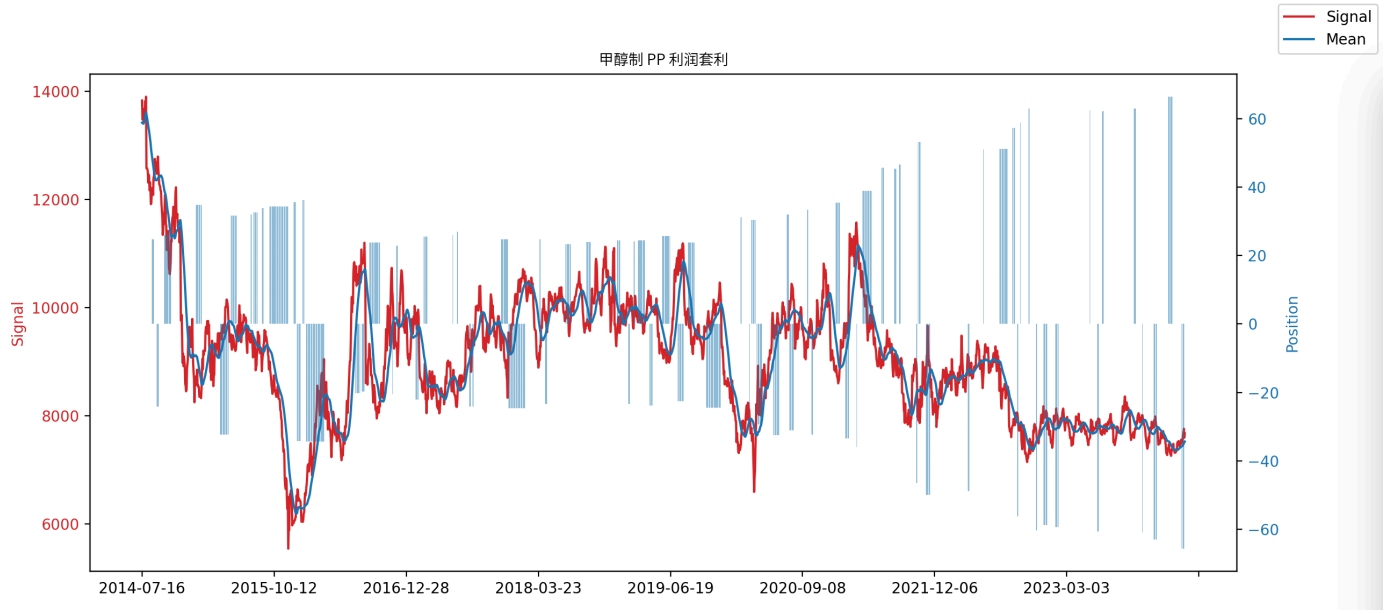
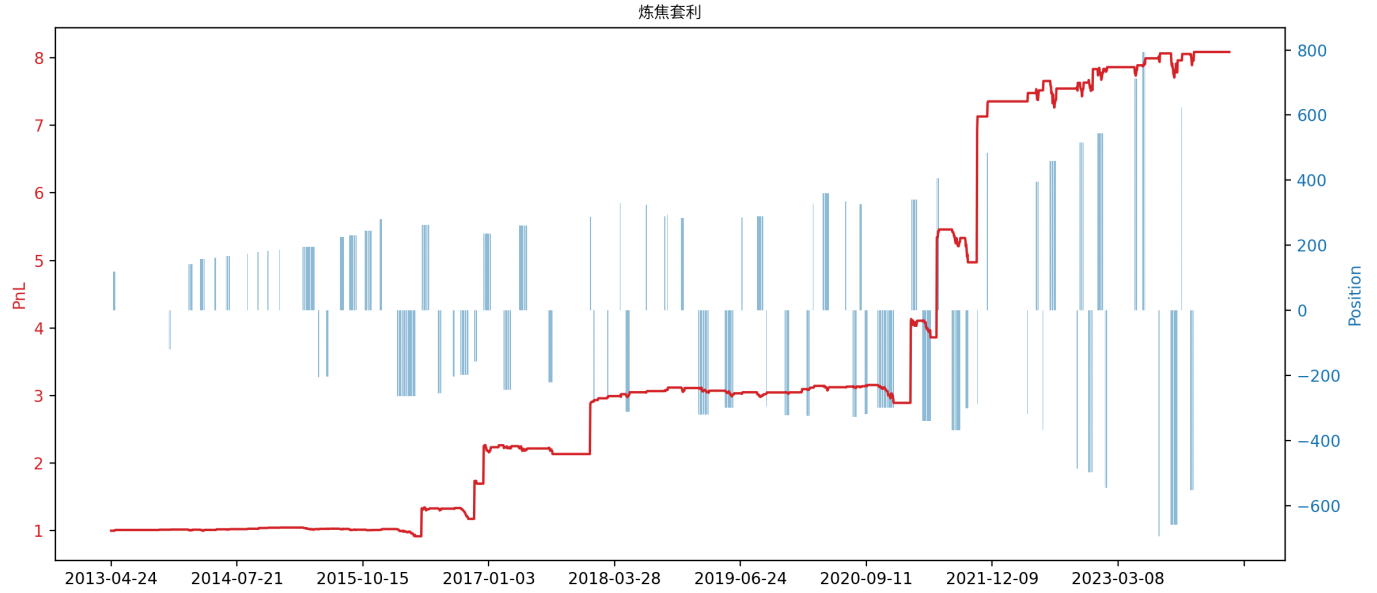
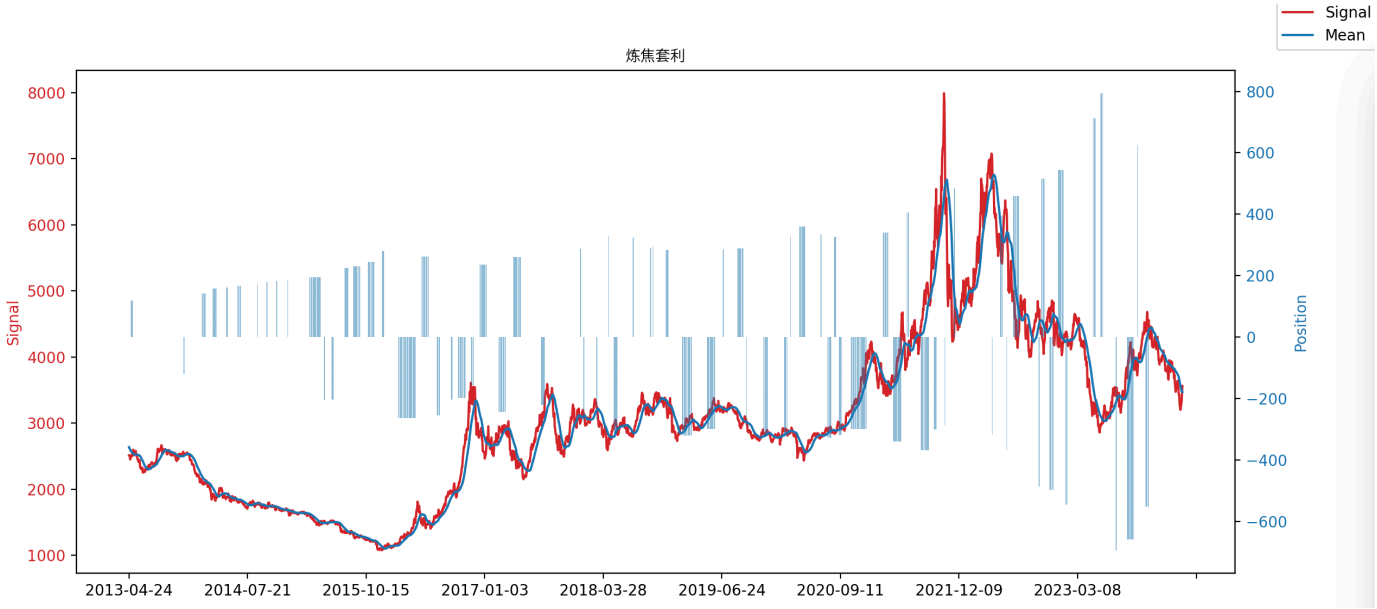


图4: 甲醇制PP利润套利信号曲线

指标	数值	指标	数值
年化收益率	0.0532	最大回撤率	0.5577
平均持仓周期	9.0875	Calmar	0.09542
胜率	0.5074	盈亏比	1.3500

4.3 炼焦套利表现





指标	数值	指标	数值
年化收益率	0.2184	最大回撤率	0.1433
平均持仓周期	9.5131	Calmar	1.5241
胜率	0.4898	盈亏比	2.2232

5. 总结

本文测试了三种产业链下的套利，其中甲醇制PP套利并不理想，而钢厂和炼焦套利表现较好。三次测试选取同一套固定的参数，即开仓条件为过去21日均值加减$0.8 \sim 1$倍的标准差。此前测试过采用原文的过去10日均值，表现相差较大。由于时间原因未完成敏感性分析，这是本策略可以完善的一点。

进步研究的重点是最大回撤发生的原因，需要参考当时经济状况、行业形势，从而对回撤进行归因，并设法解决。

代码附件

```
from causis_api.const import get_version
from causis_api.const import login
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib
login.username = 'dafu.zhu'
login.password = 'kexiYYDS20'
login.version = get_version()
from causis_api.data import *
import pandas as pd
import numpy as np

zh_font1 = matplotlib.font_manager.FontProperties(fname=r"font/SourceHanSansSC-Normal.otf")


class CommodityFutureArbitrage:
    """
    主要参数：止损线、开仓条件（多少sigma）
    """
    def __init__(self, contracts, proportion):
        self.contracts = contracts  # 拉取数据
        self.proportion = proportion   # 原料产出配比
        self.names = [name.split('.')[3] for name in self.contracts]
        self.price_df = pd.DataFrame()
        # 当前的做空、做多或空仓状态
        self.status = 0
        self.position = pd.DataFrame()
        self.current_cash = 1000000    # 初始化总资金100万
        self.contracts_value = 0    # 期货合约价值
        self.values = []     # 总资产随时间变化
        self.alpha = 0   # 交易几份spread
        self.alphas = []  # 仓位变化情况，用于画仓位图
        self.risk_day = 0   # 最近一次止损
        self.sensitivity_record = []

    # 获取数据
    def fetch_data(self):
        for contract in self.contracts:
            tmp_df = get_price(contract, start_date='2013-3-13')[['CLOCK', 'CLOSE']]
            tmp_df.set_index('CLOCK', inplace=True)
            self.price_df = tmp_df if self.price_df.empty else pd.merge(self.price_df,
tmp_df, on='CLOCK')

        self.price_df.columns = self.names

    # 计算开仓平仓信号
    def calculate_signals(self, window=21, close=0.8, distal=1.0):
        # 利润回归模型
        # 计算价差
        self.price_df['spread'] = sum(x * self.price_df[y] for x, y in zip(self.proportion,
self.names))
        self.price_df['mean'] = self.price_df['spread'].rolling(window=window).mean()
        self.price_df['std'] = self.price_df['spread'].rolling(window=window).std()
        self.price_df['lower_short'] = self.price_df['mean'] + close * self.price_df['std']
        self.price_df['upper_short'] = self.price_df['mean'] + distal * self.price_df['std']
        self.price_df['lower_long'] = self.price_df['mean'] - distal * self.price_df['std']
```

```
        self.price_df['upper_long'] = self.price_df['mean'] - close * self.price_df['std']
        self.price_df = self.price_df.dropna()
        self.position = self.price_df.copy().iloc[:, :len(self.names)]
        self.position.iloc[:, :len(self.names)] = 0

    # 开始交易
    def trade(self):
        """
        总共4种情况：未开仓->开仓；开仓->平仓；未开仓->未开仓；开仓->开仓
        :return:
        """
        df = self.price_df
        s0 = 0   # 初始化上一期的spread
        s0_bar = 0   # 初始化上一期spread对应的前10天mean
        init_position = -1 * np.array(self.proportion)   # 空首位
        for idx, row in df.iterrows():
            s = row['spread']
            s_bar = row['mean']

            # 开仓条件1：未开仓且不在止损后10天内
            if (self.status == 0) and (df.index.get_loc(idx) - self.risk_day > 10 or
self.risk_day == 0):
                # 合约金额为总金额的30%
                self.alpha = 0.3 * self.current_cash / s

                # 前两个情况为开仓，确认方向
                if row['lower_short'] < s < row['upper_short'] and s < s0:
                    # print("Entering a short position")
                    self.position.loc[idx] = self.alpha * init_position
                    self.status = 1
                    # 构造合约组合支付的费用
                    self.contracts_value = np.sum(np.array(row[self.names]) *
self.position.loc[idx])
                    # 现金减去支付费用为留存现金
                    self.current_cash -= self.contracts_value
                    self.alphas.append(-self.alpha)    # 负数表示做空螺纹钢

                elif row['lower_long'] < s < row['upper_long'] and s > s0:
                    # print("Entering a long position")
                    self.position.loc[idx] = -self.alpha * init_position
                    self.status = 1
                    # 构造合约组合支付的费用
                    self.contracts_value = np.sum(np.array(row[self.names]) *
self.position.loc[idx])
                    # 现金减去支付费用为留存现金
                    self.current_cash -= self.contracts_value
                    self.alphas.append(self.alpha)

                # 后一个情况为保持未开仓状态，不做操作
                else:
                    self.alphas.append(0)
                    pass

                self.values.append(self.current_cash + self.contracts_value)

            # 平仓条件1：已开仓
            elif self.status:
                # 上一期头寸配置
```

```
                previous_position = self.position.shift(1).loc[idx]
                # 开仓->平仓
                if (s - s_bar) * (s0 - s0_bar) < 0:   # 穿过MA线
                    # 抹平头寸
                    self.position.loc[idx] = 0 * init_position
                    self.status = 0
                    # 清点平仓时合约组合收益，用上期头寸数据
                    self.current_cash += np.sum(np.array(row[self.names]) *
previous_position)
                    # 合约价值清零
                    self.contracts_value = 0
                    self.values.append(self.current_cash + self.contracts_value)
                    self.alphas.append(0)

                # 保持开仓
                else:
                    # 更新合约组合价值，头寸不做改动
                    self.position.loc[idx] = previous_position
                    self.contracts_value = np.sum(np.array(row[self.names]) *
self.position.loc[idx])
                    self.values.append(self.current_cash + self.contracts_value)
                    self.alphas.append(self.alphas[-1])
                    # 止损指令
                    if self.values[-1] / self.values[-2] - 1 < -0.02:
                        self.position.loc[idx] = 0 * init_position
                        self.risk_day = df.index.get_loc(idx)
                        self.status = 0

            # 未开仓，但在risk_day的10天内
            else:
                self.values.append(self.values[-1])
                self.alphas.append(0)

            s0 = s
            s0_bar = s_bar

    # 存储结果
    def save_data(self, file_name):
        self.price_df.to_excel(f'results/price_df_{file_name}.xlsx')
        self.position.to_excel(f'results/position_{file_name}.xlsx')
        value = pd.DataFrame(self.values, index=self.price_df.index)
        value.to_excel(f'results/values_{file_name}.xlsx')

    # 仓位柱状图，Pnl曲线
    def pnl(self, title):
        # 生成示例数据
        dates = self.price_df.index.tolist()
        positions = self.alphas  # 仓位柱状图
        pnl = np.array(self.values) / self.values[0]  # Pnl曲线

        fig, ax1 = plt.subplots(figsize=(15, 6))

        # PnL曲线，现在将其设置为左侧Y轴
        color = 'tab:red'
        ax2 = ax1.twinx()
        ax2.plot(pnl, color=color, label='PnL')
        ax2.set_ylabel('PnL', color=color)
        ax2.tick_params(axis='y', labelcolor=color)
```

```
        ax2.yaxis.tick_left()  # 将PnL的Y轴移至左侧
        ax2.yaxis.set_label_position('left')  # 设置PnL标签位置到左侧

        # 仓位柱状图，现在将其设置为右侧Y轴
        color = 'tab:blue'
        ax1.bar(dates, positions, color=color, alpha=0.5)
        ax1.set_ylabel('Position', color=color)
        ax1.tick_params(axis='y', labelcolor=color)
        ax1.xaxis.set_major_locator(ticker.MultipleLocator(300))
        ax1.yaxis.tick_right()  # 将仓位的Y轴移至右侧
        ax1.yaxis.set_label_position('right')  # 设置仓位标签位置到右侧

        # 调整布局
        # fig.tight_layout()
        plt.title(title, fontproperties=zh_font1)
        plt.show()

    # 标的价格曲线 + 信号曲线
    def signal_curve(self, title):
        # 用spread充当信号
        df = self.price_df
        signal = df['spread']
        positions = self.alphas
        dates = df.index.tolist()

        fig, ax1 = plt.subplots(figsize=(15, 6))

        # 信号曲线
        color = 'tab:red'
        ax1.plot(dates, signal, color=color, label='Signal')
        ax1.set_ylabel('Signal', color=color)
        ax1.tick_params(axis='y', labelcolor=color)
        ax1.xaxis.set_major_locator(ticker.MultipleLocator(300))
        ax1.plot(df['mean'], label='Mean')

        # 仓位柱状图，现在将其设置为右侧Y轴
        color = 'tab:blue'
        ax2 = ax1.twinx()
        ax2.bar(dates, positions, color=color, alpha=0.5)
        ax2.set_ylabel('Position', color=color)
        ax2.tick_params(axis='y', labelcolor=color)
        ax2.xaxis.set_major_locator(ticker.MultipleLocator(300))
        ax2.yaxis.tick_right()  # 将仓位的Y轴移至右侧
        ax2.yaxis.set_label_position('right')  # 设置仓位标签位置到右侧
        fig.legend()

        # fig.tight_layout()
        plt.title(title, fontproperties=zh_font1)
        plt.show()


def sensitivity_test(contracts, proportion):
    calmar_ratios = []
    strategy = CommodityFutureArbitrage(contracts, proportion)
    strategy.fetch_data()
    # 条带长度控制在0.2*sigma
    pairs = [[x, x+0.2] for x in np.arange(0, 2, 0.1)]
    for pair in pairs:
```

```
        strategy.calculate_signals(close=pair[0], distal=pair[1])
        strategy.trade()
        measures = measure(strategy.values, strategy.position)
        calmar_ratios.append(measures['calmar'])

    plt.plot(calmar_ratios)
    plt.show()


# 套利
def arbitrage(contracts, proportion, title):
    strategy = CommodityFutureArbitrage(contracts,  proportion)
    strategy.fetch_data()
    strategy.calculate_signals()
    strategy.trade()
    strategy.save_data(title)
    strategy.pnl(title)
    strategy.signal_curve(title)
    return strategy.values, strategy.position


class Measures:
    def __init__(self, values, position):
        self.values = values
        self.position = position

    def arr(self):
        end = self.values[-1]
        start = self.values[0]
        years = 1 / 252 * len(self.values)
        return (end / start) ** (1 / years) - 1

    def max_drawdown(self):
        values = self.values
        max_dd = 0
        peak = values[0]
        for value in values:
            if value > peak:
                peak = value
            dd = peak / value - 1
            if dd > max_dd:
                max_dd = dd
        return max_dd

    def sharpe(self):
        arr = self.arr()
        std = np.std(self.values)
        return arr / std

    def calmar(self):
        arr = self.arr()
        max_dd = self.max_drawdown()
        return arr / max_dd

    # 此处为每日胜率，但调仓胜率更合理
    def win_rate(self):
        values = np.array(self.values)
        count = np.sum((values[1:] - values[:-1]) > 0)
```

```
        num = np.sum((values[1:] - values[:-1]) != 0)
        return count / num

    # 盈亏比
    def pcr(self):
        values = np.array(self.values)
        chg = values[1:] - values[:-1]
        profit = np.sum(chg[chg > 0]) / np.sum(chg > 0)
        coss = np.sum(chg[chg < 0]) / np.sum(chg < 0)
        return np.abs(profit / coss)

    # 平均持仓周期
    def avg_hold(self):
        filtered_position = self.position[(self.position != 0).all(axis=1)]
        total_period = filtered_position.shape[0]
        unique_position = filtered_position.drop_duplicates().shape[0]
        return total_period / unique_position


def measure(values, position):
    measures = Measures(values, position)
    dist = {
        'arr': measures.arr(),
        'max_drawdown': measures.max_drawdown(),
        'sharpe': measures.sharpe(),
        'calmar': measures.calmar(),
        'win_rate': measures.win_rate(),
        'pcr': measures.pcr(),
        'avg_hold': measures.avg_hold()
    }
    series = pd.Series(dist, index=dist.keys())
    return series


if __name__ == '__main__':
    # 钢厂产业链套利策略净值
    values1, position1 = arbitrage(['R.CN.SHF.rb.0004', 'R.CN.DCE.i.0004',
'R.CN.DCE.j.0004'], [10, -1.6, -0.5], '钢厂产业链套利')
    measure_data1 = measure(values1, position1)
    measure_data1.to_excel('results/钢厂产业链套利表现.xlsx')
    # 甲醇制 PP 利润套利
    values2, position2 = arbitrage(['R.CN.DCE.pp.0004', 'R.CN.CZC.MA.0004'], [2, -3], '甲醇制
PP 利润套利')
    measure_data2 = measure(values2, position2)
    measure_data2.to_excel('results/甲醇制 PP 利润套利表现.xlsx')
    # 炼焦套利
    values3, position3 = arbitrage(['R.CN.DCE.j.0004', 'R.CN.DCE.jm.0004'], [0.6, 1.4], '炼焦
套利')
    measure_data3 = measure(values3, position3)
    measure_data3.to_excel('results/炼焦套利表现.xlsx')
    print(measure_data3['sharpe'])
```