第 5 章

CHAPTER 5

基金交易策略

完成量化系统搭建后，在 4.4 节中介绍的策略基类 PortfolioStrategy 的基础上扩展策略子类实现具体策略逻辑即可。本章将以基础的单基金策略为例讲解策略编写的方法、执行过程与回测结果评估，由浅入深地讲解更为复杂策略的编写与执行。

6min

5.1 买入并持有策略

下面以最简单的策略讲解子类的编写方法，策略逻辑十分简单，即“买入并持有”：在收到基金行情的第 1 天即全仓买入该基金，并持有到最后一天。

由于此策略不需要对行情进行任何判断，所以在策略的初始化回调和启动回调中无须执行任何逻辑。对于策略执行的初步想法为在收到基金数据时获取该基金代码并判断当前仓位，如果当前有持仓，则不进行操作，否则满仓买入。在策略基类中，已经对持仓信息和可用资金进行了维护，因此容易根据基类中的 pos_symbols 和 available_capital 变量判断当前持仓和可用资金的情况，在收到基金数据回调函数中使用 buy 方法完成基金的买入即可，完整的策略代码如下：

```
//ch5/5.1/buy_and_hold_strategy.py
from typing import Dict

#pylint: disable = import - error
from strategies.base import PortfolioStrategy
from utils.object import FundData


class BuyAndHoldStrategy(PortfolioStrategy):
    """买入并持有策略"""

    author = "ouyangpengcheng"

    def on_init(self) -> None:
        """策略初始化回调"""
```

```
    def on_start(self) -> None:
        """策略启动回调"""

    def on_stop(self) -> None:
        """策略停止回调"""
        self.send_latest_data()

    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到基金数据回调"""
        super().on_fund_data(fund_data)
        symbol = list(fund_data.keys())[0]

        if not self.pos_symbols:
            # 如果没有持仓就买入
            if self.available_capital > 0:
                # 全仓买入
                amount = self.available_capital
                self.buy(symbol, amount)
```

基于第 4 章回测引擎的使用方法，不难理解如下所示的策略回测代码：

```
//ch5/5.1/main_cmd.py
from datetime import datetime
from engine.backtesting import Engine
from strategies import BuyAndHoldStrategy

from utils.constant import FUND_INFO_JSON_PATH, FUND_POOL_TXT_PATH, SERVICE_CHARGE
from utils.data_loader import get_valid_pool_codes

# 资金总量
CAPITAL = 10000
# 策略研究的基金对象代码
symbols = ["000001"]
# 基金申购赎回的手续费率
rates = {"000001": SERVICE_CHARGE}

# 初始化回测引擎
engine = Engine(
    strategy_cls=BuyAndHoldStrategy,
    symbols=symbols,
    start=datetime(2001, 1, 1),
    end=datetime(2023, 8, 31),
    rates=rates,
    capital=CAPITAL,
)

# 1. 运行回测
engine.run_backtesting()
# 2. 计算回测指标
```

```
engine.calculate_statistics()
#3. 保存并展示 QuantStats 的回测结果
engine.save_result()
```

执行如上所示的回测代码，可以得到如图 5-1 所示的回测收益结果。

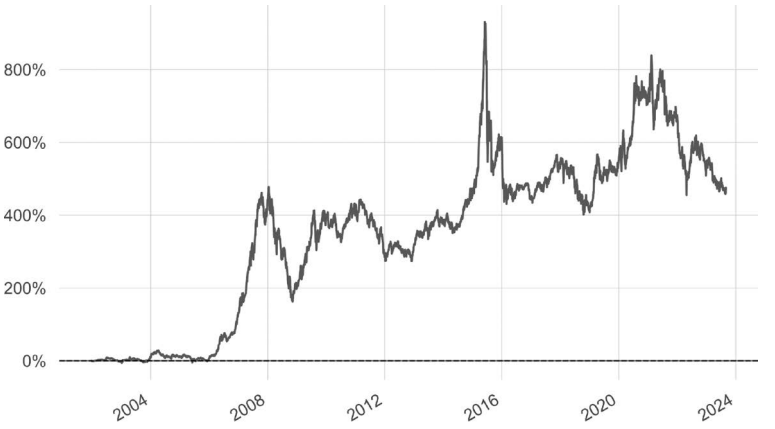


图 5-1 “买入并持有”策略的回测收益结果

从图 5-1 中可以看出，在回测的过程中，累计收益率最高达到 800%多，在回测结束时回落到 400%多，年化收益率在 5.7%左右，夏普比率为 0.37。回测结果中还会展示年终收益率(EOY Returns)，表示每年获得的收益率，如图 5-2 所示。

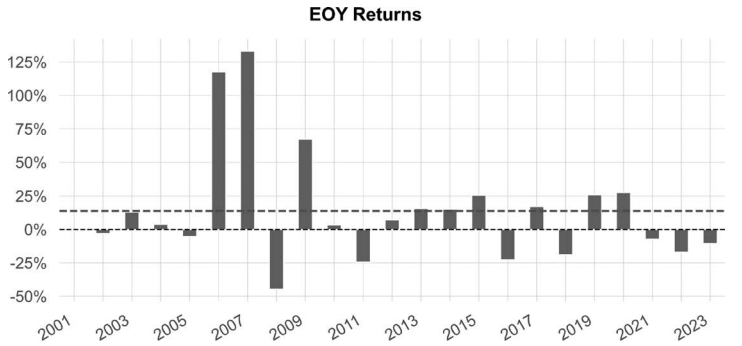


图 5-2 “买入并持有”策略的年终收益率

从图 5-2 中可以看到，策略在 2006—2007 年的收益表现很好，基本实现了资产的翻倍，而在接下来的 2008 年收益率约为－50%，由于年终收益率表示每年年终相较于年初资产的收益率，2008 年获得的－50%收益率相当于回吐了 2007 年的全年收益。在累计收益率曲线上同样可以观察到回撤的情况，QuantStats 会绘制回测过程中最严重的 5 次回撤时间段，如图 5-3 所示。

从图 5-3 中的回撤情况可以看出回撤的深度与广度：从纵轴资产曲线的下降情况可以看出回撤的深度，而从横轴时间跨度能够看出本次回撤的持续性。QuantStats 会以月作为时间单位展示每个月的收益情况热力图，如图 5-4 所示。

图 5-3　“买入并持有”策略的回撤情况

Strategy - Monthly Returns (%)

2001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.99
2002	0.20	1.10	0.99	2.25	-1.82	7.13	-2.61	0.48	-2.86	-2.94	-1.41	-3.18
2003	6.88	0.40	0.89	0.68	2.23	-4.49	1.34	-3.25	-3.56	1.74	2.03	8.27
2004	8.12	4.38	2.83	-6.08	-1.69	-5.65	1.67	-1.74	4.93	-2.07	0.96	-0.83
2005	-2.07	3.93	-4.46	1.83	-9.87	2.10	0.98	2.04	0.53	-3.03	-2.70	6.32
2006	7.20	1.56	4.98	17.30	17.00	2.08	-5.83	1.81	6.52	0.99	14.89	14.73
2007	11.58	3.69	5.38	22.66	12.27	0.50	15.56	15.85	3.87	0.81	-12.75	13.03
2008	-6.22	0.62	-11.74	-0.35	-3.04	-11.14	1.97	-10.93	-5.96	-18.81	8.51	2.72
2009	7.00	0.99	13.74	6.82	3.63	10.60	12.90	-20.60	6.92	8.01	7.28	0.14

图 5-4　“买入并持有”策略的月收益率热力图

从月收益率热力图能够以更精细的角度观察收益情况，从图 5-4 同样可以看出，在 2006—2007 年间一个月可以获得 10%以上甚至 20%的收益，而 2008 年在大部分月份出现了亏损。

本节从最简单的“买入并持有”策略入手完成策略编写基础方法的讲解，实际的策略并不会使用如此简单的逻辑，读者应通过本节讲解的策略编写方法理解策略的运行过程与策略的执行方法。

5.2　定投策略

定投即定期投资，定投策略在基金投资中十分常用，而定期投资可以根据投资份额、投资金额分为不同的策略，众多投资者从资料中了解到定投是一种“懒人理财法”，然而这种方法究竟能够带来多少收益，使用何种具体的策略可以获取多少收益，很多资料都缺乏量化的说明，本节通过编写定投策略的代码为读者呈现直观的定投策略评价。

在生活中很多读者参与过基金的定投，国家建立的养老保险基金实际上就是定投的一种应用，每个月从个人工资中拿出一部分存入养老保险基金，在退休时支取养老金。相当于

一个以月为频率的定投策略，类似于银行的零存整取。

基金的投资门槛低，因此基金定投不需要太高的成本；由于定投本身固定了投资的时机，投资者无须费心进行择时，到了定投的时间点投入资金即可，操作十分简便，同时定期投资也避免了主观情绪的影响，在定投时间点坚持买入即可。

通常来讲，定投可以分为同金额、同份额与同价值 3 种策略，最常见的定投方法是同金额定投，大多数基金软件也仅支持同金额定投。同金额定投即定期买入一定金额的基金，无论当前的净值是多少。在定投策略中，需要关心定投时间间隔（买入周期），因此策略中需要指定该变量的值，主要逻辑仍然位于行情回调函数中：当上一次买入时间到今日的时长大于时间间隔并且手头还有可用资金时，则进行同金额买入，否则继续持有，整体策略逻辑如下：

```
//ch5/5.2/automatic_invest_strategy_same_amount.py
class AutomaticInvestStrategy(PortfolioStrategy):
    """定投策略"""

    author = "ouyangpengcheng"

    #持有周期
    holding_period = 120
    #定投金额
    automatic_invest_amount = 250

    parameters = ["holding_period"]

    def __init__(self) -> None:
        """构造函数"""
        super().__init__()
        #持有日期
        self.holding_days = 0
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        symbol = list(fund_data.keys())[0]

        #从最近一次买入到今日如果大于持有时长，则需要买入
        if self.holding_days >= self.holding_period:
            if self.available_capital > 0:
                #买入定投的金额
                amount = min(
                    self.available_capital,
                    self.automatic_invest_amount
                )
                self.buy(symbol, amount)
                #如果有买入行为，则重置持仓时长
                self.holding_days = 0

        self.holding_days += 1
```

策略中每半年(120 天)买入一次基金,以同金额定投基金 000001 为例,得到回测的结果如图 5-5 所示。

图 5-5　同金额定投策略的收益率曲线

相比于图 5-1 所示的“买入并持有”的收益率曲线,图 5-5 所示的收益率曲线走势类似,但是收益率更低,这是因为定投每次买入金额较低,在投资的过程中获取的收益也较低。

从图 5-5 可以看出该基金在大多数时间呈上涨趋势,而在上涨中不断买入会导致成本不断变高,也会造成最终收益较低,由此可以改进原来的定投策略:在最新净值低于持仓成本的时候才买入,用于降低持仓成本。

策略基类维护了持仓信息,包括每个持仓标的平均价格与份额等信息,因此很容易判断最新净值与持仓均价的关系,只需在买入之前增加一次判断逻辑,代码如下:

```
//ch5/5.2/automatic_invest_strategy_same_amount_adj1.py
…
def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
    """收到行情回调"""
    super().on_fund_data(fund_data)
    symbol = list(fund_data.keys())[0]

    #从最近一次买入到今日如果大于持有时长,则需要买入
    if self.holding_days >= self.holding_period:
        if self.available_capital > 0:
            #最新的净值
            latest_av = self.symbol_latest_adjust_val(symbol)
            #持仓均价
            pos_price = self.pos_symbol_info[symbol].get(self.price_key)
            if pos_price is None or latest_av < pos_price:
                #只有当前净值小于持仓价格才买入
                #买入定投的金额
                amount = min(
                          self.available_capital,
                          self.automatic_invest_amount
```

```
        )
                self.buy(symbol, amount)
                #如果有买入行为,则重置持仓时长
                self.holding_days = 0

    self.holding_days += 1
```

运行策略得到如图 5-6 所示的收益率曲线,结果与上文的预想有所差异,同样收益率曲线走势类似,然而获得的收益率却更低。

图 5-6 低价买入同金额定投策略的收益率曲线

通过对回测中的买入记录分析可以知道,在 2006—2007 年由于净值增长较快,而之前买入的持仓均价较低,所以在 2007 年之后净值没有小于持仓成本的时间,在 2007 年之后没有继续买入基金。相当于在回测过程中只在 2007 年以前执行每半年的同金额买入操作,在 2007 年以后则一直持有到回测结束,这样进一步减小了资金使用率,也导致了收益降低。

在净值小于持仓均价的情况下进行买入的策略保持了每半年买入一次的逻辑,实际上“净值小于持仓均价”这一条件充当了买入的过滤器,可以保证持仓均价不会升高(至少不会升高过快,因为对于买入的净值预测不一定准确),因此可以减小买入的时间间隔,本书采取了极端值,即减小买入时间间隔为 1,可以得到如图 5-7 所示的收益率曲线。

相较于图 5-6,图 5-7 所示的收益率曲线走势类似,然而通过观察纵轴可以发现,其大幅提升了收益率,验证了上文提出的想法。

对于买入时机的过滤相当于提高了买入带来的收益率,而通过观察图 5-7 可以发现在回测过程中常常获得高额收益后由于没有卖出,所以遭受了重大的回撤,可以继续在卖出的时机进行改进,如当前净值超过持仓净值一定比例后卖出基金,如下代码展示了基金的卖出逻辑:

```
//ch5/5.2/automatic_invest_strategy_same_amount_adj3.py
class AutomaticInvestStrategy(PortfolioStrategy):
    """定投策略"""
```

```
author = "ouyangpengcheng"
…
def clear_pos(self):
    """清仓"""
    for _symbol in self.pos_symbol_info:
        self.sell(
            _symbol, abs(self.pos_symbol_info.get(_symbol, {}).get(self.volume_key))
        )

def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
    """收到行情回调"""
    super().on_fund_data(fund_data)
    symbol = list(fund_data.keys())[0]

    #从最近一次买入到今日如果大于持有时长,则需要买入
    if self.holding_days >= self.holding_period:
        if self.available_capital > 0:
            #最新的净值
            latest_av = self.symbol_latest_adjust_val(symbol)
            #持仓均价
            pos_price = self.pos_symbol_info[symbol].get(self.price_key)
            if pos_price is None or latest_av < pos_price:
                #只有当前净值小于持仓价格才买入
                #买入定投的金额
                amount = min(self.available_capital, self.automatic_invest_amount)
                self.buy(symbol, amount)
                #如果有买入行为,则重置持仓时长
                self.holding_days = 0
            elif pos_price is not None and latest_av >= pos_price * 10:
                #如果产生了一定盈利,则全部卖出
                self.clear_pos()
                self.holding_days = 0

    self.holding_days += 1
```

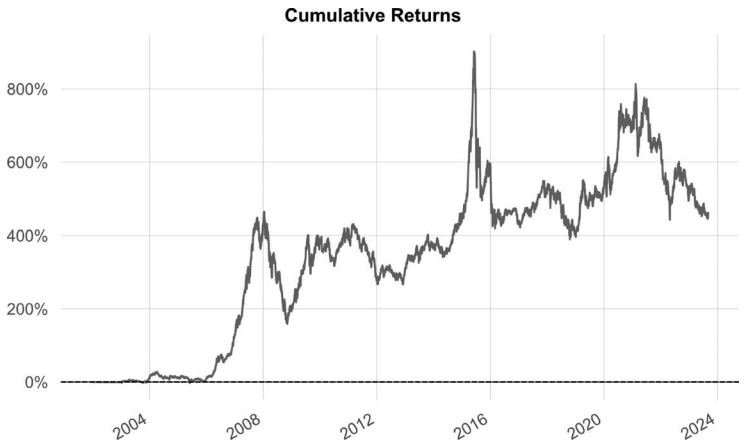


图 5-7　同金额定投策略的收益率曲线

加入卖出逻辑后，运行回测可以得到如图 5-8 所示的收益率曲线，从图中可以看出相比之前的收益率曲线波动更小，说明加入的基金卖出逻辑生效，然而回测周期内获得的收益降低了，因此基金的收益率阈值需要仔细研究并确定合适的取值。

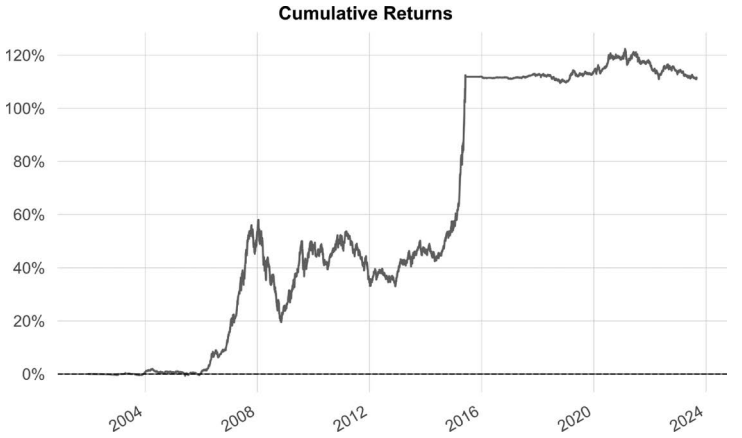


图 5-8　加入卖出逻辑的定投策略收益率曲线

除了同金额定投策略以外，还有同份额定投策略，即每隔一定时间买入相同份额的基金，但是基金的申购最终是以金额进行计算的，无法直接以确定的份额进行认购，因此以前一天的净值乘以定投份额得到申购的金额，总体来讲在标准的同金额定投策略的代码的基础上进行较小改动即可得到同份额定投策略，代码如下：

```
//ch5/5.2/automatic_invest_strategy_same_share.py
class AutomaticInvestStrategy(PortfolioStrategy):
    """定投策略"""

    author = "ouyangpengcheng"

    #持有周期
    holding_period = 120
    #定投份额
    automatic_invest_share = 50
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        symbol = list(fund_data.keys())[0]

        #从最近一次买入到今日如果大于持有时长，则需要买入
        if self.holding_days >= self.holding_period:
            if self.available_capital > 0:
                latest_av = self.symbol_latest_adjust_val(symbol)
                #买入定投的金额 = 最近的净值 * 定投份额
                share_amount = self.automatic_invest_share * latest_av
                amount = min(self.available_capital, share_amount)
```

```
                self.buy(symbol, amount)
                #如果有买入行为,则重置持仓时长
                self.holding_days = 0

        self.holding_days += 1
```

回测后可以得到如图 5-9 所示的结果,对比图 5-5 所示的同金额定投收益率,图 5-9 所示的收益更低,因为同份额定投在净值低时投入的金额低,而在净值高时投入的金额高,是一种会快速提高持仓成本的做法,因此相较于同金额定投策略,它的表现会更差。

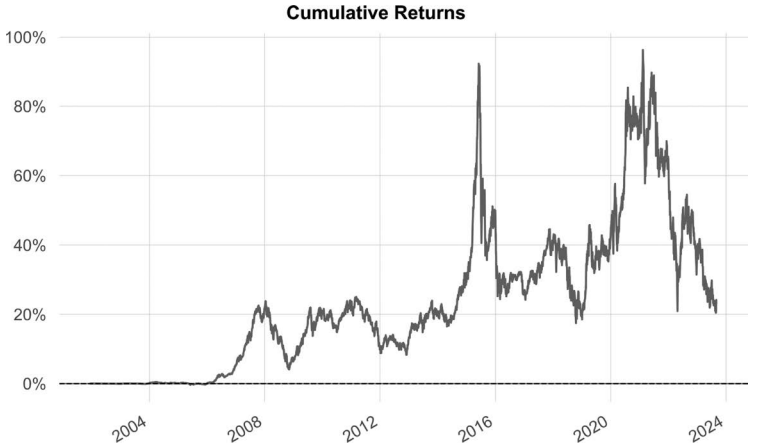


图 5-9　同份额定投策略的收益率曲线

一个更加合理的投资策略应该是：在净值较低时买入更多,用于降低持仓成本,而在净值较高时应轻仓买入甚至获利减仓,这个思想正好与同份额定投策略相反,恰恰也对应了另一种定投策略：同价值定投策略。

同金额定投策略与同份额定投策略的研究对象都是每次买入的金额或者份额,而同价值定投策略的研究对象是每次购买完成后手中基金的价值。例如首次申购 100 元的基金,由于市场的波动造成持仓的价值减少到 50 元,那么在第 2 次则应该申购 150 元的基金,这样才能保证持仓价值为 200 元,而由于市场波动造成持仓价值增加到 280 元,则第 3 次申购的金额为 20 元,保证持仓的价值为 300 元。从以上描述可以知道同价值定投策略是一种在行情上涨时轻仓买入而在下跌时加仓买入的策略,带有天然的择时能力。

同价值定投策略需要计算持仓价值,使用持仓的份额乘以前一天的净值(准确来讲应该乘以当天的净值,但是其无法在盘中获得)得到,再计算实际持仓价值与今日应达到的持仓价值之差,即本次应买入的金额。由于买入间隔时间较长造成净值增长较多,所以造成实际持仓价值已经超过应达到的持仓价值,此时以阶梯的形式继续累加直到应达到的持仓价值刚好超过实际持仓价值,逻辑代码如下：

```
//ch5/5.2/automatic_invest_strategy_same_value.py
class AutomaticInvestStrategy(PortfolioStrategy):
    """定投策略"""
```

```
author = "ouyangpengcheng"

#持有周期
holding_period = 120
#计算开仓金额的次数
calc_times = 0
#第1次买入的金额
first_buy_amount = 250
…
def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
    """收到行情回调"""
    super().on_fund_data(fund_data)
    symbol = list(fund_data.keys())[0]

    #从最近一次买入到今日如果大于持有时长,则需要买入
    if self.holding_days >= self.holding_period:
        latest_av = self.symbol_latest_adjust_val(symbol)
        symbol_volume = self.pos_symbol_info[symbol].get(self.volume_key, 0)
        #计算持仓价值
        pos_value = symbol_volume * latest_av

        #本次投资结束应有的持仓价值
        automatic_invest_cum_value = (self.calc_times + 1) * self.first_buy_amount
        #本次应投资的金额
        amount = automatic_invest_cum_value - pos_value

        #保证每次买入的金额大于0
        while amount <= 0:
            self.calc_times += 1
            automatic_invest_cum_value = (
                self.calc_times + 1
            ) * self.first_buy_amount
            amount = automatic_invest_cum_value - pos_value

        if self.available_capital > 0:
            amount = min(amount, self.available_capital)
            self.buy(symbol, amount)

            self.calc_times += 1
            #如果有买入行为,则重置持仓时长
            self.holding_days = 0

    self.holding_days += 1
```

回测得到如图5-10所示的收益率曲线,相比于图5-5和图5-9可以发现,同价值定投策略可以获得更高的收益,得益于同价值定投策略带有的择时能力。

定投策略作为一种简单有效的交易策略,读者应了解其交易思想并且熟悉代码的编写方法。实际使用的交易软件通常支持以同金额的形式执行定投,有的支持价值被低估时增加定投的金额,这种思想实际上与同价值定投较为类似,都有降低持仓成本的效果。

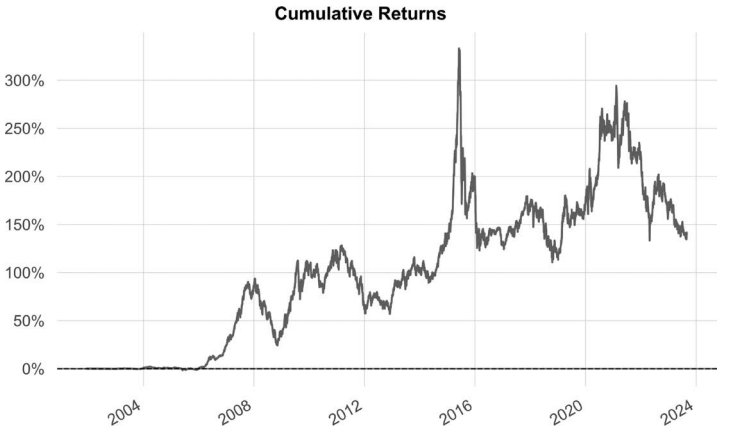


图 5-10　同价值定投策略的收益率曲线

本节介绍的同份额与同价值定投策略都以最基础的同金额定投策略改写而来，读者可以使用类似的方法对其进行改进并评估回测表现。

5.3　双均线策略

计算时间序列的均线可以起到捕捉主要趋势、忽略短期波动的作用。一般使用移动平均线 MA(t)作为时间序列的均线计算基准，MA(t)表示计算过去长度为 t 的时间序列的平均值，计算方法如式(5-1)所示。

$$\mathrm{MA}(t)=\frac{1}{t}\sum_{i=T-t+1}^{T}x_i \tag{5-1}$$

式(5-1)中，T 表示序列$\{x_i \mid x\in[1,T]\}$的总长度。由于进行了均值计算，当数据出现异常波动时，其对均线的影响程度会比原序列更小，均线起到了减小波动的作用。计算式(5-1)所示均值的代码如下：

```
//ch5/5.3/explore/ma.py
def ma(data, period):
    """计算均线"""
    sma = []
    for i in range(len(data)):
        if i < period:
            # 当当前下标小于窗口长度时, 均值为 NaN
            sma.append(np.NaN)
        else:
            # 计算均值
            sma.append(np.mean(data[i - period : i]))
    return sma
```

对于金融时间序列来讲，常使用的均线周期包括 20 日（月线）、60 日（季线）、120 日（半年线）和 240 日（年线），在如上所示的函数中将 period 参数指定为相应的周期即可，使用基金 000001 的复权净值计算均线并绘图，代码如下：

```
//ch5/5.3/explore/ma.py
file_name = "000001.csv"
content = pd.read_csv(file_name)
#获取复权净值
adjust_val = content["adjust_val"]
…
#绘制复权净值走势
plt.plot(adjust_val, label = "adjust_val")
#绘制月线均值
plt.plot(ma(adjust_val, 20), label = "ma20")
#绘制季线均值
plt.plot(ma(adjust_val, 60), label = "ma60")
#绘制半年线均值
plt.plot(ma(adjust_val, 120), label = "ma120")
#绘制年线均值
plt.plot(ma(adjust_val, 240), label = "ma240")

plt.legend()
plt.show()
```

运行绘图代码可以得到如图 5-11 所示的图像。

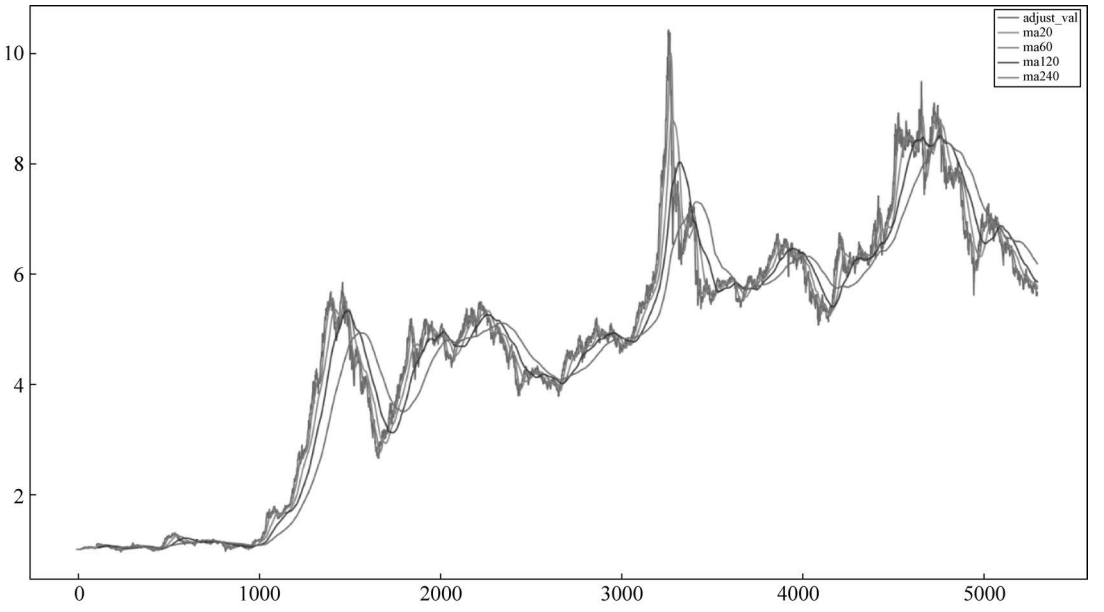


图 5-11　复权净值与均线图像

从图 5-11 可以看出，随着均线周期的增大，均线逐渐平滑（从 20 日均线与 240 日均线

对比更加显著),同时随着周期的增大,均线呈现出的滞后性也越强,即表示的走势特征时间长度越长(从不同均线呈现的极值可以看出),因此短周期均线表示短时间内的走势特征,而长周期均线表示的是长时间内的走势特征,双均线策略基于此进行买入与卖出:当短期均线从下向上穿过长期均线时,说明短时间内的上涨走势强于长时间的上涨走势,因此在交叉的时候应该买入基金,这种均线交叉情形被称作“金叉”,而短期均线从上向下穿过长期均线时,说明短时间内的下跌趋势强于长时间的下跌趋势,则此时应该卖出,这种情形被称作“死叉”。

移动平均值的计算逻辑在 TA-Lib 中已经有现成的实现,使用 SMA 方法即可完成式(5-1)的计算,函数接收名为 timeperiod 的参数表示均值计算的周期。

通过在策略收到行情回调函数中实现“金叉”时买入并持有到“死叉”卖出,代码如下:

```
//ch5/5.3/double_ma_strategy.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""

    author = "ouyangpengcheng"

    #快线周期
    fast_period = 20
    #慢线周期
    slow_period = 120

    parameters = ["fast_period", "slow_period"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = max(self.fast_period, self.slow_period) + 1
        #基类中回看天数为默认一年, 当预取天数大于默认值时, 需要修改为较大值
        self.look_back_size = self.prefetch_days

    def on_init(self) -> None:
        """策略初始化回调"""
        self.prefetch_data(self.prefetch_days)

    def on_start(self) -> None:
        """策略启动回调"""

    def on_stop(self) -> None:
        """策略停止回调"""
        self.send_latest_data()

    def clear_pos(self):
        """清仓"""
        for _symbol in self.pos_symbol_info:
```

```
            self.sell(
                _symbol, abs(
                    self.pos_symbol_info.get(_symbol, {})
                    .get(self.volume_key))
            )

    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]

            #获取历史复权净值
            symbol_adjust_vals = np.asarray(
                    self.symbol_history_adjust_vals(symbol)
        )

            if len(symbol_adjust_vals) >= self.prefetch_days:
                #当复权净值数据量大于预取数据量时，计算均线值
                fast_av = talib.SMA(symbol_adjust_vals, self.fast_period)
                slow_av = talib.SMA(symbol_adjust_vals, self.slow_period)

                fast_0, fast_1 = fast_av[-2], fast_av[-1]
                slow_0, slow_1 = slow_av[-2], slow_av[-1]

                #金叉
                if fast_0 < slow_0 and fast_1 > slow_1:
                    amount = self.available_capital
                    if amount > 0:
                        self.buy(symbol, amount)
                #死叉
                if fast_0 > slow_0 and fast_1 < slow_1:
                    self.clear_pos()
```

使用上述代码中的参数(短周期为 20，长周期为 120)执行回测，可以得到如图 5-12 所示的收益率曲线。

从图 5-12 可以看出，在曲线部分时间以水平的形式呈现，说明此时未有任何持仓，并且这种情形在收益率减小的时候发生，说明触发了策略中的“死叉”执行逻辑，顺利地完成了离场操作。

将策略中的均线计算周期修改为 20 和 60 可以得到如图 5-13 所示的回测结果。

相比于图 5-12 所示的结果，将均线周期减小后的策略表现更差，因为当均线周期减小时，策略会受到更多的短期波动影响而频繁地进行买入和卖出操作，一方面频繁地进行操作容易产生假信号，从而导致错误的持仓操作，另一方面频繁地进行操作也会增加手续费的支出，从而进一步降低收益率。

将均线计算周期分别修改为 120 与 240，再次进行回测可以得到如图 5-14 所示的回测结果。

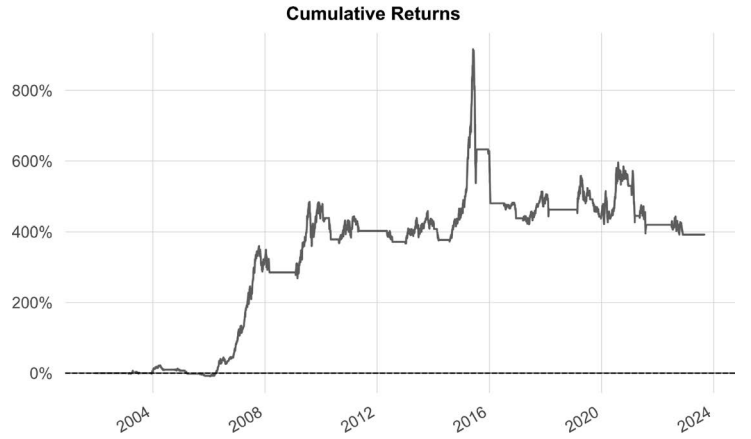


图 5-12　周期为 20/120 的双均线策略收益率曲线

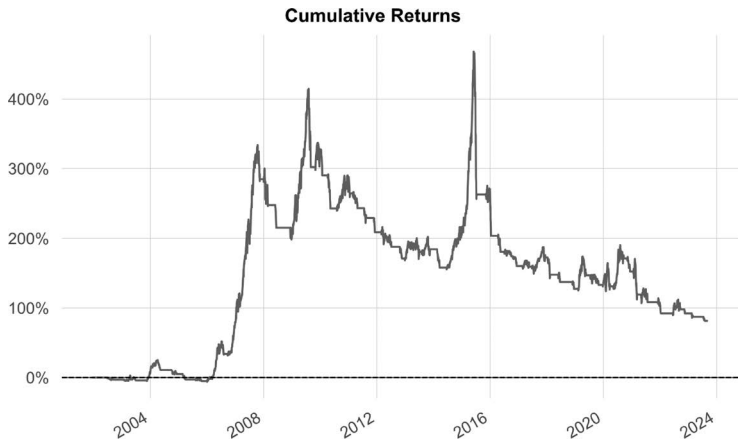


图 5-13　周期为 20/60 的双均线策略收益率曲线

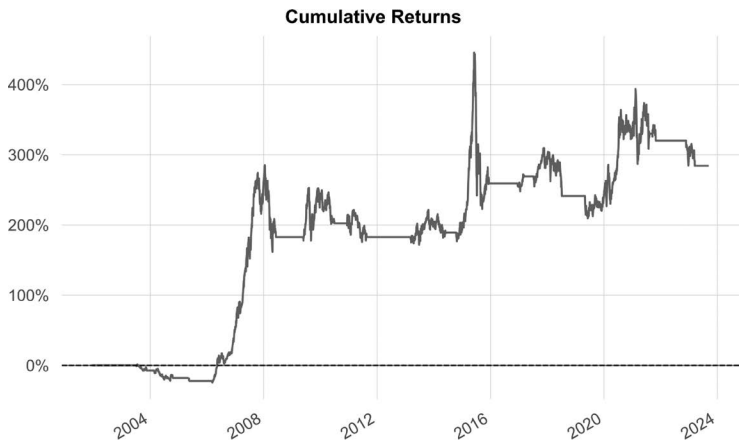


图 5-14　周期为 120/240 的双均线策略收益率曲线

从图 5-14 中可以看出，在均线周期值选取较大时，止损操作会变得十分迟钝，常常在一次较大回撤之后的一段时间才被触发；同理买入操作也会随着大周期值变得迟钝，导致买入与卖出的时机较晚，很难及时捕捉到趋势的启动与停止。

从上面不同参数的选取结果来看，小周期均线值虽然反应迅速，但容易造成频繁操作，受虚假信号的干扰，而大周期均线值反应迟钝，难以及时根据趋势出入场。一种思路是，为了改善小周期值的频繁操作问题，可以在策略中加入最短持仓时间的限制：当根据“金叉”买入后，本次买入需要持仓一定天数之后，判断是否在当日之前已经出现过“死叉”信号，或在持仓天数过后出现了“死叉”信号才能进行卖出。根据上述想法修改策略的代码如下：

```
//ch5/5.3/double_ma_strategy_adj.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""

    author = "ouyangpengcheng"

    #快线周期
    fast_period = 20
    #慢线周期
    slow_period = 120
    #最短持有天数
    min_hold_days = 90

    parameters = ["fast_period", "slow_period", "min_hold_days"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = max(self.fast_period, self.slow_period) + 1
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days
        #持有天数
        self.holding_days = 0
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]

            #获取历史复权净值
            symbol_adjust_vals = np.asarray(
                        self.symbol_history_adjust_vals(symbol)
                )

            if len(symbol_adjust_vals) >= self.prefetch_days:
                #当复权净值数据量大于预取数据量时，计算均线值
                fast_av = talib.SMA(symbol_adjust_vals, self.fast_period)
```

```
slow_av = talib.SMA(symbol_adjust_vals, self.slow_period)

fast_0, fast_1 = fast_av[-2], fast_av[-1]
slow_0, slow_1 = slow_av[-2], slow_av[-1]

#有持仓且还没有到最小持仓时间
if self.pos_symbols and \
        self.holding_days <= self.min_hold_days:
    self.holding_days += 1
    return

#金叉
if fast_0 < slow_0 and fast_1 > slow_1:
    amount = self.available_capital
    if amount > 0:
        self.buy(symbol, amount)
        #重置持有天数
        self.holding_days = 0
#卖出的条件(死叉或到持有期限之前就已经出现死叉)
if fast_1 < slow_1:
    self.clear_pos()
```

使用上述策略代码进行回测可以得到如图 5-15 所示的回测结果，相较于图 5-12 所示的结果，加入持仓最短时间限制之后获得了更高的收益，同时从收益情况来看，加入最短市场时间限制的策略没有影响正常的止损操作。当然在最短持仓时间内可能会损失部分买入的机会，但是当到达最短持仓时间后，策略仍然能够以较高的灵敏度捕捉到上涨趋势而买入基金。

图 5-15　加入持有期限限制的双均线策略收益率曲线

双均线策略思想十分简单，有时是一种“大道至简”的方法，但是对于其参数的选取往往十分困难，读者需要根据自身的投资经验或数理分析选取合适的均线周期。在最简单的双均线的基础上进行改进通常也能获得意想不到的效果，读者应该在理解双均线策略思想的基础上根据参数选取的优缺点进行改进。

5.4 MACD 策略

指数平滑异同平均线(Moving Average Convergence and Divergence,MACD)的核心思想是计算一对长短周期的指数移动平均线 EMA^{short} 与 EMA^{long} 和两者差值的指数移动平均线,然后计算长短周期指数移动平均线的差值的变化关系作为 MACD 值。MACD 的思想与 5.3 节中介绍的双均线策略有一定相似之处,相比之下 MACD 的计算方法更加复杂,并且衡量的是长短周期均线差值的变化情况。首先计算不同长短周期的指数移动平均线,如式(5-2)所示。

$$\begin{aligned} EMA_i^{short} &= EMA_{i-1}^{short} \times \frac{short-1}{short+1} + x_i \times \frac{2}{short+1} \\ EMA_i^{long} &= EMA_{i-1}^{long} \times \frac{long-1}{long+1} + x_i \times \frac{2}{long+1} \end{aligned} \tag{5-2}$$

式(5-2)中,short 和 long 分别为长短周期的值,x_i 为序列中第 i 个元素,接着计算两条指数移动平均线的离差值 DIF:

$$DIF_i = EMA_i^{short} - EMA_i^{long} \tag{5-3}$$

使用式(5-4)计算离差值 DIF 的指数移动平均,其中 dea 为计算周期:

$$DEA_i = DEA_{i-1} \times \frac{dea-1}{dea+1} + DIF_i \times \frac{2}{dea+1} \tag{5-4}$$

最后,MACD 的计算方法为 DIF 与 DEA 之间的离差值的两倍,使用式(5-5)计算:

$$MACD_i = 2 \times (DIF_i - DEA_i) \tag{5-5}$$

从式(5-2)~式(5-5)所示的计算过程来看,指数移动平均值的计算是 MACD 指标的核心,DIF 与 0 值的关系表示短期与长期均线的“金叉”或“死叉”,而 MACD 与 0 值的关系则表示价格长短期价格差值之间与其均线之间的“金叉”或“死叉”,因此对于 MACD 指标值的运用可以类似双均线策略中所介绍的方法,同样可以使用“金叉”与“死叉”进行交易。

MACD 指标的典型参数为 short=12,long=26,dea=9,在 TA-Lib 中计算 MACD 十分简单,使用 talib.MACD 即可,其返回 3 个值:macd、macdsignal 和 macdhist,分别对应上文中提到的 DIF、DEA 和 $\frac{1}{2}$MACD,由于 MACD 值常与 0 值比较,所以前面的常数项对趋势的判断不会产生影响。使用 MACD 的交易策略的代码如下:

```
//ch5/5.4/macd_strategy.py
class MacdStrategy(PortfolioStrategy):
    """MACD 策略"""

    author = "ouyangpengcheng"

    #快线周期
    short_term = 12
```

```
    #慢线周期
    long_term = 26
    #MACD 周期
    macd_term = 9

    parameters = ["short_term", "long_term", "macd_term"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = max(self.short_term, self.long_term, self.macd_term) + 10
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days

    def on_init(self) -> None:
        """策略初始化回调"""
        self.prefetch_data(self.prefetch_days)

    def on_start(self) -> None:
        """策略启动回调"""

    def on_stop(self) -> None:
        """策略停止回调"""
        self.send_latest_data()

    def clear_pos(self):
        """清仓"""
        for _symbol in self.pos_symbol_info:
            self.sell(
                _symbol,
                abs(
                        self.pos_symbol_info.get(_symbol, {})
                        .get(self.volume_key)
                )
            )

    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]

            #获取历史复权净值
            symbol_adjust_vals = np.asarray(
                        self.symbol_history_adjust_vals(symbol)
        )

            if len(symbol_adjust_vals) >= self.prefetch_days:
                dif, dea, macd = talib.MACD(
```

```
            symbol_adjust_vals,
            fastperiod = self.short_term,
            slowperiod = self.long_term,
            signalperiod = self.macd_term,
        )

        # TA-Lib 计算得到的 MACD 值为普遍使用的 MACD 值的一半
        macd *= 2

        # 当 MACD 上穿 0 轴并且 DIF 大于 0 时说明此时为多头走势，应买入
        if (macd[-2] < 0 and macd[-1] > 0) and dif[-1] > 0:
            if not self.pos_symbols:
                amount = self.available_capital
                self.buy(symbol, amount)

        # 当 MACD 下穿 0 轴并且 DIF 小于 0 时说明此时为空头走势，应卖出
        if (macd[-2] > 0 and macd[-1] < 0) and dif[-1] < 0:
            if self.pos_symbols:
                self.clear_pos()
```

在策略收到行情回调函数中，当 MACD 值由负转正并且此时的离差值 DIF 为正时买入基金，当 MACD 由负转正时说明短期的涨势较猛，其相对于长期的涨势优势在扩大或相对于长期走势的劣势在缩小，而 DIF 为正值则说明短期涨势优于长期涨势，因此将两者同时作为判断条件的含义为“短期走势好于长期走势并且短期的优势在不断扩大”，此时买入基金；反之当“短期走势劣于长期走势并且短期的劣势在不断扩大”时则卖出基金。

使用默认参数进行回测可以得到如图 5-16 所示的结果。

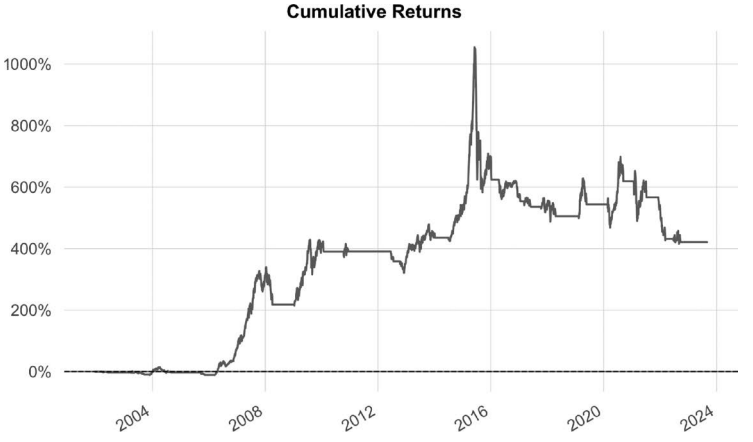


图 5-16　默认参数的 MACD 策略的收益率曲线

从图 5-16 所示的收益率曲线走势来看，基于 MACD 的策略买入卖出的次数较少，收益率曲线中平行于横轴的部分较多。在策略中可以通过凯利公式对买入金额进行控制，凯利公式相关的介绍在 4.5.5 节已经介绍过，使用 QuantStats 可以根据收益率曲线方便地计算出买入金额的比例。为了使用凯利公式，需要在策略运行的过程中记录资产序列(收益率序

列），该信息在策略基类中已经实现，在具体策略中直接使用即可，改写 MACD 策略部分逻辑，如下加粗代码所示。

```
ch5//5.4//macd_strategy_adj.py
class MacdStrategy(PortfolioStrategy):
    """MACD 策略"""
     …
    def __init__(self) -> None:
        super().__init__()
        …
        #记录持仓资产序列
        self.pos_amount_list: list[float] = []
        #是否开始使用凯利公式
        self.kelly_activated = False
     …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            …
            if len(symbol_adjust_vals) >= self.prefetch_days:
                if self.pos_symbols:
                    self.pos_amount_list.append(self.pos_amount)
                …
                # 当 MACD 上穿 0 轴并且 DIF 大于 0 时说明此时为多头走势，应买入
                if (macd[-2] < 0 and macd[-1] > 0) and dif[-1] > 0:
                    if not self.pos_symbols:
                        #没有持仓
                        if self.kelly_activated:
                            #使用凯利公式
                            kelly_ratio = qs.stats.kelly_criterion(
                                pd.Series(self.pos_amount_list)
                            )
                            if kelly_ratio < 0:
                                #如果计算出的比率为负数，则全仓买入
                                kelly_ratio = 1
                        else:
                            #如果不适用凯利公式,则全仓买入
                            kelly_ratio = 1

                        #本次买入的金额
                        amount = self.available_capital * kelly_ratio
                        if amount > 0:
                            self.buy(symbol, amount)

                # 当 MACD 下穿 0 轴并且 DIF 小于 0 时说明此时为空头走势，应卖出
                if (macd[-2] > 0 and macd[-1] < 0) and dif[-1] < 0:
                  if self.pos_symbols:
                      self.clear_pos()
```

```
                # 有一次买入与卖出之后才评价策略表现
                # 在下一次使用凯利公式进行控制
                self.kelly_activated = True
```

由于凯利公式的计算依赖于对已执行的历史决策表现进行评价，所以代码在执行了一次买入与一次卖出操作后才启用凯利公式进行计算，读者也可以在策略表现稳定后再启用凯利公式。使用带有凯利公式计算的 MACD 策略回测结果如图 5-17 所示。

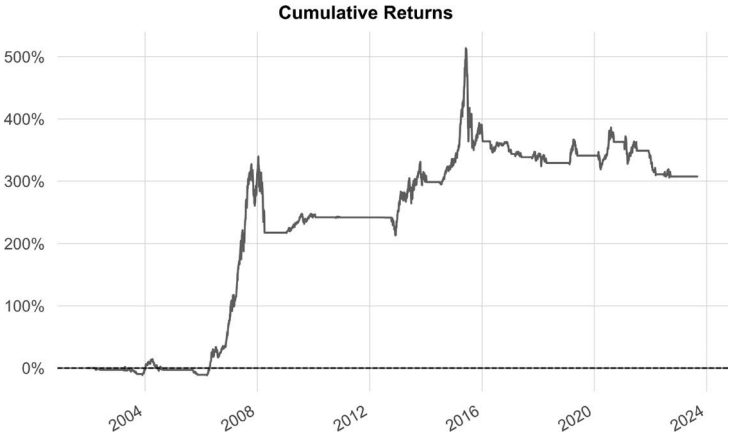


图 5-17　加入凯利公式的 MACD 策略的收益率曲线

相对于未添加凯利公式的回测结果，图 5-17 所示的结果虽然收益低一些，但是其遭受的回撤同样也更小，同时风险回报评价指标的表现更优。除了可以使用“金叉”或“死叉”完成买入与卖出操作，MACD 指标的常用方法还有背离等，读者可以自行编写代码并回测。

5.5　BIAS 策略

BIAS 为乖离率，其表示当前价格距离平均线的偏离程度，计算十分简单，如式(5-6)所示。

$$\mathrm{BIAS}_i = \frac{\mathrm{price}_i - \mathrm{MA}(x, \mathrm{bias})}{\mathrm{MA}(x, \mathrm{bias})} \times 100 \tag{5-6}$$

在式(5-6)中，MA 表示简单移动平均线，bias 为计算移动均线的周期，BIAS 指标衡量的是序列当前值偏离均线的程度，序列波动总会在平均线周围，因此认为 BIAS 值过大时(当前价格相对于均线过高)，市场中多头可能获利了结，从而导致价格下跌，此时也应该卖出，而当 BIAS 值过小时(当前价格相对于均线过低)，市场处于低估值区间，价格将会上涨，因此 BIAS 与均线系统不同，均线是趋势性指标：当均线向上时认为趋势会得到延续继续上涨，而 BIAS 是反转指标，认为走势与 BIAS 值存在反向关系。借助 TA-Lib 可以很简单地计算出移动平均值，进而根据式(5-6)计算出 BIAS 值，代码如下：

```
//ch5/5.5/bias_strategy.py
import talib
import numpy as np

def bias(price, period: int):
    """ 计算乖离率 """
    price = np.asarray(price)
    ma = talib.SMA(price, timeperiod = period)
    last_price = price[ - 1]
    return (last_price - ma[ - 1]) / ma[ - 1] * 100
```

BIAS 指标常用的均线周期值为 6、12 和 24 等，这 3 个周期值对应的操作典型值分别为±5、±7 和±11，当周期为 6 的 BIAS 值小于－5 时买入，而大于 5 时卖出，对于周期值为 12 和 24 的 BIAS 值也是类似的操作逻辑，根据该逻辑可以写出如下的策略代码：

```
//ch5/5.5/bias_strategy.py
…
class BiasStrategy(PortfolioStrategy):
    """BIAS 策略"""

    author = "ouyangpengcheng"

    bias_term1 = 6
    bias1_thresh = 5

    bias_term2 = 12
    bias2_thresh = 7

    bias_term3 = 24
    bias3_thresh = 11

    parameters = ["bias_term1", "bias_term2", "bias_term3"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = max(
                                self.bias_term1,
                                self.bias_term2,
                                self.bias_term3
                                )
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days

    def on_init(self) -> None:
        """策略初始化回调"""
        self.prefetch_data(self.prefetch_days)

    def on_start(self) -> None:
```

```
        """策略启动回调"""

    def on_stop(self) -> None:
        """策略停止回调"""
        self.send_latest_data()

    def clear_pos(self):
        """清仓"""
        for _symbol in self.pos_symbol_info:
            self.sell(
                _symbol,
                abs(self.pos_symbol_info.get(_symbol, {})
                                        .get(self.volume_key))
            )

    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]

            #获取历史复权净值
            symbol_adjust_vals = np.asarray(
                                self.symbol_history_adjust_vals(symbol)
                            )

            if len(symbol_adjust_vals) >= self.prefetch_days:
                bias1 = bias(symbol_adjust_vals, self.bias_term1)
                bias2 = bias(symbol_adjust_vals, self.bias_term2)
                bias3 = bias(symbol_adjust_vals, self.bias_term3)

                #当 3 个 BIAS 值同时发出信号时
                if (
                    bias1 < -self.bias1_thresh
                    and bias2 < -self.bias2_thresh
                    and bias3 < -self.bias3_thresh
                ):
                    amount = self.available_capital
                    if amount > 0:
                        self.buy(symbol, amount)
                elif (
                    bias1 > self.bias1_thresh
                    and bias2 > self.bias2_thresh
                    and bias3 > self.bias3_thresh
                ):
                    self.clear_pos()
```

交易策略的回测结果如图 5-18 所示。

图 5-18　BIAS 策略的收益率曲线

BIAS 策略的回测表现十分一般，读者在实际回测中遇到这种情况应当分析当前策略是否适用于当前的行情走势，或修改策略买卖信号逻辑，使其表现更好。如上的策略代码中使用了逻辑与对信号进行判断，在一定程度上造成了信号的延后与迟钝，将买卖信号的逻辑与换成逻辑或进行回测，代码如下：

```
//ch5/5.5/bias_strategy_adj1.py
class BiasStrategy(PortfolioStrategy):
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
    …
                #某个 BIAS 发出信号
                if (
                    bias1 < -self.bias1_thresh
                    or bias2 < -self.bias2_thresh
                    or bias3 < -self.bias3_thresh
                ):
                    …
                elif (
                    bias1 > self.bias1_thresh
                    or bias2 > self.bias2_thresh
                    or bias3 > self.bias3_thresh
                ):
                    self.clear_pos()
```

得到的回测结果如图 5-19 所示。

从图 5-19 可以看出，修改后的策略表现更差，这说明对于基金 000001 来讲，默认的参数或者信号产生方法不一定适用，通过离线绘制默认参数的乖离率与净值曲线进行观察，代码如下：

```
//ch5/5.5/explore/search_param.py
file_name = "000001.csv"
content = pd.read_csv(file_name)
```

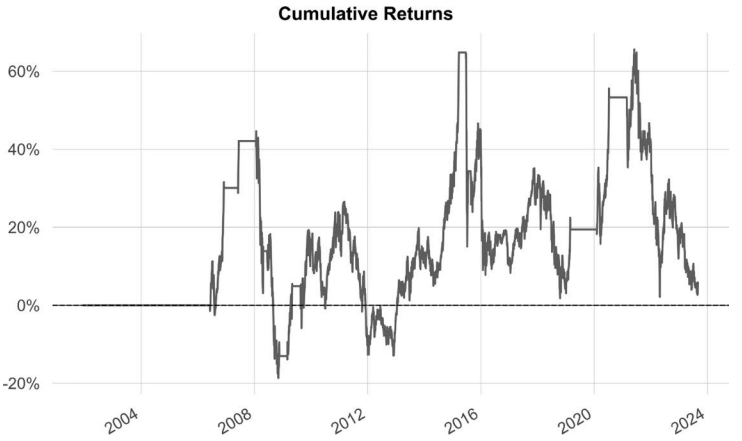


图 5-19　宽松条件的 BIAS 策略收益率曲线

```
#获取复权净值
adjust_val = content["adjust_val"]

#乖离率计算周期
period1 = 6
period2 = 12
period3 = 24


def bias(price, period: int):
    """计算乖离率"""
    …

#记录净值与乖离率的列表
ps = []
b1s = []
b2s = []
b3s = []

for i in range(period3, len(adjust_val)):
    #获取窗口数据
    window_data = adjust_val[i - period3 : i]
    bias1 = bias(window_data, period1)
    bias2 = bias(window_data, period2)
    bias3 = bias(window_data, period3)

    ps.append(adjust_val[i])
    b1s.append(bias1)
    b2s.append(bias2)
    b3s.append(bias3)

#绘制净值曲线
```

```
ax1 = plt.subplot(111)
ax1.plot(ps, label = "vals", linewidth = 2)

#绘制乖离率曲线
ax2 = ax1.twinx()
ax2.plot(b1s, label = "bias1", linestyle = "--", color = "r")
ax2.plot(b2s, label = "bias2", linestyle = "--", color = "g")
ax2.plot(b3s, label = "bias3", linestyle = "--", color = "y")
plt.legend()
plt.show()
```

运行以上代码，可以得到如图 5-20 所示的结果。

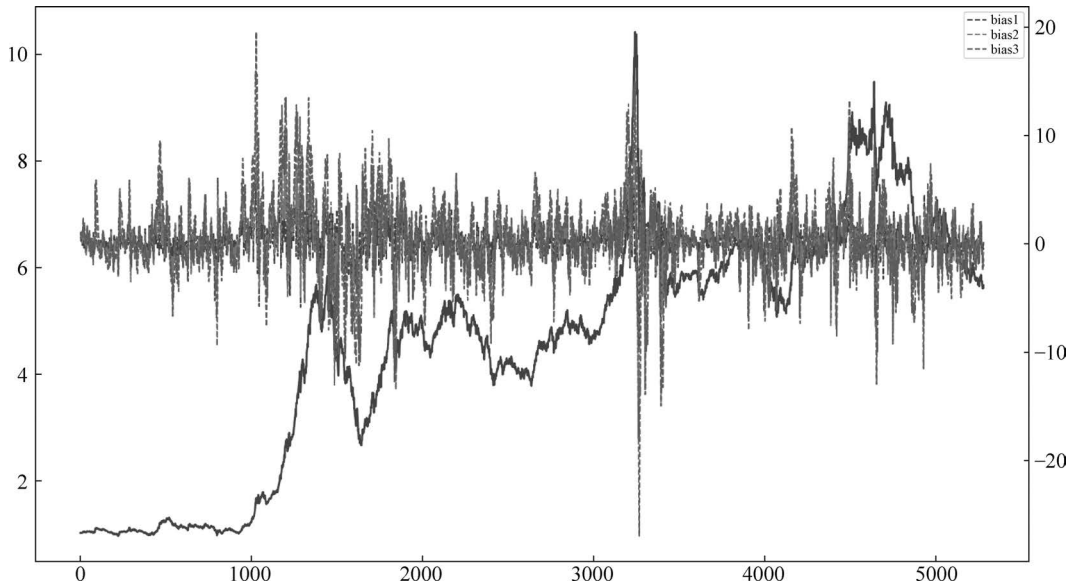


图 5-20　默认参数的 BIAS 曲线与净值曲线变化情况

图 5-20 中可以看到，如果按照反转策略的逻辑，当 BIAS 值较大时卖出，而当该值较小时买入，可以发现会错过大的上涨趋势并且卖出时机过晚，也会造成亏损。对于基金 000001 而言，反转策略的信号可能并不适合交易，可以将 BIAS 信号的使用方法改进为类似“金叉”与“死叉”的趋势性策略方法，在此之前，需要选取一个合适的 BIAS 周期，保证其信号的准确性，绘制不同周期的 BIAS 曲线方法，代码如下：

```
//ch5/5.5/explore/search_param.py
for i in range(1, 10 + 1):
    #计算不同周期
    p = 2 ** i
    ps = []
    bs = []

    for i in range(p, len(adjust_val)):
```

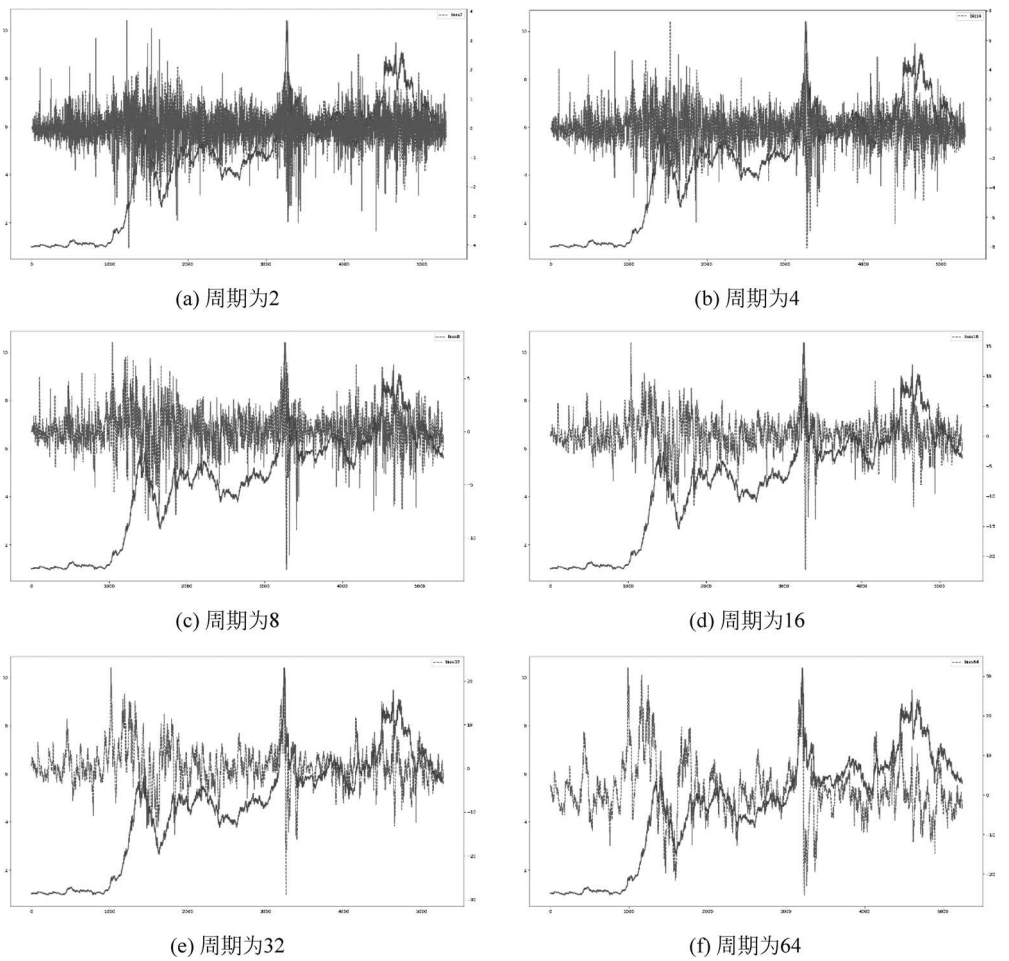
```
        window_data = adjust_val[i - p : i]
        bias_p = bias(window_data, p)

        ps.append(adjust_val[i])
        bs.append(bias_p)

    ax1 = plt.subplot(111)
    ax1.plot(ps, label = "vals", linewidth = 2)

    ax2 = ax1.twinx()
    ax2.plot(bs, label = f"bias{p}", linestyle = " -- ", color = "r")
    plt.legend()
    plt.show()
```

上面的代码以 2 的幂作为周期绘制图像，运行以上代码可以得到如图 5-21 所示的 10 张图像。

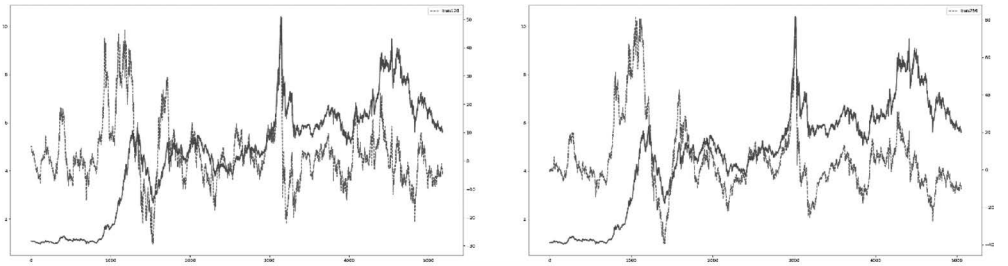


(a) 周期为2　(b) 周期为4

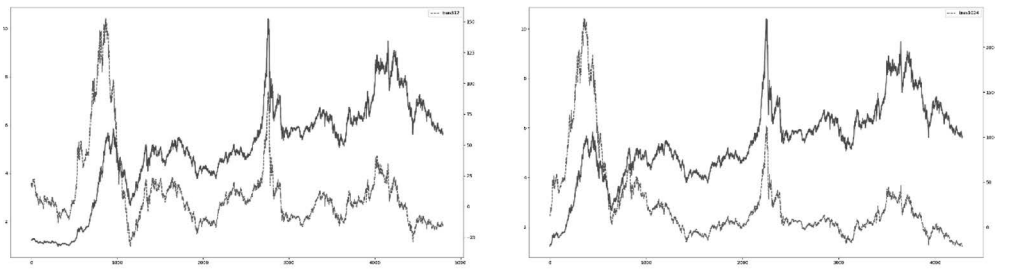
(c) 周期为8　(d) 周期为16

(e) 周期为32　(f) 周期为64

图 5-21　不同周期的 BIAS 曲线与净值曲线走势情况



(g) 周期为128　　(h) 周期为256



(i) 周期为512　　(j) 周期为1024

图 5-21　（续）

从图 5-21 中可以看出，随着 BIAS 的均线周期增大，BIAS 曲线的走势也愈发清晰，而根据大周期的 BIAS 值模仿“金叉”与“死叉”的逻辑也更容易捕捉大趋势。例如对于周期为 1024 的 BIAS 曲线，将“金叉”阈值设置为 25，将“死叉”阈值设置为 100 可以在大趋势中提早布局并适时出场，将 BIAS 指标的使用方法改为趋势性策略写法的代码如下：

```
//ch5/5.5/bias_strategy_adj2.py
…
class BiasStrategy(PortfolioStrategy):
    """BIAS 策略"""

    author = "ouyangpengcheng"

    bias_term1 = 1024
    bias1_thresh_pos = 25
    bias1_thresh_neg = 100

    parameters = ["bias_term1"]

    def __init__(self) -> None:
        super().__init__()
        # 定义预取数据的天数
        self.prefetch_days = self.bias_term1
        # 基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days
        # 记录历史 BIAS 值
        self.bias1_list = []
```

```
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            …
            if len(symbol_adjust_vals) >= self.prefetch_days:
                bias1 = bias(symbol_adjust_vals, self.bias_term1)

                self.bias1_list.append(bias1)
                if len(self.bias1_list) >= 2:
                    if (
                        self.bias1_list[-1] > self.bias1_thresh_pos
                        and self.bias1_list[-2] < self.bias1_thresh_pos
                    ):
                        #BIAS 值金叉
                        amount = self.available_capital
                        if amount > 0:
                            self.buy(symbol, amount)
                    elif (
                        self.bias1_list[-1] < self.bias1_thresh_neg
                        and self.bias1_list[-2] > self.bias1_thresh_neg
                    ):
                        #BIAS 值死叉
                        self.clear_pos()
```

使用上述代码进行回测，可以得到如图 5-22 所示的结果。

图 5-22　使用趋势方法执行 BIAS 信号的策略收益率曲线

从图 5-22 可以看出，使用趋势性策略的信号产生方法后，收益表现优化很大，读者可以在进行参数优化时使用更加精细的粒度进行优化。

5.6　布林带策略

布林带利用标准差衡量窗口内序列的波动情况，并使用均线与二倍标准差绘制出价格的上轨、中轨与下轨，公式如下：

$$\begin{aligned} \text{MIDDLE} &= \text{MA}(x, \text{middle}) \\ \text{UPPER} &= \text{MIDDLE} + 2 \times \text{std}(x) \\ \text{LOWER} &= \text{MIDDLE} - 2 \times \text{std}(x) \end{aligned} \tag{5-7}$$

式(5-7)中最简单的是中轨 MIDDLE 的计算，直接计算序列的移动平均值即可，其中 middle 表示均线的计算周期，而上轨和下轨的计算分别为中轨相加减 2 倍序列的标准差(式中 std)。标准差衡量了近期价格的波动情况，若波动大，则标准差的值也相应偏大，此时价格如果触及上轨，则有可能后期会回调，反之触及下轨，则有可能会出现反弹。总体而言，布林带的默认信号使用方法也属于反转型策略：下跌时买入而上涨时卖出，这与 BIAS 策略的默认使用思想类似。

式(5-7)中均线的周期默认取值为 20，使用 TA-Lib 计算布林带时可以指定均线周期与上下轨的标准差倍数，代码如下：

```
//ch5/5.6/boll_strategy.py
class BollStrategy(PortfolioStrategy):
    """布林带策略"""

    author = "ouyangpengcheng"

    #均线周期
    boll_period = 20
    #上轨标准差倍数
    nbdev_up = 2
    #下轨标准差倍数
    nbdev_down = 2
    #均线类型
    ma_type = 0

    parameters = ["boll_period", "nbdev_up", "nbdev_down", "ma_type"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = self.boll_period + 1
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
```

```
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]

            #获取历史复权净值
            symbol_adjust_vals = np.asarray(
                        self.symbol_history_adjust_vals(symbol)
                )

            if len(symbol_adjust_vals) >= self.prefetch_days:
                upper, middle, lower = talib.BBANDS(
                    symbol_adjust_vals,
                    timeperiod=self.boll_period,
                    nbdevup=self.nbdev_up,
                    nbdevdn=self.nbdev_down,
                    matype=self.ma_type,
                )

                if (
                    symbol_adjust_vals[-2] > lower[-2]
                    and symbol_adjust_vals[-1] < lower[-1]
                ):
                    #当净值下穿下轨时买入
                    amount = self.available_capital
                    if amount > 0:
                        self.buy(symbol, amount)

                if (
                    symbol_adjust_vals[-2] < upper[-2]
                    and symbol_adjust_vals[-1] > upper[-1]
                ):
                    #当净值上穿上轨时卖出
                    self.clear_pos()
```

使用默认参数运行以上代码,可以得到如图 5-23 所示的结果。

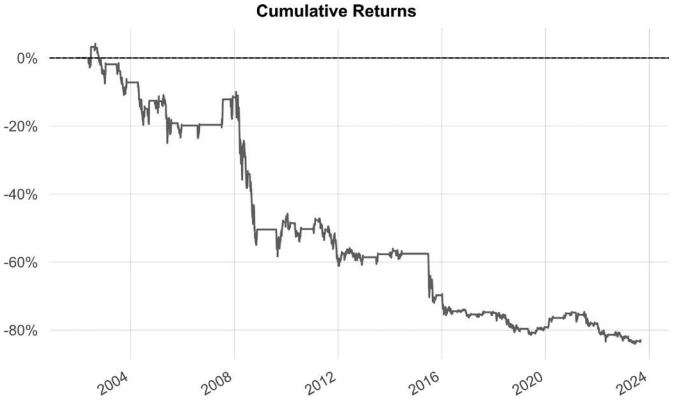


图 5-23 默认参数的布林带策略回测收益率曲线

如图 5-23 所示的收益率说明，默认的布林带策略在回测过程中表现很差，与默认的 BIAS 策略类似，它们都适用于在震荡行情中获利，反转策略的信号使用方法不适用于基金 000001 的回测。类似于对 BIAS 策略的优化方法，可以将布林带策略信号的使用改为趋势类策略的使用方法：当价格下穿下轨时认为行情在加速下跌，此时应该卖出，而当价格上穿上轨时则说明行情在加速上涨，此时应该买入。简单修改默认策略的买卖逻辑即可：

```
//ch5/5.6/boll_strategy_adj1.py
    class BollStrategy(PortfolioStrategy):
    """布林带策略"""
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            …
            if len(symbol_adjust_vals) >= self.prefetch_days:
                …
                if (
                    symbol_adjust_vals[-2] > lower[-2]
                    and symbol_adjust_vals[-1] < lower[-1]
                ):
                    #当净值下穿下轨时卖出
                    self.clear_pos()

                if (
                    symbol_adjust_vals[-2] < upper[-2]
                    and symbol_adjust_vals[-1] > upper[-1]
                ):
                    #当净值上穿上轨时买入
                    amount = self.available_capital
                    if amount > 0:
                        self.buy(symbol, amount)
```

使用趋势类信号处理方法进行回测后，得到的收益率曲线如图 5-24 所示。

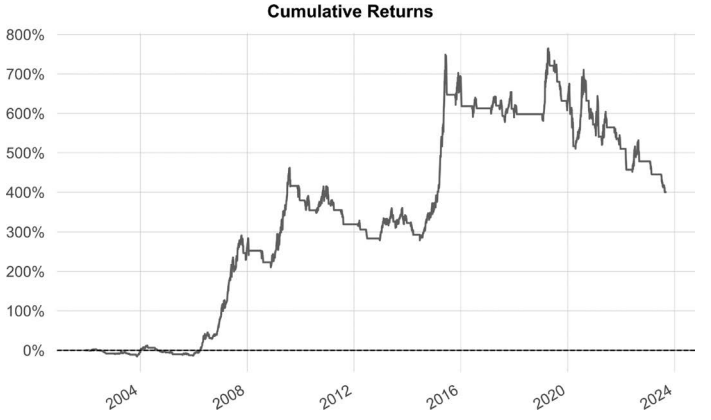


图 5-24　使用趋势类信号处理方法的布林带策略回测收益率曲线

对比图 5-23，将布林带信号转换为趋势类使用方法可以获得较为可观的收益，说明基金 000001 的走势行情适用于趋势类的交易信号。

在基金交易中只存在单方向交易，因此当净值触及下轨卖出时是为了获取收益或止损。相对于基金的买入来讲，卖出需要更加敏感的信号，可以使用不对称的布林带上下轨参数，将下轨标准差倍数改为 1，并且使用类似优化 BIAS 参数的步骤进一步将布林带的均线参数取值优化为 128，修改后的代码如下：

```
//ch5/5.6/boll_strategy_adj2.py
class BollStrategy(PortfolioStrategy):
    """布林带策略"""

    author = "ouyangpengcheng"

    #均线周期
    boll_period = 128
    #上轨标准差倍数
    nbdev_up = 2
    #下轨标准差倍数
    nbdev_down = 1
    ...
```

回测得到的收益率曲线如图 5-25 所示。

图 5-25　优化后使用趋势类信号处理方法的布林带策略回测收益率曲线

相较于未改进的布林带策略，最终优化后的策略版本进一步地获得了更高的收益与夏普比率。在其他基于布林带的策略中，其中轨也会作为信号产生的基准，读者可以尝试并改进其他基于布林带的交易策略。

5.7　网格策略

网格交易策略适用于震荡行情，顾名思义该策略使用了类似渔民捕鱼的思想，使用“渔网”获取一定范围内的利润。网格交易策略通常会指定策略执行的上界与下界，在该界限内

执行“高抛低吸”。上下界的范围内被划分为若干网格，每次当行情上涨到某个网格时执行卖出操作，每次当行情下跌到某个网格时执行买入操作，如图 5-26 所示。

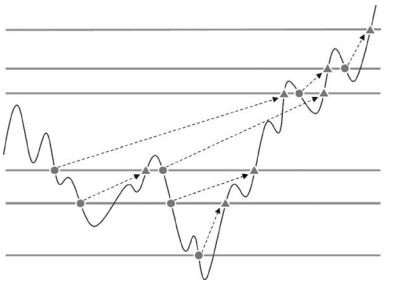


图 5-26　网格交易策略原理

图 5-26 中的行情走势被分为不同的网格，每当价格向下穿过某个网格边界时会买入，并且只有当价格重新上涨到买入网格的上一格时才会平仓，因此只要不是单边趋势，在震荡市使用网格交易策略的每一对交易（一买一卖）的盈利可能性很高。在图 5-26 中，分别使用圆形和三角形表示买卖时机，虚线连接表示成对的交易。

使用网格交易的核心在于网格中枢价的确定，通常在震荡市中以一段时间内价格的最高价与最低价的中间值作为中枢价，当确定网格大小后，使用当前价与中枢价之间的差值决定需要买入或者卖出的份额数，代码如下：

```
//ch5/5.7/grid_strategy.py
class GridStrategy(PortfolioStrategy):
    """网格策略"""

    author = "ouyangpengcheng"

    #网格大小
    grid_size = 0.01
    #网格策略的回看数据长度
    period = 22

    parameters = ["grid_size", "period"]

    def __init__(self) -> None:
        super().__init__()
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.period
        self.base_val = None

    def on_init(self) -> None:
        """策略初始化回调"""

    def on_start(self) -> None:
        """策略启动回调"""

    def on_stop(self) -> None:
        """策略停止回调"""
        self.send_latest_data()

    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
```

```
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]
            val = self.symbol_latest_adjust_val(symbol)
            pos = self.pos_symbol_info.get(symbol, {})
                        .get(self.volume_key, 0)

            if self.base_val is None:
                period_adjust_val = self.symbol_history_adjust_vals(symbol)[
                    -self.period :
                ]
                #使用过去一个月净值最大值与最小值的中间值作为网格中枢价
                self.base_val = (
                    np.max(period_adjust_val) + np.min(period_adjust_val)
                ) / 2

            #计算目标买入量
            buy_direction_steps = (self.base_val - val) / self.grid_size
            buy_volume = buy_direction_steps

            #计算目标卖出量
            sell_direction_steps = (val - self.base_val) / self.grid_size
            sell_volume = sell_direction_steps

            #没有持仓并且当前应该买入的量为非正数
            if buy_volume <= 0 and pos == 0:
                self.base_val = None

            #价格下跌时买入
            if buy_volume > 0:
                buy_amount = min(self.available_capital, buy_volume * val)
                if buy_amount > 0:
                    self.buy(symbol, buy_amount)
            #价格上涨时卖出
            elif sell_volume > 0:
                if pos > 0:
                    if pos <= sell_volume:
                        self.base_val = None
                    self.sell(symbol, min(pos, sell_volume))
```

代码中以每次清仓后的第 1 次买入时间点的前一段时间内的净值最大值和最小值的均值作为网格中枢价，并且在清仓之前该中枢价不发生变化，使用以上代码执行回测可以得到如图 5-27 所示的收益率曲线。

基金 000001 的长期行情总体而言并不在某个固定的区间内震荡，所以策略的表现并不理想，如果能够更多地以趋势的思路使用网格策略，则应该能够获得更好的收益。提高中枢价，甚至使其在近期最高价之上是一种改进网格策略的思路。中枢价提高之后买入的机会更多，并且当价格越低时，根据网格策略的计算方法将会买入更多的份额，此时网格策略演

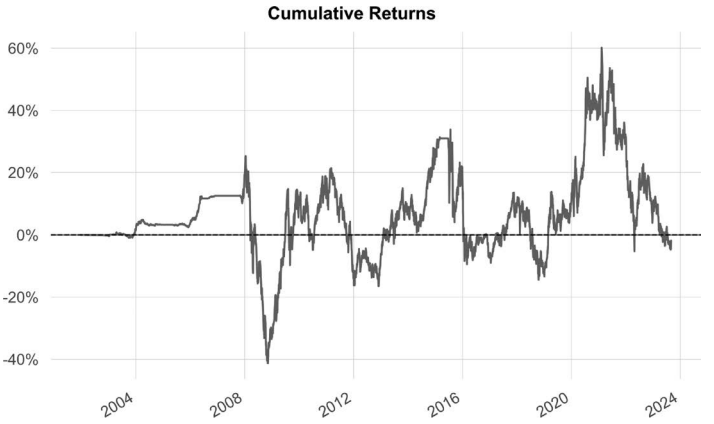


图 5-27　默认网格交易策略的收益率曲线

化成为一种辅助仓位管理的方法，同理当价格上涨至中枢价以上后，距离中枢价越远则会卖出越多，从而获利，改进后的策略代码如下：

```
//ch5/5.7/grid_strategy_adj.py
class GridStrategy(PortfolioStrategy):
    """网格策略"""

    author = "ouyangpengcheng"

    #网格大小
    grid_size = 0.01

    parameters = ["grid_size"]

    def __init__(self) -> None:
        super().__init__()
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = 1
        self.base_val = None
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            symbol = list(fund_data.keys())[0]
            val = self.symbol_latest_adjust_val(symbol)

            if self.base_val is None:
                #如果已经没有持仓，则以当前净值的两倍作为中枢价
                self.base_val = val * 2
    …
```

使用改进后的策略回测可以得到如图 5-28 所示的收益率曲线。

图 5-28 改进中枢价网格交易策略的收益率曲线

与上文预期相同，使用改进后的网格策略后，收益表现相较于图 5-27 有了较大的提升。

5.8 如何改进策略

上文介绍了几种经典的交易策略并介绍了部分策略的改进方法，读者也可以通过各种渠道获取更多优秀的交易策略并学习它们的交易思想与源码。本节将会从不同角度介绍与总结改进交易策略的方法。

5.8.1 选取合适的标的

在 5.1～5.7 节所示的策略中，回测直接使用了基金 000001 的历史数据，而在策略的实际使用过程中对交易标的的选取也十分关键。目前市场上有数以万计的不同基金，可以通过一些启发式规则进行过滤筛选，例如本书选取的条件如下：过往业绩好于同类平均值、基金资产净值在 10 亿元以上、过去三年与五年的晨星评级在 3 星以上，这样可以筛选得到一个仅有几十只基金的小基金池，相对于分析几万只基金的行情，分析经过初筛的小基金池会更有针对性，并且更加容易分析到业绩出色的基金。

当策略收到若干标的行情时，需要从中选取最值得投资的标的，通常需要一个强弱指标来衡量产生信号的强弱。例如对于双均线策略，其买入条件为快速均线上穿慢速均线，当不同标的在同一天产生了买入的“金叉”信号时，此时需要一个指标衡量“金叉”的强度，选择出强度最大的标的买入。

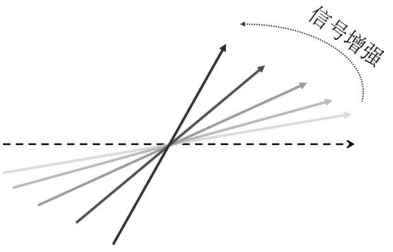


图 5-29 双均线信号强度衡量方法

图 5-29 展示了不同的双均线“金叉”情形，图中虚线表示慢线，不同颜色的实线表示快线上穿慢线的不同情况。

从直观上来讲，当快线以更加“陡峭”的角度上穿慢线时表示短期的上涨趋势更加强烈，因此可以通过快线与慢线的斜率之差表示快线趋势强于慢线的程度。对于某一只基金 i 来讲，可以使用式(5-8)计算“金叉”的程度：

$$\text{crossover\_degree}^i = (\text{fast}_1^i - \text{fast}_0^i) - (\text{slow}_1^i - \text{slow}_0^i) \tag{5-8}$$

式(5-8)中的 fast_0^i 和 fast_1^i 分别表示快线的最后两个值，慢线的字段同理。由于不同基金的净值不同，使用式(5-8)得到的程度值需要归一化后才可比较，将式(5-8)加入归一化可以得到式(5-9)所示的计算规则：

$$\text{crossover\_degree}^i = \frac{\text{fast}_1^i - \text{fast}_0^i}{\text{fast}_0^i} - \frac{\text{slow}_1^i - \text{slow}_0^i}{\text{slow}_0^i} = \frac{\text{fast}_1^i}{\text{fast}_0^i} - \frac{\text{slow}_1^i}{\text{slow}_0^i} \tag{5-9}$$

在收到行情回调函数中，对每个发生“金叉”的标的都计算其程度，同时过滤发生“死叉”的标的，返回“金叉”程度最高的标的与发生“死叉”的标的集合，如果当前持仓中有发生“死叉”的标的，则卖出，如果无持仓，则买入“金叉”程度最高的标的，逻辑代码如下：

```
//ch5/5.8/5.8.1/double_ma_strategy.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""
    …
    def select_symbol(self, fund_data: Dict[str, FundData]) -> str:
        """选取买入和卖出的标的"""
        buy_target_symbol = None
        sell_target_symbols = set()
        max_rise_ratio = -1

        for _symbol in fund_data:
            _symbol_av = np.asarray(
                        self.symbol_history_adjust_vals(_symbol)
            )

            if len(_symbol_av) >= self.prefetch_days:
                #当复权净值数据量大于预取数据量时，计算均线值
                fast_av = talib.SMA(_symbol_av, self.fast_period)
                slow_av = talib.SMA(_symbol_av, self.slow_period)

                fast_0, fast_1 = fast_av[-2], fast_av[-1]
                slow_0, slow_1 = slow_av[-2], slow_av[-1]

                #金叉
                if fast_0 < slow_0 and fast_1 > slow_1:
                    #计算金叉强度
                    relative_rise = fast_1 / fast_0 - slow_1 / slow_0
                    if relative_rise > max_rise_ratio:
                        #记录金叉强度最大的标的
                        max_rise_ratio = relative_rise
                        buy_target_symbol = _symbol
                #死叉
```

```
                if fast_0 > slow_0 and fast_1 < slow_1:
                    #记录发生死叉的标的集合
                    sell_target_symbols.add(_symbol)

        return buy_target_symbol, sell_target_symbols

    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            #获取买入和卖出的标的
            buy_symbol, sell_symbols = self.select_symbol(fund_data)
            symbols_to_sell = sell_symbols & self.pos_symbols
            if symbols_to_sell:
                #如果持仓有选取的卖出标的代码，则卖出
                for _ss in symbols_to_sell:
                    _pos = self.pos_symbol_info
                                .get(_ss, {}).get(self.volume_key)
                    self.sell(_ss, _pos)
            elif buy_symbol and not self.pos_symbols:
                #如果有买入的标的并且无持仓，则买入
                amount = self.available_capital
                if amount > 0:
                    self.buy(buy_symbol, amount)
```

执行回测之后，得到如图 5-30 所示的收益率曲线，相对于图 5-12 所示的表现，选择标的后的双均线策略能够获得更高的回报与夏普比率。

图 5-30　选取标的的双均线交易策略收益率曲线

实际上，交易标的提前初筛的基金池结果也应该是与时间相关的变量，否则可能存在数据泄露的风险。通常来讲，可挑选标的策略会比单标的交易策略的表现更好。

5.8.2　优化策略参数

本书通过将不同 BIAS 的均线周期参数与行情对比图一一绘制寻找最优参数以改进策

略，除此之外还可以使用传统的数值优化方法对参数寻优，本节将介绍使用 Optuna 寻优参数的方法。

为了实现参数寻优，需要对现有的回测框架进行部分修改，目前的框架参数在代码中直接设置，需要改为从外部输入参数的方式完成寻优，因此在初始化回测引擎时需要从外部添加策略的参数并完成其初始化，如加粗代码所示。

```
//ch5/5.8/5.8.2/backtesting.py
class Engine:
    """
    组合策略回测引擎
    """

    batch_days = ANNUAL_DAYS * 2

    def __init__(
        self,
        strategy_cls: "PortfolioStrategy",
        symbols: list[str],
        start: datetime,
        rates: dict[str, float],
        capital: int = 1_000_000,
        end: datetime = None,
        risk_free: float = FIXED_DEPOSIT_5Y,
        strategy_settings: Optional[dict] = None,
    ) -> None:
        """构造函数"""
        …
        #待回测的策略
        self.strategy: "PortfolioStrategy" = strategy_cls(strategy_settings)
        …
```

策略的参数以字典的形式表示，键为参数名称，值为需要设置的参数值。在具体策略收到参数的时候使用 setattr 方法完成参数值的设定，改动部分如加粗代码所示。

```
//ch5/5.8/5.8.2/double_ma_strategy.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""
    …
    def __init__(self, settings: Optional[dict] = None) -> None:
        super().__init__()

        #设置外部传入的参数
        if settings is not None:
            for k, v in settings.items():
                setattr(self, k, v)
    …
```

完成引擎与策略端的改造后，需要定义 Optuna 的参数优化空间与优化目标，引擎的 calculate_statistics 方法会计算并返回盈亏统计量，可以将优化目标设置为最大化期末的资

产总额。对于双均线策略，需要优化的参数包括长期与短期均线周期值，它们都是整型变量，因此使用 Optuna 中的 suggest_int 方法指定二者的优化区间，本书将短期均线的优化空间指定为[2，240]，将长期均线的优化空间指定为[2，480]，当取出的参数非法（短期均线周期大于或等于长期均线周期）时，直接返回一个小值，以便表示非法情形。定义参数优化空间与优化目标的代码如下：

```
//ch5/5.8/5.8.2/optimize.py
def objective(trial):
    """优化目标"""
    #快线参数范围为[2, 240]
    fast_period = trial.suggest_int("short_period", 2, ANNUAL_DAYS)
    #慢线参数范围为[2, 480]
    slow_period = trial.suggest_int("long_period", 2, 2 * ANNUAL_DAYS)

    if fast_period >= slow_period:
        #当快线周期大于或等于慢线周期时，非法参数直接返回 0
        return 0

    #初始化回测引擎
    engine = Engine(
        strategy_cls = DoubleMaStrategy,
        symbols = symbols,
        start = datetime(2001, 1, 1),
        end = datetime(2023, 8, 31),
        rates = rates,
        capital = CAPITAL,
        #从外部注入策略参数
        strategy_settings = {
            "fast_period": fast_period,
            "slow_period": slow_period,
        },
    )

    #1. 运行回测
    engine.run_backtesting()
    #2. 计算回测指标
    result = engine.calculate_statistics()

    #以期末的资产值作为优化目标
    return result.get("end_balance", 0)
```

使用 Optuna 优化目标的代码如下：

```
//ch5/5.8/5.8.2/optimize.py
#定义最大化目标
study = optuna.create_study(direction = "maximize")
study.optimize(objective, n_trials = 100)
#查看最优参数
bp = study.best_params
print(bp)
```

执行优化后，可以得到最优的快速与慢速均线的周期分别为 27 和 159，其回测的收益率曲线如图 5-31 所示。

图 5-31　优化参数后的双均线交易策略收益率曲线

相比于图 5-12，经过参数优化的策略能够获得更好的表现。本书不推荐在对参数优化的时候使用纯数值优化的方式，如在上述过程中优化得到的均线周期并不具备经济意义，读者在优化参数时应该优先从经济意义的角度入手，例如尝试周期使用周线、月线、季线、半年线等典型值，可以使用 Optuna 中的 suggest_categorical 方法指定离散型的优化空间，读者可以在本节代码的基础上修改优化空间定义方式并尝试完成回测。本书更建议读者从实际的经济意义或如 5.5 节中的绘图观察入手以进行参数优化，因此随书的系统代码不包含数值优化功能，如果读者想要使用该功能，则可参考本节的代码修改方式。

5.8.3　设置动态参数

在前几节的策略中，使用的参数都为定值，在回测时间较长、行情结构变化较大的情形下有一定的局限性，因此可以设计一种参数作为对行情自适应调整的方式。考夫曼自适应均线是一种参数自适应调整的方法，它设计了一种价格变化效率的衡量指标，称为效率系数，其使用价格变化除以波动，如式(5-10)所示。

$$\text{efficient\_ratio}=\frac{\text{abs}(x_n-x_1)}{\sum_{i=2}^{n}\text{abs}(x_i-x_{i-1})} \tag{5-10}$$

式(5-10)中 n 为序列长度，abs 表示绝对值函数。分子表示序列终止值与序列初始值之差，分母则表示序列所有波动之和，可以将其分子与分母类比为物理中“位移”与“路程”的区别，前者只关心初始与最终态，而后者是路径依赖的，以两者比值表示序列变化的效率。对仅含有一个元素的序列考查效率系数是没有意义的。当周期内初始值与终止值相同时，无论其在过程中如何变化(序列过程有实际性变化)，效率系数都为 0，而当序列单调变化时，效率系数为 1，容易验证效率系数的值域为[0,1]。考夫曼认为当有行情来临(效率系数

大)时,应该使用较小的均线周期敏锐地捕捉趋势,而当行情震荡时(效率系数小)为了避免过小周期均线频繁地发出错误信号,应该使用较大的均线周期。考夫曼自适应均线接着使用效率系数推算平滑系数,最终得到均线的具体周期。本节不具体介绍考夫曼自适应均线的使用方法,感兴趣的读者可以自行查询相关资料并使用 TA-Lib 实现默认参数的考夫曼自适应均线策略回测。

式(5-10)所示的效率系数提供了一种为策略设置动态参数的思路,并且参数与行情走势是强相关的。式(5-10)中对分子取绝对值后损失了行情的方向信息,对上涨与下跌的行情使用相同的对待方式,本书认为在实际操作中可以对下跌行情使用更小的周期更加敏锐地进行止损,而相对来讲在上涨行情中,则需要更加"稳健"的判断,可以使用相对来讲大一些的周期,因此本书在计算效率系数时会保留行情方向信息,分子不计算绝对值,如式(5-11)所示。

$$\text{efficient\_ratio} = \frac{x_n - x_1}{\sum_{i=2}^{n} \text{abs}(x_i - x_{i-1})} \tag{5-11}$$

式(5-11)所示的效率系数值域相应变为[-1,1],当值小于 0 时表示行情下跌的效率,当值大于 0 时表示行情上涨的效率。由上文分析可知行情强势(效率系数绝对值大)时,周期值应该更小,效率系数的绝对值与周期值呈现反相关的关系,因此需要设计效率系数到周期值调整系数的函数关系。

当行情有下跌迹象(效率系数为小负值)时,应该相对敏感地调整均线周期,在负半轴接近 0 的位置,函数一阶导数更大,函数图像应大致呈现如图 5-32 所示的结果。

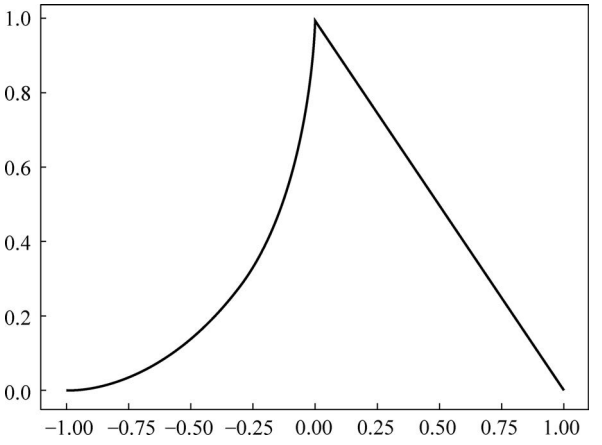


图 5-32 周期调整系数的函数图像

如图 5-32 所示,图像的分段函数如式(5-12)所示。

$$f(x) = \begin{cases} 1 - \sqrt{1 - (x+1)^2}, & -1 \leqslant x < 0 \\ 1, & x = 0 \\ 1 - x, & 0 < x \leqslant 1 \end{cases} \tag{5-12}$$

式(5-12)只是一种变换函数的设计方式，读者也可以尝试更多其他变换函数。从式(5-12)与图 5-32 不难看出，本书使用的变换函数定义域与值域分别为[－1，1]和[0，1]，得到变换后的周期调整系数 ρ 后，使用式(5-13)变换得到均线周期即可：

$$\text{real\_period} = \text{period}_0 + \rho(\text{period}_1 - \text{period}_0) \tag{5-13}$$

式(5-13)中，period_0 和 period_1 分别为预定义的均线取值区间，例如将快速均线的取值区间定义为 [20，60]，那么根据 ρ 的不同取值，实际使用的均线计算周期将在区间 [20，60] 内。下面的代码实现了上述的参数自适应调整逻辑：

```
//ch5/5.8/5.8.3/double_ma_strategy.py
def er(x):
    """涨跌强度评价指标"""
    x = np.asarray(x)
    if len(x) < 2:
        return 0
    return (x[-1] - x[0]) / np.sum(np.abs(np.diff(x)))


def transformer(val):
    """强度指标到均线系数的转换器"""
    if val == 0:
        return 1

    if val > 0:
        return 1 - val
    return 1 - np.sqrt(1 - (val + 1) ** 2)


class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""

    author = "ouyangpengcheng"

    #效率系数评价周期
    er_period = 20
    #快线周期
    fast_interval = (20, 60)
    #慢线周期
    slow_interval = (60, 240)

    parameters = ["er_period", "fast_interval", "slow_interval"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = max(
              self.fast_interval[-1], self.slow_interval[-1]) + 1
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days
    …
```

```
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            ...
            if len(symbol_adjust_vals) >= self.prefetch_days:
                #计算 er 评价指标
                er_val = er(symbol_adjust_vals[-self.er_period:])
                mp = transformer(er_val)

                #计算快速均线周期
                fast_period = self.fast_interval[0] + mp * (
                    self.fast_interval[-1] - self.fast_interval[0]
                )
                #计算慢速均线周期
                slow_period = self.slow_interval[0] + mp * (
                    self.slow_interval[-1] - self.slow_interval[0]
                )

                fast_av = talib.SMA(symbol_adjust_vals, fast_period)
                slow_av = talib.SMA(symbol_adjust_vals, slow_period)

                fast_0, fast_1 = fast_av[-2], fast_av[-1]
                slow_0, slow_1 = slow_av[-2], slow_av[-1]

                #金叉
                if fast_0 < slow_0 and fast_1 > slow_1:
                    amount = self.available_capital
                    if amount > 0:
                        self.buy(symbol, amount)
                #死叉
                if fast_0 > slow_0 and fast_1 < slow_1:
                    self.clear_pos()
```

回测的结果如图 5-33 所示，相较于原始的双均线交易策略，可以看出使用自适应参数的策略可以取得更好的表现。

图 5-33　自适应参数的双均线策略回测收益率曲线

设置自适应参数有多种方法，本节仅列举了其中的一种。将参数自适应化后可能会引入更多的参数（例如本节的方法相对于原始方法多引入了 3 个参数），对于可能造成的策略泛化能力下降问题需要格外注意。

5.8.4　过滤有效信号

在上文提到的所有交易策略中，每次策略信号的产生都是独立的，以双均线为例，当产生“金叉”时就买入，当产生“死叉”时就卖出，各次交易之间没有任何关系，而如果某一次产生“金叉”买入的结果（一买一卖）是亏损的，则策略在下一次出现类似的情形时不应该重复上一次导致亏损的操作。

由于策略只有满仓买入和清仓两种操作，每次卖出操作都对应上一次买入操作，所以记录买入时的均值很容易计算本次交易是否获利。本节为一对操作（一买一卖）定义两种类别：盈利（类别 1）和亏损（类别 0），此时问题转换为一个二分类问题，当产生初步信号时，为模型输入当前市场状态，如果模型给出买入二次确认的信号，则真正买入，否则不执行买入动作。

本节选用 SVM 作为二分类模型，在双均线交易策略的基础上进行改进，SVM 输入的因子包括产生“金叉”时快线增长比例、产生“金叉”时慢线增长比例、产生“金叉”时快线前值落后于慢线前值的比例、产生“金叉”时快线最新值领先于慢线最新值的比例及 5.8.1 节中提到的“金叉”程度值。

由于基金以金额申购，净值在成交时才会得到确认，所以在收到成交的回调函数 on_trade 中记录基金买入时的净值，同样在 on_trade 中收到卖出确认回报时判断本次交易是否盈利，为当前的因子打类别标签（盈利或亏损），并重新使用最新的数据训练 SVM。on_trade 函数的实现代码如下：

```
//ch5/5.8/5.8.4/double_ma_strategy.py
…
def on_trade(self, trade: TradeData):
    """收到成交回报的回调"""
    super().on_trade(trade)

    if trade.direction == Direction.BUY:
        #如果是买入，则记录当前买入的基金净值
        self.last_buy_adjust_val = trade.val
    elif trade.direction == Direction.SELL:
        #如果是卖出，则检查本次交易是否获利
        #并向训练集中添加相应数据
        if trade.val > self.last_buy_adjust_val:
            self.train_set.append((self.last_factor, 1))
        else:
            self.train_set.append((self.last_factor, 0))

        #完成一对交易后重置字段
```

```
        self.last_buy_adjust_val = None
        self.last_factor = None

        #生成训练数据及其标签
        x_train = [x[0] for x in self.train_set]
        y_train = [x[1] for x in self.train_set]

        if len(set(y_train)) > 1:
            #当训练集中有两个类别的数据时才训练模型并激活模型
            self.model.fit(x_train, y_train)
            self.svc_activated = True
…
```

当训练数据不足以训练时(只有一个类别的数据),不使用 SVM 模型辅助确认信号,在每次产生“金叉”信号时记录当前因子,代码如下:

```
//ch5/5.8/5.8.4/double_ma_strategy.py
…
def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
    """收到行情回调"""
    super().on_fund_data(fund_data)
    if self.ready:
        symbol = list(fund_data.keys())[0]

        #获取历史复权净值
        symbol_adjust_vals = np.asarray(
                                self.symbol_history_adjust_vals(symbol)
                            )

        if len(symbol_adjust_vals) >= self.prefetch_days:
            #当复权净值数据量大于预取数据量时,计算均线值
            fast_av = talib.SMA(symbol_adjust_vals, self.fast_period)
            slow_av = talib.SMA(symbol_adjust_vals, self.slow_period)

            fast_0, fast_1 = fast_av[-2], fast_av[-1]
            slow_0, slow_1 = slow_av[-2], slow_av[-1]

            #金叉
            if fast_0 < slow_0 and fast_1 > slow_1:
                #记录当前金叉的因子
                self.last_factor = (
                    #快线增长程度
                    fast_1 / fast_0 - 1,
                    #慢线增长程度
                    slow_1 / slow_0 - 1,
                    #快线起点相对慢线起点的落后程度
                    fast_0 / slow_0,
                    #快线终点相对慢线终点的领先程度
                    fast_1 / slow_1,
```

```
                ＃金叉程度
                fast_1 / fast_0 - slow_1 / slow_0,
            )

            confirmation = True
            if self.svc_activated:
                ＃类别为买入时得到二次确认
                confirmation = self.model.predict(
                                    [self.last_factor]
                                )[0] == 1

             amount = self.available_capital
             if amount > 0 and confirmation:
                 ＃当有可用资金并且金叉信号有效时买入
                 self.buy(symbol, amount)
        ＃死叉
        if fast_0 > slow_0 and fast_1 < slow_1:
            self.clear_pos()
```

在策略初始化时定义好模型所需要的额外变量：

```
//ch5/5.8/5.8.4/double_ma_strategy.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""

    author = "ouyangpengcheng"

    ＃快线周期
    fast_period = 20
    ＃慢线周期
    slow_period = 120

    parameters = ["fast_period", "slow_period"]

    def __init__(self) -> None:
        super().__init__()
        ＃定义预取数据的天数
        self.prefetch_days = max(self.fast_period, self.slow_period) + 1
        ＃基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days

        ＃训练集，格式为(因子，类别)
        self.train_set = []
        ＃上一个因子
        self.last_factor = None
        ＃上一次买入基金时的净值
        self.last_buy_adjust_val = None
        ＃SVC 模型
        self.model = svm.SVC(
```

```
                    C = 10, kernel = "rbf", decision_function_shape = "ovr"
                )
        # SVC 是否被激活
        self.svc_activated = False
...
```

本节的方法通过给模型输入当前信号与市场状态进一步完成交易信号的确认，使用历史业绩作为当前买入的辅助判断依据，只有得到模型二次确认的买入信号才会被执行，有效的信号被过滤，进一步提高了买入的置信度。使用改进后的策略进行回测，表现如图 5-34 所示。

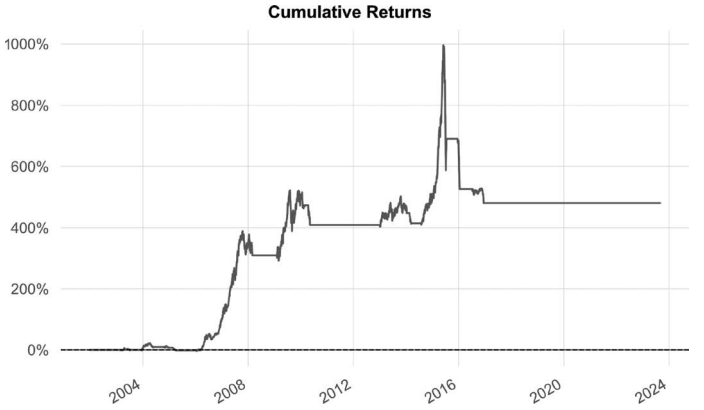


图 5-34　过滤有效信号的双均线策略收益率曲线

相比于原始的回测收益率曲线（见图 5-12），改进后的策略大幅减少了错误的买入行为，在市场内持仓的时间也大幅减少，同时对于回撤的控制要好于原始策略。

5.8.5　管理仓位

策略产生有效的信号固然重要，在投资中对仓位的有效管理更是保住收益的生命线。良好的仓位管理策略符合以下几个特点：①没把握时应轻仓入场，行情不好时轻仓入场，甚至在场外观望；②有把握时重仓入场，大机会来临时重仓买入，甚至加杠杆入场。在 4.5.5 节的第 14 部分介绍的凯利公式正是这样一种管理仓位的工具，通过量化策略的胜率表示策略对行情的把握，控制每次投资的仓位。在 5.4 节中对 MACD 策略使用了凯利公式进行改进，读者可以参考该节中的代码编写方法。

5.8.6　执行多信号协同

在上文介绍过的策略中都是以单指标信号作为买入或卖出的依据进行操作的，事实上可以通过类似 Bagging 的思路将不同指标集成至一个策略中，例如集成双均线与 MACD 信号：当其中任一指标发出买入信号时（不同指标可能捕捉到的是不同周期的交易信号）就执行买入操作，当所有指标都发出卖出信号时（不同周期的行情趋势在此时都发生了改变）就

执行卖出操作。双均线与 MACD 策略在上文的 5.3 节与 5.4 节都详细介绍过，本节只将两者的信号融合即可，代码如下：

```
//ch5/5.8/5.8.6/double_ma_strategy.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""

    author = "ouyangpengcheng"

    #均线参数
    #快线周期
    fast_period = 20
    #慢线周期
    slow_period = 120
    #MACD 参数
    #快线周期
    short_term = 12
    #慢线周期
    long_term = 26
    #MACD 周期
    macd_term = 9

    parameters = ["fast_period", "slow_period"]

    def __init__(self) -> None:
        super().__init__()
        #定义预取数据的天数
        self.prefetch_days = (
            max(
                self.fast_period,
                self.slow_period,
                self.short_term,
                self.long_term,
                self.macd_term,
            )
            + 10
        )
        #基类中回看天数为默认一年，当预取天数大于默认值时，需要修改为较大值
        self.look_back_size = self.prefetch_days
    …
    def on_fund_data(self, fund_data: Dict[str, FundData]) -> None:
        """收到行情回调"""
        super().on_fund_data(fund_data)
        if self.ready:
            …
            if len(symbol_adjust_vals) >= self.prefetch_days:
                #当复权净值数据量大于预取数据量时，计算均线值
                fast_av = talib.SMA(symbol_adjust_vals, self.fast_period)
                slow_av = talib.SMA(symbol_adjust_vals, self.slow_period)
```

```
fast_0, fast_1 = fast_av[-2], fast_av[-1]
slow_0, slow_1 = slow_av[-2], slow_av[-1]

#金叉
ma_buy = fast_0 < slow_0 and fast_1 > slow_1
#死叉
ma_sell = fast_0 > slow_0 and fast_1 < slow_1

dif, dea, macd = talib.MACD(
    symbol_adjust_vals,
    fastperiod=self.short_term,
    slowperiod=self.long_term,
    signalperiod=self.macd_term,
)

#TA-Lib计算得到的MACD值为普遍使用的MACD值的一半
macd *= 2

#当MACD上穿0轴并且DIF大于0时说明此时为多头走势，应买入
macd_buy = (macd[-2] < 0 and macd[-1] > 0) and dif[-1] > 0
#当MACD下穿0轴并且DIF小于0时说明此时为空头走势，应卖出
macd_sell = (macd[-2] > 0 and macd[-1] < 0) and dif[-1] < 0

if ma_buy or macd_buy:
    if not self.pos_symbols:
        amount = self.available_capital
        self.buy(symbol, amount)
if ma_sell and macd_sell:
    if self.pos_symbols:
        self.clear_pos()
```

回测得到的收益率曲线如图 5-35 所示。

图 5-35　双均线与 MACD 信号协同策略的收益率曲线

相对于双均线与 MACD 策略各自的收益率表现，两者协同策略能够获取更高的收益。由于基金 000001 在回测周期内大部分时间处于上涨的状态，为买入基金的时机设置更加宽松的条件（当任一指标发出买入信号时），而为卖出条件设置更加严格的条件能够增加获利的可能性，但如果回测周期内的行情处于震荡或下行时，则需要使用其他信号协同的方法进行组合，本书鼓励读者自行尝试更多信号协同的方式，更多策略集成/协同的思路可以参考笔者的拙作《Python 量化交易实战——使用 vn.py 构建交易系统》。

5.8.7　考虑更多因素

在交易中，除了可以简单地通过低买高卖盈利，还需要考虑更多的因素，只有当一对交易的盈利能够覆盖交易成本时才算真正的盈利，交易成本中最显而易见的部分就是交易手续费。在回测中设置的手续费率为 1%，而由于买入和卖出涉及两次交易，一对交易的手续费率共为 2%，换言之当一对交易赚取的利润超过 2%时才算作真正盈利。

在 5.8.4 节中使用 SVM 作为分类器辅助双均线信号判断时，可以改进标签的判别方法，在打标签的时候将交易手续费率考虑进去，代码如下：

```
//ch5/5.8/5.8.7/double_ma_strategy.py
class DoubleMaStrategy(PortfolioStrategy):
    """双均线策略"""

    author = "ouyangpengcheng"

    #快线周期
    fast_period = 20
    #慢线周期
    slow_period = 120
    #手续费率
    rates = 0.01
    …

    def on_trade(self, trade: TradeData):
        """收到成交回报的回调"""
        super().on_trade(trade)

        if trade.direction == Direction.BUY:
            #如果是买入,则记录当前买入的基金净值
            self.last_buy_adjust_val = trade.val
        elif trade.direction == Direction.SELL:
            #如果是卖出,则检查本次交易是否获利
            #并向训练集中添加相应数据
            if trade.val > self.last_buy_adjust_val * (1 + 2 * self.rates):
                self.train_set.append((self.last_factor, 1))
            else:
                self.train_set.append((self.last_factor, 0))
    …
```

执行回测的结果如图 5-36 所示。

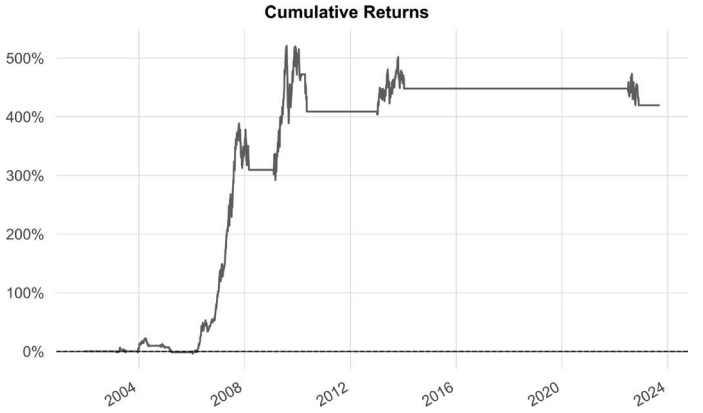


图 5-36　考虑交易成本的双均线策略收益率曲线

从图 5-36 中可以看出，因为考虑了交易成本，所以买入基金盈利的可能性更高，回测整体的夏普比率也更高，但同时由于买入基金变得更加保守，所以造成错过了不少机会。每个交易者应该有各自需要考虑纳入交易的因素，本书列举的交易手续费只是交易成本中通用的一部分，例如还可以考虑盈利相较于负债的情况、持有基金的时间成本等因素。

5.9　小结

5.1～5.7 节介绍了 7 种不同的基金交易策略，有最简单的“买入并持有”和定投策略，也有适用于趋势行情的双均线与 MACD 策略，最后介绍了震荡市常用的 BIAS、布林带和网格策略，读者应该在充分理解了各策略的思想与适用情形后再运用。

5.8 节介绍了一些改进策略的角度，从优化信号、优化参数、优化仓位等方面都可以大幅提升策略的稳健性。本节大多数策略以一个固定的基金作为投资标的，这更有利于读者专注于策略本身的交易逻辑，而在实际投资中常采取类似 5.8.1 节中所述的方法，先从基金池中选取适合交易的若干只(5.8.1 节中选取的是一只)基金，缩小范围后再着重分析并做出相应决策，更多投资组合的内容将在第 6 章中介绍。

读者在学习公开策略之后，应该结合自身状况与投资目标改进或开发交易策略，找到最适合的交易节奏与方法。