

```

view_function = MultiTraceIf.app.route_output(path)
view_function(self.df_ohlc, self.graph_curr, val)
self.graph_attr(**kwargs[key])
plt.show()
finally:
    sys.stdout = oldstdout
def graph_attr(self, **kwargs):
    if 'title' in kwargs.keys():
        self.graph_curr.set_title(kwargs['title'])

    if 'legend' in kwargs.keys():
        self.graph_curr.legend(loc=kwargs['legend'], shadow=True)

    if 'xlabel' in kwargs.keys():
        self.graph_curr.set_xlabel(kwargs['xlabel'])

    self.graph_curr.set_ylabel(kwargs['ylabel'])
    self.graph_curr.set_xlim(0, len(self.df_ohlc.index)) # 设置一下x轴的范围
    self.graph_curr.set_xticks(range(0, len(self.df_ohlc.index), kwargs['xticks'])) #
    x 轴刻度设定 每15天标一个日期

    if 'xticklabels' in kwargs.keys():
        self.graph_curr.set_xticklabels(
            [self.df_ohlc.index.strftime(kwargs['xticklabels'])][index] for index in
            self.graph_curr.get_xticks()]) # 标签设置为日期

    # x-轴每个ticker 标签都向右倾斜45度
    for label in self.graph_curr.xaxis.get_ticklabels():
        label.set_rotation(45)
        label.set_fontsize(10) # 设置标签字体
    else:
        for label in self.graph_curr.xaxis.get_ticklabels():
            label.set_visible(False)

```

9.2 经典择时策略进阶之股票量化交易

股票交易需要做的有3件事：选股、择时和仓位管理。关于择时方面的交易策略，目前市面上有复杂的、也有简单的，种类繁多。其实各种纷繁复杂的量化策略的理论基础是均值回归或趋势追踪。

趋势追踪策略也称为“惯性”或“动量”策略，认为之前价格的上涨预示着之后一段时期内也会上涨。均值回归策略认为股价是围绕着价值在上下波动的，之前股价上涨/下跌只是暂时的，价格随后会下跌/上涨，回归到一个相对正常的水平。

其实股价的涨跌本质是多空双方之间的博弈，选择交易的时机也就是判断多空双方之间抗衡的局势。作为一名股票交易者，制定策略的出发点更多的是为了拨开股价现象看到多空

双方动能的本质，而不是一味地追求复杂的策略机制。

与此同时，在量化交易策略中存在着必不可少的设计环节，包括如何止盈/止损、如何最优化参数、如何管理仓位，如何以可视化方法呈现信号的触发，如何回测分析策略的执行效果等，这些都是在制定量化交易策略时需要考虑到的。

本节选取量化策略中经典的趋势型策略作为案例，介绍制定量化交易择时策略的全过程，以此带领读者从更全面的角度进阶股票量化交易。

本节仍然使用格力电器 2018-06-01 至 2019-06-01 这一年的历史股票行情数据。将 `df_stockload` 以副本形式传入函数中使用，函数中对应的形参定义为 `stock_dat`。

9.2.1 唐奇安通道突破策略的思想

“我们要培养交易者，就像新加坡人养海龟一样。”著名的交易大师理查德·丹尼斯在新加坡时聚精会神地观察着一个海龟农场，突然脱口说出了这样一句话，而著名的“海龟交易试验”正是取名于此。

“海龟交易试验”的起因是理查德·丹尼斯想弄清伟大的交易员是天生造就的还是后天培养的。为此，他在 1983 年招募了 13 个人，教授给他们期货交易的基本概念，以及他自己的交易方法和原则，学员们被称为“海龟”。在随后的 4 年中，“海龟”们取得了年均复利 80% 的收益。

“海龟交易试验”也因此成为了金融史上著名的实验，在实验中运用的“海龟交易法则”非常适合应用于量化分析，以至于在近几年的量化投资热浪中再一次成为热门模式。

“海龟交易法则”具备了一个完整交易系统所应有的所有成分，包括市场、入市、头寸规模、止损/止盈、退出、买卖策略等。其中介绍了一种趋势类的择时策略——唐奇安通道突破策略。

策略的核心思想为：将 N_1 天内最高价构成唐奇安通道的上轨，当天收盘价超过上轨，则认为上升趋势成立，作为买入信号；将 N_2 天内最低价构成唐奇安通道的下轨，当天收盘价低于下轨，则认为下跌趋势成立，作为卖出信号。

也就是说，唐奇安通道上轨突破买入即为 N_1 日创新高买入，当股价创出阶段性新高或历史新高后，一方面说明该股有资金在运作，相对比较强势，更容易顺势而上；另一方面创新高后近期买入的投资者都有获利，上档的套牢盘比较少，股价上冲的阻力也较小，更容易继

续上涨。反之，下轨趋势跌破时卖出的逻辑思维一样成立。

9.2.2 唐奇安通道突破策略的实现

1. 制定唐奇安通道突破策略

创建 `get_ndays_signal()` 函数，该函数的作用是在原股票行情数据基础上增加 '`N1_High`' '`N2_Low`' '`Signal`' 3 列数据。如下所示：

```
print(get_ndays_signal(df_stockload.copy(deep=True))) # 海龟策略-唐奇安通道突破(N日突破) 买入/卖出信号
# 打印结果
"""
      High  Low  Open  ...   N1_High  N2_Low  Signal
Date
2018-06-01  47.3  46.3  47.3  ...    47.3    46.3   -1.0
2018-06-04  48.3  47.0  47.0  ...    48.3    46.3   -1.0
2018-06-05  48.8  47.9  48.0  ...    48.8    46.3    1.0
2018-06-06  48.8  48.1  48.5  ...    48.8    46.3    1.0
2018-06-07  48.9  47.9  48.8  ...    48.9    46.3    1.0
...
2019-05-27  54.5  53.0  54.0  ...    56.6    52.9   -1.0
2019-05-28  55.3  53.8  54.0  ...    56.6    52.9   -1.0
2019-05-29  54.7  53.7  54.1  ...    56.6    52.9   -1.0
2019-05-30  53.8  52.6  53.6  ...    56.6    52.6   -1.0
2019-05-31  53.5  52.2  52.9  ...    56.6    52.2   -1.0
[238 rows x 8 columns]
"""

```

在函数中需完成以下步骤。

(1) 确定 N 日突破策略的参数。关于 N 日突破策略的参数 N_1 、 N_2 的选取，假定我们侧重于中线周期的交易，此处选择 N_1 参数为 15 天， N_2 参数为 5 天，至于参数 N_1 大于 N_2 的原因是为了打造一个非均衡胜负收益的环境，因为我们从事量化交易的目标是要赢大于亏。

(2) 计算股票 N_1 个交易日的滚动最大值。此处使用 `df.rolling().max()` 这种方式，只需提供最高价和移动时间窗口大小即可，对使用者来说是非常方便快捷的，如下所示：

```
stock_dat['N1_High'] = stock_dat.High.rolling(window=N1).max() # 计算最近N1个交易日最高价
```

(3) 由于是从第 N_1 天开始滚动计算该周期内的最大值，因此前 N_1 个数值都为 `NaN`，我们用前 N_1 个交易日滚动最大值来填充 `NaN`，如下所示：

```

stock_dat['N1_High'] = stock_dat.High.rolling(window=N1).max() # 计算最近N1个交易日最高价
expan_max = stock_dat.High.expanding().max() # 滚动计算当前交易日为止的最大值
stock_dat['N1_High'].fillna(value=expan_max,inplace=True) # 填充前N1个nan
print(stock_dat.head())
# 打印结果
"""
      High  Low  Open  Close  Volume  N1_High
Date
2018-06-01  47.3  46.3  47.3  46.6  5.0e+05    47.3
2018-06-04  48.3  47.0  47.0  47.8  1.0e+06    48.3
2018-06-05  48.8  47.9  48.0  48.5  1.0e+06    48.8
2018-06-06  48.8  48.1  48.5  48.4  5.5e+05    48.8
2018-06-07  48.9  47.9  48.8  48.0  5.6e+05    48.9
"""

```

(4) 同理，计算股票 N_2 个交易日的滚动最小值。如下所示：

```

stock_dat['N2_Low'] = stock_dat.Low.rolling(window=N2).min() # 计算最近N2个交易日最低价
expan_min = stock_dat.Low.expanding().min()
stock_dat['N2_Low'].fillna(value=expan_min,inplace=True) # 目前出现过的最小值填充前N2个nan
print(stock_dat.head())
# 打印结果
"""
      High  Low  Open  ...  Volume  N1_High  N2_Low
Date
2018-06-01  47.3  46.3  47.3  ...  5.0e+05    47.3   46.3
2018-06-04  48.3  47.0  47.0  ...  1.0e+06    48.3   46.3
2018-06-05  48.8  47.9  48.0  ...  1.0e+06    48.8   46.3
2018-06-06  48.8  48.1  48.5  ...  5.5e+05    48.8   46.3
2018-06-07  48.9  47.9  48.8  ...  5.6e+05    48.9   46.3
[5 rows x 7 columns]
"""

```

(5) 根据突破定义构建买卖信号。在当天的收盘价超过 N_1 天内最高价时，给出买入股票信号；在当天的收盘价跌破 N_2 天内最低价时，给出卖出股票信号。接下来选取符合买入条件的时间序列 buy_index，以及符合卖出条件的时间序列 sell_index。用 Dataframe 矢量化处理数据的方式，仅需简单的几行代码就可以实现，如下所示：

```

# 收盘价超过N1最高价 买入股票
buy_index = stock_dat[stock_dat.Close > stock_dat.N1_High.shift(1)].index
# 收盘价超过N2最低价 卖出股票
sell_index = stock_dat[stock_dat.Close < stock_dat.N2_Low.shift(1)].index

```

此处 shift(1) 的作用是在 index 不变的情况下对序列的值向右移动一个单位，这么做的目的是获取昨天为止的最高/最低价格，表示当日收盘价突破昨日为止的最高/最低价格时买入/卖出股票。

(6) 构建新的序列 Signal, 表示触发的突破信号。在符合买入/卖出条件的时间序列下, 将买入当天的 Signal 值设置为 1, 代表买入, 将卖出当天的 Signal 设置为 -1, 代表卖出。

```
stock_dat.loc[buy_index, 'Signal'] = 1
stock_dat.loc[sell_index, 'Signal'] = -1
```

由于未涉及仓位管理的方法, 此处设定为一旦触发买入信号则全仓买入, 一旦卖出信号触发则全仓卖出。因此在第一个信号触发后, 由于是全仓买入或全仓卖出, 即使后续仍有信号发出也不执行, 也就是说连续的信号只有第一个有实际的操作意义。

(7) 由于收盘价格是在收盘后才确定, 那么第二天才能执行给出的买卖操作, 此处将 signal 序列使用 shift(1) 方法右移更接近真实情况, 代码如下所示:

```
stock_dat['Signal'] = stock_dat.Signal.shift(1)
```

(8) 对于 Signal 序列中的 NaN 值, 使用 fillna() 方法将所有 NaN 值与前面元素值保持一致, 即参数 method='ffill', 这样符合一旦状态被设置为 1 (买入持有), 只有遇到 -1 (卖出空仓) 时 Signal 状态才会改变, 代码如下所示:

```
stock_dat['Signal'].fillna(method = 'ffill', inplace = True) # 与前面元素值保持一致
```

(9) 对于 Signal 序列最前面几个 NaN 值并不起作用, 此时需要再一次使用 fillna() 方法, 选择用 -1 值填充序列最前面的几个 NaN 值, 代码如下所示:

```
stock_dat['Signal'].fillna(value = -1, inplace = True) # 序列最前面几个 NaN 值用-1 填充
```

完整的唐奇安通道突破策略的实现代码, 如下所示:

```
def get_ndays_signal(stock_dat, N1 = 15, N2 = 5):
    # 海龟策略-唐奇安通道突破(N日突破) 买入/卖出信号
    stock_dat['N1_High'] = stock_dat.High.rolling(window=N1).max() # 计算最近N1个交易日最高价
    expan_max = stock_dat.High.expanding().max() # 滚动计算当前交易日为止的最大值
    stock_dat['N1_High'].fillna(value=expan_max,inplace=True) # 填充前N1个nan
    stock_dat['N2_Low'] = stock_dat.Low.rolling(window=N2).min() # 计算最近N2个交易日最低价
    expan_min = stock_dat.Low.expanding().min()
    stock_dat['N2_Low'].fillna(value=expan_min,inplace=True) # 目前出现过的最小值填充前N2个nan
    # 收盘价超过N1最高价, 买入股票
    buy_index = stock_dat[stock_dat.Close > stock_dat.N1_High.shift(1)].index
    # 收盘价超过N2最低价, 卖出股票
    sell_index = stock_dat[stock_dat.Close < stock_dat.N2_Low.shift(1)].index
    stock_dat.loc[buy_index, 'Signal'] = 1
    stock_dat.loc[sell_index, 'Signal'] = -1
    stock_dat['Signal'] = stock_dat.Signal.shift(1)
    stock_dat['Signal'].fillna(method = 'ffill', inplace = True) # 与前面元素值保持一致
    stock_dat['Signal'].fillna(value = -1, inplace = True) # 序列最前面几个NaN值用-1填充
    return stock_dat
```

2. 可视化唐奇安通道突破策略

在 K 线图上绘制唐奇安通道以及突破通道的指示信号，如图 9.10 所示。代码如下所示：

```

def draw_ndays_annotate(stock_dat):
    # 绘制唐奇安通道突破/N 日突破
    signal_shift = stock_dat.Signal.shift(1)
    signal_shift.fillna(value=-1, inplace=True) # 序列最前面的 NaN 值用-1 填充
    list_signal = np.sign(stock_dat.Signal - signal_shift) # 计算买卖点

    down_cross = stock_dat[list_signal < 0]
    up_cross = stock_dat[list_signal > 0]

    layout_dict = {'figsize': (14, 7),
                   'index': stock_dat.index,
                   'draw_kind': ('ochl':
                                  {'Open': stock_dat.Open,
                                   'Close': stock_dat.Close,
                                   'High': stock_dat.High,
                                   'Low': stock_dat.Low
                                  },
                                  'line':
                                  {'N1_High': stock_dat.N1_High,
                                   'N2_Low': stock_dat.N2_Low
                                  }),
                   'annotate':
                   (u'down':
                    {'andata': down_cross,
                     'va': 'top',
                     'xy_y': 'N2_Low',
                     'xytext': (-10, -stock_dat['Close'].mean()),
                     'fontsize': 8,
                     'arrow': dict(facecolor='green', shrink=0.1)
                    },
                    u'up':
                    {'andata': up_cross,
                     'va': 'bottom',
                     'xy_y': 'N1_High',
                     'xytext': (-10, stock_dat['Close'].mean()),
                     'fontsize': 8,
                     'arrow': dict(facecolor='red', shrink=0.1)
                    }
                   )
                  },
    'title': u"000651 格力电器-唐奇安通道突破",
    'ylabel': u"价格",
    'xlabel': u"日期",
    'xticks': 15,
    'legend': u'best',
    'xticklabels': '%Y-%m-%d'}
    app.fig_output(**layout_dict)

```

观察是否能够改善交易过程。

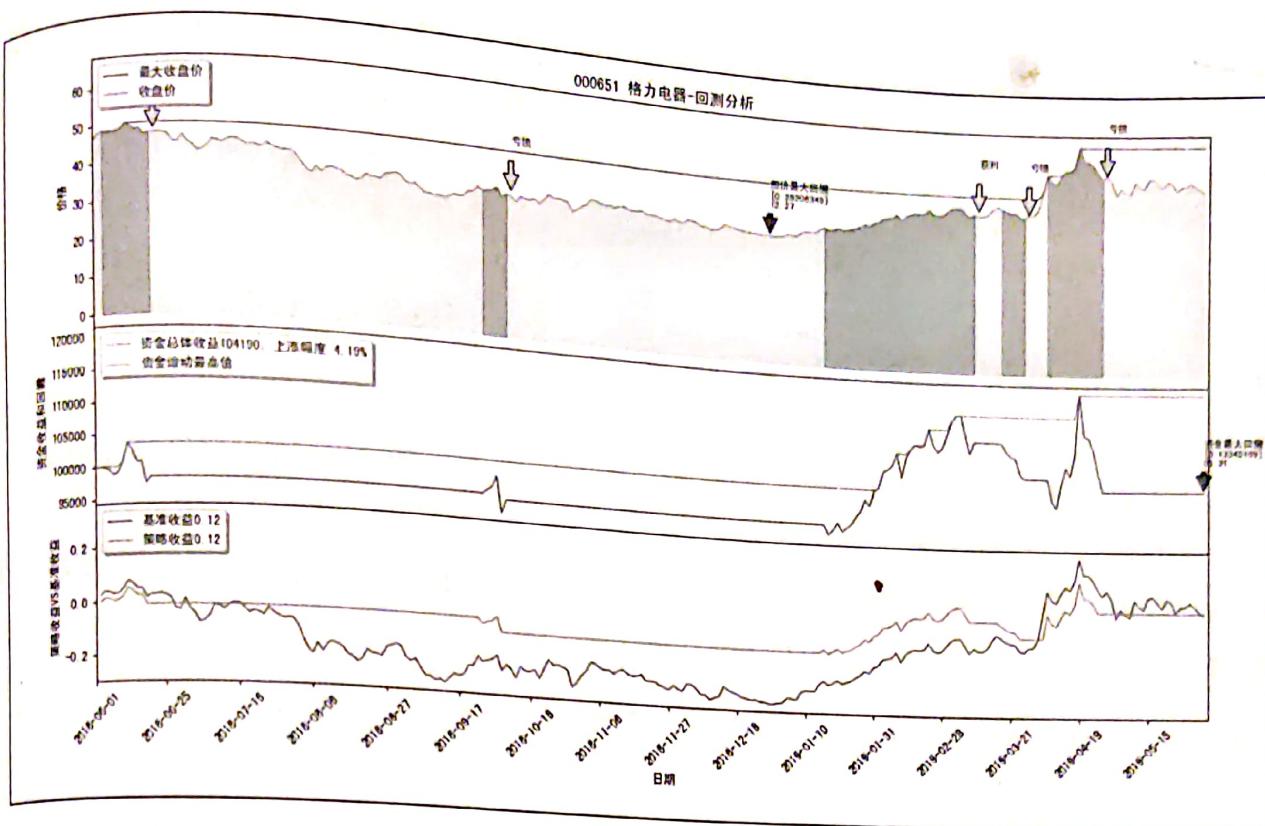


图 9.11

logtrade.txt						
	BuyTime	SellTime	BuyPrice	SellPrice	DiffPrice	PctProfit
0	18.06.05	18.06.20	48.5	47.9	-0.7	-1.4
1	18.09.25	18.10.09	39.0	38.8	-0.2	-0.6
2	19.01.16	19.03.11	39.3	45.4	6.1	15.4
3	19.03.19	19.03.26	47.2	44.8	-2.4	-5.1
4	19.04.10	19.04.26	57.1	56.2	-0.9	-1.6

亏损次数:4, 盈利次数:1, 胜率:20.0%
 平均亏损:-2.17% 平均盈利:15.42%
 股价最大回撤29.21% 从2018-06-12 00:00:00开始至2018-12-27 00:00:00结束
 资金最大回撤13.35% 从2019-04-19 00:00:00开始至2019-05-31 00:00:00结束
 资金总体收益104190; 上涨幅度 4.19%
 资金相对收益: 策略收益0.12 VS 基准收益0.12

图 9.12

9.3 融入 ATR 跟踪止盈/止损策略

唐奇安通道突破择时策略的核心思想为：当天收盘价超过 N1 天内最高价，认为上升趋势成立，作为买入信号；当天收盘价低于 N2 天内最低价格，认为下跌趋势成立，作为卖出信号。

我们发现该策略存在的问题是：当买入股票后，在趋势上涨时并不会出现卖点，而是当趋势下跌成立后才出现卖点，此时已经回撤了较大部分盈利；当策略判断错误而给出买入信号时，股价出现不断下跌的走势，如果直到策略给出卖点信号才卖出股票，此时已经亏损较

多资金。

在择时策略中通常会融入多个因子协同触发信号，对此我们可以在唐奇安通道突破择时策略的基础上引入止盈/止损策略因子，既能使自己的盈利最大化，又能有效规避市场风险。跟踪止盈/止损策略指根据当前价格上涨或下跌幅度，动态地调节止损/止盈位置。本节介绍如何融入基于 ATR 基准值的跟踪止盈/止损策略，根据行情的波动幅度自适应调节并跟踪止盈/止损通道。

9.3.1 ATR 技术指标的实现

技术指标大体分为趋势型和震荡型两类。唐奇安通道突破、双均线突破属于趋势型指标，趋势型指标背后的逻辑是假设之前价格上涨预示着之后一段时间内仍然会上涨。而震荡型指标侧重于波动幅度的分析，例如之前介绍的 KDJ 指标，本节介绍的 ATR 指标也属于震荡型指标。

ATR (Average True Range) 又称平均真实波动范围，由 J.Welles Wilder 所发明。ATR 指标的计算分为以下两步。

- 第一步：计算真实波幅 TR。TR 指的是今日振幅、今日最高价与昨日收盘价之间的波幅、昨日收盘价与今日最低价之间的波幅，取这三者之中的最大值，即 $TR=MAX[(\text{当日最高价}-\text{当日最低价}), \text{abs}(\text{当日最高价}-\text{昨日收盘价}), \text{abs}(\text{昨日收盘价}-\text{当日最低价})]$ 。
- 第二步：对真实波幅 TR 进行 N 日移动平均计算。 $ATR=MA(TR,N)$ ，常用参数 N 为 14 日或 21 日。

ATR 指标主要用来衡量市场波动的强烈程度，用于反应市场变化率的指标。较低的 ATR 值表示市场交易气氛比较冷清，而较高的 ATR 则表示市场交易气氛比较旺盛，较低的 ATR 或较高的 ATR 都可以看作价格趋势的反转，例如股价横盘整理预示着变盘，股价波幅加剧表明有主力资金进出。

使用 TA-Lib 库中的 `talib.ATR()` 接口函数可以直接计算得到 ATR 波动幅度序列。使用时需要输入 `numpy.ndarray` 类型的最高价、最低价和收盘价序列，以及指定 ATR 移动平均参数 N。

接下来以格力电器 2018-6-1 至 2019-6-1 这一年的历史股票行情数据为例程数据，分别计算 ATR14 和 ATR21 指标序列，并且使用可视化接口绘制 ATR 指标。绘制效果如图 9.13 所示，完整的代码如下所示：

```

def draw_atr_chart(stock_dat):
    stock_dat['atr14'] = talib.ATR(stock_dat.High.values, stock_dat.Low.values, stock_dat.
close.values, timeperiod=14) # 计算 ATR14
    stock_dat['atr21'] = talib.ATR(stock_dat.High.values, stock_dat.Low.values, stock_dat.
close.values, timeperiod=21) # 计算 ATR21
    layout_dict = {'figsize': (14, 5),
                  'index': stock_dat.index,
                  'draw_kind': {'line':
                                 {'atr14': stock_dat.atr14,
                                  'atr21': stock_dat.atr21
                                 },
                               },
                  'title': u"000651 格力电器-ATR",
                  'ylabel': u"波动幅度",
                  'xlabel': u"日期",
                  'xticks': 15,
                  'legend': u'best',
                  'xticklabels': '%Y-%m-%d') # strftime
    app.fig_output(**layout_dict)

draw_atr_chart(df_stockload.copy(deep=True))

```

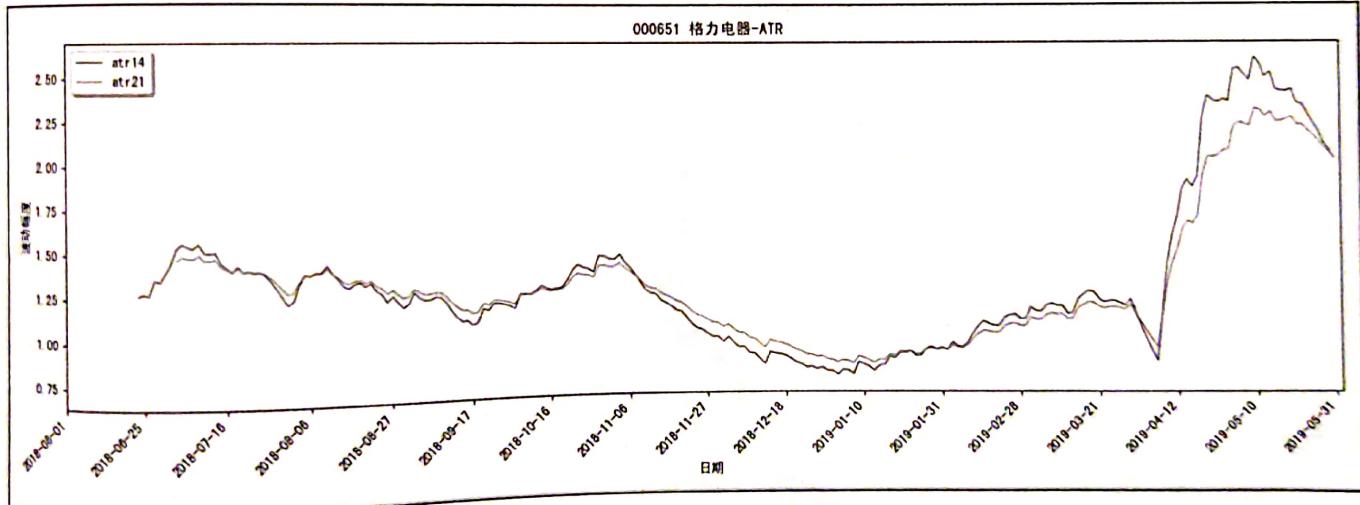


图 9.13

9.3.2 止盈/止损策略的实现

关于止盈/止损的实现,将 ATR 值作为止盈/止损的基准值,止盈值设置为 n_win 倍的 ATR 值,止损值设置为 n_loss 倍的 ATR 值, n_win 和 n_loss 分别为最大止盈系数和最大止损系数,

此处设置最大止盈系数为 3.5，最大止损系数为 1.8，倾向于盈利值要大于亏损值。触发止盈/止损条件为：

- 当 $n_win \times ATR$ 值 < (今日收盘价格 - 买入价格)，触发止盈信号，卖出股票；
- 当 $n_loss \times ATR$ 值 < (买入价格 - 今日收盘价格)，触发止损信号，卖出股票。

接下来我们在 9.2 节 get_ndays_signal() 函数的基础上融入 ATR 止盈/止损策略因子的判断。将买入价存储于变量 buy_price 之中，而后与每个交易日的收盘价对比，无论是到达止盈价格还是止损价格都会触发卖出信号，当卖出后将变量 buy_price 清零，如下所示：

```

def get_ndays_atr_signal(stock_dat, N1=15, N2=5, n_win=3.5, n_loss=1.8):
    # 海龟策略-唐奇安通道突破(N 日突破) 买入/卖出信号
    stock_dat['N1_High'] = stock_dat.High.rolling(window=N1).max() # 计算最近N1个交易日最高价
    expan_max = stock_dat.High.expanding().max() # 滚动计算当前交易日为止的最大值
    stock_dat['N1_High'].fillna(value=expan_max, inplace=True) # 填充前N1个nan
    stock_dat['N2_Low'] = stock_dat.Low.rolling(window=N2).min() # 计算最近N2个交易日最低价
    expan_min = stock_dat.Low.expanding().min()
    stock_dat['N2_Low'].fillna(value=expan_min, inplace=True) # 目前出现过的最小值填充前N2个
    nan

    stock_dat['ATR21'] = talib.ATR(stock_dat.High.values, stock_dat.Low.values, stock_dat.
Close.values, timeperiod=21) # 计算ATR21
    # 收盘价超过N1 最高价 买入股票
    buy_index = stock_dat[stock_dat.Close > stock_dat.N1_High.shift(1)].index
    # 收盘价超过N2 最低价 卖出股票
    sell_index = stock_dat[stock_dat.Close < stock_dat.N2_Low.shift(1)].index
    stock_dat.loc[buy_index, 'Signal'] = 1
    stock_dat.loc[sell_index, 'Signal'] = -1
    stock_dat['Signal'] = stock_dat.signal.shift(1)
    buy_price = 0

    for kl_index, today in stock_dat.iterrows():
        if (buy_price == 0) and (today.Signal == 1):
            buy_price = today.Close
            # 到达收盘价少于买入价后触发卖出
        elif (buy_price != 0) and (buy_price > today.Close) and ((buy_price - today.Close) > n_loss * today.ATR21):
            print(f'止损时间:{kl_index.strftime("%Y.%m.%d")}, 止损价格:{round((today.Close, 2))}')
            stock_dat.loc[kl_index, 'Signal'] = -1
            buy_price = 0
            # 到达收盘价多于买入价后触发卖出

```

```

    elif (buy_price != 0) and (buy_price < today.Close) and ((today.Close - buy_price)
> n_win * today.ATR21):
        print(f'止盈时间:{kl_index.strftime("%Y.%m.%d")}, 止盈价格:{round(today.Close,
2)}')
        stock_dat.loc[kl_index, 'Signal'] = -1
        buy_price = 0
    elif (buy_price != 0) and (today.Signal == -1):
        stock_dat.loc[kl_index, 'Signal'] = -1
        buy_price = 0
    else:
        pass
stock_dat['Signal'].fillna(method='ffill', inplace=True) # 与前面元素值保持一致
stock_dat['Signal'].fillna(value=-1, inplace=True) # 序列最前面几个 NaN 值用 0 填充
return stock_dat

```

9.3.3 ATR 止盈/止损策略回测

接下来对格力电器的历史行情数据进行回测，评估在海龟交易法则唐奇安策略基础上融入 ATR 止盈/止损策略后的效果如何，可以直接调用 9.1.6 节的回测分析界面。对应的回测分析可视化效果和回测评估指标分别如图 9.14 和图 9.15 所示。

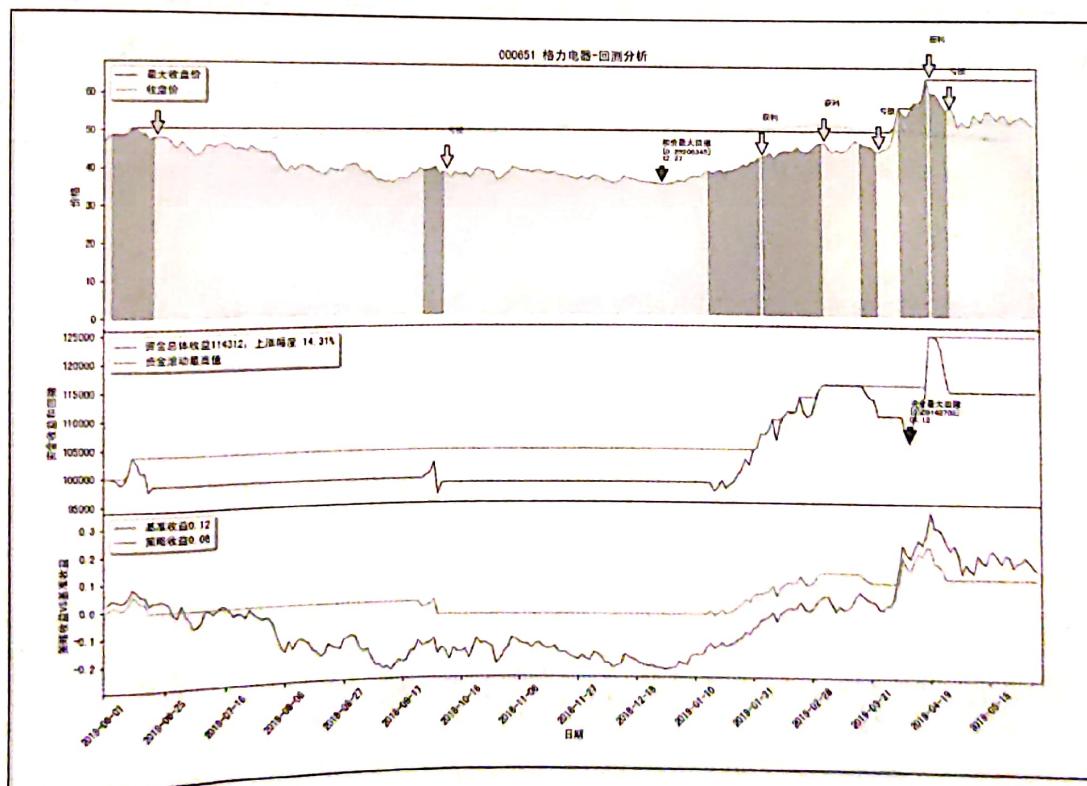


图 9.14

	BuyTime	SellTime	BuyPrice	SellPrice	DifffPrice	PctProfit
0	18.06.01	18.06.20	48.1	47.4	-0.7	-1.4
1	18.09.25	18.10.09	39.0	38.0	-0.2	-0.6
2	19.01.16	19.02.11	39.3	43.1	3.8	9.6
3	19.02.12	19.03.05	43.2	47.1	3.9	8.9
4	19.03.19	19.03.25	47.2	45.0	-2.2	-4.7
5	19.04.10	19.04.19	57.1	65.0	7.9	13.8
6	19.04.22	19.04.26	61.1	56.2	-5.0	-8.1

亏损次数:4, 盈利次数:3, 胜率:42.86%
 平均亏损:-3.69% 平均盈利:10.76%
 股价最大回撤29.21% 从2018-06-12 00:00:00开始至2018-12-27 00:00:00结束
 资金最大回撤 9.14% 从2019-03-08 00:00:00开始至2019-04-12 00:00:00结束
 资金总体收益114312; 上涨幅度 14.31%
 资金相对收益: 策略收益0.09 VS 基准收益0.12

图 9.15

未采用 ATR 止盈/止损策略和采用 ATR 止盈/止损策略的收益和风险值对例如表 9.3 所示。

表 9.3

	未采用 ATR 止盈止损	采用 ATR 止盈止损
资金总体收益	104190	114312
上涨幅度	4.19%	14.31%
资金相对收益	策略收益 0.12 与基准收益 0.12	策略收益 0.09 与基准收益 0.12
股价最大回撤	29.21% 从 2018-06-12 00:00:00 开始至 2018-12-27 00:00:00 结束	29.21% 从 2018-06-12 00:00:00 开始至 2018-12-27 00:00:00 结束
资金最大回撤	13.35% 从 2019-04-19 00:00:00 开始至 2019-05-31 00:00:00 结束	9.14% 从 2019-03-08 00:00:00 开始至 2019-04-12 00:00:00 结束

采用 ATR 止盈/止损策略后, 触发的止盈/止损交易如下所示, 可知总共触发了 2 笔止损交易, 3 笔止盈交易。止损阈值参数设置过低所带来的问题是在波动较剧烈的行情中容易被洗出局。

止盈时间:19.02.11 止盈价格:43.07
 止盈时间:19.03.05 止盈价格:47.08
 止损时间:19.03.25 止损价格:44.99
 止盈时间:19.04.19 止盈价格:65.0
 止损时间:19.04.26 止损价格:56.18

通过对比可以看出, 未加入止盈/止损模块的资金曲线回撤大, 单笔亏损较大, 且不可控, 资金曲线相当不稳; 而加入了回撤百分比止盈/止损后, 回撤变小, 资金曲线变得更加平滑, 更重要的是风险得到了控制。将 ATR 止盈/止损策略作为风险管理因子与 N 日突破择时的策略相融合, 将多个策略作为因子作用在一起判断走势, 可以从不同的维度保证交易的可靠性, 从而避免策略的不确定性所带来的交易风险。

9.4 蒙特卡洛法最优化策略参数

在股票技术面分析过程中大家是否注意到, 在多数的股票行情软件中默认的均线参数普