法律声明

本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

加班主任入群

《初阶！量化交易：策略编写及系统搭建》第6期

第2课：认识量化信号系统

主　讲：汪 浩

内容介绍

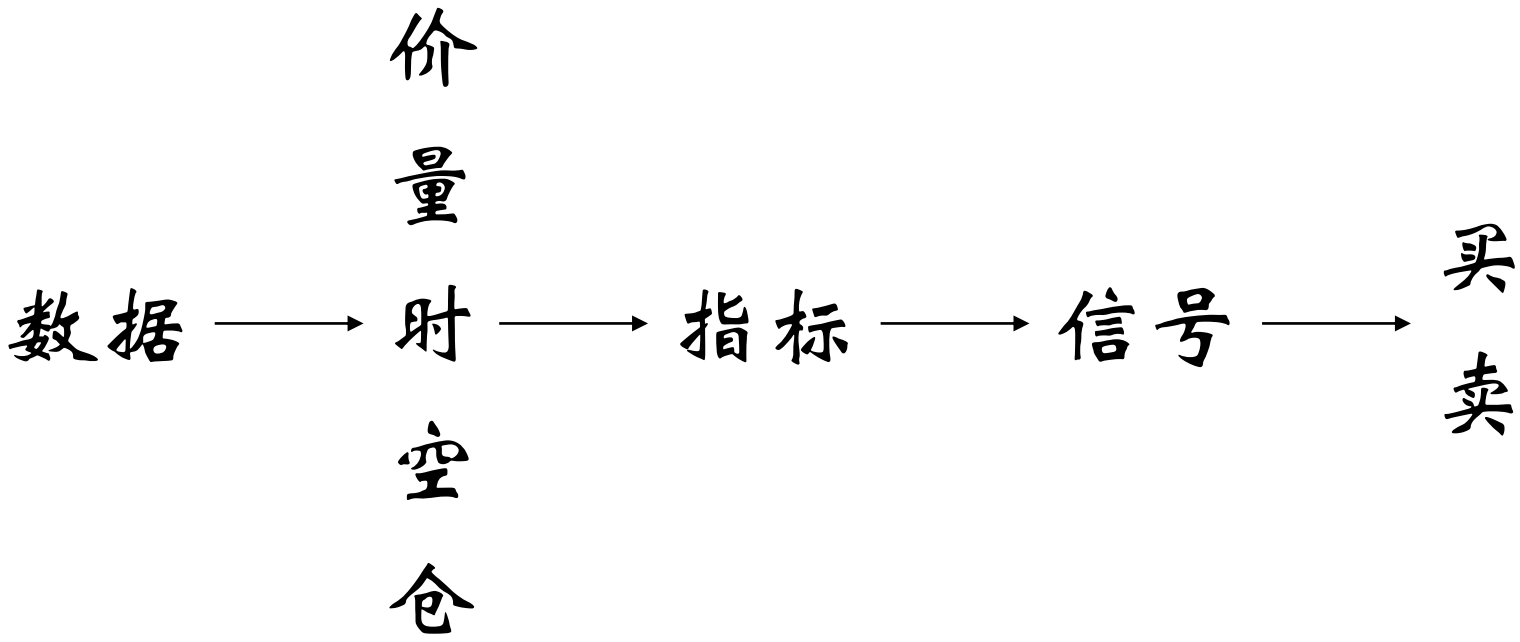
- 技术指标
- 基于MACD的交易信号开发
- 基于RSI的交易信号开发
- 基于Boll的交易信号开发
- 基于分形的交易信号开发

目标

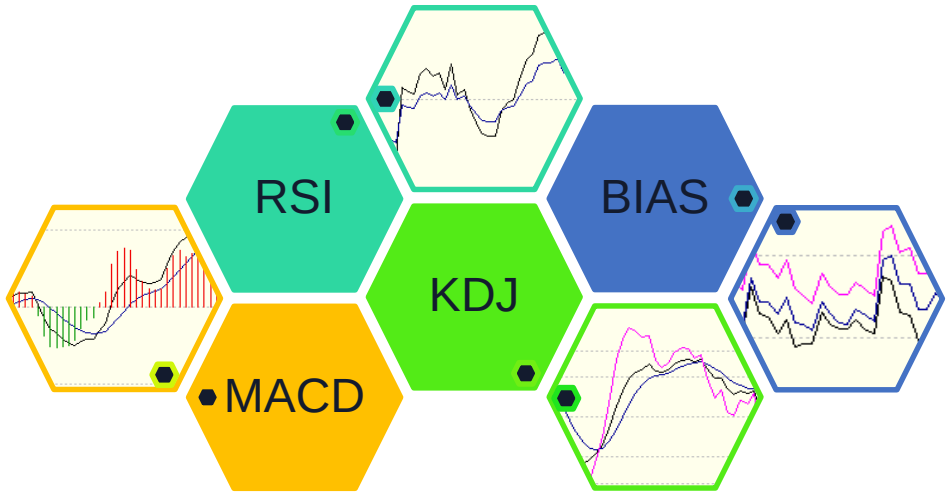
实现MACD、Boll、RSI和分型等技术信号的计算，以便能够在下一课的回测流程中使用这些信号完成择时

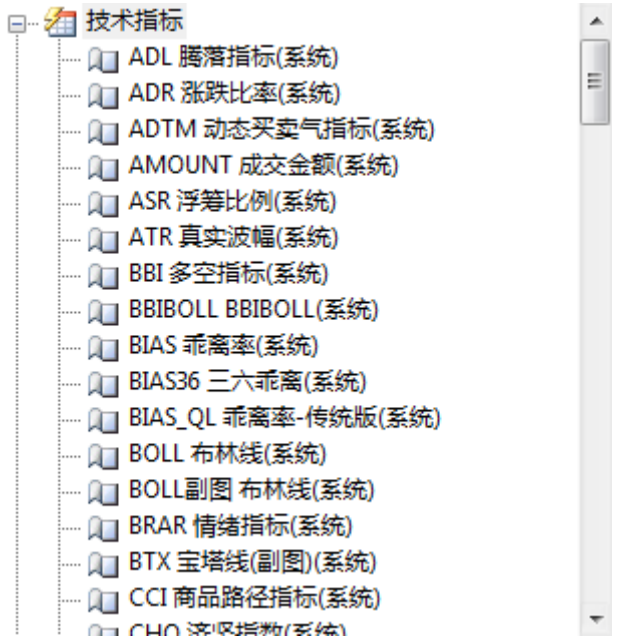
技术指标

交易决策过程简图



什么是技术指标？





- 基于行情数据，通过特定数学公式或模型计算得出的、用于辅助交易决策的数值序列

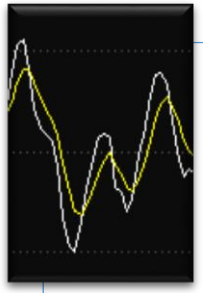
技术指标的分类

均线型

- 反映一段时间内的平均成本
- 具有一定的压力或支撑作用
- MA、EXPMA、BBI、...

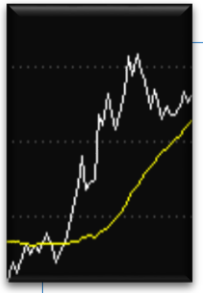
趋势型

- 适用于趋势类的行情
- 检测趋势的启动、延续，还有可能的转折
- MACD、SAR、ASI、DMI、...



摆动型

- 适用于震荡类的行情
- 检测超买超卖、波动走势的可能转折点
- KDJ、RSI、CCI、WR、BOLL、.....



能量型

- 度量涨跌的力度，预示价格位移的可持续性
- 依据是“量在价先，量价配合”
- OBV、VOL、VR、...

趋势即牛熊


上证指数(周线,前复权)
设置均线
6124.04
1067.41
VOL(5,10) VOLUME: 669975696.000 MAVOL1: 886359693.400 MAVOL2: 821095118.400
改参数
加指标
换指标

三个关系

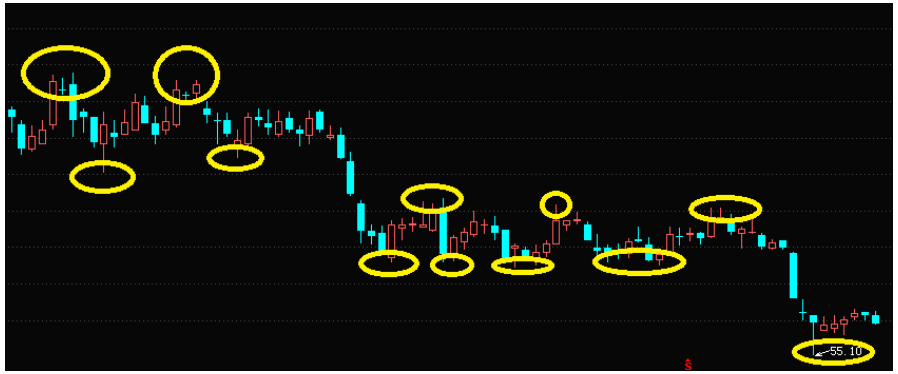
供需关系	• 趋势的动力来自于供需关系的不平衡
因果关系	• 趋势形成之前需要准备过程
努力与结果	• 成交量的增长没有使价格大幅增长，这是走势停止行为

趋势线

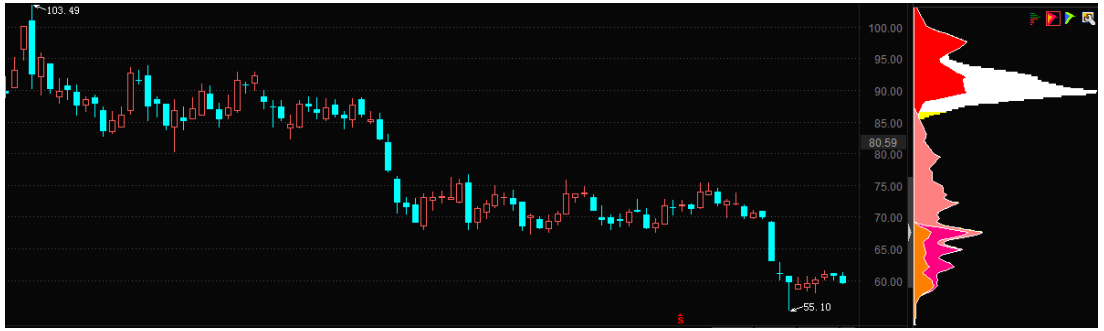
公众对支撑和压力的误解

均线系统能描述
压力和支撑吗?

转折点呢?



筹码分布呢?



正解在这里

支撑

- 在某个价位购买力超过了抛售压力，需求吸收了全部供应
- 当价格再次回到支撑位，反弹力度表明需求质量

压力

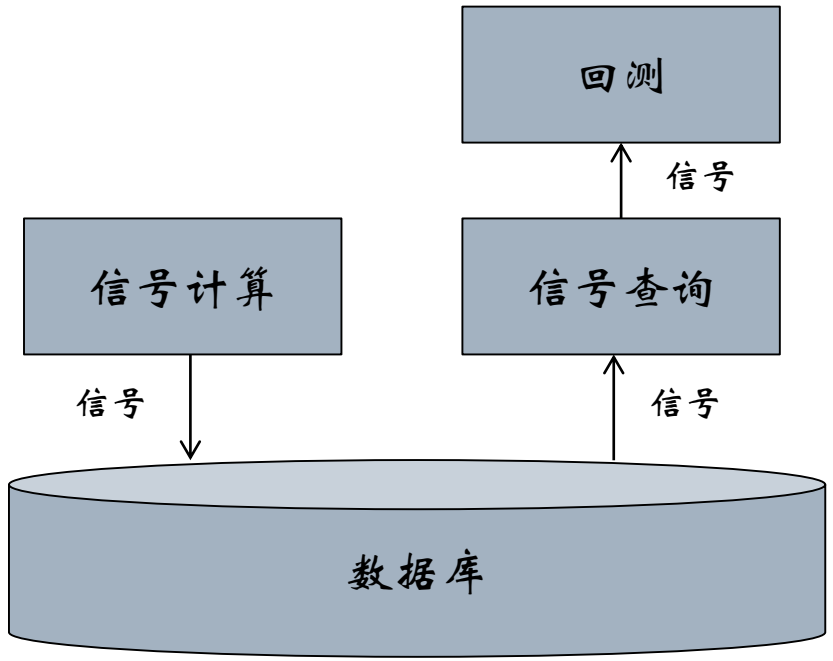
- 某个价位抛售力量超过了购买力，供应超过了需求
- 当价格再次回到压力位，价格回落力度表明供应是否扩大

要实现的指标

- MACD
  - 金叉和死叉
- RSI
  - 超买和超卖
- Boll
  - 突破上轨和突破下轨
- 分型
  - 顶分型和底分型

功能实现

- 信号计算
  - 实现信号算法
  - 检测历史信号
  - 保存到数据库
- 信号使用
  - 提供查询接口



基于MACD的交易信号开发

MACD

DIFF（MACD线）：短时EMA – 长时EMA
DEA（信号线）：DIF的EMA
MACD（红绿柱）：(DIF – DEA) [x2]

计算方法

- $EMA_i = 2/(N+1)(CLOSE_i - EMA_{i-1}) + EMA_{i-1}$
  $EMA_0 = CLOSE_0$
- EMA1 = EMA（CLOSE，short） 短时EMA
- EMA2 = EMA（CLOSE，long） 长时EMA
- DIFF = EMA1 - EMA2
- DEA = EMA（DIFF， m）

short = 12　　long = 26　　m= 9

金叉与死叉

- 死叉
  - DIFF下穿DEA
  - $DIFF_{i-1} >= DEA_{i-1}$ && $DIFF_i < DEA_i$
- 金叉
  - DIFF上穿DEA
  - $DIFF_{i-1} <= DEA_{i-1}$ && $DIFF_i > DEA_i$

Tips

- EMA的衰减
  - $EMA_i = 2/(N+1)(CLOSE_i - EMA_{i-1}) + EMA_{i-1}$
  - $EMA_0 = CLOSE_0$
- 第三方库的默认实现

pandas.DataFrame

pandas.DataFrame.ewm

`DataFrame.ewm(com=None, span=None, halflife=None, alpha=None, min_periods=0, adjust=True, ignore_na=False, axis=0)` [source]

Provides exponential weighted functions

New in version 0.18.0.

Parameters:

com : *float, optional*
Specify decay in terms of center of mass, $\alpha = 1/(1 + com)$, for $com \geq 0$

span : *float, optional*
Specify decay in terms of span, $\alpha = 2/(span + 1)$, for $span \geq 1$

halflife : *float, optional*
Specify decay in terms of half-life,
$\alpha = 1 - exp(log(0.5)/halflife)$, for $halflife > 0$

alpha : *float, optional*
Specify smoothing factor α directly, $0 < \alpha \leq 1$
New in version 0.18.0.

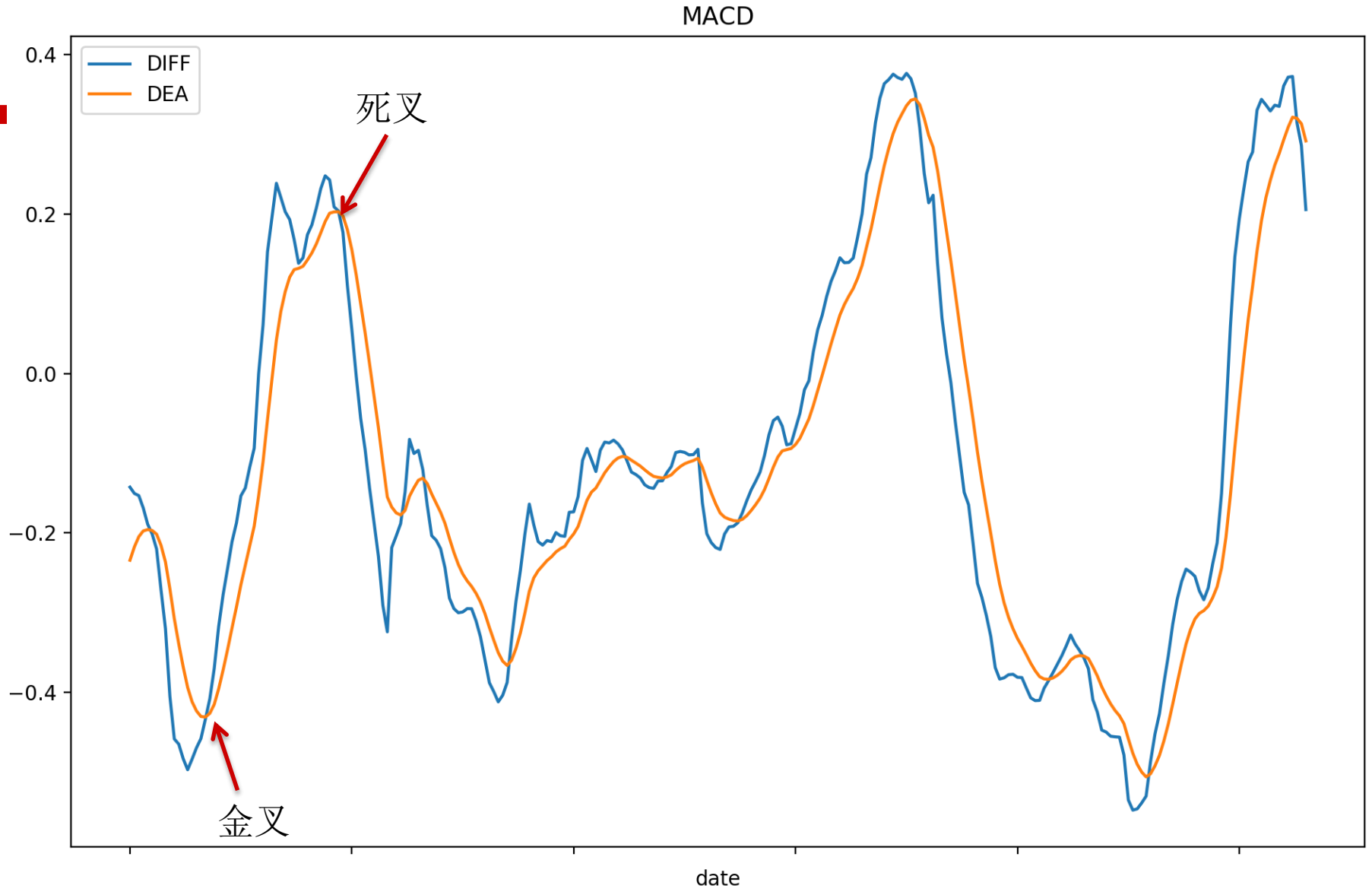
min_periods : *int, default 0*
Minimum number of observations in window required to have a value (otherwise result is NA).

adjust : *boolean, default True*
Divide by decaying adjustment factor in beginning periods to account for imbalance in relative weightings (viewing EWMA as a moving average)

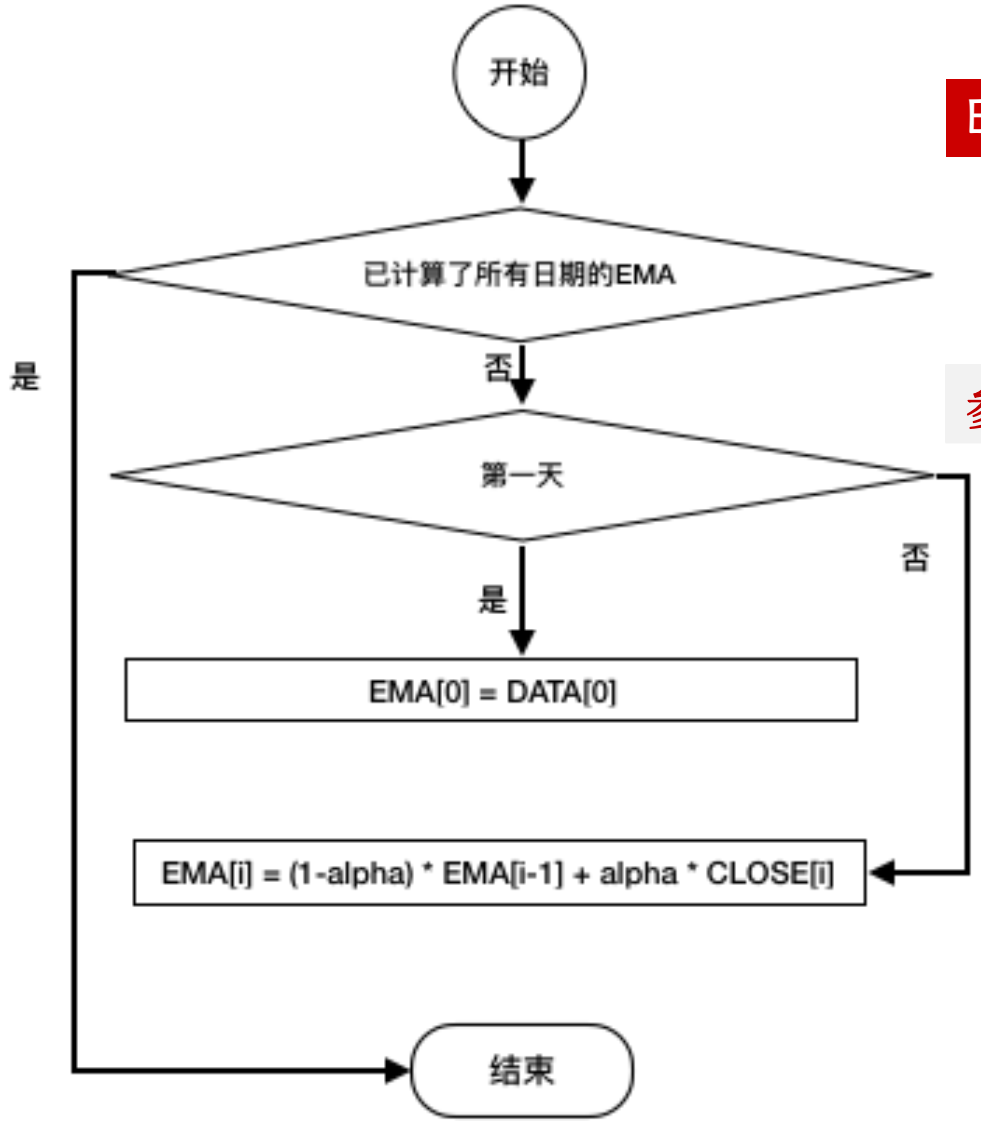
ignore_na : *boolean, default False*
Ignore missing values when calculating weights; specify True to reproduce pre-0.15.0 behavior

Returns:

a Window sub-classed for the particular operation


MACD
DIFF
DEA
死叉
金叉
0.4
0.2
0.0
−0.2
−0.4
date

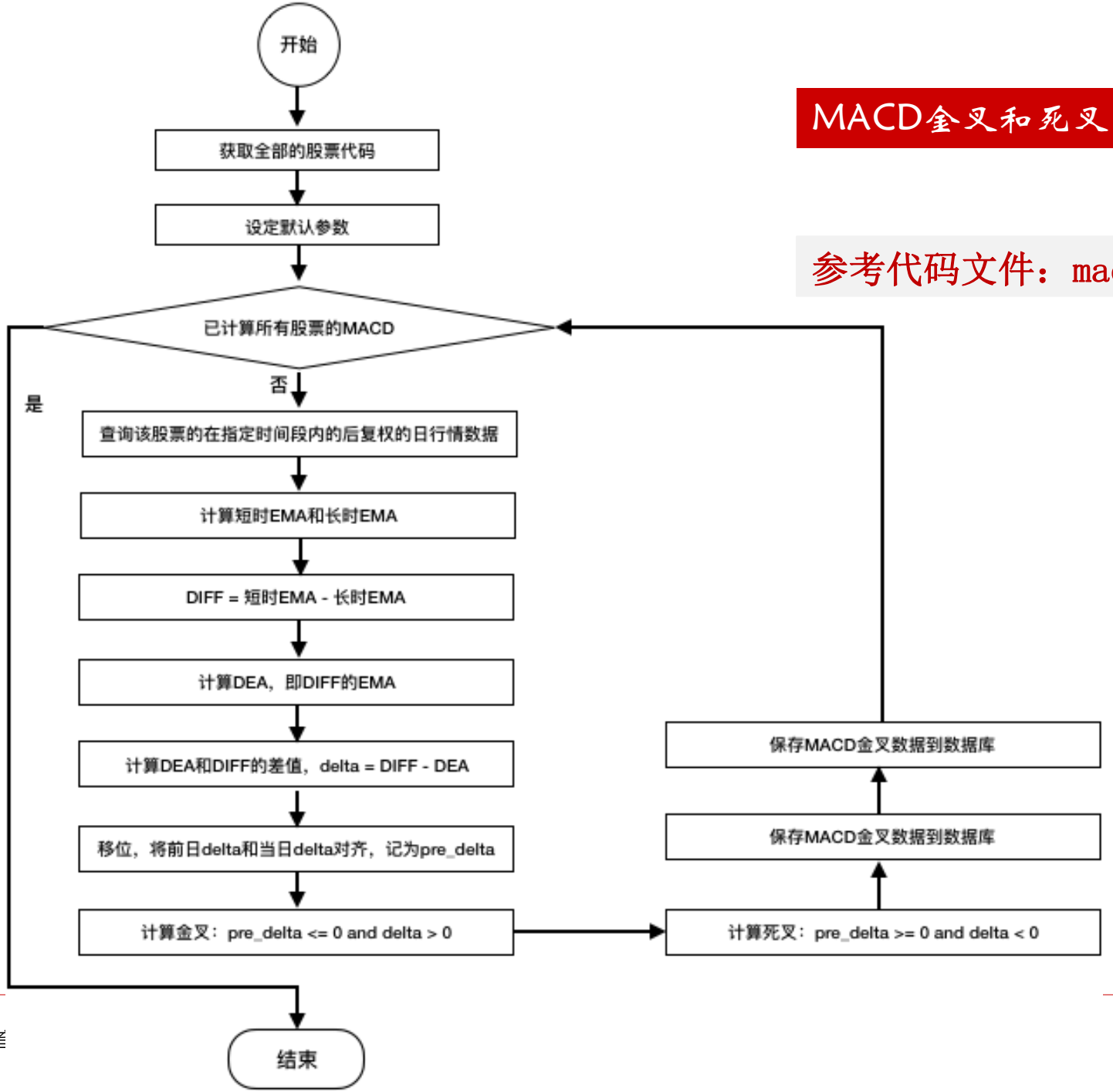
EMA的计算流程



参考代码文件：macd_factor.py

MACD金叉和死叉的计算流程

参考代码文件：macd_factor.py



参考代码文件：macd_factor.py

```
from database import DB_CONN
from stock_util import get_all_codes
from pymongo import ASCENDING, UpdateOne
from pandas import DataFrame
import traceback


def compute_macd(begin_date, end_date):
    """
    计算给定周期内的MACD金叉和死叉信号，把结果保存到数据库中
    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    """
    下面几个参数是计算MACD时的产生，这几个参数的取值都是常用值
    也可以根据需要调整
    """
    # 短时
    short = 12
    # 长时
    long = 26
    # 计算DIFF的M值
    m = 9

    # 获取所有股票代码
    codes = get_all_codes()
```

```
# 循环检测所有股票的MACD金叉和死叉信号
for code in codes:
    try:
        # 获取后复权的价格，使用后复权的价格计算MACD
        daily_cursor = DB_CONN['daily_hfq'].find(
            {'code': code, 'date': {'$gte': begin_date, '$lte': end_date}, 'index': False},
            sort=[('date', ASCENDING)],
            projection={'date': True, 'close': True, '_id': False}
        )

        # 将数据存为DataFrame格式
        df_daily = DataFrame([daily for daily in daily_cursor])
        # 设置date作为索引
        df_daily.set_index(['date'], 1, inplace=True)

        # 计算EMA
        # alpha = 2/(N+1)
        # EMA(i) = (1 - alpha) * EMA(i-1) + alpha * CLOSE(i)
        #        = alpha * (CLOSE(i) - EMA(i-1)) + EMA(i-1)
        index = 0
        # 短时EMA列表
        EMA1 = []
        # 长时EMA列表
        EMA2 = []
        # 每天计算短时EMA和长时EMA
        for date in df_daily.index:
            # 第一天EMA就是当日的close，也就是收盘价
            if index == 0:
                # 初始化短时EMA和长时EMA
                EMA1.append(df_daily.loc[date]['close'])
                EMA2.append(df_daily.loc[date]['close'])
            else:
                # 短时EMA和长时EMA
                EMA1.append(2/(short + 1) * (df_daily.loc[date]['close'] - EMA1[index - 1]) + EMA1[index - 1])
                EMA2.append(2/(long + 1) * (df_daily.loc[date]['close'] - EMA2[index - 1]) + EMA2[index - 1])

            index += 1
```

参考代码文件：macd_factor.py

参考代码文件：macd_factor.py

```
# 将短时EMA和长时EMA作为DataFrame的数据列
df_daily['EMA1'] = EMA1
df_daily['EMA2'] = EMA2

# 计算DIFF，短时EMA - 长时EMA
df_daily['DIFF'] = df_daily['EMA1'] - df_daily['EMA2']

# 计算DEA，DIFF的EMA，计算公式是： EMA(DIFF，M)
index = 0
DEA = []
for date in df_daily.index:
    if index == 0:
        DEA.append(df_daily.loc[date]['DIFF'])
    else:
        # M = 9 DEA = EMA(DIFF, 9)
        DEA.append(2/(m+1) * (df_daily.loc[date]['DIFF'] - DEA[index - 1]) + DEA[index - 1])
    index += 1

df_daily['DEA'] = DEA

# 计算DIFF和DEA的差值
df_daily['delta'] = df_daily['DIFF'] - df_daily['DEA']
# 将delta的移一位，那么前一天delta就变成了今天的pre_delta
df_daily['pre_delta'] = df_daily['delta'].shift(1)
# 金叉，DIFF上穿DEA，前一日DIFF在DEA下面，当日DIFF在DEA上面
df_daily_gold = df_daily[(df_daily['pre_delta'] <= 0) & (df_daily['delta'] > 0)]
# 死叉，DIFF下穿DEA，前一日DIFF在DEA上面，当日DIFF在DEA下面
df_daily_dead = df_daily[(df_daily['pre_delta'] >= 0) & (df_daily['delta'] < 0)]
```

参考代码文件：macd_factor.py

```
# 保存结果到数据库
    update_requests = []
    for date in df_daily_gold.index:
        # 保存时以code和date为查询条件，做更新或者新建，所以对code和date建立索引
        # 通过signal字段表示金叉还是死叉，gold表示金叉
        update_requests.append(UpdateOne(
            {'code': code, 'date': date},
            {'$set': {'code': code, 'date': date, 'signal': 'gold'}},
            upsert=True))

    for date in df_daily_dead.index:
        # 保存时以code和date为查询条件，做更新或者新建，所以对code和date建立索引
        # 通过signal字段表示金叉还是死叉，dead表示死叉
        update_requests.append(UpdateOne(
            {'code': code, 'date': date},
            {'$set': {'code': code, 'date': date, 'signal': 'dead'}},
            upsert=True))

    if len(update_requests) > 0:
        update_result = DB_CONN['macd'].bulk_write(update_requests, ordered=False)
        print('Save MACD, 股票代码: %s, 插入: %4d, 更新: %4d' %
              (code, update_result.upserted_count, update_result.modified_count), flush=True)
except:
    print('错误发生:  %s' % code, flush=True)
    traceback.print_exc()
```

参考代码文件：macd_factor.py

```
def is_macd_gold(code, date):
    """
    判断某只股票在某个交易日是否出现MACD金叉信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 有金叉信号，False - 无金叉信号
    """
    count = DB_CONN['macd'].count({'code': code, 'date': date, 'signal': 'gold'})
    return count == 1


def is_macd_dead(code, date):
    """
    判断某只股票在某个交易日是否出现MACD死叉信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 有死叉信号，False - 无死叉信号
    """
    count = DB_CONN['macd'].count({'code': code, 'date': date, 'signal': 'dead'})
    return count == 1


if __name__ == '__main__':
    compute_macd('2015-01-01', '2015-12-31')
```

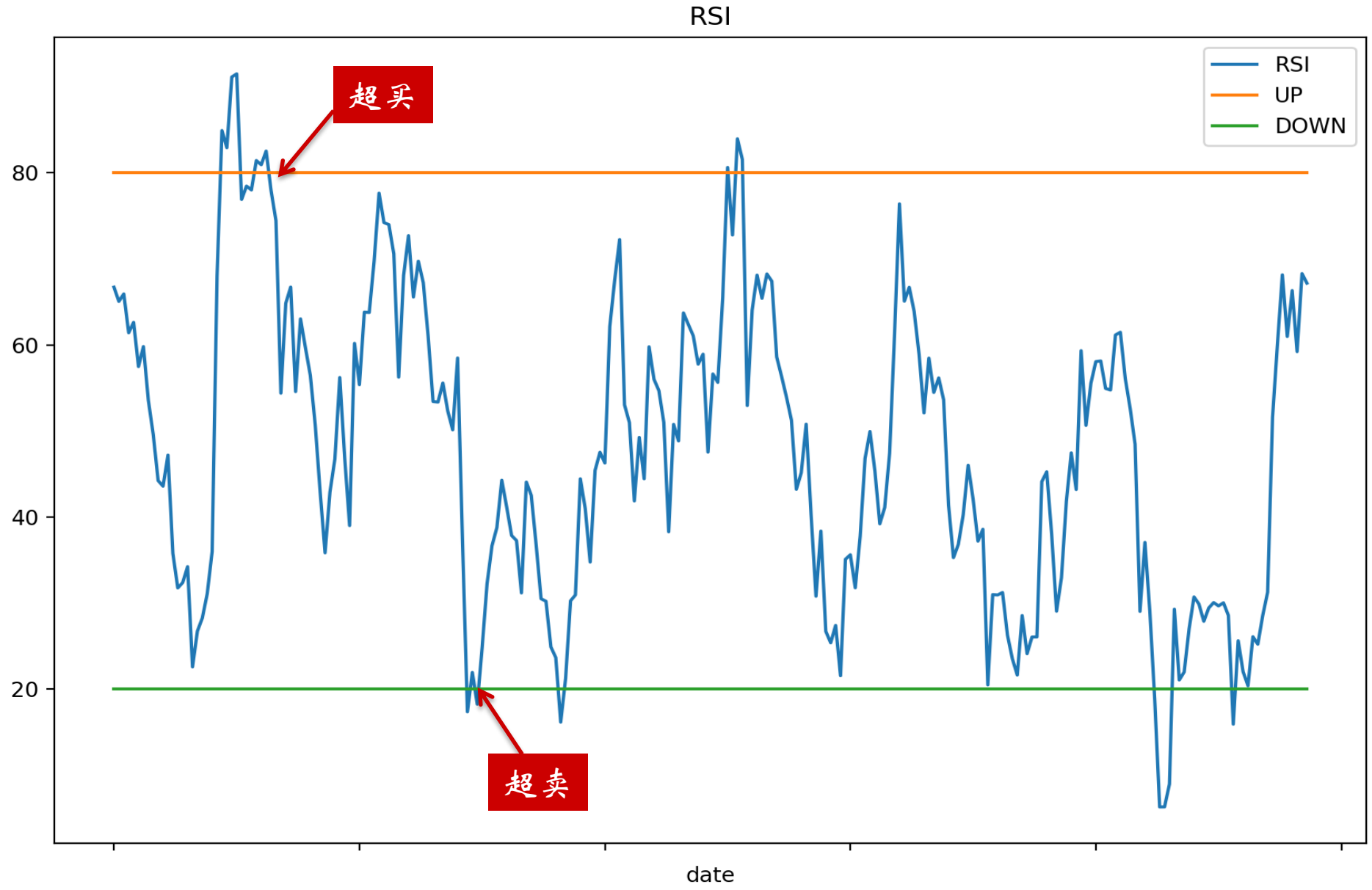
基于RSI的交易信号开发

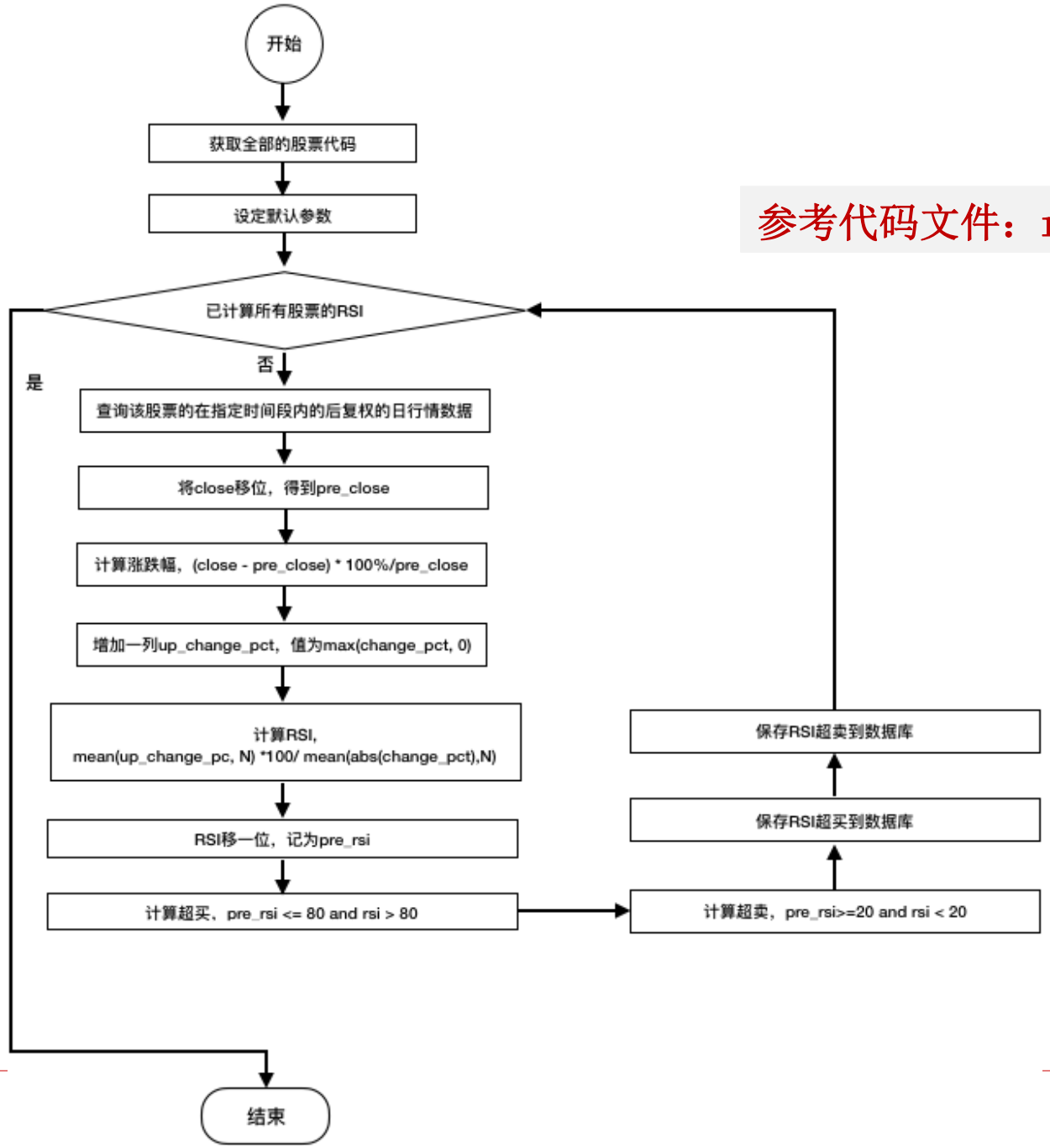
RSI –相对强弱指数

- 一定时间窗口内，上涨幅度之和占整体涨跌幅度绝对值之和的比例
- 买入意愿相对总体成交的强弱
- 超卖区：RSI < 20
- 超买区：RSI > 80
- 强弱信号
  - 超买：RSI上穿80
  - 超卖：RSI下穿20

计算公式

- change = close – prev_close
- up_change = max(change, 0)
- RSI = mean(up_change, N) * 100 /mean(abs(change), N)


RSI
RSI
UP
DOWN
超买
超卖
80
60
40
20
date



参考代码文件：rsi_factor.py

```
from pandas import DataFrame
from pymongo import ASCENDING, UpdateOne

from database import DB_CONN
from stock_util import get_all_codes


def compute_rsi(begin_date, end_date):
    """
    计算指定时间段内的RSI信号，并保存到数据库中
    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    # 获取所有股票代码
    codes = get_all_codes()

    # 计算RSI
    N = 12

    # 计算所有股票的RSI信号
    for code in codes:
        try:
            # 获取后复权的价格，使用后复权的价格计算RSI
            daily_cursor = DB_CONN['daily_hfq'].find(
                {'code': code, 'date': {'$gte': begin_date, '$lte': end_date}, 'index': False},
                sort=[('date', ASCENDING)],
                projection={'date': True, 'close': True, '_id': False}
            )

            df_daily = DataFrame([daily for daily in daily_cursor])

            # 如果查询出的行情数量还不足以计算N天的平均值，则不再参与计算
            if df_daily.index.size < N:
                print('数据量不够： %s' % code, flush=True)
                continue
```

参考代码文件：rsi_factor.py

```
 # 计算RSI
    df_daily['RSI'] = df_daily['up_pct'].rolling(N).mean() /
abs(df_daily['change_pct']).rolling(N).mean() * 100

    # 移位
    df_daily['PREV_RSI'] = df_daily['RSI'].shift(1)

    # # 超买，RSI下穿80，作为卖出信号
    df_daily_over_bought = df_daily[(df_daily['RSI'] < 80) & (df_daily['PREV_RSI'] >= 80)]
    # # 超卖，RSI上穿20，作为买入信号
    df_daily_over_sold = df_daily[(df_daily['RSI'] > 20) & (df_daily['PREV_RSI'] <= 20)]
    #
    # # 保存结果到数据库
    update_requests = []
    for date in df_daily_over_bought.index:
        update_requests.append(UpdateOne(
            {'code': code, 'date': date},
            {'$set': {'code':code, 'date': date, 'signal': 'over_bought'}},
            upsert=True))

    for date in df_daily_over_sold.index:
        update_requests.append(UpdateOne(
            {'code': code, 'date': date},
            {'$set': {'code':code, 'date': date, 'signal': 'over_sold'}},
            upsert=True))

    if len(update_requests) > 0:
        update_result = DB_CONN['rsi'].bulk_write(update_requests, ordered=False)
        print('Save RSI, 股票代码：%s, 插入：%4d, 更新：%4d' %
              (code, update_result.upserted_count, update_result.modified_count), flush=True)
except:
    print('错误发生：  %s' % code, flush=True)
```

参考代码文件：rsi_factor.py

参考代码文件：rsi_factor.py

```
# 将日期作为索引
df_daily.set_index(['date'], 1, inplace=True)
# 将close移一位作为当日的pre_close
df_daily['pre_close'] = df_daily['close'].shift(1)
# 计算当日的涨跌幅：(close - pre_close) * 100 / pre_close
df_daily['change_pct'] = (df_daily['close'] - df_daily['pre_close']) * 100 / df_daily['pre_close']
# 只保留上涨的日期的涨幅
df_daily['up_pct'] = DataFrame({'up_pct': df_daily['change_pct'], 'zero': 0}).max(1)

# 计算RSI
df_daily['RSI'] = df_daily['up_pct'].rolling(N).mean() / abs(df_daily['change_pct']).rolling(N).mean() * 100

# 移位
df_daily['PREV_RSI'] = df_daily['RSI'].shift(1)

# 超买，RSI下穿80，作为卖出信号
df_daily_over_bought = df_daily[(df_daily['RSI'] < 80) & (df_daily['PREV_RSI'] >= 80)]
# 超卖，RSI上穿20，作为买入信号
df_daily_over_sold = df_daily[(df_daily['RSI'] > 20) & (df_daily['PREV_RSI'] <= 20)]

# 保存结果到数据库，要以code和date创建索引，db.rsi.createIndex({'code': 1, 'date': 1})
update_requests = []
# 超买数据，以code和date为key更新数据，signal为over_bought
for date in df_daily_over_bought.index:
    update_requests.append(UpdateOne(
        {'code': code, 'date': date},
        {'$set': {'code': code, 'date': date, 'signal': 'over_bought'}},
        upsert=True))

# 超卖数据，以code和date为key更新数据，signal为over_sold
for date in df_daily_over_sold.index:
    update_requests.append(UpdateOne(
        {'code': code, 'date': date},
        {'$set': {'code': code, 'date': date, 'signal': 'over_sold'}},
        upsert=True))

if len(update_requests) > 0:
    update_result = DB_CONN['rsi'].bulk_write(update_requests, ordered=False)
    print('Save RSI, 股票代码：%s, 插入：%4d, 更新：%4d' %
          (code, update_result.upserted_count, update_result.modified_count), flush=True)
```

参考代码文件：rsi_factor.py

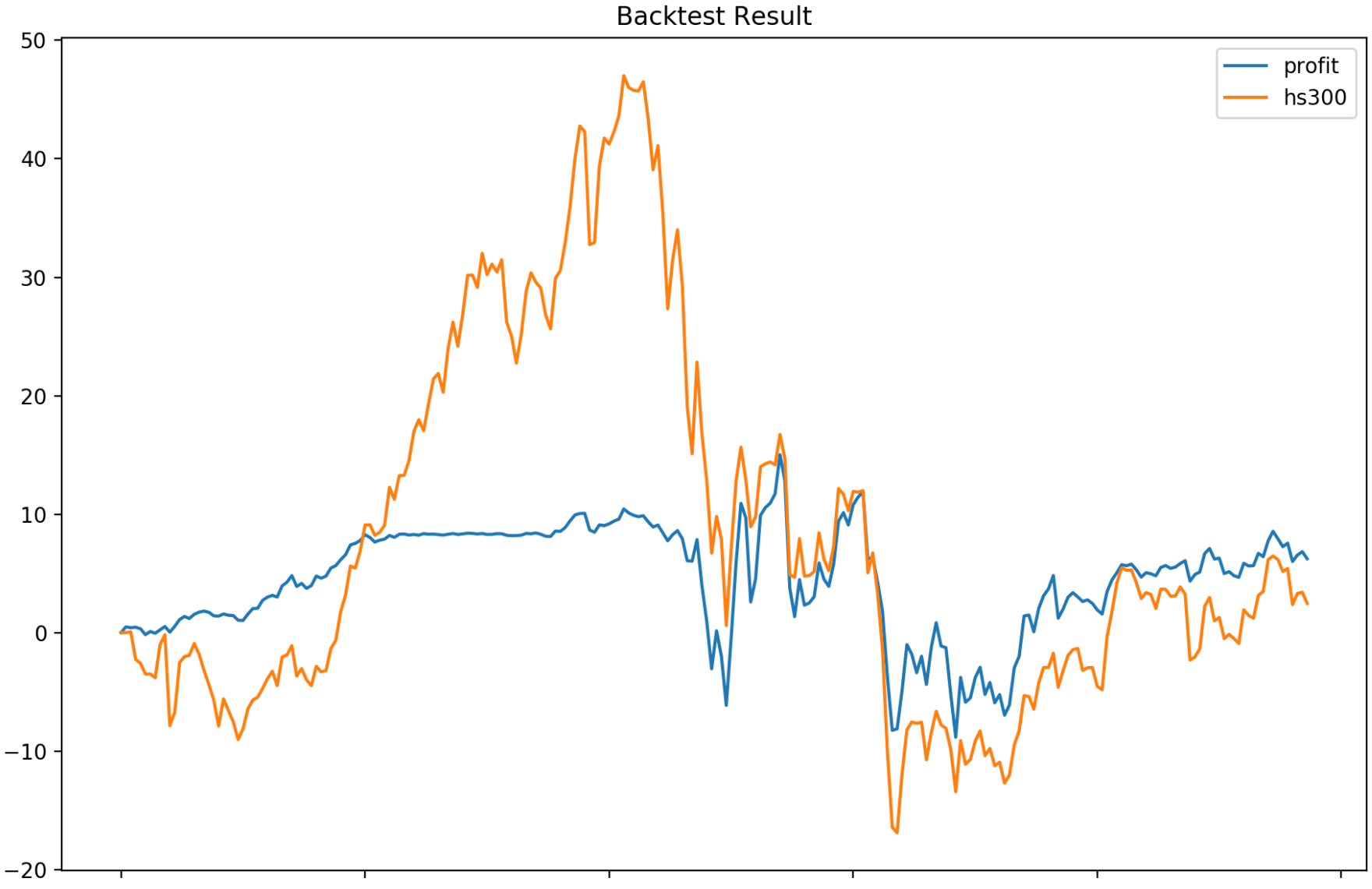
```
def is_rsi_over_sold(code, date):
    """
    判断某只股票在某个交易日是出现了超卖信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 出现了超卖信号，False - 没有出现超卖信号
    """
    count = DB_CONN['rsi'].count({'code': code, 'date': date, 'signal': 'over_sold'})
    return count == 1


def is_rsi_over_bought(code, date):
    """
    判断某只股票在某个交易日是出现了超买信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 出现了超买信号，False - 没有出现超买信号
    """
    count = DB_CONN['rsi'].count({'code': code, 'date': date, 'signal': 'over_bought'})
    return count == 1


if __name__ == '__main__':
    compute_rsi('2015-01-01', '2015-12-31')
```

```
# 检查是否有需要第二天卖出的股票
for holding_code in holding_codes:
    if is_rsi_over_bought(holding_code, _date):
        to_be_sold_codes.add(holding_code)
```

```
# 检查是否有需要第二天买入的股票
to_be_bought_codes.clear()
if this_phase_codes is not None:
    for _code in this_phase_codes:
        if _code not in holding_codes and is_rsi_over_sold(_code, _date):
            to_be_bought_codes.add(_code)
```


Backtest Result
profit
hs300
50
40
30
20
10
0
−10
−20

思考

- 时间窗口长度
- 超买/超卖区间阈值
- 信号触发的位置

休息一下

5分钟后回来

基于BOLL的交易信号开发

Boll

- 基于统计学的标准差原理
- 三条轨道
  - 上轨：压力线
  - 中轨：价格均线
  - 下轨：支撑线

计算公式：价格均值和标准差

$$MA = \frac{1}{N}\sum_{i=0}^{N} CLOSE_i \qquad (1)$$

N=20

$$STD = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(CLOSE_i - MA)^2} \qquad (2)$$

计算公式

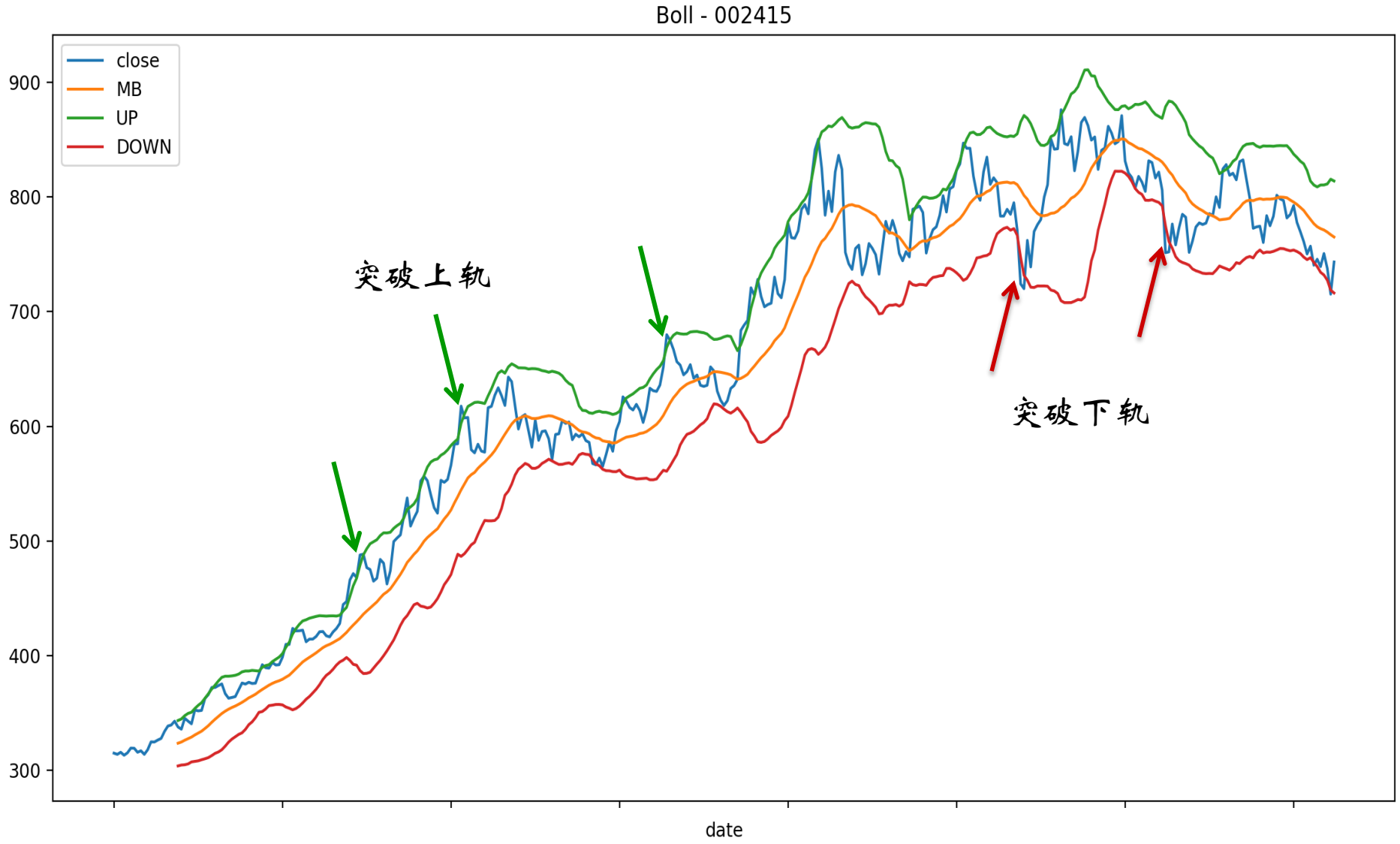
$$MB = \frac{1}{N-1}\sum_{i=0}^{N-1} CLOSE_i$$ 中轨

盘中实时计算，当日收盘价不稳定，所以计算的是前一天的均价

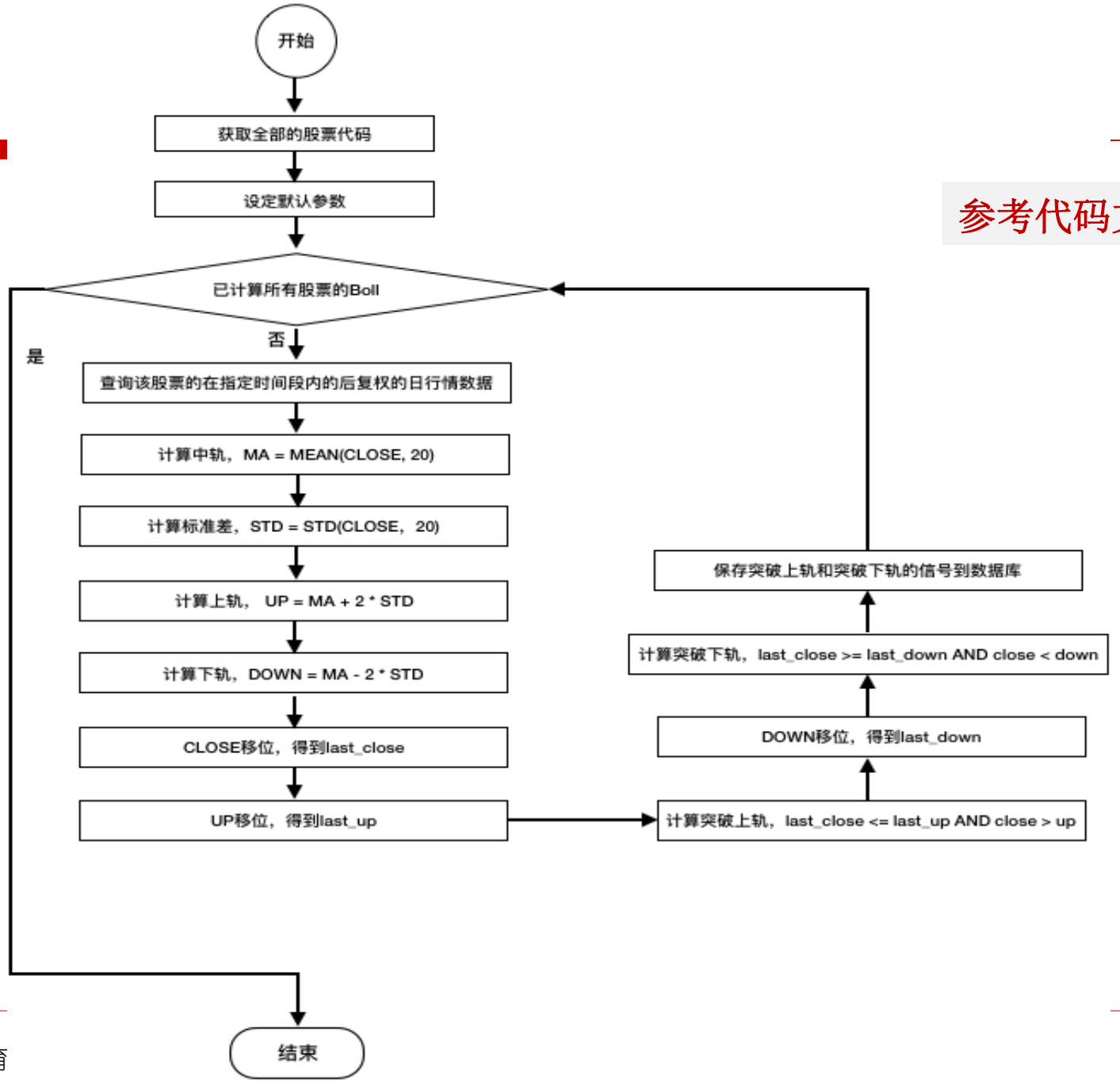
$$UP = MB + k * STD$$ 上轨

k=2

$$DOWN = MB - k * STD$$ 下轨


Boll - 002415
close
MB
UP
DOWN
900
800
700
600
500
400
300
突破上轨
突破下轨
date

参考代码文件：boll_factor.py



```
import traceback

from pandas import DataFrame
from pymongo import UpdateOne, ASCENDING

from database import DB_CONN
from stock_util import get_all_codes


def compute(begin_date, end_date):
    """
    计算指定日期内的Boll突破上轨和突破下轨信号，并保存到数据库中，
    方便查询使用
    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    # 获取所有股票代码
    all_codes = get_all_codes()

    # 计算每一只股票的Boll信号
    for code in all_codes:
        try:
            # 获取后复权的价格，使用后复权的价格计算BOLL
            daily_cursor = DB_CONN['daily_hfq'].find(
                {'code': code, 'date': {'$gte': begin_date, '$lte': end_date}, 'index': False},
                sort=[('date', ASCENDING)],
                projection={'date': True, 'close': True, '_id': False}
            )

            df_daily = DataFrame([daily for daily in daily_cursor])
```

参考代码文件：boll_factor.py

Boll

参考代码文件：boll_factor.py

```
# 计算MB，盘后计算，这里用当日的Close
df_daily['MB'] = df_daily['close'].rolling(20).mean()
# 计算STD20，计算20日的标准差
df_daily['std'] = df_daily['close'].rolling(20).std()

print(df_daily, flush=True)
# 计算UP，上轨
df_daily['UP'] = df_daily['MB'] + 2 * df_daily['std']
# 计算down，下轨
df_daily['DOWN'] = df_daily['MB'] - 2 * df_daily['std']

print(df_daily, flush=True)

# 将日期作为索引
df_daily.set_index(['date'], inplace=True)

# 将close移动一个位置，变为当前索引位置的前收
last_close = df_daily['close'].shift(1)

# 将上轨移一位，前一日的上轨和前一日的收盘价都在当日了
shifted_up = df_daily['UP'].shift(1)
# 突破上轨，是向上突破，条件是前一日收盘价小于前一日上轨，当日收盘价大于当日上轨
df_daily['up_mask'] = (last_close <= shifted_up) & (df_daily['close'] > shifted_up)

# 将下轨移一位，前一日的下轨和前一日的收盘价都在当日了
shifted_down = df_daily['DOWN'].shift(1)
# 突破下轨，是向下突破，条件是前一日收盘价大于前一日下轨，当日收盘价小于当日下轨
df_daily['down_mask'] = (last_close >= shifted_down) & (df_daily['close'] < shifted_down)

# 对结果进行过滤，只保留向上突破或者向上突破的数据
df_daily = df_daily[df_daily['up_mask'] | df_daily['down_mask']]
# 从DataFrame中扔掉不用的数据
df_daily.drop(['close', 'std', 'MB', 'UP', 'DOWN'], 1, inplace=True)
```

参考代码文件：boll_factor.py

```
        # 将信号保存到数据库
        update_requests = []
        # DataFrame的索引是日期
        for date in df_daily.index:
            # 保存的数据包括股票代码、日期和信号类型，结合数据集的名字，就表示某只股票在某日
            doc = {
                'code': code,
                'date': date,
                # 方向，向上突破 up，向下突破 down
                'direction': 'up' if df_daily.loc[date]['up_mask'] else 'down'
            }
            update_requests.append(
                UpdateOne(doc, {'$set': doc}, upsert=True))

        # 如果有信号数据，则保存到数据库中
        if len(update_requests) > 0:
            # 批量写入到boll数据集中
            update_result = DB_CONN['boll'].bulk_write(update_requests, ordered=False)
            print('%s, upserted: %4d, modified: %4d' %
                  (code, update_result.upserted_count, update_result.modified_count),
                  flush=True)
    except:
        traceback.print_exc()
```

参考代码文件：boll_factor.py

```
def is_boll_break_up(code, date):
    """
    查询某只股票是否在某日出现了突破上轨信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 出现了突破上轨信号，False - 没有出现突破上轨信号
    """
    count = DB_CONN['boll'].count({'code': code, 'date': date, 'direction': 'up'})
    return count == 1


def is_boll_break_down(code, date):
    """
    查询某只股票是否在某日出现了突破下轨信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 出现了突破下轨信号，False - 没有出现突破下轨信号
    """
    count = DB_CONN['boll'].count({'code': code, 'date': date, 'direction': 'down'})
    return count == 1


if __name__ == '__main__':
    # 计算指定时间内的boll信号
    compute(begin_date='2015-01-01', end_date='2015-12-31')
```

思考

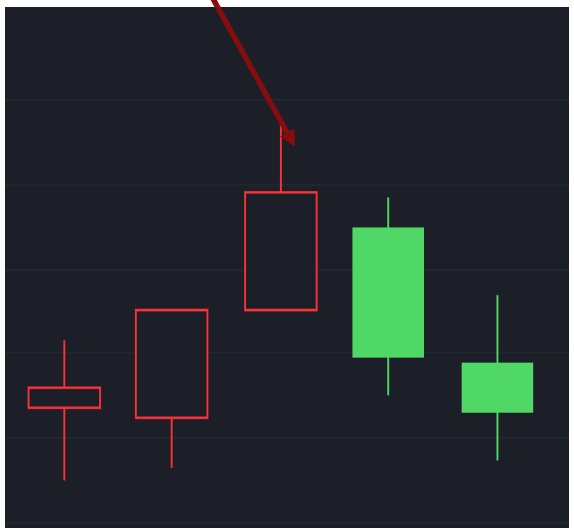
参考代码文件：boll_factor.py

- 实时价格的不稳定性
  - MB：前一日的收盘价
- 周期
  - 日
  - 周
  - 月
  - 分钟
- 可变参数
  - N
  - k

基于分形的交易信号的开发

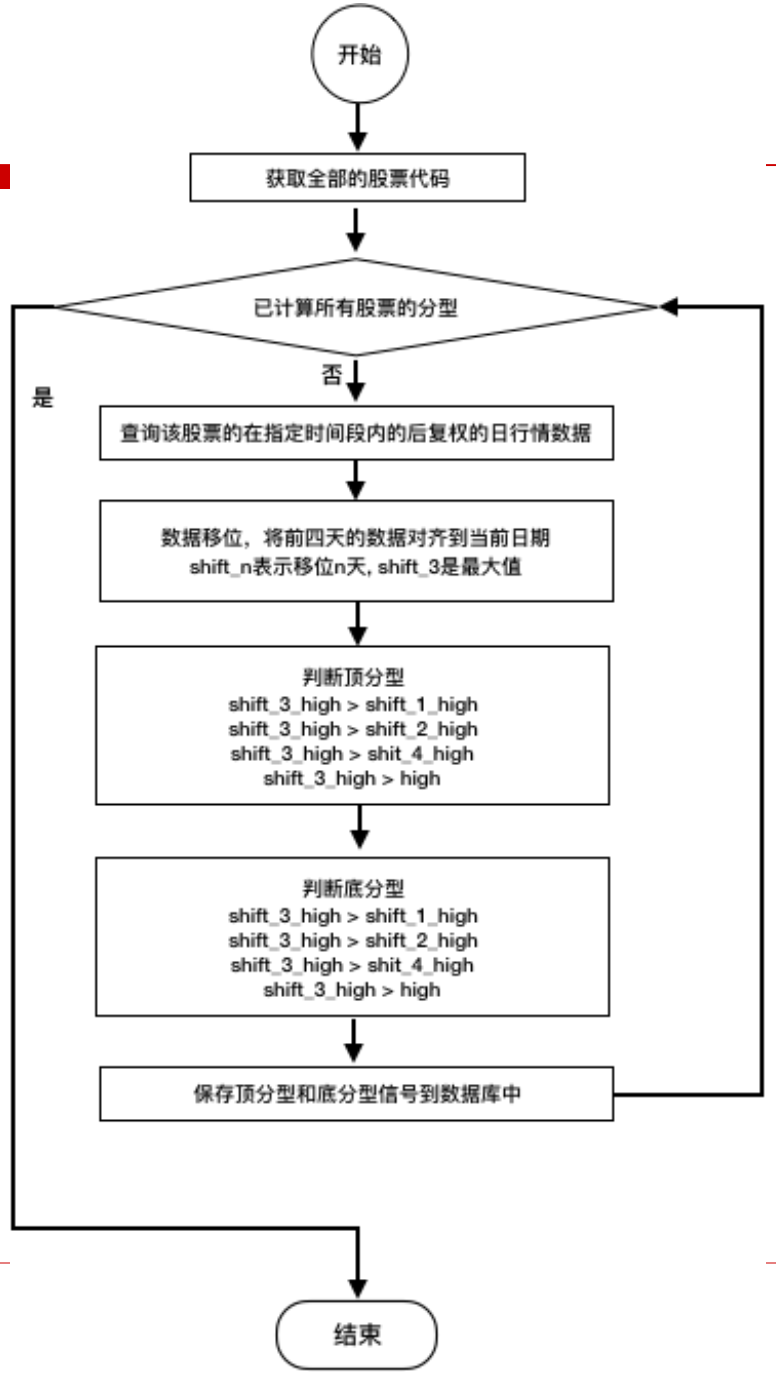
分型信号

顶分型



底分型

宽度 = 5



参考代码文件：fractal_factor.py

参考代码文件：fractal_factor.py

```
from database import DB_CONN
from stock_util import get_all_codes
from pymongo import ASCENDING, UpdateOne
from pandas import DataFrame
import traceback


def compute_fractal(begin_date, end_date):
    # 获取所有股票代码
    codes = get_all_codes()

    # 计算每个股票的信号
    for code in codes:
        try:
            # 获取后复权的价格，使用后复权的价格计算分型信号
            daily_cursor = DB_CONN['daily_hfq'].find(
                {'code': code, 'date': {'$gte': begin_date, '$lte': end_date}, 'index': False},
                sort=[('date', ASCENDING)],
                projection={'date': True, 'high': True, 'low': True, '_id': False}
            )

            df_daily = DataFrame([daily for daily in daily_cursor])

            # 设置日期作为索引
            df_daily.set_index(['date'], 1, inplace=True)
```

```
# 通过shift，将前两天和后两天对齐到中间一天
        df_daily_shift_1 = df_daily.shift(1)
        df_daily_shift_2 = df_daily.shift(2)
        df_daily_shift_3 = df_daily.shift(3)
        df_daily_shift_4 = df_daily.shift(4)

        # 顶分型，中间日的最高价既大于前两天的最高价，也大于后两天的最高价
        df_daily['up'] = (df_daily_shift_3['high'] > df_daily_shift_1['high']) & \
                         (df_daily_shift_3['high'] > df_daily_shift_2['high']) & \
                         (df_daily_shift_3['high'] > df_daily_shift_4['high']) & \
                         (df_daily_shift_3['high'] > df_daily['high'])

        # 底分型，中间日的最低价既小于前两天的最低价，也小于后两天的最低价
        df_daily['down'] = (df_daily_shift_3['low'] < df_daily_shift_1['low']) & \
                           (df_daily_shift_3['low'] < df_daily_shift_2['low']) & \
                           (df_daily_shift_3['low'] < df_daily_shift_4['low']) & \
                           (df_daily_shift_3['low'] < df_daily['low'])

        # 只保留了出现顶分型和低分型信号的日期
        df_daily = df_daily[(df_daily['up'] | df_daily['down'])]

        # 抛掉不用的数据
        df_daily.drop(['high', 'low'], 1, inplace=True)

        print(df_daily)
        # 将信号保存到数据库 ,
        update_requests = []
        # 保存的数据结果时，code、date和信号的方向
        for date in df_daily.index:
            doc = {
                'code': code,
                'date': date,
                # up: 顶分型， down：底分型
                'direction': 'up' if df_daily.loc[date]['up'] else 'down'
            }
```

参考代码文件：fractal_factor.py

```
                # 保存时以code、date和direction做条件，那么就需要在这三个字段上建立索引
                # db.fractal_signal.createIndex({'code': 1, 'date': 1, 'direction': 1})
                update_requests.append(
                    UpdateOne(doc, {'$set': doc}, upsert=True))

            if len(update_requests) > 0:
                update_result = DB_CONN['fractal_signal'].bulk_write(update_requests, ordered=False)
                print('%s, upserted: %4d, modified: %4d' %
                      (code, update_result.upserted_count, update_result.modified_count),
                      flush=True)
        except:
            print('错误发生： %s' % code, flush=True)
            traceback.print_exc()


def is_fractal_up(code, date):
    """
    查询某只股票在某个日期是否出现顶分型信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 出现顶分型信号，False - 没有出现顶分型信号
    """
    count = DB_CONN['fractal_signal'].count({'code': code, 'date': date, 'direction': 'up'})
    return count == 1


def is_fractal_down(code, date):
    """
    查询某只股票在某个日期是否出现底分型信号
    :param code: 股票代码
    :param date: 日期
    :return: True - 出现底分型信号，False - 没有出现底分型信号
    """
    count = DB_CONN['fractal_signal'].count({'code': code, 'date': date, 'direction': 'down'})
    return count == 1


if __name__ == '__main__':
    compute_fractal('2015-01-01', '2015-12-31')
```

参考代码文件：fractal_factor.py

思考

- 检测宽度
  - N∈（3，5，7，9…）
- 对称性

总结

- 信号在策略开发中的作用
- 交易信号的概念和实现
  - MACD金叉/死叉
  - RSI
  - Boll
  - 分形：顶分型、底分型

预告

- 环境准备《开发环境准备》《MongoDB安装指导》
- 注意事项
- 搭建一个简单的量化交易策略回测验证系统

问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**

THANKS