JoinQuant 量化课堂

来源：https://www.joinquant.com/study

新手专区

聚宽新手指南

如果你想量化交易快速入门

1. 十行代码带你量化交易入门，文章以简单的实例介绍了在聚宽做量化交易最核心的流程——策略编写、策略回测、建立模拟、发送信号，绝对是量化交易急速入门教程。

如果你是编程苦手

1. 向导式策略生成器已上线，无需编程、快速验证，聚宽最新上线向导式策略生成器，堪称小白的入门福音，大神的偷懒神器。结合编程模式搭配使用，便能做到无编程快速验证方向，有编程自由精制策略。

如果你想要更多的学习资源

1. 量化课堂，量化课堂里提供了编程，数学，策略实例，统计研究，金融市场等量化相关的知识，尤其是在量化核心的数理方面，比其更好的几乎业界难觅。
2. 社区干货遴选与整理（持续进行中），上百篇的聚宽社区好文，从心得技巧到策略分享，从机器学习到股指套利，可谓是成吨的量化交易干货。

如果你想发现好策略或你有好策略

1. 策略擂台，旨在发现好策略，发现好宽客，聚宽将助其实现其价值，寻求合作共赢的机会，让人人真正地成为靠策略赚钱的宽客。
2. 另外，策略擂台目前可以免费订阅牛人的策略信号，跟着交易，躺着赚钱不是梦。我们了解到的最厉害的，用户跟策略半年多，6万变到12万。

如果你想了解数据方面

1. 聚宽提供的数据
2. 数据常见疑问汇总（持续汇总中...）
3. 数据调整记录

相信多数数据方面的问题都可以从中找到答案。

如果你还有疑问

1. 常见问题里解答了基础的使用方面的疑问。
2. API文档介绍了聚宽提供的各种功能的API。
3. 【有用功】教你如何调试程序(Debug)为编程新手介绍了基础的代码纠错方法。
4. 善用社区的搜索功能，你的疑问很可能之前已经有人问过了。
5. 可以在社区问答区发帖提问，必要时可以艾特聚宽工作人员蛋蛋-zy与JoinQuant-TWist。
6. 可以加入聚宽qq群，那里经常有很多聚宽用户交流，相信他们愿意友好地帮助你，当然你也可以艾特群的管理员。

JoinQuant聚宽4群 589457056
JoinQuant聚宽3群 524080396

Welcome to JoinQuant ！

【新手入门教程】十行代码带你量化交易入门

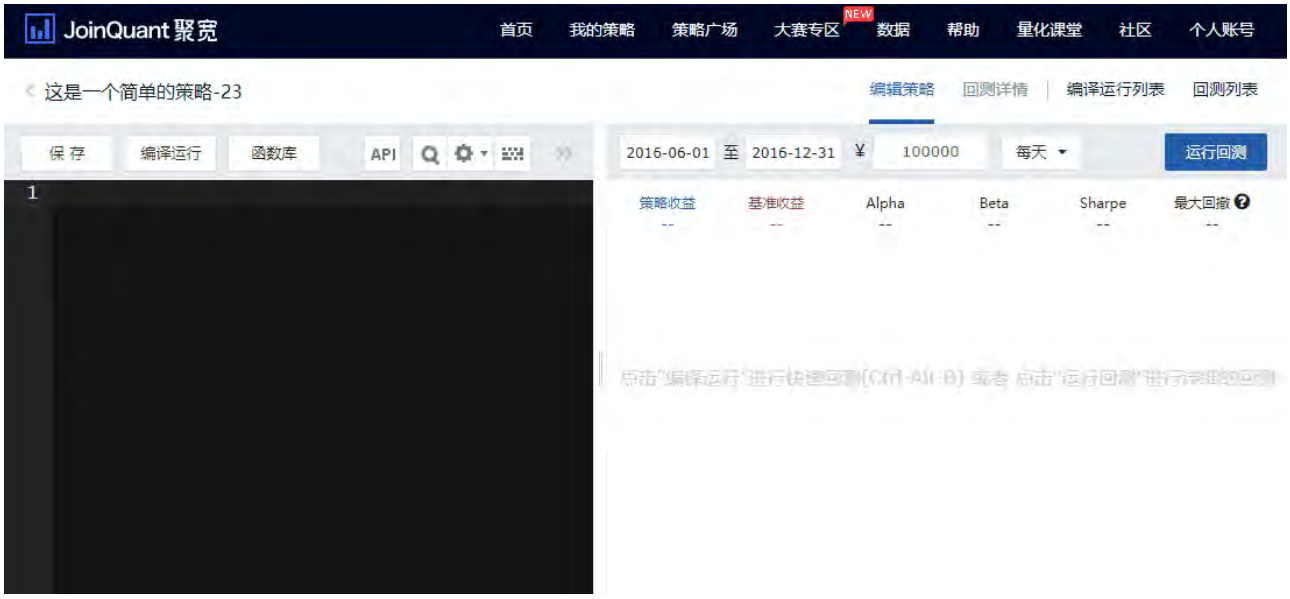
下一篇文章更新：【新手入门教程】多股票策略

说起量化交易入门，很多时候得到的答案要么是谈理论、要么列书单、要么就过程繁琐难以实践，结果往往是让对量化交易感兴趣的人迈不出那最初的一步。另外一方面，如果已经学了一些Python基础又希望能够用上，学以致用，那么这篇文章将教你简单的使用Python去写一个 策略。

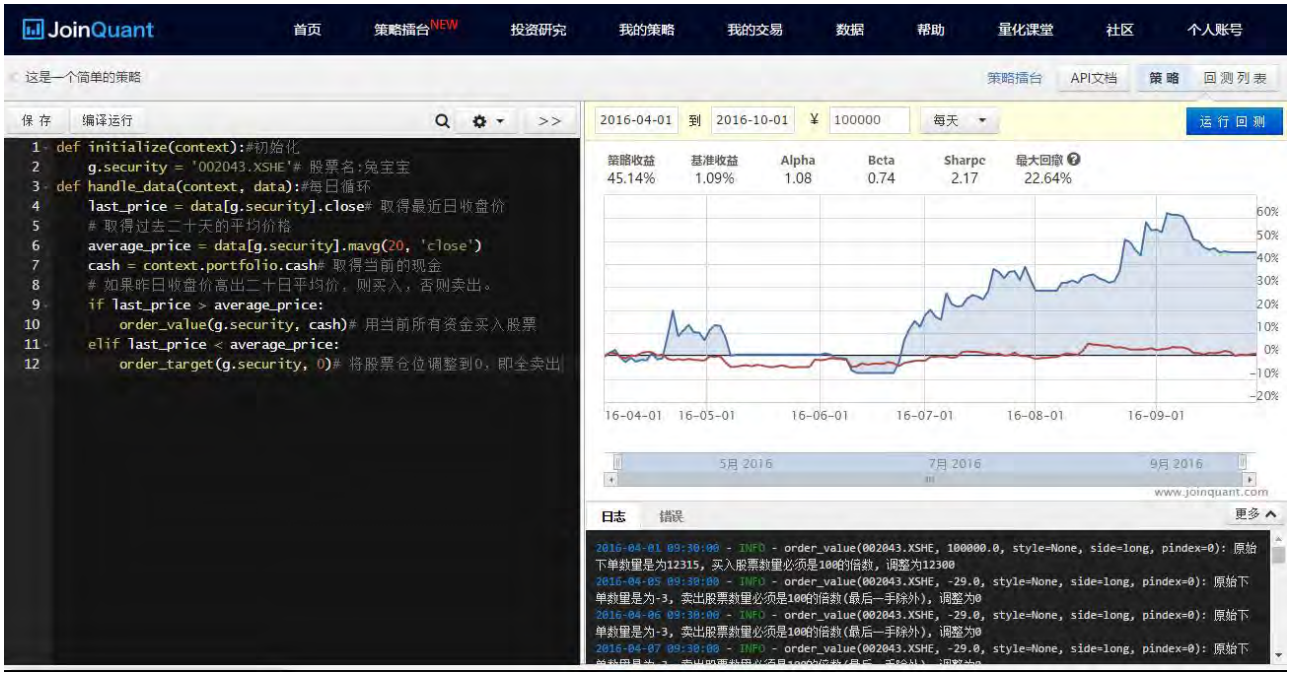
这篇文章，力求以简单的例子，手把手教你如何用十行代码，写一个能根据实际行情发送交易信号到微信的量化交易策略！

>学习内容：
>- 学会写一个简单的量化交易策略
>- 理解策略的基本框架
>- 学会建立连接实盘的模拟交易，并使其自动发送交易的信号到微信

首先，进入JoinQuant，新建策略并清空原本模板代码，如下图。注意，未登录用户无法保存编写的策略以及查看回测详情，最好先登录，免得编的策略保存不了白做了。



左侧是编写策略代码，右侧是策略运行结果。我们就在左侧写策略代码。（图片是老版示例，跟上图不相连，只看大体界面就好）



下面教你用10行代码写个量化交易策略——单股票均线策略

1 确定策略内容与框架

我们明确下策略内容：
若昨日收盘价高出过去20日平均价今天开盘买入股票
若昨日收盘价低于过去20日平均价今天开盘卖出股票

只操作一只股票，很简单对吧，但怎么用代码说给计算机听呢？

想想人是怎么操作的，应该包括这样两个部分

1. 既然是单股票策略，事先决定好交易哪一个股票。
2. 每天看看昨日收盘价是否高出过去20日平均价，是的话开盘就买入，不是开盘就卖出。每天都这么做，循环下去。

对应代码也是这两个部分

```
def initialize(context):
    用来写最开始要做什么的地方
def handle_data(context,data):
    用来写每天循环要做什么的地方
```

> 答疑与延伸：
> - def后面的空格和最后的冒号不能少！
> - 符号都要用英文输入法！
> - 为什么这么写？就这么规定的，先别管了：）
> - handle_data 按天循环时，如此处，其中的操作都是在9:30执行。

几乎所有策略都基于这个基本的策略框架：先初始化，然后循环操作
1 初始化，即最开始要做的事情，如选定股票，设置变量、参数等等
2 周期循环：即每个周期要做的事情，如计算指标，买入卖出等，周期可能是分钟，天等，本文策略的周期是一天。当你要做一些盘中短线操作的时候，周期就要调成分钟，先别着急会遇到的。

2 初始化

我们要写设置要交易的股票的代码，比如 兔宝宝（002043），真的有这个股票哦。

```
def initialize(context):
    g.security = '002043.XSHE'# 存入兔宝宝的股票代码
```

>答疑与延伸：
>- "g." 是什么？全局变量前都要写"g."，全局变量就是全局都能用的变量，一般变量只能在该函数下使用。如security不加"g.",只能在第一部分即initialize里用，不能在第二部分handle_data里用。
>- 什么是变量？，可以当变量是各种存放数据的容器，每个都要有个名字，比如`g.security = '002043.XSHE'`，就是把数据'002043.XSHE'放到变量`g.security`中，如果变量中里面有别的数据会替换掉。具体到量化课堂的python编程里学习下基础内容，或者问问百度。
>- "XSHE" 是什么？ 股票代码使用时要加后缀，深交所股票代码后缀为 ".XSHE "，上交所股票代码后缀为 ".XSHG"。
>- 代码中“#" 是什么？"#“后的内容都是注释，是为代码做说明的，不会被计算机当做代码处理。

3 获取收盘价与均价

首先，获取昨日股票的收盘价

```
# 用法：变量 = data[股票代码].close
last_price = data[g.security].close# 取得最近日收盘价，命名为last_price
```

然后，获取近二十日股票收盘价的平均价

```
# 用法：变量 = data[股票代码].mavg(天数，‘close’)
# 获取近二十日股票收盘价的平均价，命名为average_price
average_price = data[g.security].mavg(20, 'close')
```

>答疑与延伸：
>- data是什么?
>- data[股票代码]后面除了平均价.mavg还能接什么?
>- 这些写法其实都是python写法，如果你觉得不好“理解”可以先记住。

4 判断是否买卖

数据都获取完，该做买卖判断了

```
# 如果昨日收盘价高出二十日平均价，则买入，否则卖出
if last_price > average_price:
    买入
elif last_price < average_price:
    卖出
```

问题来了，现在该写买卖下单了，但是拿多少钱去买我们还没有告诉计算机，所以每天还要获取账户里现金量。

```
# 用法：变量 = context.portfolio.cash
cash = context.portfolio.cash# 取得当前的现金量，命名为cash
```

>答疑与延伸：
>- context.portfolio 是什么?

这句看着有点复杂，先记住吧。然后我们看看买入卖出怎么写。

5 买入卖出

```
# 用法：order_value(要买入股票股票的股票代码，要多少钱去买）
order_value(g.security, cash)# 用当前所有资金买入股票
# 用法：order_target(要买卖股票的股票代码，目标持仓金额）
order_target(g.security, 0)# 将股票仓位调整到0，即全卖出
```

>答疑与延伸：
>- 为什么没有指定交易价格？此策略是按天回测进行的且使用的较为简单的市价单下单方法，交易价格为开盘价(加上滑点)
>- 无法交易的情况？涨跌停，停牌，T+1制度等无法交易的情况，系统会自动使下单不成交并在日志中发出警告。
>- 滑点是什么？简言之是为成交误差留出余地。
>- 下单方法有哪些?

6 策略代码写完，进行回测

把买入卖出的代码写好，策略就写完了，如下

```
def initialize(context):#初始化
    g.security = '002043.XSHE'# 股票名:兔宝宝
def handle_data(context, data):#每日循环
    last_price = data[g.security].close# 取得最近日收盘价
    # 取得过去二十天的平均价格
    average_price = data[g.security].mavg(20, 'close')
    cash = context.portfolio.cash# 取得当前的现金
    # 如果昨日收盘价高出二十日平均价，则买入，否则卖出。
    if last_price > average_price:
        order_value(g.security, cash)# 用当前所有资金买入股票
    elif last_price < average_price:
        order_target(g.security, 0)# 将股票仓位调整到0，即全卖出
```

现在，在策略回测界面右上部，设置回测时间从20140101到20160601，设置初始资金100000，设置回测频率，然后点击运行回测。

> 答疑与延伸：

> - 什么是回测？回测是量化交易策略研究中的关键，是指给定一段时间的历史数据（如此处是20140101到20160601的每日数据），计算机按照所编写的策略进行模拟仿真交易，以测试策略效果好坏。

如果你代码没有问题，就会顺利的进行回测，回测结果如下图：

至此，你就完成了一个简单策略的回测了。

> 答疑与延伸：

> - 如何根据回测结果评价策略好坏？很初级地讲，有三：

> 1. 盈利能力：策略收益与年化收益高，则说明盈利能力强。盈利能力不行说啥都没用。

> 1. 盈利稳定性：最大回撤要低。最大回撤是指最大亏损幅度，50%则意味着历史上看最大亏损率为50%。

> 1. 回测可靠性：交易次数要多。交易次数越多意味着经历了越多次的检验，回测的结果也越可靠。

> 更多说明见：风险指标说明

> - 这个策略回撤大，交易次数少，只交易一只股票，并不靠谱。但是结构简单适合新手入门理解整个流程。

7 建立模拟交易，使策略和行情实时连接自动运行

策略写好，回测完成，点击回测结果界面（如上图）右上部红色模拟交易按钮，新建模拟交易如下图。

写好交易名称，设置初始资金，数据频率，此处是每天，设置好后点提交。

>答疑与延伸：

>- 模拟交易创建成功后，需要等待A股至少开盘一次，才能查看模拟交易结果。

8 开启微信通知，接收交易信号

点击聚宽导航栏我的交易，可以看到创建的模拟交易，如下图。

点击右边的微信通知开关，将OFF调到ON，按照指示扫描二维码，绑定微信，就能微信接收交易信号了。

当策略买卖操作，微信会收到信号提醒类似下图。

>答疑与延伸：

>- 能不能自动下单？目前不能，国家管制。你可根据信号手动下单买卖，施行策略。

自测与自学

1. 能否理解整个策略框架。

2. 能否成功编写单股票均线策略，成功回测，建立模拟，开启微信通知。
3. 能否理解年化收益，最大回撤。
4. 浏览聚宽新手指南，干货，答疑，指路，一应俱全。

摘要回顾：

1. 确定策略内容与框架
2. 初始化
    - 全局变量 g.
    - 股票后缀
    - 注释格式 #
3. 获取收盘价与均价
    - 获取二十日均价
    - 获取收盘价
4. 判断是否买卖
    - 利用 if 进行判断
    - 获取当前账户现金
5. 买入卖出
    - 下市价单
    - 滑点
    - 下单方法有哪些
    - 无法交易的情况
6. 进行回测
    - 回测含义及其方法
    - 如何根据回测结果评价策略
7. 建立模拟交易，行情实时连接
8. 开启微信通知，接收交易信号

【量化课堂】什么是量化投资?

什么是量化投资?

提起量化投资，就不得不提量化投资的标杆——华尔街传奇人物詹姆斯·西蒙斯(James Simons)。

通过将数学理论巧妙融合到投资的实战之中，西蒙斯成为了投资界中首屈一指的“模型先生”。由其运作的大奖章基金(Medallion)在1989-2009的二十年间，平均年收益率为35%，若算上44%的收益提成，则该基金实际的年化收益率可高达60%，比同期标普500指数年均回报率高出20多个百分点，即使相较金融大鳄索罗斯和股神巴菲特的操盘表现，也要遥遥领先十几个百分点。最为难能可贵的是，纵然是在次贷危机全面爆发的2008年，该基金的投资回报率仍可稳稳保持在80%左右的惊人水准。西蒙斯通过将数学模型和投资策略相结合，逐步走上神坛，开创了由他扛旗的量化时代。

说回量化投资，量化投资就是利用计算机技术并且采用一定的数学模型去实践投资理念，实现投资策略的过程。

价值投资和趋势投资（技术分析）是引领过去一个世纪的投资方法，随着计算机技术的发展，已有的投资方法和计算机技术相融合，产生了量化投资。

量化投资的优势

量化投资的优势在于纪律性、系统性、及时性、准确性和分散化。

1. 纪律性：严格执行投资策略，不是投资者情绪的变化而随意更改。这样可以克服人性的弱点，如贪婪、恐惧、侥幸心理，也可以克服认知偏差。
2. 系统性：量化投资的系统性特征包括多层次的量化模型、多角度的观察及海量数据的观察等。多层次模型包括大类资产配置模型、行业选择模型、精选个股模型等。多角度观察主要包括对宏观周期、市场结构、估值、成长、盈利质量、市场情绪等多个角度分析。此外，海量数据的处理能力能够更好地在广大的资本市场捕捉到更多的投资机会，拓展更大的投资机会。
3. 及时性：及时快速地跟踪市场变化，不断发现能够提供超额收益的新的统计模型，寻找新的交易机会。
4. 准确性：准确客观评价交易机会，克服主观情绪偏差，从而盈利。
5. 分散化：在控制风险的条件下，量化投资可以充当分散化投资的工具。表现为两个方面：一是量化投资不断地从历史中挖掘有望在未来重复的历史规律并且加以利用，这些历史规律都是较大概率取胜的策略；二是依靠筛选出股票组合来取胜，而不是一只或几只股票取胜，从投资组合的理念来看也是捕捉大概率获胜的股票，而不是押宝到单个股票。

如何进行量化投资呢?

使用量化策略是进行量化投资的有效方式。
量化交易新手指南
什么是量化策略?
如何使用JoinQuant编写量化策略?

【量化课堂】什么是量化策略?

什么是策略?

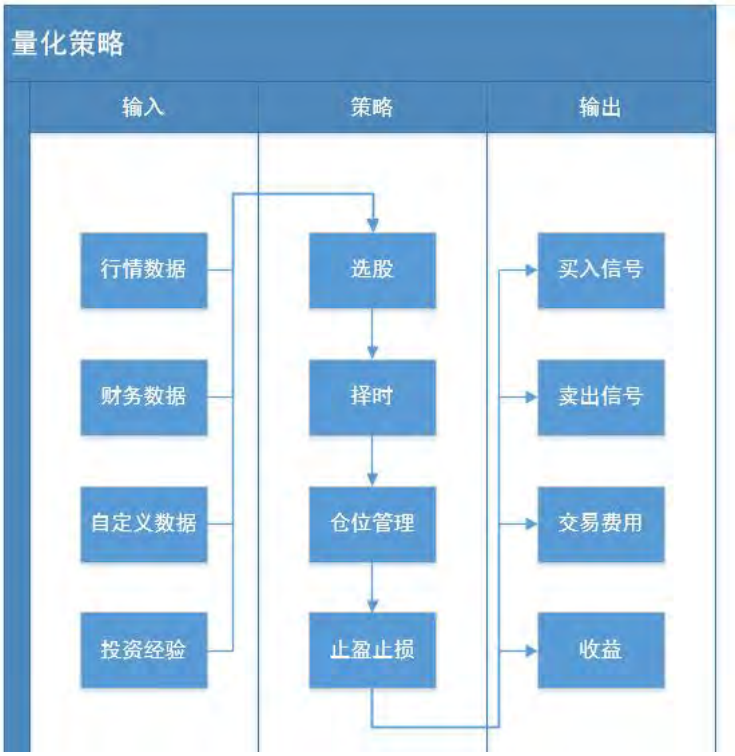
策略，可以实现目标的方案集合；在证券交易中，策略是指当预先设定的事件或信号发生时，就采取相应的交易动作。

什么是量化策略?

量化策略是指使用计算机作为工具，通过一套固定的逻辑来分析、判断和决策。
量化策略既可以自动执行，也可以人工执行。

一个完整的量化策略包含哪些内容?

一个完整的策略需要包含输入、策略处理逻辑、输出；策略处理逻辑需要考虑选股、择时、仓位管理和止盈止损等因素。



选股

量化选股就是用量化的方法选择确定的投资组合，期望这样的投资组合可以获得超越大盘的投资收益。常用的选股方法有多因子选股、行业轮动选股、趋势跟踪选股等。

1 多因子选股

多因子选股是最经典的选股方法，该方法采用一系列的因子（比如市盈率、市净率、市销率等）作为选股标准，满足这些因子的股票被买入，不满足的被卖出。比如巴菲特这样的价值投资者就会买入低PE的股票，在PE回归时卖出股票。

2 风格轮动选股

风格轮动选股是利用市场风格特征进行投资，市场在某个时刻偏好大盘股，某个时刻偏好小盘股，如果发现市场切换偏好的规律，并在风格转换的初期介入，就可能获得较大的收益。

3 行业轮动选股

行业轮动选股是由于经济周期的原因，有些行业启动后会有其他行业跟随启动，通过发现这些跟随规律，我们可以在前者启动后买入后者获得更高的收益，不同的宏观经济阶段和货币政策下，都可能产生不同特征的行业轮动特点。

4 资金流选股

资金流选股是利用资金的流向来判断股票走势。巴菲特说过，股市短期是投票机，长期看一定是称重机。短期投资者的交易，就是一种投票行为，而所谓的票，就是资金。如果资金流入，股票应该会上涨，如果资金流出，股票应该下跌。所以根据资金流向就可以构建相应的投资策略。

5 动量反转选股

动量反转选股方法是利用投资者投资行为特点而构建的投资组合。索罗斯所谓的反身性理论强调了价格上涨的正反馈作用会导致投资者继续买入，这就是动量选股的基本根据。动量效应就是前一段强势的股票在未来一段时间继续保持强势。在正反馈到达无法持续的阶段，价格就会崩溃回归，在这样的环境下就会出现反转特征，就是前一段时间弱势的股票，未来一段时间会变强。

6 趋势跟踪策略

当股价在出现上涨趋势的时候进行买入，而在出现下降趋势的时候进行卖出，本质上是一种追涨杀跌的策略，很多市场由于羊群效用存在较多的趋势，如果可以控制好亏损时的额度，坚持住对趋势的捕捉，长期下来是可以获得额外收益的。

择时

量化择时是指采用量化的方式判断买入卖出点。如果判断是上涨，则买入持有；如果判断是下跌，则卖出清仓；如果判断是震荡，则进行高抛低吸。
常用的择时方法有：趋势量化择时、市场情绪量化择时、有效资金量化择时、SVM量化择时等。

仓位管理

仓位管理就是在你决定投资某个股票组合时，决定如何分批入场，又如何止盈止损离场的技术。
常用的仓位管理方法有：漏斗型仓位管理法、矩形仓位管理法、金字塔形仓位管理法等

止盈止损

止盈，顾名思义，在获得收益的时候及时卖出，获得盈利；止损，在股票亏损的时候及时卖出股票，避免更大的损失。
及时的止盈止损是获取稳定收益的有效方式。

策略的生命周期

一个策略往往会经历产生想法、实现策略、检验策略、运行策略、策略失效几个阶段。

产生想法

任何人任何时间都可能产生一个策略想法，可以根据自己的投资经验，也可以根据他人的成功经验。

实现策略

产生想法到实现策略是最大的跨越，实现策略可以参照上文提到的“一个完整的量化策略包含哪些内容？”

检验策略

策略实现之后，需要通过历史数据的回测和模拟交易的检验，这也是实盘前的关键环节，筛选优质的策略，淘汰劣质的策略。

实盘交易

投入资金，通过市场检验策略的有效性，承担风险，赚取收益。

策略失效

市场是千变万化的，需要实时监控策略的有效性，一旦策略失效，需要及时停止策略或进一步优化策略。

【JoinQuant 的正确打开方式】纯小白的社区使用攻略

更新：聚宽新手指南

分享一下自己的一些使用心得，希望给后来的小白们提供一些方便。

大神们可以出门左转，当然也欢迎补充~

以下是JQ的正确打开方式：

1.API文档——2.干货贴——3.多交流，有问题随时提——4.实战

1.API文档

先过一遍，不用细究。随用随查。我自己是打印了一份放手边方便查询。
JQ的API文档感觉写的很详细全面了。

2.干货贴

【干货合集目录】你想要的，都在这里！

这篇帖子对我个人而言帮助很大。
集合了很多量化资料，python量化编程的教学，还有很多示例策略。
最关键的是照着源码、照着策略学习会很快。无论是编程还是交易思想。

3.多交流，有问题随时提

提问
一处是QQ群，不是很显眼的位置。。一般使用中遇到的各种问题，JQ的人会很快解答。

提需求
还有一处是提需求的帖子，搬运过来：小伙伴，你希望JoinQuant添加哪些内容呢?

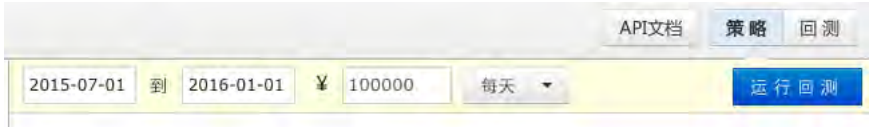
4.实战

研究功能

- 还没怎么了解，先搬运一份使用帮助：研究功能使用帮助

策略编写

- 很好用。我在学习阶段自己编写的比较少，都是克隆社区了的策略研究学习。



- 一个策略可以更改代码和回测区间以及回测频率等做很多次回测。

运行回测后，会展示收益曲线图。

- 常用的功能在页面左侧，包括每日持仓&收益，交易详情，可以调出来一笔笔分析买卖的效果，对策略做改进。

模拟交易

- 自己觉得满意的策略，运行回测后可以在右上角看到模拟交易。
- 这个功能很喜欢，绑定微信之后，策略发出买卖信号会直接微信收到提醒。

最后

社区里庸俗神父整理的一些心得，包括很多API使用的小技巧，看看可以避免之后遇到同样的问题不知道怎么解决。
搬运一下：
JoinQuant 心得——股票行情数据
JoinQuant 心得——时间持仓资金数据

JoinQuant 心得——基本面数据
JoinQuant 心得——订单
JoinQuant 心得——回测功能性完善
JoinQuant 心得——数据存取
JoinQuant 心得——小技巧

【量化课堂】如何使用JoinQuant 编写策略?

【新】新手向文章 十行代码带你量化交易入门

写在前面的话:

使用JoinQuant编写量化策略需要具备一定的金融知识和编程基础，如果你不会使用Python编程，推荐你用几天时间学习一下这篇Python基础教程

如何使用JoinQuant 取数据?

(1)取行情数据
详见：股票行情数据
(2)取财务数据
详见：基本面数据

如何设置股票池?

可以使用get_index_stocks、get_industry_stocks或是自行设置股票池列表。
例如:

```
# 获取所有沪深300的股票，设为股票池
stocks = get_index_stocks('000300.XSHG')
set_universe(stocks)
```

或者指定自定义的股票池:

```
stocks = ['000009.XSHE','002222.XSHE','000005.XSHE','000002.XSHE']
set_universe(stocks)
```

如何买入卖出股票?

详见：订单
下订单方法

```
1. order
```

API原文：order
买卖一定数量（单位：股）股票。

```
2. order_target
```

API原文：order_target
通过买卖，将股票仓位调整至一定数量（单位：股）。

```
3. order_value
```

API原文：order_value
买卖一定价值量（单位：元）股票。

```
4. order_target_value
```

API原文：order_target_value
通过买卖，将股票仓位调整至一定价值量（单位：元）。

如何止盈止损?

context.portfolio.positions中包含持有的某个股票的信息，详见：positions
可以使用持有的每只股票的持仓成本与当前价格实现止盈或止损，示例如下:

```
avg_cost = context.portfolio.positions[stock].avg_cost
price = context.portfolio.positions[stock].price
# 收益50%止盈
if (price/avg_cost) >= 1.5:
    order_target(stock, 0)
# 亏损10%止损
if (price/avg_cost) <= 0.9:
    order(stock, amount)
```

如何实现一个最简单的策略?

API文档中提供了几个策略，这几个策略包含了大部分的使用方法。
详见：策略示例
下面对用户需要实现几个函数做下简单介绍：

```
1. [initialize](https://www.joinquant.com/api#initialize)
```

初始化方法，在整个回测、模拟实盘中最开始执行一次，用于初始一些全局变量，如设置基准、交易的手续费、股票池或滑点等等。示例如下：

```
def initialize(context):
    # 设定沪深300为基准
    set_benchmark('000300.XSHG')
    # 调用此函数设置手续费，每笔交易时的手续费是，买入时万分之三，卖出时万分之三加千分之一印花税，每笔交易最低扣5块钱
    set_commission(PerTrade(buy_cost=0.0001, sell_cost=0.001, min_cost=5))
    # 调用此函数设置滑点
    set_slippage(PriceRelatedSlippage(0.002))
```

```
2. [handle_data](https://www.joinquant.com/api#handledata)
```

该函数每个单位时间会调用一次, 如果按天回测,则每天调用一次,如果按分钟,则每分钟调用一次。
函数内部就是你的交易思路，详情可参考示例代码。

```
3. [before_trading_start](https://www.joinquant.com/api#beforetradingstart-可选) 和 [after_trading_end](https://www.joinquant.com/api#aftertradi
```

这两个函数与handle_data基本相同，只是没有传入data参数。before_trading_start 会在每天开始交易前被调用一次，而after_trading_end 会在每天结束交易后被调用一次。

如何使用自定义消息?

JoinQuant提供了微信消息推送的功能，API为send_message
教程见:send_message用法

实例代码：

(1) 清仓止损（发送消息）

```
def before_trading_start(context):
    g.is_stop = dp_stoploss(kernel=2, n=10, zs=0.03)
    if g.is_stop:
        if len(context.portfolio.positions.keys())>0:
            for stock in context.portfolio.positions.keys():
                order_target(stock, 0)
        send_message("清仓")
        return
```

(2) 购买股票（发送股票池）

```
def before_trading_start(context):
    g.is_stop = dp_stoploss(kernel=2, n=10, zs=0.03)
    df = get_fundamentals(query(
                    valuation.code, valuation.market_cap
                ).filter(
                    valuation.code.in_(chosed_stocks)
                ).order_by(
                # 按市值降序排列
                valuation.market_cap.asc()
            ))
    g.per_buylist = list(df['code'])
    send_message(g.per_buylist)
```

如何实现常用技术指标?

指数平滑均线

```
# 指数平滑均线函数，以收盘价计算，可以换开盘价等其他价格，N为时间周期，m用于计算平滑系数a=m/(N+1)
def f_expma(N,m):
    # 获取EXPMA初始值为回测时间段和上市交集的前N-1个收盘价均值
    global init_price,EXPMA2,EXPMA1
    close_price = history(N, unit='1d', field='close', security_list=None)
    if init_price is None and not math.isnan(close_price[security][1]):
        init_price = close_price[security].mean()#nan和N-1个数，mean为N-1个数的均值
        EXPMA2 = init_price
    EXPMA1 = EXPMA2
    #log.info(EXPMA1)
    #log.info(EXPMA2)
    if not math.isnan(close_price[security][0]):
        # 回测T时间，取得T-1的收盘价
        current_close = close_price[security][-1]
```

```
        a = m/(N+1)
        EXPMA2 = a * current_close + (1 - a)*EXPMA1
        log.info(EXPMA1)
        log.info(EXPMA2)
    return EXPMA1,EXPMA2#1为前一天值，2为后一天值
```

MACD

```
import talib
def MACD(prices, fastperiod=12, slowperiod=26, signalperiod=9):
    '''
    参数设置:
        fastperiod = 12
        slowperiod = 26
        signalperiod = 9

    返回: macd - signal
    '''
    macd, signal, hist = talib.MACD(prices,
                                    fastperiod=fastperiod,
                                    slowperiod=slowperiod,
                                    signalperiod=signalperiod)
    return macd[-1] - signal[-1]
```

KDJ死叉金叉

```
#定义KDJ计算函数，输入为基期长度count、平滑因子a，输出为KDJ指标值。
#K1为前一日k值,D1为前一日D值,K2为当日k值,D2为当日D值,J为当日J值
def KDJ(count,a,b,K1,D1):
    h = attribute_history(security, count, unit='1d',fields=('close', 'high', 'low'),skip_paused=True)
    # 取得过去count天的最低价格
    low_price = h['low'].min()
    # 取得过去count天的最高价格
    high_price = h['high'].max()
    # 取得当日收盘价格
    current_close = h['close'][-1]
    if high_price!=low_price:
        #计算未成熟随机值RSV(n)＝（Ct－Ln）/（Hn-Ln）×100
        RSV = 100*(current_close-low_price)/(high_price-low_price)
    else:
        RSV = 50
    #当日K值=(1-a)×前一日K值+a×当日RSV
    K2=(1-a)*K1+a*RSV
    #当日D值=(1-a)×前一日D值+a×当日K值
    D2=(1-b)*D1+b*K2
    #计算J值
    J2 = 3*K2-2*D2
    return K1,D1,K2,D2,J2
```

RSI指数

```
def calRSI(stocks):
    rsi = []
    for stock in stocks:
        # 获取股票的收盘价数据,talib参数取14，前14天的rsi无法计算，所以取15天的数据
        prices = attribute_history(stock, 15, '1d', ('close'))
        # 创建RSI买卖信号，包括参数timeperiod
        # 注意：RSI函数使用的price必须是narray
        rsi += [talib.RSI(prices['close'].values, timeperiod=14)[-1]]

    return rsi
```

OBV指标

```
#定义多空力量比率加权修正成交量的obv函数
def obv(df,init_obv):
    obv1 = []
    for i in range(0,len(df)):
        jinge = ((df['close'].values[i]-df['low'].values[i])-(df['high'].values[i]-df['close'].values[i]))/(df['high'].values[i]-df['low'].value
        if i==0:
            obv1.append(init_obv+jinge*df['volume'].values[i])
        else:
            obv1.append(obv1[-1]+jinge*df['volume'].values[i])
    obv1=np.array(obv1)
    return obv1
```

JoinQuant（聚宽）量化交易平台是为量化爱好者、宽客量身打造的云平台，我们致力于为用户打造最高效、易用的量化交易平台，我们坚信“人人皆为宽客”—任何人只要对量化交易感兴趣，就可以成为一名宽客，我们希望降低量化交易的门槛，让更多人有机会参与进来；目前我们主要针对沪深A股、ETF提供服务；我们免费提供高质量数据、强大的研究平台、顶级回测体验、顶尖模拟交易、量化交流社区。

我们提供的服务

我们免费为量化爱好者、宽客提供如下服务：

1 高质量数据

我们的数据基于2005年至今完整的Level-2数据，包含完整的停牌、复权等信息，盘后及时更新。 目前提供的数据主要基于沪深A股、ETF，后续我们会逐步支持期货、现货、外汇等其他金融衍生品。

2 强大的研究平台

我们免费提供IPython Notebook研究平台，提供分钟级数据，采用Docker技术隔离，资源独立、安全性更高、性能更好，同步支持Python2、Python3。
立即体验>>

3 顶级回测体验

我们支持对沪深A股、ETF进行回测。 我们支持每日、分钟两级回测，提供简洁、强大的API，回测结果实时显示、快速响应、数据全面。
立即体验>>

4 顶尖模拟交易

我们提供最准确、实时的沪深A股、ETF模拟交易工具，支持基于tick级的模拟交易。
立即体验>>

5 量化交流社区

我们为量化爱好者提供线上交流社区，便于用户交流量化策略、学习量化知识，一起成长。
立即体验>>

6 实盘交易

努力开发中，敬请期待
我们希望提供最准确、实时的实盘交易工具，帮助您使用策略赚钱。

7 其他金融衍生品

努力开发中，敬请期待
我们希望为所有量化爱好者服务，我们会逐步支持期货、现货、外汇等其他金融衍生品。

我们的优势

1 用户第一

追求极致的用户体验，快速响应用户需求，为用户创造价值。

2 尊重知识产权

策略是宽客最大的财富，用户对策略拥有100%的知识产权。

3 保障策略安全

我们提供业界最高的安全级别，通过https传输、加密存储、沙箱保护保障您的策略安全，任何人无法获取您的策略。

4 高质量数据

我们提供的数据基于2005年至今完整的Level-2数据，包含完整的停复牌、复权、退市等信息，盘后及时更新。

5 强大的研究平台

我们提供分钟级数据，采用Docker技术隔离，资源独立、安全性更高、性能更好，同步支持Python2、Python3。

6 顶级回测体验

我们支持每日、分钟两级回测，提供简洁、强大的API，回测结果实时显示、快速响应、数据全面。

7 顶尖模拟交易

我们提供最准确、实时的沪深A股、ETF模拟交易工具，支持基于tick级的模拟交易。

8 量化交流社区

我们为量化爱好者提供线上交流社区，便于用户交流量化策略、学习量化知识，一起成长。

我们的团队

我们的创始团队具有丰富的金融、互联网从业经验，既有超过10年炒股经验、持续跑赢大盘的炒股大师，也有多年从事基金、证券行业的金融精英，还有BAT的技术大牛，我们致力于打造最高效、易用的量化交易平台。

常见问题

1 JoinQuant 收费吗?

JoinQuant（聚宽）量化交易平台是为量化爱好者、宽客量身打造的云平台，我们提供的服务均免费。 我们免费为您提供高质量数据、顶级回测服务、顶尖模拟交易、量化交流社区、IPython Notebook研究平台，便于您快速实现、使用自己的量化交易策略。

2 策略归谁所有?

策略是宽客最大的财富，您对策略拥有100%的知识产权。

3 我的策略是否安全?

我们提供业界最高的安全级别，通过https传输、加密存储、沙箱保护保障您的策略安全，除了您本人，任何人无法获取您的策略。

联系我们

欢迎您通过社区和我们沟通交流。

如果您有任何疑问，请发邮件到hi@JoinQuant.com或者加QQ群429620025告诉我们，谢谢。

【看过来】PM 教你计算股息率（报告期）！（内附函数，可copy 走直接使用）

JoinQuant 近来新增了国泰安的数据，其中包含分红数据，看到有用户获取股息率的需求，现在小编便教大家使用分红数据计算股息率。

股息率的计算公式:

$$股息率 = \frac{每股分红(税前)}{股票当前价} \times 100\%$$

小编定义了 dividend_yield_ratio 函数，可以取到股息率(报告期)，并跟Wind进行了对比，结果是一样的。

现在你可以拷贝走直接放在自己的策略中调用，详情参见下面分享的研究模块:

获取股息率¶

JoinQuant 近来新增了国泰安的数据，其中包含分红数据，看到有用户获取股息率的需求，现在小编便教大家使用分红数据计算股息率。

股息率的计算公式:

$$股息率 = \frac{每股分红(税前)}{股票当前价} \times 100\%$$

下面定义了 dividend_yield_ratio 函数，可以取到股息率(报告期)，具体用法见函数说明:

In [3]:

```
def dividend_yield_ratio(stock, start_date=None, count=None, date=None, end_date=None, paymentdate=2015, no_data_return=NaN, skip_pau
    '''
    stock: 股票代码
    start_date: （start_date 与 count二选一）获取股息率开始日期
    count:  （start_date 与 count二选一）指定获取之前几个交易日
    end_date: 获取股息率结束日期
    paymentdate: 报表年度，默认报告年底2015年
    no_data_return: 选填 NaN 或 0，决定没有股息率的股票返回的数据是 NaN 或 0
    skip_paused: 跳过停牌，默认不跳过
    '''
    def get_div(df, paymentdate):
```

```
        import pandas as pd
        df.PAYMENTDATE = pd.to_datetime(df.PAYMENTDATE)
        try:
            if len(df)>0:
                div = df[[x.year == (int(paymentdate)+1) for x in df.PAYMENTDATE]]['DIVIDENTBT'].iloc[-1]
                return float(div)
            else:
                return no_data_return
        except:
            return no_data_return

    # 导入jqdata库
    from jqdata import gta

    # 获取股票六位代码
    symbol = stock[:6]

    # 获取
    df = gta.run_query(query(
            gta.STK_MKT_DIVIDENT.SYMBOL,
            gta.STK_MKT_DIVIDENT.PAYMENTDATE,
            gta.STK_MKT_DIVIDENT.DIVIDENTBT,
            ).filter(gta.STK_MKT_DIVIDENT.SYMBOL.in_([symbol])
            ).order_by(gta.STK_MKT_DIVIDENT.PAYMENTDATE
        ))

    div = get_div(df, paymentdate)

    # 获取股票最近几天的价格
    price = get_price(stock, start_date=start_date, count=count, end_date=end_date, \
                      fields=['close'], frequency='daily', fq=None)

    # 计算股息率
    price['close'] = float(div)/price['close']*100

    # 将股息率精确到小数点后4位
    price.close = price.close.round(4)

    # 返回结果
    return price
```

以平安银行('000001.XSHE')为例，讲解一下计算方法：

In [4]:

```
security = '000001.XSHE'
count = 3
starttime = '2016-07-19'
endtime = '2016-07-21'
paymentdate = 2015

test_dyr = dividend_yield_ratio(stock=security, start_date=None, count=count, end_date=endtime,\
                                paymentdate=paymentdate, no_data_return=0)
test_dyr
```

Out[4]:

	close
2016-07-19	1.7057
2016-07-20	1.7076
2016-07-21	1.7019

国农科技('000004.XSHE')如下：

In [5]:

```
security = '000004.XSHE'
count = 3
starttime = '2016-07-19'
endtime = '2016-07-21'
paymentdate = 2015

test_dyr_0 = dividend_yield_ratio(stock=security, start_date=None, count=count, end_date=endtime,\
                                paymentdate=paymentdate, no_data_return=0)
test_dyr_0
```

Out[5]:

	close
2016-07-19	0
2016-07-20	0
2016-07-21	0

也可指定返回值为 `NaN` ,只需指定 `no_data_return=NaN`, 方法如下：

In [6]:

```
test_dyr_nan = dividend_yield_ratio(stock=security, start_date=None, count=count, end_date=endtime,\
                                paymentdate=paymentdate, no_data_return=NaN)
test_dyr_nan
```

Out[6]:

	close
2016-07-19	NaN
2016-07-20	NaN
2016-07-21	NaN

也许你会怀疑小编计算的方法对不对，请看下图，内容截自Wind：

股息率（报告期），7月19-7月21日，报告年度2015

序号	证券代码	证券简称	股息率(报告期) [交易日期] 2016-07-19 [报表年度] 2015 [单位] %	股息率(报告期) [交易日期] 2016-07-20 [报表年度] 2015 [单位] %	股息率(报告期) [交易日期] 2016-07-21 [报表年度] 2015 [单位] %
1	000001.SZ	平安银行	1.7057	1.7076	1.7019
2	000002.SZ	万科A	4.2081	4.2155	4.2303
3	000004.SZ	国农科技	0.0000	0.0000	0.0000
4	000005.SZ	世纪星源	0.0000	0.0000	0.0000

再来一张报告年度为 2014 年的进行测试一下：

股息率（报告期），7月19-7月21日，报告年度2014

序号	证券代码	证券简称	股息率(报告期) [交易日期] 2016-07-19 [报表年度] 2014 [单位] %	股息率(报告期) [交易日期] 2016-07-20 [报表年度] 2014 [单位] %	股息率(报告期) [交易日期] 2016-07-21 [报表年度] 2014 [单位] %
1	000001.SZ	平安银行	1.9398	1.9420	1.9355
2	000002.SZ	万科A	2.9223	2.9274	2.9377
3	000004.SZ	国农科技	0.0000	0.0000	0.0000
4	000005.SZ	世纪星源	0.0000	0.0000	0.0000

平安银行：¶

In [7]:

```
security = '000001.XSHE'
count = 3
starttime = '2016-07-19'
endtime = '2016-07-21'
paymentdate = 2014

test_dyr = dividend_yield_ratio(stock=security, start_date=None, count=count, end_date=endtime,\
                                paymentdate=paymentdate, no_data_return=0)
test_dyr
```

Out[7]:

	close
2016-07-19	1.9398
2016-07-20	1.9420
2016-07-21	1.9355

万科A：¶

In [8]:

```
security = '000002.XSHE'
count = 3
starttime = '2016-07-19'
```

```
endtime = '2016-07-21'
paymentdate = 2014

test_dyr = dividend_yield_ratio(stock=security, start_date=None, count=count, end_date=endtime,\
                                paymentdate=paymentdate, no_data_return=0)
test_dyr
```

Out[8]:

	close
2016-07-19	2.9223
2016-07-20	2.9274
2016-07-21	2.9377

测试结果是一样的，小伙伴们可以放心的使用了！

In [ ]:

```
```

金融期货策略研究框架介绍及Demo演示

值JoinQuant 聚宽一周年生日之际，伴随着广大宽粉的期盼，我们推出了金融期货策略研究框架Beta版，现在面向全员进行公测。

> ### 主要功能:

1. 提供股指期货的回测与模拟
2. 可同时交易股指期货以及股票，提供各种风险指标
3. 支持分账户操作：股票、股指期货分账户操作，完全无干扰
4. 支持账户之间的资金转移功能，灵活调控每个账户的资金
5. 提供持仓均价, 开仓均价，盯市浮盈, 平仓盈亏、保证金等其他期货软件上都有的数据
6. 可自行设置交易手续费、平今仓手续费、保证金比例等
7. 在保证金不足时，会强制平仓
8. 交割日未卖出时，自动进行交割平仓
9. 完全符合历史上股指期货的开盘、闭盘时间
10. 实时更新保证金，并提供保证金预警功能，随时查看保证金的状态，避免因保证金不足导致的强制平仓

更多功能详见API

还有更多功能等你来发现，小伙伴们赶快来试用吧~~~

> ### 下面是一个简单的股指期货策略demo —— 期现套利：

股票现货市场和股指期货市场紧密相连，根据股指期货的制度设计，期货价格在合约到期日会与现货市场标的指数的价格相等。但实际行情中，期货指数价格常受多种因素影响而偏离其合理的理论价格，与现货指数之间的价格差距往往出现过大或过小的情况，一旦这种偏离出现，就会带来在期货市场和现货市场之间套利的机会，我们把这种跨越期市和现市同时进行交易的操作称之为期现套利。

期现套利有两种类型：当现货指数被低估，某个交割月份的期货合约被高估时，投资者可以卖出该期货合约，同时根据指数权重买进成份股，建立套利头寸。当现货和期货价格差距趋于正常时，将期货合约平仓，同时卖出全部成份股，可以获得套利利润，这种策略称为正向基差套利。

当现货指数被高估，某个交割月份的期货合约被低估时，如果允许融券，投资者可以买入该期货合约，同时按照指数权重融券卖空成份股，建立套利头寸。当现货和期货价格趋于正常时，同时平仓，获利了结，这是反向基差套利。

期现套利的实质是对现货指数和期货指数的基差进行投机。基差的变动是可以分析和预测的，分析正确可以获利，即使分析失误套利的风险也远比单向投机的风险低。

因为A股市场没有办法做空，这里只是进行了正向基差套利，我们选择300ETF代替一揽子股票，进行指数跟踪。

策略基本思路：

1. 当基差大于100则买入ETF，卖出股指期货；
2. 当基差缩小至20内，则获利清仓；

【有用功】教你如何调试程序(Debug)

在编写策略中经常会遇到各种各样的错误，十分繁琐，但这又是程序化交易必不可少的一步。

下面小编就教大家如何在 JoinQuant 进行代码调试。

其实在 JoinQuant 运行出错的原因类型也就那么几点。

1. python代码格式错误，如缩进、用法不对等 —— 这个自行百度或者 google 一下错误提示就基本可以搞定
2. 数据处理的问题，如对DataFrame、list、dict操作失误造成的 —— 学习一下相关处理方法即可，可在量化课堂Python 教学中学到相关的基础知识。
3. 还有一些奇奇怪怪的问题，学习随着自己写的程序多了，见的多了，可自行解决。

如下两图所示就是错误提示：

```
Traceback (most recent call last):
  File "kuanke/user_space.py", line 146, in exec_msg
    return getattr(self, func)(*msg['args'], **msg['kwargs'])
  File "kuanke/user_space.py", line 228, in init_code
    exec code_obj in code_scope
  File "user_code.py", line 34, in
    record(stock_price=current_price)
NameError: name 'current_price' is not defined
```

```
  File "user_code.py", line 25
    order_value(security, cash)
    ^
IndentationError: unexpected indent
```

调试的方法可概括为如下几点：
一、先看一下出错代码第几行，自行查看一下，先排除是拼写或者缩进等简单错误。

二、百度 或者 google 一下错误提示代码，如查询 `python unexpected indent`, 会有非常多的结果可供参考。
当然也可以去 Stack Overflow 查询，十分好用，强烈推荐。

三、把错误处前后变量打印出来看一下，是不是所需要的数据对象格式不同造成的。
可以使用 print 或者 try...excpt... ，把相关的变量输出看一下。

示例：

如下代码：

```
1 def initialize(context):
2     g.security = '000001.XSHE'
3     set_universe([g.security])
```

```
4
5 def handle_data(context, data):
6     get_lists(context)
7     print 'done'
8
9 def get_lists(context):
10    lists = get_all_securities('stock')
11    log.info(lists)
12    h = history(3, '1d', field='close', security_list = lists, df=False)
13    return [stock for stock in lists if h[stock][-1] < h[stock][0]]
```

运行之后会报错：

```
2016-06-01 09:30:00 - INFO -             display_name  name  start_date    end_date   type
000001.XSHE         平安银行  PAYH  1991-04-03  2200-01-01  stock
000002.XSHE         万　科A   WKA  1991-01-29  2200-01-01  stock
000004.XSHE         国农科技  GNKJ  1990-12-01  2200-01-01  stock
........

2016-06-01 09:30:00 - ERROR - Traceback (most recent call last):
  File "kuanke/user_space.py", line 146, in exec_msg
    return getattr(self, func)(*msg['args'], **msg['kwargs'])
  File "kuanke/user_space.py", line 319, in handle_data
    self.func_handle_data(self.user_context, user_space_api.SecuritiesData(context.current_dt))
  File "user_code.py", line 6, in handle_data
    get_lists(context)
  File "user_code.py", line 12, in get_lists
    h = history(60, '1d', field = ('close'), security_list = lists)
  File "kuanke/user_space_api.py", line 978, in history
    HistoryOptions(fq=fq, skip_paused=bool(skip_paused), df=bool(df)))
  File "/home/server/y/envs/kuanke/lib/python2.7/site-packages/functools32/functools32.py", line 380, in wrapper
    result = user_function(*args, **kwds)
  File "kuanke/user_space_api.py", line 943, in history_daily
    options=options,
  File "kuanke/user_space_utils.py", line 465, in get_price_daily_single
    a = get_all_day_data(security)
  File "/home/server/y/envs/kuanke/lib/python2.7/site-packages/functools32/functools32.py", line 380, in wrapper
    result = user_function(*args, **kwds)
  File "kuanke/user_space_utils.py", line 169, in get_all_day_data
    api_proxy.prepare_all_day_data(security)
  File "kuanke/user_space.py", line 73, in func
    raise ret
Exception: 股票"display_name"不存在
```

日志打印如上所示，看着眼花缭乱，有木有~~~~
虽说程序给出了 Exception，但明显不是看到不应该是这个问题（机器也没有那么智能）。

再看一下错误提示，发现提示程序第六行出现错误。
第六行我们调用了 `get_lists(context)` 函数，这里出现错误。
因此我们要查询一下 get_list(contest) 函数。
接着看错误提示，提示 line 12 出现错误，之后又提示了一大堆的错误。
接着我们要核查一下是不是 line 12，history 函数的问题, history 函数如下所示：

```
history(count, unit='1d', field='avg', security_list=None, df=True, skip_paused=False, fq='pre')
```

这种函数一般的错误就是传入参数的格式不对，可能是因为传参的顺序出错或者本身传入的格式就有问题。
我们对比一下程序发现，`count, unit, field`没什么错误，错误基本是 security_list 的问题。带着怀疑，我们应该打印一下 lists 看一下，是不是一个list。

为了方便，我的程序已经在第11行打印了传入的lists。
发现结果是一个多列的 DataFrame。因为我是想要返回结果的 index 数据。因此应该把 第10行改为`lists = get_all_securities('stock').index`,并输出 lists 看一下。lists 的结果如下：

```
2015-06-01 09:30:00 - INFO - Index([u'000001.XSHE', u'000002.XSHE', u'000004.XSHE', u'000005.XSHE',
       u'000006.XSHE', u'000007.XSHE', u'000008.XSHE', u'000009.XSHE',
       u'000010.XSHE', u'000011.XSHE',
       ...
       u'603969.XSHG', u'603979.XSHG', u'603986.XSHG', u'603988.XSHG',
       u'603989.XSHG', u'603993.XSHG', u'603996.XSHG', u'603997.XSHG',
       u'603998.XSHG', u'603999.XSHG'],
      dtype='object', length=3010)

2015-06-01 09:30:00 - INFO - done

2015-06-02 09:30:00 - INFO - Index([u'000001.XSHE', u'000002.XSHE', u'000004.XSHE', u'000005.XSHE',
       u'000006.XSHE', u'000007.XSHE', u'000008.XSHE', u'000009.XSHE',
       u'000010.XSHE', u'000011.XSHE',
       ...
       u'603969.XSHG', u'603979.XSHG', u'603986.XSHG', u'603988.XSHG',
       u'603989.XSHG', u'603993.XSHG', u'603996.XSHG', u'603997.XSHG',
       u'603998.XSHG', u'603999.XSHG'],
```

```
2015-06-02 09:30:00 - INFO - done

.....................
```

程序运行成功

当然在代码调试的过程中还会遇到其他各种各样的错误，需要你慢慢积累。

如 https://www.joinquant.com/post/1936 这篇帖子中的问题，程序会提示：

```
  File "user_code.py", line 101
    def chk_position(context):
      ^
SyntaxError: invalid syntax
```

但仔细检查，并没错误，真正的错误是由于第98行最后缺少一个括号，程序认为之后的是属于98行括号内的内容。

总结来说，大多数的错误其实通过搜索相关错误提示是可以找到解决办法；再者通过简单的调试与分析，也可以基本解决问题。

调试路漫漫，还望各位稍安勿躁、平心静气，毕竟修行是一辈子的事情~

【有用功】模拟盘替换代码注意事项

内容概述

1. 替换自己定义的函数
2. 修改变量值
3. 修改run_daily 运行时间
4. process_initialize 函数说明

正文

如果需要替换自定义编写的函数，直接使用新函数回测，并替换代码即可。

模拟盘替换代码之后，恢复过程如下，具体参看[模拟盘注意事项]:

1. 加载策略代码, 因为python是动态语言, 编译即运行, 所以全局的(在函数外写的)代码会被执行一遍.
2. 使用保存的状态恢复 g, context, 和函数外定义的全局变量.
3. 执行 process_initialize, 每次启动时都会执行这个函数.
4. 如果策略代码和上一次运行时发生了修改，而且代码中定义了 after_code_changed 函数，则会运行 after_code_changed 函数.

重启后不再执行 initialize 函数, initialize 函数在整个模拟盘的生命周期中只执行一次. 即使是更改回测后, initialize 也不会执行.

模拟盘更改回测之后上述的全局变量(包括 g 和 context 中保存的)不会丢失, run_daily() 函数依然会执行. 新代码中 initialize 不会执行.

如果需要修改原来的值, 可以在 after_code_changed 函数里面修改。

如果需要指定新的定期执行的函数, 调用 unschedule_all 清除原有 run_daily() 运行时间，再重新指定 run_daily() 运行时间。

比如, 原来代码是:

```
def initialize(context):
    g.stock = '000001.XSHE'
    run_daily(buy_stock, 'open')
```

现在想修改`g.stock = '000002.XSHE'`, run_daily 运行时间为 `before_open`，必须定义 after_code_changed:

```
def after_code_changed(context):
    unschedule_all()
    g.stock = '000002.XSHE'
    run_daily(buy_stock, 'before_open')
```

这样，程序便替换了。

下面再讲讲解一下`process_initialize(context)`函数:

现在模拟交易中 initialize 函数在整个模拟盘的生命周期中只执行一次，重启后不再执行，而一些初始化的数据不能被通过pickle序列化, 在模拟交易时进程重启后会丢失状态。因此我们构建了该函数。

process_initialize 函数会在每次模拟盘/回测进程重启时执行, 一般用来初始化一些不能持久化保存的内容.（如 get_fundamentals 中的 query 对象）

执行顺序是在 initialize 后执行.

因为模拟盘会每天重启, 所以这个函数会每天都执行.

示例:

```
def process_initialize(context):
    # query 对象不能被 pickle 序列化，所以不能持久保存，所以每次进程重启时对它初始化
    # 以两个下划线开始，系统序列化 [g] 时就会自动忽略这个变量，更多信息，请看 [g] 和 [模拟盘注意事项]
    g.__q = query(valuation
                ).filter(valuation.code == '000001.XSHE'
                ))

def handle_data(context, data):
    get_fundamentals(g.__q)
```

这样，g.__q 每天都会被初始化一遍，在 handle_data 中调用 get_fundamentals 不会出错。

【更新说明】新超额收益 和 对数轴

最近我们更新了超额收益的算法，以及增加了对数轴。接下来就为大家解释一下：

一、 超额收益

1. 错误的思路——减法版差额收益

一个最简单的想法是使用简单的超额收益，也就是“策略净值-基准净值”。但是这样做除了能看出我们策略比大盘多赚了多少钱，就没有别的作用了。

像这种回测倒还好啦，策略和基准的差异不是特别大，从黄线上升的趋势可以看出策略有稳定的超额收益。但是策略和基准的差距变大的话就彻底残废了。

你看由于策略的净值高出大盘好几十倍，所以减去基准的黄线和策略的蓝线基本都重合了，要这条黄线还有什么用?

我们看超额收益的一个想法的就是想知道如果按照策略进行交易，并且卖空基准，能得到多少的额外收益。那么这个简单相减的差值是个什么意思？假设策略净值和基准净值都是 100 万，那么相减出来的线表达了每买多一块钱策略并且卖空一块钱基准能获得的超额收益，这倒没问题。但是当我们的策略净值涨了十倍，而基准只涨了一倍，相减出来线告诉我们买多五块钱策略并且卖空一块钱基准能得到的差额，这时的对冲比例就没有任何的逻辑和意义了，它也不能展示给我们任何有用的信息。

再者说，一般我们也想知道策略在某一段时间中的表现是强于大盘还是弱于大盘。当大盘和策略净值一样的时候确实可以直接相减来看，但是假设经过一段时间的回测，策略净值是基准的十倍，然后在之后的一年里策略收益2%并且大盘收益 10%，所以策略在这一年里实际上是若于大盘的，但是却因为策略净值基数较大导致策略在这一年的净增长是大盘的两倍，所以相减的超额收益是上涨的，完全不能反应出策略弱于基准的这个事实，那么我们还需要更改回测的起始时间才能看出这个现象。

很明显相减而来的超额收益是不可取的，所以我们要另寻方法。

2. 正确的方法——除法版超额收益

由于投资的资产变动是有复利效应的，所以净值的变动是符合几何增长过程的，在这种情况中，最自然的“减法”不是减法，而是除法，所以一个更合理的超额收益算法是:

$$< br > \text{策略收益}/\text{基准收益} - 100\% < br >$$

先看一眼这条线的效果，第一印象就可以发现它和策略线分开了，不是继续黏在一起。

回到上面的例子，假如在时间T时策略的净值S是基准B的十倍，那么超额收益线就是:

$$< br > \frac{S}{B} - 100\% = 900\% < br >$$

假设在从 T 之后的一年里，策略涨幅 2% 并且基准涨幅 10%，那么，一年之后的超额收益线是:

$$< br > \frac{S < em > 1.02}{B < /em > 1.10} - 100\% = 1000\% * 92.7\% - 100\% = 827\% < br >$$

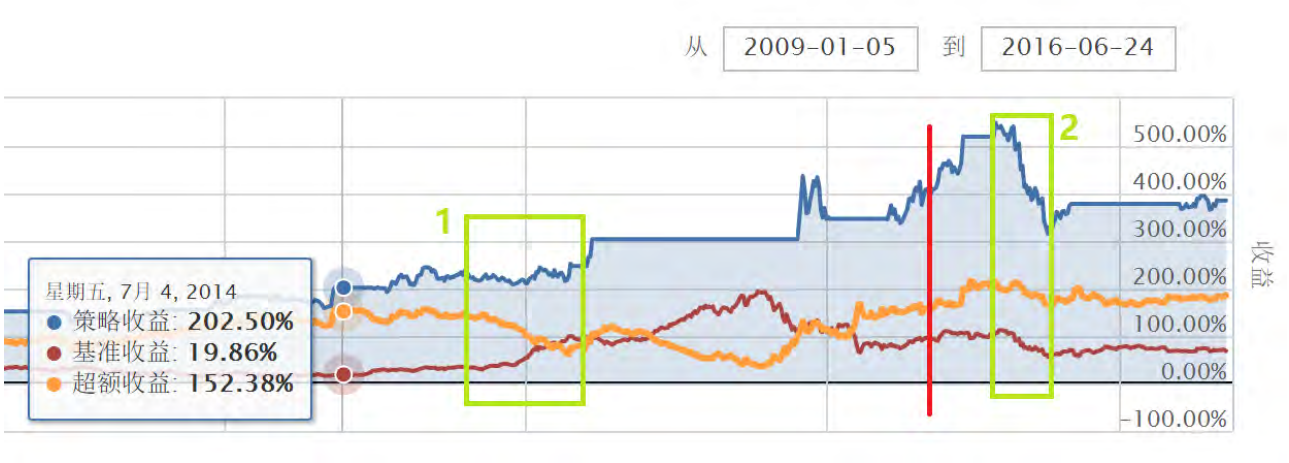
低于了之前的 900%，它表明了在这一年里策略跑输了基准。跑输了多少呢？算一下

$$< br > \frac{827\% + 100\%}{900\% + 100\%} = \frac{S < em > 1.02}{B < /em > 1.10} * \frac{B}{S} = \frac{1.02}{1.10} = 92.7\% < br >$$

告诉我们如果我们在时间T的时候用一块钱策略和一块钱基准一起跑（注意这很重要，不是十块钱策略对一块钱基准），那么在一年之后策略的净值只有基准的 92.7%。

使用除法产生的超额收益线上任意两点的数值都可以进行上面的计算，当然这里要展示的并不是教大家如何去算这个数，而是要让大家明白，只要除法版的超额收益线发生了回撤，就说明在这段时间里策略跑输了基准，而只要超额收益上涨了，就说明策略跑赢了基准，我们不需要再改时间重新回测就可以知道这个信息了。

举一个可以看出效果的例子，看下图标为 1 和 2 的部分:



- 在 1 的地方，策略和基准都涨了。谁涨的多？看黄线下跌了，所以是基准涨的多。
- 在 2 的地方策略和基准都跌了。谁跌的多？看黄线又是下跌了，所以是策略跌的多。
- 在 1 和 2 哪个地方策略输于基准更厉害？在1的地方黄线挖了一个坑，而在 2 的地方黄线下降并不多，所以虽然看起来2的地方策略跌得很厉害，但其实1的那段输于基准更多。

除此之外，我们知道回测在14年7月4号的超额收益指标是 150% 左右，并且在16年6月24号是接近 200%，并且根据这条黄线的形状，我们可以判断出：如果我从14年7月4号开始运行这个策略，它会在很长一段时间里跑输基准，然后在红色竖线的地方追平，最后在16年6月24号的地方净值大于基准；我们甚至不需要知道策略和基准的收益曲线都可以做出这个判断。

二、 对数轴

上面的的除法版超额收益线解决了多个时间序列在截面上比对的问题，但是同一序列在不同时间的对比还存在着问题。什么问题呢？就是我们没法看清策略在两个不同时间段的涨跌幅的区别。再回顾之前的一个回测的话，

要不是图上标出了最大回撤的位置，我们肯定目测不出最大回撤的位置。这是因为08年的时候策略净值还相对小，回撤个80%所损失的钱也不是那么多；而15年的时候净值是之前的几十倍，就算回撤20%损失也比之前回撤80%的损失多，所以在图上根本对比不出来。再者说，基准线呢？根本都看不见了啊。

和之前一样，这是由于策略净值有着复合增长效应，导致净值上很难表现出很多的信息。这里的解决办法就是把竖轴做一个变换，改成对数轴。

在对数轴的图上，策略（或者基准）在时间T时显示的高度是:

$$< br > log\left(\text{在时间}T\text{的净值}\right) < br >$$

就像我们之前从减法改成用除法一样，对数log可以把乘法变加法，把除法变减法，

$$< br > log\left(x * y\right) = log\left(x\right) + log\left(y\right) < br >$$

$$< br > log\left(x/y\right) = log\left(x\right) - log\left(y\right) < br >$$

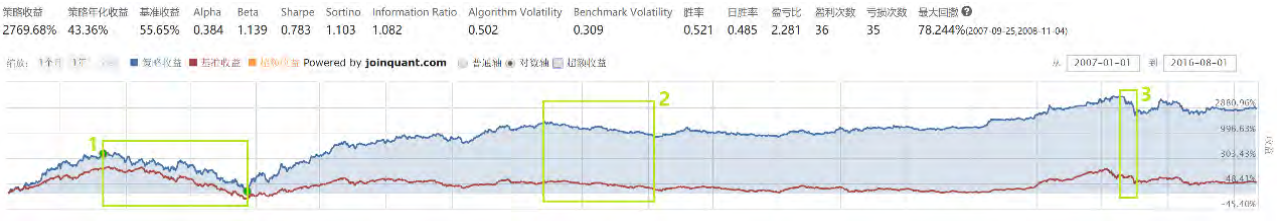
这样，在对数轴的图上，涨跌幅的倍数就不再是乘除关系而是加法关系，这样回测图上就能看出更丰富的信息了。

举例来说，策略在时间T的时候净值是 S，在 T+1 时是 2S，在 T+2 时是 4S，也就是说在每个时间段都翻了一倍。那么在普通轴上我们可以看出 T+1 的净值比 T 的净值高很多，但是 T+2 的净值比 T+1 高出更多，然而哪个是阶段的涨幅更大却很难看出。如果改用对数轴，那么在 T 时对数轴上高度是 $log(S)$；在

T+1 时是 $log(2S) = log(2) + log(S)$；在 T+2 时是 $log(4S) = 2log(2) + log(S)$。这三个数构成一个等差数列，也就是说在对数轴上它们相互之间的距离是一样的，很容易通过目测看出每个时间段的涨跌幅是一样的。

再举个最大回撤的例子。假设一策略在 08 年高峰时的净值是 S，并且在随后的股灾中回撤 80%，即损失 0.8S；它在 15 年高峰时净值有之前的五十倍，50S，并在股灾中回撤 50%，即 25S。在普通轴上我们根本不可能目测出哪个回撤更大，但如果换到对数轴上，在08年高峰和低点的对数值分别是 $log(S)$ 和 $log(S/5) = log(S) - log(5)$，那么对数轴上的回撤是 $log(5)$；在15年高峰和低点的对数值分别是 $log(50S)$ 和 $log(25S) = log(50S/2) = log(50S) - log(2)$，在对数轴上的回撤是 $log(2)$。由于 Misplaced &，我们可以很明显地看出 08 年的回撤大于 16 年。

除此之外，策略和基准在同一时间段中的对比也更简单了。假设基准在一段时间里翻了一倍，策略只涨了 50%，那么如果策略净值太大的话我们在普通轴上根本看不出基准有什么变动。在上边我们提到可以根据除法版超额收益的回撤来观测，而另一个方法就是在对数轴上看：在这段时间里基准在对数轴上上升了$log(2)$ 个单位，而基准只上升了 $log(1.5)$ 个单位，很直观，上升多的那个就是赢了。



我们把之前的回测放到对数轴上，净值规模造成的问题荡然无存。08 年回撤巨大，在图上清晰可见。相比之下，16 年的股灾（标注 3 的地方）虽然净值损失很多，但是实际上回撤比 08 年温柔很多，同时凭目测可以看出标注2的阴跌部分回撤和 16 年股灾中差不多。另外，基准的表现也一下清楚了是不是。

三、 对数轴上的超额收益

对数轴上的超额收益的计算方法为：

$$< br > log\left(\text{策略净值}/\text{基准净值}\right) = log\left(\text{策略净值}\right) - log\left(\text{基准净值}\right) < br >$$

也就是说，普通轴上的除法版超额收益可以很方便地移植到对数轴上，只要取一个 log 就好。这条曲线集成了上面说过的所有优点。首先，由于log函数是单调的（x小于y的话就有log(x)小于log(y)），那么只要超额收益线在对数轴上上升或下降，就说明它在普通轴上同样也上升或下降，于是可以看出策略在这段时间里是跑赢还是跑输了大盘。另外，在对数轴上，净值的规模效应也被消除了。举例来说，在时间T的时候策略净值 S，基准净值 B，一年之后变动为 3S 和 2B，再假设在多年后的某个时间点策略和基准的净值分别是 10S 和 B，一年后变成 30S 和 2B。在这两个一年期里都是策略翻两倍基准翻一倍，实际上是一样的涨跌幅。在基准轴上超额收益分别是从 S/B-100% 变到3S/2B-100%以及从 10S/B-100% 变到30S/2B-100%，由于净资产规模的影响，后者在图像上的变动是前者的 10 倍。但是如果改用对数轴，我们看到两个一年期内超额收益的对数分别是从 $log(S/B)$ 变到 $log(3/2) + log(S/B)$ 以及从 $log(10S/B)$ 变到 $log(3/2) + log(10S/B)$，都是得到了$log(3/2)$ 的增长，反映了在这两个一年期里，策略相对于大盘的表现是一样的。

如何看一个策略是否有稳定的alpha收益？最直观的方法莫过于看它在对数轴上的超额收益线了，如果那条线是稳定斜向向上的就对了。

先看这个回测，它有着整体很高的超额收益，但是我们也发现了几个明显的黄线回撤，说明它在这些时段跑输大盘的地方。而下面这个策略虽然收益不如上面的高，但是超额收益高稳定得多，黄线看不出明显回撤说明它基本没有跑输大盘的时候。

进一步，我们也可以看出这个策略的超额收益都产生于哪些时候。在第一根红线之前超额收益超额收益斜率较大，说明超额收益很高；两根红线的之间的超额收益基本为零；第二根红线之后又开始有了超额收益，但远没有第一个阶段里高。策略什么时候强什么时候弱，哪些时段需要额外的分析研究，或者很多其他的重要信息，都在一条线上一目了然，这是一条不简单的线。

【答疑】是否开启动态复权(真实价格)模式对模拟交易的影响

近来，很多用户反馈在模拟盘看到的有些股票价格与在炒股软件上看到的不一样，对此表示很疑惑。

这是因为在模拟交易中，在未开启动态复权(真实价格)模式时，我们是使用基于模拟交易创建日期的后复权价格。

后复权模式示意图如下图所示：

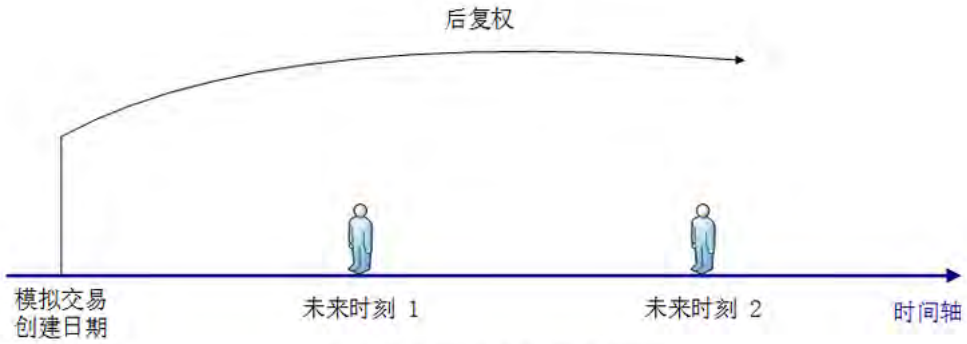


图1 未使用动态复权(真实价格)模式

不开启真实价格模拟盘的运算结果是没有错误，只是会让您理解起来更费劲一些。

用户如果想知道今天的真实价格，还需知道模拟创建的日期，并进行复权计算。

为了让用户使用更便于理解、更真实的模拟系统，我们强烈建议您开启动态复权(真实价格)模式。开启方式：用户可在代码中调用`set_option('use_real_price', True)`.

开启动态复权(真实价格)模式示意图如下图所示：

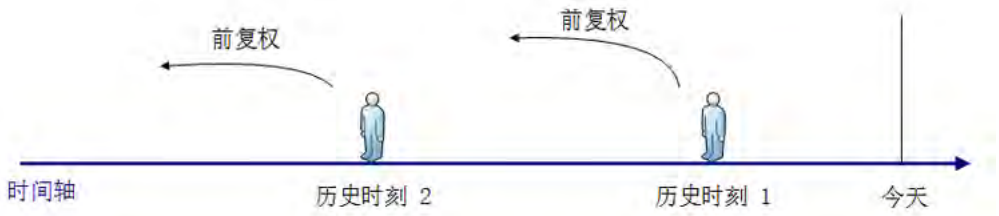


图2 使用动态复权(真实价格)模式

开启动态复权(真实)模式后，您看到的价格都是最新的，每到新的一天, 如果持仓中有股票发生了拆合或者分红或者其他可能影响复权因子的情形, 我们会根据复权因子自动调整股票的数量. 但不要跨日期缓存这些 API 返回的结果

关于真实价格的详细说明，请见帖中详述。

我们强烈建议您开启动态复权(真实价格)模式，进行模拟与回测!

注意:

1. 开启真实价格回测之后，为了让编写代码简单, 通过history/attribute_history/get_price/SecurityUnitData.mavg/vwap 等 API 拿到的都是基于当天日期的前复权价格. 另一方面, 你在不同日期调用 history/attribute_history/get_price/SecurityUnitData.mavg/vwap 返回的价格可能是不一样的, 因为我们在不同日期看到的前复权价格是不一样的. 所以不要跨日期缓存这些 API 返回的结果.
2. 每到新的一天, 如果持仓中有股票发生了拆合或者分红或者其他可能影响复权因子的情形, 我们会根据复权因子自动调整股票的数量, 如果调整后的数量是小数, 则向下取整到整数, 最后为了保证context.portfolio.portfolio_value不变, context.portfolio.cash可能有略微调整.
3. 更多详情请见 API - set_option

JoinQuant 心得——数据存取

by 庸俗神父

聚宽使用两个月，分享些心得，方便后来人。

若朋友们觉得有帮助，回复一下，权当鼓励。

当然，若觉得有疑问或建议，也请告知。

API文档已有一定的说明，故只写重点，但附上API连接，请自行查阅。

- JoinQuant 心得——股票行情数据
- JoinQuant 心得——时间持仓资金数据
- JoinQuant 心得——基本面数据
- JoinQuant 心得——订单
- JoinQuant 心得——回测功能性完善
- JoinQuant 心得——数据存取

存取数据

- API原文 read_file

- API原文 write_file
- 策略和研究中都能用。
- 文件内容只能是 str 或者unicode 。
- List与Dict等常规非字符数据,写入需`json.dumps()`，读取需`json.loads()`，所以需导入相应模块`import json`。
- DataFrame类型数据 写入需`.to_csv`，读取需`pd.read_csv`，`StringIO`。所以需导入相应模块`import pandas as pd`,`from six import StringIO`
- DataFrame类型数据在研究中有更简单的写法，不用`read_file`和`write_file`即直接`.to_csv`和`pd.read_csv`，但不能在策略中用，具体说明待更新。

样例（输出结果插入到相应代码后了）

```
# 回测时间为20151129-20151130
# 资金100000

import pandas as pd
import numpy as np
import json
from six import StringIO

def initialize(context):
    # 定义一个全局变量, 保存要操作的股票
    g.security = ['000001.XSHE','000002.XSHE']
    # 设置我们要操作的股票池
    set_universe(g.security)

# 每个单位时间(如果按天回测,则每天调用一次,如果按分钟,则每分钟调用一次)调用一次
def handle_data(context, data):

# 以下代码可以在研究中使用，但需导入相应模块

    security = ['000001.XSHE','000002.XSHE']
    tb={'a':1,'b':2}
    df=get_price(security=security, start_date='2015-11-26'
                , end_date='2015-11-27'
                , frequency='1d'
                , fields=['open','high','low','close'])
    print df['close']
# 输出
# 2015-11-30 09:30:00 - INFO - 000001.XSHE  000002.XSHE
# 2015-11-26        12.23        14.50
# 2015-11-27        11.73        14.26

    # List 存取
    write_file('test.json',json.dumps(security))
    temp= json.loads(read_file('test.json'))
    print temp
# 输出
# 2015-11-30 09:30:00 - INFO - [u'000001.XSHE', u'000002.XSHE']

    # Dict 存取
    write_file('test.json',json.dumps(tb))
    temp= json.loads(read_file('test.json'))
    print temp
# 输出
# 2015-11-30 09:30:00 - INFO - {u'a': 1, u'b': 2}

    # DataFrame 存取
    write_file('test.csv', df['close'].to_csv(), append=False)
    close= pd.read_csv(StringIO( read_file('test.csv')),index_col=[0]) # index_col=[0]是指定DataFrame的index位置在第0列
    print close
# 输出
# 2015-11-30 09:30:00 - INFO - 000001.XSHE  000002.XSHE
# 2015-11-26        12.23        14.50
# 2015-11-27        11.73        14.26
```

【新手入门教程】多股票策略

上一篇文章：十行代码带你量化交易入门
下一篇文章：【新手入门教程】简单小市值轮动策略

>学习内容：
>- 学会使用for语句和list数据类型
>- 学会写多股票策略

1 确定策略内容

前文中，我们写的单股票的均线策略的策略内容是这样的：

>若昨日收盘价高出过去20日平均价今天开盘买入股票
>若昨日收盘价低于过去20日平均价今天开盘卖出股票

现在，我们想利用计算机强大的数据处理能力，同时监视市场上多只股票，如果满足条件就进行相应交易。简言之，对多个股票分别实行原本的单股票策略，策略内容应该是这样的：

>若多只股票某只昨日收盘价高出过去20日平均价今天开盘买入该股票
>若多只股票某只昨日收盘价低于过去20日平均价今天开盘卖出该股票

那怎么用代码说给计算机听呢？老办法，先想清楚人要做的话要怎么做，再一点点翻译成代码。

1. 多股票究竟是都是哪些？即要确定股票的范围。
2. 每天看看每一只股票昨日收盘价是否高出过去20日平均价，是的话开盘就买入，不是开盘就卖出。每天都这么做，循环下去。

接下来就按之前讲得基本框架的套路出牌就好了，即初始化加周期循环。

2 用list 数据类型初始化股票列表

我们要确定股票的选择范围，此处举例就简单点只选两个了，比如 兔宝宝（002043） 和 好想你（002582）。代码如下：

```
def initialize(context):
    g.security = ['002043.XSHE','002582.XSHE']# 存入兔宝宝、好想你 的股票代码
```

对比下，单股票策略中初始化代码是这样的：

```
def initialize(context):
    g.security = '002043.XSHE'# 存入兔宝宝 的股票代码
```

可以看到，多个股票代码之间用逗号隔开了，并且两侧被中括号包在了一起。这种被中括号包在一起的数据的类型叫做list 。形如：

```
[x1,x2,x3,...,xn]
```

当多个股票代码被包在一起成为一个list后就是一个整体，我们就可以给他们一起命名（例如此处命名是g.security），就好像我们把多个股票代码包在一个盒子里，在盒子外面写上名字，如此以后你想让计算机把那些股票代码拿到哪里去计算，或是怎么样的时候，只要跟她讲g.security，她就知道是那个盒子了，而不必一一地把每个股票名字再交代一遍。

> 答疑与延伸：
>- 关于list 的详细介绍？： 请看 Python入门（2）- 数据类型之列表

初始化完成，开始周期循环的部分。

3 for语句

之前我们已经发现了，所谓的多股票策略，就是对多个股票逐个地实行单股票策略，所以对于所选股票只有两个的时候，只要把原本单股票策略对每个股票再写一遍就好了，比如这样:

```
def initialize(context):
    g.security1 = '002043.XSHE'
    g.security2 = '002582.XSHE'

def handle_data(context, data):
    last_price = data[g.security1].close
    average_price = data[g.security1].mavg(20, 'close')
    cash = context.portfolio.cash
    if last_price > average_price:
        order_value(g.security1, cash)
    elif last_price < average_price:
        order_target(g.security1, 0)

    last_price = data[g.security2].close
    average_price = data[g.security2].mavg(20, 'close')
    cash = context.portfolio.cash
    if last_price > average_price:
        order_value(g.security2, cash)
    elif last_price < average_price:
        order_target(g.security2, 0)
```

（不是很重要，也避免代码太长，注释就省了）

基本上就是原来的单股票代码写两遍，两只股票写两遍，上千只股票的话还不写死。。。

所以我们要用到for，来告诉计算机，对多只股票都逐个进行相同的一系列的操作。
for的用法如下：

```
# 把x中的数据依次取出暂时放入i中
for i in x:
    描述操作的代码

# 其中x的数据类型需要是list。
```

这段代码的含义可以理解成面试：
（x像一队的等待的面试者，i就是面试的房间，操作就是面试的过程）

取出x中的第一个数据放到i中，进行操作。
（排在最前面的人进去面试然后出来）
然后从x中再取出第二个数据覆盖掉i中原本的数据，进行操作。
（第二个人进去面试然后出来）
如此继续，直到x中最后一个数据取到并进行操作。
（直到最后一个人面试完）

> 答疑与延伸：
>- 从x中取出的次序？：x的数据类型是list，list类型里面的数据是有次序的，所以从x中取出的次序就是x里自带的次序。如for i in [2,7,3]的取出次序就是，如所见到的从左到右2,7,3的次序。（[2,7,3]是个list）
>- 关于for的更多内容？：请看文章 Python入门（5）- 条件与循环：if、while、for中关于for的部分。

for应用在我们的多股票策略中后，我们的策略就是这样的：

```
def initialize(context):
    # 存入兔宝宝、好想你 的股票代码
    g.security = ['002043.XSHE','002582.XSHE']

def handle_data(context, data):
    # 把g.security中的股票代码依次取出，逐个进行单股票均线策略
    for i in g.security:
        # 获取取得最近日收盘价，命名为last_price
        last_price = data[i].close
        # 获取近二十日股票收盘价的平均价，命名为average_price
        average_price = data[i].mavg(20, 'close')
        # 获取当前现金数量，命名为cash
        cash = context.portfolio.cash
        # 如果昨日收盘价高出二十日平均价，则买入，否则卖出。
        if last_price > average_price:
            order_value(i, cash)# 用per_cash的资金量买入股票i
        elif last_price < average_price:
            order_target(i, 0)# 将股票仓位调整到0，即全卖出
```

至此，已经是一个完整的可运行的策略了。

但是，我们应用原本单股票策略的买入卖出方法到多股票后，逻辑上会出现这样一种情况，只要多只股票中一只股票满足买入条件了，就用所有资金买入了，从而没有资金买别的股票了，即便余下的股票也有满足条件的，即策略一直最多持有一只股票。

当然，这种买入卖出逻辑并没什么错，但一般来说，多股票策略相比单股票的策略的优势，除了可以更大范围内的寻找机会外，能同时持有多只股票能帮助我们分散风险。

故，我们有必要继续研究下每次花多少钱去买股票，使策略可同时持有多只股票。

4 各个股票买多少？

每次交易信号发生，不全额买卖，该买卖多少额度呢？这是个复杂的问题，每个人对于每个策略都可能有不同的看法，并无定法。此处只做简单的处理，即将资金按股票数量分配预留，哪只股票发出信号，就将该股票的那份资金全额交易。

详细表述与代码如下：

1. 将资金平分成两份（"兔宝宝"一份，"好想你"一份），每份资金量为per_cash.

   ```
   # cash除以g.security中的股票数，得到per_cash
   per_cash = cash/len(g.security)
   ```

   >- 答疑与延伸：
   len(g.security) 什么意思？：len()是用来求list长度，即list中包含多少个东西。本例中len(g.security)就是求g.security中的股票数，结果为2.

2. 如果昨日收盘价高出二十日平均价, 则用per_cash的资金量买入的该股票；
   否则卖出全部该股票。

翻译成代码：

```
if last_price > average_price:
    order_value(i, per_cash)# 用per_cash的资金量买入股票i
elif last_price < average_price:
    order_target(i, 0)# 将股票i持有量调整到0，即全卖出
```

5 策略代码写完，进行回测

把买入卖出的代码写好，策略就写完了，如下

```
def initialize(context):
    # 存入兔宝宝、好想你 的股票代码
    g.security = ['002043.XSHE','002582.XSHE']

def handle_data(context, data):
# 把g.security中的股票代码依次取出，逐个进行单股票均线策略
    for i in g.security:
```

```
        # 获取取得最近日收盘价，命名为last_price
        last_price = data[i].close
        # 获取近二十日股票收盘价的平均价，命名为average_price
        average_price = data[i].mavg(20, 'close')
        # 获取当前现金数量，命名为cash
        cash = context.portfolio.cash
        # cash除以g.security中的股票数，得到per_cash
        per_cash = cash/len(g.security)
        # 如果昨日收盘价高出二十日平均价，则买入，否则卖出。
        if last_price > average_price:
            order_value(i, per_cash)# 用资金买入股票
        elif last_price < average_price:
            order_target(i, 0)# 将股票仓位调整到0，即全卖出
```

现在，点击运行回测，如果你代码没有问题，就会顺利的进行回测，回测结果见下文

至此，你就完成了一个简单策略的回测了。

>- 答疑与延伸：
>- 代码编辑区上方的编译运行按钮是什么？：编译是简化版的回测，相比回测少做了很多统计工作，比如每日持仓，交易详情等，所以运行会快很多。所以策略前期需要反复调试的时候，点编译运行，而策略完善后需要生成详细的报告，就点运行回测。两个都试试就知道了。

自测与自学

1. 能否理解并学会使用list数据类型。
2. 能否理解并学会使用for语句。
3. 试着调整多股票的数量，比如将选股范围调整为沪深300指数的成分股。（提示：使用获取指数成份股的API）
4. 试着调整买入卖出条件，比如将买买卖条件变为：如果昨日收盘价高出二十日平均价5%, 则买入；如果昨日收盘价低出二十日平均价5%，则卖出。（提示：乘法的代码是" * "，a的5%用代码表示为：a * 0.05）

【新手入门教程】简单市值轮动策略

上一篇文章：【新手入门教程】多股票策略
下一篇文章：【新手入门教程】市值轮动策略2.0

> 学习内容：
> - 学会使用run_daily进行周期循环
> - 学会取用市值数据、持仓数据、指数成分股数据
> - 学会写小市值策略

这一次我们要学写一个小市值轮动策略。

1 确定策略内容

简单小市值轮动策略内容是这样的：

> 每隔若干个交易日，等金额持有市值排名最小的前几只股票，卖出其他股票。

进一步明确下，把量确定，例如这样：

> 每隔10个交易日，等金额持有市值排名最小的前5只股票，卖出其他股票。

接下来按之前讲的套路出牌就好了，即初始化加周期循环。（想想之前的内容，这次具体该怎么做）

2 初始化

确定轮动频率，即每隔多少个交易日进行买卖。比如就是10个交易日。

确定持有最小市值股票数，即持有市值排名最小的前多少只股票。比如就是前5只。

则初始化部分代码为：

```
def initialize(context):
    g.stocksnum = 5 # 持有最小市值股票数
    g.period = 10 # 轮动频率
```

> 答疑与延伸：
>- 这也是多股票策略，怎么不确定股票范围了？股票范围我们希望是能在全市场，但它是变化的，有新股上市，也有退市的，所以股票范围不能事先确定好不变，所以要在周期循环的部分不断更新并确定股票范围。

3 周期循环的另一种写法--run_daily

那么用`run_daily`的我们的初始化-周期循环的策略框架怎么写呢？

我们的新套路如下：

```
def initialize(context):
    写初始化的代码
    run_daily(daily,time='every_bar')
```

```
def daily(context):
    写周期循环的代码
```

首先，可以看出run_daily要写在initialize里。
其次，需要知道的是，你把什么函数替换掉daily,谁就会像原来的handle_data一样进行周期循环。虽然名字不一定是daily，可以自己改，但不能很聚宽回测引擎已经有的函数重名，比如不能叫handle_data、initialize。

> 答疑与延伸：
>- time='every_bar' 是什么意思？ time这个参数控制循环行为的，当time='every_bar'时，循环的行为就和handle_data一样，按天回测时就每天开盘时运行，按分钟回测时就每分钟开始时运行。当然time还可以为别的，从而循环的行为会有所不同，从而实现一些handle_data做不到的事，具体见API原文：定时运行
>- 推荐以后都用run_daily来进行周期循环，聚宽系统上handle_data将会弃用。

4 开始写周期循环部分

按之前的设定，我们希望策略每十天进行一次循环，进行判断，买入卖出等，即轮动频率为10天。但策略回测只提供了基本的每分钟循环和每日循环，要做到每10天进行一次循环需要写代码编程解决。

怎么编呢？想法很简单，就每日循环好了，第一天正常进行判断买入卖出，之后九天什么都不做，然后，第十一天再正常判断买入卖出，再闲九天，这不就搞定了。

因此，我们需要一个变量，记着现在是策略进行第几天了，如果能被轮动频率10整除余1就干活，进行买卖。否则，就闲着，什么都不做。

所以，至此我们的代码应当是这样写的：

```
def initialize(context):
    g.stocksnum = 5 # 持有最小市值股票数
    g.period = 10 # 轮动频率
    run_daily(daily,time='every_bar')# 周期循环daily
    g.days = 1 # 记录策略进行到第几天，初始为1

def daily(context):
    # 判断策略进行天数是否能被轮动频率整除余1
    if g.days % g.period == 1:
        写进行选股、判断、买入卖出等代码
    else:
        pass # 什么也不做
    g.days = g.days + 1 #策略经过天数增加1
```

> 答疑与延伸：
>- g.days=1为什么要写到initialize里？因为g.days需要独立在周期循环之外来记录策略运行天数，而且还要在周期循环中使用，所以需要在initialize里设置为全局变量并赋予初值1。如果写在周期循环里，每次循环都将被重置为1，无法记录运行天数。
>- g.days % g.period什么意思？ 意为求g.days整除g.period后的余数，“%”是取余运算。例如，那么12%10就是等于2，因为12整除10得1余2，更多的，3%10等于3，3%5等于3。
>- pass什么意思？ 在python里就是什么也不做的意思。

接下来，我们关注点将在“写进行选股、判断、买入卖出等代码”这个关键代码，其余代码暂时放到视野外。

5 找出全市场上市值最小的5只股票

用context结构获取策略回测数据中的当前时间；用get_index_stocks获取指数成分股，从而确定股票范围；用get_fundamentals来使用财务数据中的市值数据，同时也能获取到相应股票代码。

代码如下：

```
# 获取当前时间
date=context.current_dt.strftime("%Y-%m-%d")
# 获取上证指数和深证综指的成分股代码并连接，即为全A股市场所有股票的股票代码
scu = get_index_stocks('000001.XSHG')+get_index_stocks('399106.XSHE')
# 选出在scu内的股票的股票代码，并按照当前时间市值从小到大排序
df = get_fundamentals(query(
        valuation.code # 获取 市值表-股票代码
    ).filter( # 条件筛选
        valuation.code.in_(scu)# 市值表-股票代码在scu中
    ).order_by(# 排序
        valuation.market_cap.asc()# 按市值表-市值 从小到大排序
    ), date= date
    )
# 取出前g.stocksnum名的股票代码，并转成list类型，buylist即为选中的股票的代码列表
buylist =list(df['code'][:g.stocksnum])
```

get_fundamentals可用来获取财务数据，用法复杂而强大，但常用的基本可以套用这个公式：（可对比上文代码理解）

```
get_fundamentals(query(
        # 获取的数据,多个数据项用逗号隔开
        数据表.数据项，数据表.数据项...
    ).filter( # 按条件筛选
        # 具体条件 多个条件用逗号连接，且要求是要同时满足
```

```
        数据表.数据项>xx,数据表.数据项<xx,数据表.数据项.in_(scu),...
    ).order_by( # 排序
        # .asc()是从小到大 .desc()是从大到小
        数据表.数据项.asc()
    ), date=日期
    )
```

其中，数据表.数据项怎么找呢？到导航栏-数据-财务数据中找，比如数据表valuation，数据项market_cap，如下图：

`get_index_stocks`的用法就比较简单，就是在导航栏-数据-指数数据中找到要用的指数，之后放到括号里。返回的是指数成分股list，所以获取多个指数成分股list后用“+”连接起来。具体看文档API文档 get_index_stocks。

> 答疑与延伸：
>- 使用更复杂的条件筛选财务数据？get_fundamentals中的与或非
>- `get_fundamentals`的更多介绍API文档 get_fundamentals。
>- `list(df['code'][:g.stocksnum])`什么意思？ df是取出的按市值从小到大排好序的股票代码，是个dataframe，`df['code']`就是选取df中的code列，即股票代码。`df['code'][:g.stocksnum]`就是取df中的code列中的前g.stocksnum个。最后`list()`是把`df['code'][:g.stocksnum]`转成将会用到的list的数据类型。dataframe的用法详情请见：dataframe 专题指南 , pandas库之数据查看、选择

6 分配资金进行买卖交易

要买的股票已经选好放在buylist里了，只要将资金分好，相应的买入就好了，如果看过之前那篇多股票策略的话，应该很好理解。

特别的是，存在这样的情况，某股票市值是最小前五的而被选入buylist，而且也买入持有了，但过了一段时间，其股票市值不是最小前五了，从而不再当下的buylist里，这时我们应当卖出这个股票。因为我们的策略就是10天为周期地持有市场上市值最小的五只股票。

代码如下：

```
# 对于每个当下持有的股票进行判断：现在是否已经不在buylist里，如果是则卖出
for stock in context.portfolio.positions:
    if stock not in buylist: #如果stock不在buylist
        order_target(stock, 0) #调整stock的持仓为0，即卖出

# 将资金分成g.stocksnum份
position_per_stk = context.portfolio.cash/g.stocksnum
# 用position_per_stk大小的g.stocksnum份资金去买buylist中的股票
for stock in buylist:
    order_value(stock, position_per_stk)
```

> 答疑与延伸：
>- `context.portfolio.positions`什么意思？ 是当前策略的持有股票数据是个dict，不妨自己输出看看，参考内容： 回测账户数据——context、Python入门-数据类型之字典。

7 策略完成，进行回测

至此，我们的策略都已经用代码实现好了，完整代码如下：

```
 def initialize(context):
    g.stocksnum = 5 # 持有最小市值股票数
    g.period = 10 # 轮动频率
    run_daily(daily,time='every_bar')# 周期循环daily
    g.days = 1 # 记录策略进行到第几天，初始为1

def daily(context):
    # 判断策略进行天数是否能被轮动频率整除余1
    if g.days % g.period == 1:

        # 获取当前时间
```

```
        date=context.current_dt.strftime("%Y-%m-%d")
        # 获取上证指数和深证综指的成分股代码并连接，即为全A股市场所有股票
        scu = get_index_stocks('000001.XSHG')+get_index_stocks('399106.XSHE')

        # 选出在scu内的股票的股票代码，并按照当前时间市值从小到大排序
        df = get_fundamentals(query(
                valuation.code,valuation.market_cap
            ).filter(
                valuation.code.in_(scu)
            ).order_by(
                valuation.market_cap.asc()
            ), date=date
            )

        # 取出前g.stocksnum名的股票代码，并转成list类型，buylist为选中的股票
        buylist =list(df['code'][:g.stocksnum])

        # 对于每个当下持有的股票进行判断：现在是否已经不在buylist里，如果是则卖出
        for stock in context.portfolio.positions:
            if stock not in buylist: #如果stock不在buylist
                order_target(stock, 0) #调整stock的持仓为0，即卖出

        # 将资金分成g.stocksnum份
        position_per_stk = context.portfolio.cash/g.stocksnum
        # 用position_per_stk大小的g.stocksnum份资金去买buylist中的股票
        for stock in buylist:
            order_value(stock, position_per_stk)
    else:
        pass # 什么也不做

    g.days = g.days + 1 # 策略经过天数增加1
```

进行回测，回测结果如图：

自测与自学

1. 是否学会使用run_daily进行周期循环
2. 是否学会取用市值数据、持仓数据、指数成分股数据
3. 是否学会编写简单小市值轮动策略
4. 调整下策略，比如轮动频率，持仓股票数，回测看看效果如何？
5. 思考此文中，虽然每次都等资金的买入，为何5只股票的持仓总价值其实还是不同，而且可能越差越大？

社区干货精选与整理（持续进行中）

> 故不积跬步，无以至千里；不积小流，无以成江海。--《荀子·劝学篇》

更新按月查看目录（2016年10月开始）

新手入门

编程知识

心得技巧

十行代码带你量化交易入门
你知道吗，你的回测结果可能都是错的
【资料分享】Python、研究报告、计量经济学、投资书籍、R语言等！(Book+Video)
几个投资者经常用到的网站
JoinQuant 心得——股票行情数据
JoinQuant 心得——基本面数据
【答疑】是否开启动态复权(真实价格)模式对模拟交易的影响
计算股息率
蚂蚁量化看中国股市
规范你的代码，构建你自己的交易系统
高斯分布下的凯利公式
仓位控制：多标的凯利公式
头寸调整策略 —— 波动性百分比调仓
分析某段代码的执行时间

量化黑科技

策略常用函数库--JoinQuant-PM
【重磅】教你如何连接“雪球组合”！！！(更新：20160909)
平安证券一键下单教程
发布聚宽功能增强器，极大降低你的编码难度！
原生使用 easytrader，以及使用 easytrader 关联 模拟交易 和 雪球组合
【教程】如何在JoinQuant的回测及研究中发送邮件！
指数信息，实时打印或者定时推送到微信
微信提醒条目太多难查看？我有HTML邮件
如何在回测及研究中发送邮件
研究模块调用回测功能
策略调参 并行一次10个回测
一个小工具助你了解策略的交易状况
和行情软件（同花顺、大智慧、通达信等）一致的KDJ和RSI以及MACD算法
多策略组合利器——分仓管控技术【非交易策略】
自动抓取并更新股票黑名单(12-18更新:抓取限售解禁股,增减持股票信息)
面向对象重构策略(一):二八轮动小市值 v2.0.7 策略组合器稚形
【工具】大家好，我是widgets，叫我帅逼就可以了
你的大蟒是不是太慢了？？？
【量化课堂】Statsmodels 统计包之 OLS 回归
【教程】send_message用法
【教程】 PrettyTable介绍 - 让你的日志数据更美观
Ta-Lib用法介绍！
talib 的K线模式识别使用 用于识别晨星、乌鸦、三兵、锤线、碑线等等形态
talib的各种MA，以及同花顺通达信等软件的SMA

数理知识

【量化课堂】凯利公式，你用对了吗?
【量化课堂】信息增益入门
【量化课堂】朴素贝叶斯入门
【量化课堂】协整的直观认识
【量化课堂】数学规划简介
【量化课堂】无约束的非线性规划问题 -- 线搜索方法
【量化课堂】拉格朗日乘子
【量化课堂】MPT 模型的解析解（上）

统计研究

JQ小市值指数编制方案--宏观经济占卜师
全市场估值-等权PE PB
指数PE统计
ETF量化投资--指数估值计算源代码
小市值策略的探索性研究（三）
【量化课堂】指标效果的统计分析：思路之一
【量化课堂】因子研究系列之一 -- 估值和资本结构因子
【量化课堂】因子研究系列之二 -- 成长因子
【量化课堂】因子研究系列之三 -- 技术因子
【量化课堂】因子研究系列之四 -- 市值与行业的中性化
回测及参数优化
【研究】关于波动率指标SD与ATR的研究（二）
【组合管理】——投资组合理论（有效前沿）
【研究】运用HMM模型的择时策略
指标效果的统计分析：思路之一

自编指数研究上证指数一直涨, 创业板指数一直跌 小市值择时该何去何从?——别担心，自编指数为你指路!
傅里叶变换求功率谱
卡尔曼滤波去流动性误差
因子中性化探讨（以pb_ratio为例）
中性化的研究
因子策略的行业偏离情况探索
一个工具用来检测数据中的波峰以及波谷
大数据系统基础B_因子有效性分析
开盘价低与(股市/基金)下跌的关系统计
预估年股息率 -- 初冰暖阳

机器学习

机器学习用于选股，对财务数据的特征学习，居然还是小市值NB....
机器学习，海量数据预测股票的未来趋势，+Model的研究
基于SVM的机器学习策略
机器学习之神经网络入门
随机森林入门
SVM原理入门
【量化课堂】强化学习入门
【量化课堂】手把手教你如何应用机器学习
【机器学习】时间序列波动率估计
深度学习简介
【机器学习】非参数型聚类分析
【机器学习】缠论中的线性回归
【机器学习】上证指数十年走势
【机器学习方法研究】——思路整理、支持向量机
一只兔子帮你理解 kNN
【量化课堂】kd 树算法之思路篇
【量化课堂】kd 树算法之详细篇
【量化课堂】scikit-learn 之 kNN 分类
基于scikit的理念设计的择时寻找最优参数思路1
机器学习预测沪深300近两年516天涨跌准确率稳定在55%以上，值得研究--一心想错
零基础小白生撸《机器学习实战》笔记——第3章 决策树(逐行注释代码，持续更新)--一心想错
一键即得标准化、规范化、二值化等多种机器学习数据预处理方式 - 我爱长颈鹿咕咕

金融&市场

效用模型
风险模型
CAPM+APT多因子模型
股票手续费，一文全学会！！！
MPT 模型
CAPM 模型和公式
“深综君”和“深成君”的前生今世 （追忆篇）
“深综君”和“深成君”的前生今世 （星霜篇）
白话分级基金之一——一家“鸡”不说两家话
白话分级基金之二——成长的烦恼（上）
白话分级基金之二——成长的烦恼（下）
白话分级基金之三——让小A飞
白话分级基金之四 ——进击的小B
白话分级基金之五——攻克套利

经典策略

【量化课堂】CAPM模型应用策略
【量化课堂】彼得·林奇的成功投资
彼得林奇修正 PEG
【量化课堂】PEG 动态选股策略
【量化课堂】羊驼交易系统
神奇的鳄鱼法则交易系统——避开盘整，抢占趋势先机
【回测来啦】——鳄鱼法则交易系统，15年至今114%
【量化课堂】海龟策略
【网格交易策略-年化30%+】-网格大法好，熊市不用跑~
【量化课堂】动量策略入门
Ichimoku Kinko 的云图指标策略的回测结果
【经典策略系列】之 Dual Thrust交易策略
费雪转换策略研究
基于阻力的市场投资策略 -- 国泰君安量化报告的代码实现
【量化课堂】Foster Friess 积极成长策略

华泰价值选股之FFScore模型--小兵哥
价值投资 -- 三一投资管理公司价值选股法
TMOM趋势策略
带收益预测的Markowitz动态平衡策略
【量化课堂】基于修正TD指标的指数择时策略
【羊群效应系列】--寻找行业轮动中的龙头股
【羊群效应系列】--识别股市中的羊群效应

统计类策略

【量化课堂】均值回归入门
【量化课堂】均值回归进阶策略
【钟摆系列3】——单股票价值中枢动态调仓
OLS回归的量化之旅 -- sly
OLS回归的量化之旅Chapter 2 -- sly
【量化课堂】基于协整的搬砖策略
【量化课堂】Hurst指数与应用（单股票）
仓位控制：风险价值法（VaR）（算法有变更）
一致性风险度量（桥水全天候为例）

基本面策略

从标普红利机会指数学到的 —— 限制行业比重上限，很好很强大--小兵哥
价值投资--低估价值选股策略
质量因子选股策略 - 我爱长颈鹿咕咕
【苍老师推荐】价值投资 -- 成长股内在价值投资
市销率与毛利率策略，简单实用 -- foolmouse
【量化课堂】高息股策略
三高五低，一种基本面选股思路的验证
巴菲特说毛利率就是护城河 - 囚徒
【量化选股专题】在稳定增长中寻找超额收益
白马股筛选策略 - sly
ST摘帽行情研究 -- bucherren
基于财报公布日期的策略 -- 我爱长颈鹿咕咕
基于借壳公告的驱动的买入策略 - 长街千堆雪
行业上下游预测策略的多行业实现+止损 - 长街千堆雪
电影好看 影视类股票收益率同样精彩！ - Quant中找米吃的阿鼠
银行轮动（中、农、工、商）无止损，年化77%
【量化课堂】雪球云蒙银行股搬砖

均线类策略

【量化课堂】双均线策略
关于MA均线回归的研究结论（终）
【简单的多均线择时策略】那个天台排队的孩子，我给你讲个故事
基于凸组合优化的均线交叉策略
【量化课堂】多头趋势回踩策略

多因子策略

【量化课堂】多因子策略入门
【量化课堂】多因子策略-APT模型
【量化课堂】Fama-French三因子火锅
【量化课堂】Fama-French五因子模型
双因子加指标模型
动态因子策略探讨 - 止一之路
605更新：11年 100倍以上的多因子策略-四因子选股策略研究（2）
【研究】量化选股——多因子模型
多因子模型+资产组合优化
这位骚年，看你骨骼惊奇，跟我一起挖掘多因子策略吧!
11年36倍收益的四因子选股策略研究（1）
【量化课堂】多因子换档反转策略
多因子rank小市值 年化80.3% - ipqhjjybj
面向对象策略框架升级版: 多因子选股+多因子权重排序示例策略。 - 晚起的小虫

小市值&二八轮动

二八轮动小市值优化版 v2.0.7 更新于2016.11.16
二八轮动小市值策略改进版07年至16年4000多倍
小市值策略，剔除了停牌，st，*st，加了简单的止损【收益340000%】

基于Morningstar'二八轮动小市值优化版 v2.0.7' 代码模块化版本
二八轮动+小市值，修改了个bug，直接实盘去
小市值二八轮动，计算股数买卖、增加过滤退市、轮动开关
基于上证指数macd择时的小市值（20161222更新：指数跌幅超过4%且破ma60就空仓休息）
【量化课堂】斗牛蛋卷二八轮动原版策略实现
蛋卷斗牛二八轮动系统
蛋卷二八轮动择时 + 小市值择股
二八轮动2.0
次新小盘策略，2014以来20倍，年化200%
蛋卷斗牛二八轮动
改进版二八轮动策略
主力脉冲 + 模块化小市值策略编辑器
整合了 VaR 仓位控制的小市值策略 -- jdd108
通过回测“二八轮动小市值优化版 v2.0.7 ”，对小市值股票的一些疑问 -- barrytan
小市值策略，也许，还能用……？ - 止一之路

期货类策略

小市值20只组合不择时不止损IC对冲——股指期货对冲研究成果应用
“【量化课堂】股指期货对冲策略”之学习笔记
多空对冲策略，大盘风险为 0
银行股搬砖轮动+300指数对冲策略
金融期货策略研究框架介绍及Demo演示
【量化课堂】股指期货跨期套利策略
【量化课堂】股指期货对冲策略

基金类策略

原来B基还可以这么玩，终于可以躺着数钱。
分级A轮动策略
RSI衍生指标择时，轮动A股ETF
关于ETF品种轮动的研究（四）
基于价值投资原理的ETF投资策略详解
基金智能定投法： 均线偏离法——遗传算法寻优
达康书记带你用ETF判断行业轮动 - Quant中找米吃的阿鼠

其它类策略

脉冲法抓庄股1.0版本
【淡手辑略】低开买（跌停不买），高开卖（涨停不卖）——Total Returns 73984.45%
《高开低开》屌丝版 -- w11
再论高开卖、低开买 -- jqz1226
【缠论】日线分笔&画图显示--李二少
【量化缠论】之分型、笔、线段识别
牛熊分界+取强舍弱+均线动量指标择时选股策略
《MACD背离》技术研究
【滚动复利策略】的量化实现-改进v1.0
股灾是系统的试金石 -- 11年400倍
一日游短线策略
总有一些策略，让你觉得日了狗
一个没劲的波动策略
融资功能初体验~ -- 北极圈

\>>策略擂台

【干货】JQ量化学习视频合集

视频不定期持续更新中～您的留言和收藏可以加速进度条哦～

HOT【校友论坛】首届华南理工大学校友量化投资论坛-回顾视频

演讲一：《量化投资教学与实践》
主讲人：于孝建，华南理工大学金融工程中心 副主任 副教授

演讲二：《基本面量化投资》
主讲人：温尚清，阿巴马资产合伙人、研究总监

演讲三：《华工系量化团队的科技与投资之路》
主讲人：蔡伟真，玄同量化CEO

演讲四：《公募基金的量化投资实践》
主讲人：霍华明，广发基金量化投资基金经理

演讲五：《量化投资的互联网商业模式探索》
主讲人：王恒鹏，JoinQuant聚宽 华南区总经理

演讲六：《广发证券量化江湖》
主讲人：闫伟，广发证券量化交易服务 博士 资深经理

圆桌论坛《主题：机器学习在量化投资中的应用、量化投资与母校产学研结合点探讨》

HOT【量化分享】魔法小鹏的“价值成长跟踪策略”分享

1、视频回看
2、PPT资料
3、订阅该策略

HOT【聚宽两周年线下聚会】

1、【北京站】聚宽两周年，宽客线下聚会
2、【深圳站】聚宽两周年，宽客线下聚会
3、【上海站】聚宽两周年，宽客线下聚会

资料：现场PPT（链接：https://pan.baidu.com/s/1eS7x2ee#list/path=%2F）

【聚宽API讲解】

1、handle_data、initialize—初始化
2、get_price—获取历史数据
3、set_order_cost—设置交易手续费
4、set-benchmark—设置基准
5、set_slippage—设置滑点
6、set_option—设置动态复权

【对接实盘】免费申请一键跟单，打通实盘最后一公里

1、使用演示视频
2、功能申请说明

【2017.04.20】20170417量化系列课程串讲

主讲人：李瑞（量化研究员）
内容：量化投资的定义、优势、工具简介、系列课程简介、课程实例串讲～
资料：
视频课件（ 链接: https://pan.baidu.com/s/1qYRm5sg 密码: 83c2 ）
kelly公式理论推导与策略应用1（链接: https://pan.baidu.com/s/1nvFhe6d 密码: 7rvb）
kelly公式理论推导与策略应用2（链接: https://pan.baidu.com/s/1i5l3ckH 密码: epj5）
kelly公式理论推导与策略应用3（链接: https://pan.baidu.com/s/1dFILkoL 密码: 2cwr）

【2017.03.23】大小市值轮动 — 研究和策略

主讲人：肖睿（量化课堂编辑）
内容：什么是大小市值轮动？如何实现大小市值轮动？如何减小回撤？如何止损？深入研究的方向？
资料：课程PPT（链接: https://pan.baidu.com/s/1bp6DTTX 密码: k8tx）

【2015.12.14】量化投资经验交流会

主讲人：高斯蒙（聚宽CEO）、刘小川（聚宽联合创始人）

【2015.10.14】高总带你用3分钟了解聚宽

主讲人：高斯蒙（聚宽CEO）

量化资料

（一）python 的相关资料 链接：http://pan.baidu.com/s/1gdZ7lzd 密码：b41d
（二）投资阅读书籍 链接：http://pan.baidu.com/s/1bqCyMa 密码：jgut
Quant Interview Books 链接：http://pan.baidu.com/s/1hrxlKsG 密码：ws0p
（三）计量经济学 链接：http://pan.baidu.com/s/1pJRjxF1 密码：fmth
python
（四）视频

R语言基础、进阶、七武器（quantmod、ggplot2....）
金融工程 89集 郑振龙 厦门大学
链接：http://pan.baidu.com/s/1ntWDjOt 密码：79jz
链接：http://pan.baidu.com/s/1sjVPb6d 密码：jsrj
链接：http://pan.baidu.com/s/1mhaITa4 密码：augf

Python 编程

Python 入门（1）- Python 介绍

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的！

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了 Python 是什么？ 以及教你写一个简单的程序！ Come on~

【Python 入门（1）】Python 介绍¶

致谢：Python教程部分内容参考了社区用户"冰柠檬"，以及廖雪峰老师的博文，在此对他们表示深深的感谢！

注：教程所用环境为 Python 2.7

一、Python 是什么？¶

Python是一种计算机程序设计语言。你可能已经听说过很多种流行的编程语言，比如非常难学的C语言，非常流行的Java语言，适合初学者的Basic语言，适合网页编程的JavaScript语言等等。

那Python是一种什么语言？

首先，我们普及一下编程语言的基础知识。用任何编程语言来开发程序，都是为了让计算机干活，比如下载一个MP3，编写一个文档等等，而计算机干活的CPU只认识机器指令，所以，尽管不同的编程语言差异极大，最后都得“翻译”成CPU可以执行的机器指令。而不同的编程语言，干同一个活，编写的代码量，差距也很大。

比如，完成同一个任务，C语言要写1000行代码，Java只需要写100行，而Python可能只要20行。

所以Python是一种相当高级的语言。

你也许会问，代码少还不好？代码少的代价是运行速度慢，C程序运行1秒钟，Java程序可能需要2秒，而Python程序可能就需要10秒。

那是不是越低级的程序越难学，越高级的程序越简单？表面上来说，是的，但是，在非常高的抽象计算中，高级的Python程序设计也是非常难学的，所以，高级程序语言不等于简单。

但是，对于初学者和完成普通任务，Python语言是非常简单易用的。连Google都在大规模使用Python，你就不用担心学了会没用。

二、第一个Python 程序¶

依照传统，学习一门新语言，写的第一程序都叫"Hello World！"，因为这个程序所要做的事情就是显示"Hello World！"。在Python中，它是这个样子：

In [1]:

```
print "Hello World! "
```

```
Hello World!
```

这是print语句的一个示例。print并不会真的往纸上打印文字，而是在屏幕上输出值。程序中的引号表示输出的文本的开始和结束，在输出结果中并不显示。

print语句也可以跟上多个字符串，用逗号“,”隔开，就可以连成一串输出：

In [2]:

```
print 'The quick brown fox', 'jumps over', 'the lazy dog'
```

```
The quick brown fox jumps over the lazy dog
```

print也可以打印整数，或者计算结果：

In [3]:

```
print 300
```

```
300
```

In [4]:

```
print 100 + 200
```

```
300
```

因此，我们可以把计算`100 + 200`的结果打印得更漂亮一点：

In [5]:

```
print '100 + 200 =', 100 + 200
```

```
100 + 200 = 300
```

三、变量¶

编程语言最强大的功能之一是操作变量的能力。变量是指向一个值的名称。

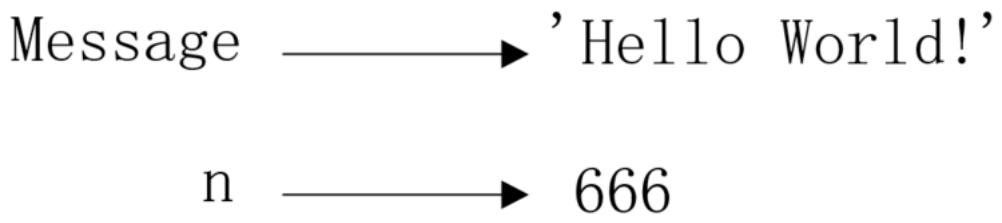
赋值语句可以建立新的变量，并给他们赋值：

In [7]:

```
message = 'Hello World!'
n = 666
```

这个例子有两个赋值。第一个将字符串'Hello World!'赋值给一个叫做message的变量；第二个将666赋值给n。

他的状态图如下所示，它显示了每个变量所在的状态。



四、导入模块¶

是不是常常看到程序的开始处有很多的import...，他的作用就时导入模块（模块：指包含一组相关的函数的文件）。

使用import可以导入模块之后，就可以使用这个模块内包含的函数了。

jqdata是JoinQuant提供的一个模块，内部包含很多API，如获取全部交易日、获取融资融券信息等等。

下面以平台的jqdata模块为例做一个示例：

导入jqdata，调用get_money_flow()函数获取股票的资金流数据。

In [8]:

```
import jqdata
jqdata.get_money_flow('000001.XSHE', '2015-12-25', '2015-12-30', fields="change_pct")
```

Out[8]:

	change_pct
0	0.57
1	-3.47
2	0.92
3	0.08

每次都要在前面加上jqdata，是不是很烦?

你可以使用from jqdata import XX 的方法加入导入模块中单个的文件。
或者使用from jqdata import * 的方法加入导入模块中所有的文件。

In [9]:

```
from jqdata import get_money_flow
get_money_flow('000001.XSHE', '2015-12-25', '2015-12-30', fields="change_pct")
```

Out[9]:

	change_pct
0	0.57
1	-3.47
2	0.92
3	0.08

刚看python是不是觉得有点累呢?

赶快休息一下，消化一下这节的内容。精彩还在后面!

Python 入门（2）- 数据类型之列表

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的!

更多内容请查看量化课堂 - Python编程板块。

本节概要： 介绍了列表的主要用法

【Python 入门（2）】基本语法-数据类型之列表¶

新手在使用平台进行量化策略实现时，往往会被各种数据类型搞乱，不知道目前获取的数据是什么类型的，可以使用什么方法，所以梳理了一下。

在Python中有多种内建的数据结构，我们这里经常遇到的会有列表、字典、集合和元组，第三方库pandas还提供DataFrame和Series。

将逐步详细介绍列表、字典、集合、元组、DataFrame和Series的基本使用方法和小技巧，然后介绍在使用平台获取数据时不同数据类型的转换。

Python 内嵌的数据类型主要包括以下两类：¶

有序：¶

List（列表），是有序集合，没有固定大小，可以通过对偏移量以及其他方法修改列表大小。列表的基本形式如：[1,2,3,4]

Tuple（元组），是有序集合，是不可变的，可以进行组合和复制运算后会生成一个新的元组。元组的基本形式比如：(1,3,6,10)

String（字符串），也是有序集合，字符串的基本形式比如：'hello'，这里不进行具体介绍。

无序：¶

Set（集合），是一个无序不重复元素的集。基本功能包括关系运算和消除重复元素。集合的基本形式如：set('abracadabra')

Dictionary（字典）是无序的键：值对 (key:value 对)集合，键必须是互不相同的(在同一个字典之内)。字典的基本形式如：{'jack': 4098, 'sape': 4139}

首先对列表进行介绍。

列表¶

List（列表） 是 Python 中最通用的序列。列表是一个任意类型对象位置的相关有序集合，它没有固定大小。不像字符串，其大小是可以变的，通过对偏移量进行赋值以及其他各种列表的方法进行调用，可以修改列表大小。

索引是从0开始而非1开始！！¶

列表中值的分割用变量[头下标:尾下标]，就可以截取相应的列表，从左到右索引默认“0”开始的，从右到左索引默认-1开始，下标可以为空表示取到头或尾。可以对列表进行索引、切片等操作，看下面例子。

In [12]:

```
# 定义列表L
L = [1,2,3,4,5,6,7,8,9]

# 使用len()函数查看列表的长度
len(L)
```

Out[12]:

```
9
```

①列表索引:

In [13]:

```
print L[0]
print L[-1] # 负数表示从后数第几个元素，-1即为列表的最后一个元素
```

```
1
9
```

②列表切片： (注意：切片并不会取到“尾下表”那个数)

In [16]:

```
L[1:5]
```

Out[16]:

```
[2, 3, 4, 5]
```

③ +操作可以拼接列表

In [17]:

```
L + [2,3,4]
```

Out[17]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4]
```

Python 的列表数据类型包含更多的方法。¶

list.append(x) 把一个元素添加到列表的结尾。

list.extend(L) 将一个给定列表中的所有元素都添加到另一个列表中。

list.insert(i, x) 在指定位置插入一个元素。第一个参数是准备插入到其前面的那个元素的索引，例如 a.insert(0, x) 会插入到整个列表之前，而 a.insert(len(a), x) 相当于 a.append(x)。

list.remove(x) 删除列表中值为 x 的第一个元素。如果没有这样的元素，就会返回一个错误。

list.pop([i]) 从列表的指定位置删除元素，并将其返回。如果没有指定索引，a.pop() 返回最后一个元素。元素随即从链表中被删除。(方法中 i 两边的方括号表示这个参数是可选的，而不是要求你输入一对方括号，这个经常会在 Python 库参考手册中遇到这样的标记。)

list.index(x) 返回列表中第一个值为 x 的元素的索引。如果没有匹配的元素就会返回一个错误。

list.count(x) 返回 x 在链表中出现的次数。

list.sort(cmp=None, key=None, reverse=False) 对列表中的元素进行排序（参数可以用来自定义排序方法，参考 sorted() 的更详细的解释）。

list.reverse() 就地倒排链表中的元素

del list[i] 有个方法可以从列表中按给定的索引而不是值来删除一个子项：del 语句。它不同于有返回值的 pop() 方法。语句 del 还可以从列表中删除切片或清空整个列表(我们以前介绍过一个方法是将空列表赋值给列表的切片)。

In [32]:

```
# 统计a中元素出现的次数
a = [1, 2, 3, 3, 1234.5]
print 'a中 1 出现的次数：',a.count(1)
print 'a中 3 出现的次数：',a.count(3)
print 'a中 x 出现的次数：',a.count('x')
```

```
a中 1 出现的次数： 1
a中 3 出现的次数： 2
a中 x 出现的次数： 0
```

In [33]:

```
# 在a的尾部添加元素
a.append(555)
a
```

Out[33]:

```
[1, 2, 3, 3, 1234.5, 555]
```

In [34]:

```
# 将[7,8,9]于a进行拼接
a.extend([7,8,9])
a
```

Out[34]:

```
[1, 2, 3, 3, 1234.5, 555, 7, 8, 9]
```

In [35]:

```
# 在a中第三个位置插入-1
a.insert(2, -1)
a
```

Out[35]:

```
[1, 2, -1, 3, 3, 1234.5, 555, 7, 8, 9]
```

In [36]:

```
# 返回2在a中的位置
a.index(2)
```

Out[36]:

```
1
```

In [37]:

```
# 反向排列a
a.reverse()
a
```

Out[37]:

```
[9, 8, 7, 555, 1234.5, 3, 3, -1, 2, 1]
```

In [38]:

```
# 对a进行排序，默认为升序!
a.sort()
a
```

Out[38]:

```
[-1, 1, 2, 3, 3, 7, 8, 9, 555, 1234.5]
```

In [39]:

```
# 逆序排列a
a.sort(reverse=True)
a
```

Out[39]:

```
[1234.5, 555, 9, 8, 7, 3, 3, 2, 1, -1]
```

In [41]:

```
# 删除列表a中值为 3 的第一个元素
a.remove(3)
a
```

Out[41]:

```
[1234.5, 555, 9, 8, 7, 2, 1, -1]
```

In [42]:

```
# 删除a中索引为0的元素
del a[0]
a
```

Out[42]:

```
[555, 9, 8, 7, 2, 1, -1]
```

In [43]:

```
# 删除a中索引为 2:4 的元素
del a[2:4]
```

```
a
```

Out[43]:

```
[555, 9, 2, 1, -1]
```

del 也可以删除整个变量，此后再引用命名 a 会引发错误(直到另一个值赋给它为止)。我们在后面的内容中可以看到 del 的其它用法。

In [44]:

```
del a
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-44-ef9d13752aff> in <module>()
      1 del a
----> 2 a

NameError: name 'a' is not defined
```

把列表当作堆栈使用¶

列表方法使得列表可以很方便的做为一个堆栈来使用，堆栈作为特定的数据结构，最先进入的元素最后一个被释放(后进先出)。用 append() 方法可以把一个元素添加到堆栈中。用不指定索引的 pop() 方法可以把一个元素从堆栈顶释放出来。例如:

In [18]:

```
stack = [3, 4, 5]
stack.append(6)
stack.append(7)
stack
```

Out[18]:

```
[3, 4, 5, 6, 7]
```

In [19]:

```
stack.pop()    #后进先出
```

Out[19]:

```
7
```

In [20]:

```
stack
```

Out[20]:

```
[3, 4, 5, 6]
```

In [21]:

```
print stack.pop()
print stack.pop()
print stack
```

```
6
5
[3, 4]
```

把列表当作队列使用¶

你也可以把列表当做队列使用，队列作为特定的数据结构，最先进入的元素最先释放(先进先出)。不过，列表这样用效率不高。相对来说从列表末尾添加和弹出很快；在头部插入和弹出很慢(因为为了一个元素，要移动整个列表中的所有元素)。

要实现队列，使用 collections.deque，它为在首尾两端快速插入和删除而设计。例如:

In [22]:

```
from collections import deque
queue = deque(["Eric", "John", "Michael"])
queue.append("Terry")
```

```
queue.append("Graham")
queue
```

Out[22]:

```
deque(['Eric', 'John', 'Michael', 'Terry', 'Graham'])
```

In [23]:

```
print queue.popleft()                # 先到的先出
print queue.popleft()
print queue
```

```
Eric
John
deque(['Michael', 'Terry', 'Graham'])
```

这一节就学到这里就结束了。

下一节会向大家介绍数据类型的另一个重要成员——字典！

Python 入门（3）- 数据类型之字典

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的！

更多内容请查看量化课堂 - Python编程板块。

本节概要： 介绍了字典的主要用法

【Python 入门（3）】基本语法-数据类型之字典¶

字典（dictionary ）¶

字典在某些语言中可能称为 联合内存 (associative memories) 或 联合数组 (associative arrays)。序列是以连续的整数为索引，与此不同的是，字典以"关键字"为索引，关键字可以是任意不可变类型，通常用字符串或数值。如果元组中只包含字符串和数字，它可以作为关键字，如果它直接或间接地包含了可变对象，就不能当做关键字。不能用列表做关键字，因为列表可以用索引、切割或者 append() 和 extend() 等方法改变。

字典是无序的键：值对 (key:value 对)集合，键必须是互不相同的(在同一个字典之内)。使用大括号创建一个空的字典：{}。初始化列表时，在大括号内放置一组逗号分隔的键：值对，这也是字典输出的方式。

字典的主要操作是依据键来存储和取值。也可以用 del 来删除键：值对(key:value)，从一个不存在的键中取值会导致错误。

In [2]:

```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
print tel
print tel['jack']
```

```
{'sape': 4139, 'jack': 4098, 'guido': 4127}
4098
```

In [3]:

```
del tel['sape']
tel['irv'] = 4127
print tel
print tel.keys()
print'guido' in tel
```

```
{'jack': 4098, 'irv': 4127, 'guido': 4127}
['jack', 'irv', 'guido']
True
```

常见字典操作方法¶

D.clear()删除字典内所有元素

D.copy()返回一个字典的复制

D.fromkeys(seq,val)创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值

D.get(key, default=None)返回指定键的值，如果值不在字典中返回default值

D.has_key(key)如果键在字典dict里返回true，否则返回false

D.items()以列表返回可遍历的(键, 值) 元组数组

D.keys()以列表返回一个字典所有的键

D.setdefault(key, default=None)和get()类似, 但如果键不存在于字典中，将会添加键并将值设为default

D.update(dict2)把字典dict2的键/值对更新到dict里

D.values()以列表返回字典中的所有值

D.pop(key)删除一个键并返回它的值，类似于列表的pop,只不过删除的是一个键不是一个可选的位置

del D[key]删除键

D[key] = 42新增或修改键

字典用法注意事项：¶

1. 序列运算无效，字典元素间是没有顺序的概念
2. 对新索引赋值会添加项
3. 键不一定总是字符串

字典键也常用于实现稀疏数据结构。例如，多维数组中只有少数位置上有存储的值：

In [5]:

```
tel.get('kkk',0) #返回指定键的值，如果值不在字典中返回default值
```

Out[5]:

```
0
```

多种构造字典方式¶

dict() 构造函数可以直接从 key-value 对中创建字典

In [6]:

```
dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

Out[6]:

```
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

In [7]:

```
dict.fromkeys(['a','b'],0) #创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值
```

Out[7]:

```
{'a': 0, 'b': 0}
```

In [8]:

```
dict(zip(['a','b','c'],[1,2,3]))
```

Out[8]:

```
{'a': 1, 'b': 2, 'c': 3}
```

In [9]:

```
{k:v for (k,v) in zip(['a','b','c'],[1,2,3])}
```

Out[9]:

```
{'a': 1, 'b': 2, 'c': 3}
```

此外，字典推导式可以从任意的键值表达式中创建字典:

In [10]:

```
{x: x**2 for x in (2, 4, 6)}
```

Out[10]:

```
{2: 4, 4: 16, 6: 36}
```

如果关键字都是简单的字符串，有时通过关键字参数指定 key-value 对更为方便:

In [11]:

```
D = dict(a=1,b=2,c=3)
D
```

Out[11]:

```
{'a': 1, 'b': 2, 'c': 3}
```

In [12]:

```
{c:c*4 for c in 'JoinQuant'}#默认是集合
```

Out[12]:

```
{'J': 'JJJJ',
 'Q': 'QQQQ',
 'a': 'aaaa',
 'i': 'iiii',
 'n': 'nnnn',
 'o': 'oooo',
 't': 'tttt',
 'u': 'uuuu'}
```

In [13]:

```
{c:c*4 for c in ['JoinQuant']}
```

Out[13]:

```
{'JoinQuant': 'JoinQuantJoinQuantJoinQuantJoinQuant'}
```

In [14]:

```
{c.lower():c*4+'!' for c in 'JoinQuant'}
```

Out[14]:

```
{'a': 'aaaa!',
 'i': 'iiii!',
 'j': 'JJJJ!',
 'n': 'nnnn!',
 'o': 'oooo!',
 'q': 'QQQQ!',
 't': 'tttt!',
 'u': 'uuuu!'}
```

注意：¶

我们一般都很愿意使用DataFrame，因为它方便，显示清楚，但是其创建和操作速度较慢，如果你对程序运行速度有较高要求, 可以考虑使用dict。目前平台中使用到的字典主要有：

handle_data(context, data)里的data是一个字典(dict), key是股票代码, value是当时的SecurityUnitData 对象

get_open_orders()，获得当天的所有未完成的订单，返回一个dict, key是order_id, value是Order对象

get_orders()，获取当天的所有订单，返回一个dict, key是order_id, value是Order对象

get_trades()，获取当天的所有成交记录, 一个订单可能分多次成交，返回一个dict, key是trade_id, value是Trade对象

get_current_data(security_list=None)，获取当天的开盘价、涨跌停价等，返回一个dict, key是股票代码, value是当天的开盘价、涨跌停价等信息

Portfolio中，positions 当前持有的可卖出的股票, 返回一个dict, key是股票代码, value是Position对象。 unsell_positions 当前持有的不可卖出的股票(比如T+1, 当前购票的股票), 一个dict, key是股票代码, value是Position对象.

history，attribute_history，get_extras中，如果df=True, 返回pandas.DataFrame, 否则返回一个dict, key是股票代码, 值是一个numpy数组numpy.ndarray, 对应上面的DataFrame的每一列。

具体可以在浏览器中按快捷键Ctrl + F，搜索“dict”，查看详细帮助。

In [ ]:

Python 入门（4）- 数据类型之元组、集合【选修篇】

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的!

更多内容请查看量化课堂 - Python编程板块。

本节概要： 介绍了元组、集合的主要用法

【Python 入门（4）】基本语法 - 数据类型3之元组、集合【选修篇】¶

下面来介绍Python的另外两种数据类型，元组和集合。这两种在实现策略中使用的相对较少，学习一遍后，之后使用中如果遇到可以到这里查询。

集合是一个无序不重复元素的集，常用到其关系运算和消除重复元素。

一、元组（Tuple）¶

1. 任意对象的有序集合 元组与字符串和列表类似，是一个位置有序的对象的集合（也就是其内容维持从左到右的顺序）。与列表相同，可以嵌入到任何类别的对象中。
2. 通过偏移存取 通过偏移而不是键来访问，例如可以使用索引，切片
3. 属于不可变序列类型 不能在原处修改（因为他们是不可变的），但可以进行组合和复制，运算后会生成一个新的元组。

创建空元组¶

In [1]:

```
tup1 = ()
```

元组中只包含一个元素时，需要在元素后面添加逗号

In [2]:

```
tup1 = (50,)
```

元组与字符串类似，下标索引从0开始，可以进行截取，组合等。元组可以使用下标索引来访问元组中的值，如下实例：

In [3]:

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

```
tup1[0]:  physics
tup2[1:5]:  (2, 3, 4, 5)
```

修改元组¶

元组与列表不同，元组中的元素值是不允许修改的，但我们可以对元组进行连接组合，如下实例：

In [4]:

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# 以下修改元组元素操作是非法的。
# tup1[0] = 100;

# 创建一个新的元组
tup3 = tup1 + tup2;
print tup3;
```

```
(12, 34.56, 'abc', 'xyz')
```

删除元组¶

元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组，实例中元组被删除后，输出变量会有异常信息，如下实例:

In [1]:

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup;
```

```
('physics', 'chemistry', 1997, 2000)
```

In [2]:

```
del tup;
print tup;
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-aaa0ed01a66f> in <module>()
      1 del tup;
----> 2 print tup;

NameError: name 'tup' is not defined
```

任意无符号的对象，以逗号隔开，默认为元组，如下实例：

In [6]:

```
print 'abc', -4.24e93, 18+6.6j, 'xyz';
x, y = 1, 2;
print "Value of x , y : ", x,y;
```

```
abc -4.24e+93 (18+6.6j) xyz
Value of x , y :  1 2
```

Python 的元组数据类型包含更多的方法。¶

tup.index(x, [start, [stop]])) 返回元组中start到stop索引中第一个值为 x 的元素在整个列表中的索引。如果没有匹配的元素就会返回一个错误。

tup.count(x) 返回 x 在元组中出现的次数。

cmp(tuple1, tuple2) 比较元组中两个元素。

len(tuple) 计算元组元素个数。

max(tuple) 返回元组中元素最大值。

min(tuple) 返回元组中元素最小值。

tuple(seq) 将列表转换为元组。

元组不提供字符串、列表和字典中的方法。如果相对元组排序，通常先得将它转换为列表并使其成为一个可变对象，才能获得使用排序方法，或使用sorted内置方法。

In [7]:

```
T = ('c','a','d','e')
tmp = list(T)
tmp.sort()
print tmp
print tuple(tmp)
```

```
['a', 'c', 'd', 'e']
('a', 'c', 'd', 'e')
```

In [8]:

```
T = ('c','a','d','e')
sorted(T)
```

Out[8]:

```
['a', 'c', 'd', 'e']
```

列表解析也可用于元组的转换

In [9]:

```
T = (1,2,3,4,5)
L = [x+20 for x in T]
L
```

Out[9]:

```
[21, 22, 23, 24, 25]
```

In [10]:

```
T = (1,2,3,2,3,5,2)
print T.index(2)
print T.index(5,2,7)
print T.count(2)
```

```
1
5
3
```

二、集合¶

Python 还包含一种数据类型 set (集合)。集合是一个无序不重复元素的集。基本功能包括关系运算和消除重复元素。比如支持 union(联合)，intersection(交)，difference(差)和 sysmmetric difference(对称差集)等数学关系运算。

大括号或 set() 函数可以用来创建集合。注：创建空集合，你必须使用 set() 而不是 {}，后者用于创建空字典

所有集合方法¶

S.issubset(t) 如果 s 是 t 的子集，则返回True，否则返回False

S.issuperset(t) 如果 s 是 t 的超集，则返回True，否则返回False

S.unoin(t) 返回一个新集合，该集合是s和t的并集，也可用s1|s2，但不能用s1+s2

S.intersection(t)返回一个新集合，该集合是s和t的交集,，也可用s1&s2

S.difference(t) 返回一个新集合，该集合是s的成员，但不是t的成员,，也可用s1-s2

S.symmetric_difference(t)对称差分是集合的异或，返回一个新集合，该集合是s或t的成员，但不是s和t共有的成员，也可用s1^s2

S.copy()返回一个新集合，该集合是s的复制

仅适合可变集合¶

S.update(t) 用t中的元素修改s，即s现在包括s或t的成员

S.intersection_update s中的成员是共同属于s和t的元素

S.difference_update s中的成员是属于s但不包含在t中的元素

S.symmetric_difference_update s中的成员更新为那些包含在s或t中，但不是s和t共有的元素

S.add(obj) 在集合s中添加对象obj

S.remove(obj) 从集合s中删除对象obj，如果obj不是集合s中的元素将有错误

S.discard(obj) 如果obj是集合s中的元素，从集合s中删除对象obj

S.pop() 删除集合s中的任意一个对象，并返回它

S.clear() 删除集合s中的所有元素

In [11]:

```
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
fruit = set(basket)
print fruit
print 'orange' in fruit
print 'crabgrass' in fruit
```

```
set(['orange', 'pear', 'apple', 'banana'])
True
False
```

In [12]:

```
a = set('abracadabra')
b = set('alacazam')
print 'a:\t',a                                  # 唯一值
print 'a - b:\t',a - b                          # 在a不在b里面
print 'a | b:\t',a | b                          # 在a或b里
print 'a & b:\t',a & b                          # a、b里面都有
print 'a ^ b:\t',a ^ b                          # 在a或b里但是不同时在两个里面
print 'a>b,a<b:\t',a>b,a<b
```

```
a:      set(['a', 'r', 'b', 'c', 'd'])
a - b:  set(['r', 'b', 'd'])
a | b:  set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
a & b:  set(['a', 'c'])
a ^ b:  set(['b', 'd', 'm', 'l', 'r', 'z'])
a>b,a<b:        False False
```

集合推导式语法

In [13]:

```
a = {x for x in 'abracadabra' if x not in 'abc'}#'abc'默认是集合
a
```

Out[13]:

```
{'d', 'r'}
```

In [14]:

```
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
fruit = set(basket)
```

列表和字典不能嵌入到集合中，但是如果需要存储复合值，元组是可以嵌入的

In [15]:

```
S = set()
S.add([1,2,3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-7338704a28cc> in <module>()
      1 S = set()
----> 2 S.add([1,2,3])

TypeError: unhashable type: 'list'
```

In [16]:

```
S.add((1,2,3))
S
```

Out[16]:

```
{(1, 2, 3)}
```

In [17]:

```
type(S)
```

Out[17]:

```
set
```

In [ ]:

```

```

Python 入门（5）- 条件与循环：if、while 、for

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的！

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了条件与循环：if、while 、for 的用法

【Python 入门（5）】条件与循环：if、while 、for¶

一、 条件if¶

条件语句格式：¶

if 判断条件：

```
执行语句……
```

else：

```
执行语句……
```

当if有多个条件时可使用括号来区分判断的先后顺序，括号中的判断优先执行，此外 and 和 or 的优先级低于>（大于）、<（小于）等判断符号，即大于和小于在没有括号的情况下会比与或要优先判断。

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

if 判断条件1：

```
执行语句1……
```

elif 判断条件2：

```
执行语句2……
```

elif 判断条件3：

```
执行语句3……
```

else：

```
执行语句4……
```

In [1]:

```
num = 5
if num == 3:            # 判断num的值
    print 'boss'
elif num == 2:
    print 'user'
elif num == 1:
    print 'worker'
elif num < 0:           # 值小于零时输出
    print 'error'
else:
    print 'roadman'     # 条件均不成立时输出
```

```
roadman
```

1.1 and、or¶

对or而言，Python会由左到右求算操作对象，然后返回第一个为真的操作对象。

Python会在其找到的第一个真值操作数的地方停止，通常叫短路计算。

In [2]:

```
2 < 3, 3 < 2
```

Out[2]:

```
(True, False)
```

In [3]:

```
2 or 3, 3 or 2
```

Out[3]:

```
(2, 3)
```

如果左边的操作数为假（空对象），Python只会计算右边的操作数并将其返回

In [4]:

```
[] or 3
```

Out[4]:

```
3
```

In [5]:

```
[] or {}
```

Out[5]:

```
{}
```

and 会停在第一个为假的对象上

In [6]:

```
2 and 3, 3 and 2
```

Out[6]:

```
(3, 2)
```

In [7]:

```
[] and {} #空 list 本身等同于 False
```

Out[7]:

```
[]
```

In [ ]:

```
#由于一个空 list 本身等同于 False，所以可以直接：
if mylist:
  # Do something with my list
else:
  # The list is empty
```

1.2 神奇的布尔值¶

从一个固定大小的集合中选择非空的对象，只要将其串在一个or表达式即可

In [ ]:

```
X = A or B or C or None #会把X设为A、B以及C中第一个非空（为真）的对象，或者所有对象都为空就设为None
```

In [ ]:

```
X = A or default # 如果A为真（或非空）的话将X设置为A，否则，将X设置为default
```

In [ ]:

```
if f1() or f2():...# 如果f1返回真值（非空），Python将不再执行f2，若要保证两个函数都执行，需要在or之前调用他们，如

tmp1, tmp2 = f1(), f2()
if tmp1 or tmp2:...
```

二、循环¶

稍后会介绍更加奇特的迭代工具，如生成器、filter和reduce。现在先从最基础的学起。

Python提供了for循环和while循环（在Python中没有do...while循环），for循环一般比while计数器循环运行得更快

break语句，在语句块执行过程中终止循环，并且跳出整个循环

continue语句，在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。

pass语句，是空语句，是为了保持程序结构的完整性。不做任何事情，一般用做占位语句。

循环else块，只有当循环正常离开时才会执行（也就是没有碰到break语句）

2.1 while¶

In [8]:

```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
```

2.2 for¶

是一个通用的序列迭代器，可以遍历任何有序的序列对象内的元素。可用于字符串、列表、元组、其他内置可迭代对象等

In [ ]:

```
for iterating_var in sequence:
   statements(s)
```

In [10]:

```
for letter in 'Python':     # 第一个实例
   print '当前字母 :', letter

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # 第二个实例
   print '当前字母 :', fruit
```

```
当前字母 : P
当前字母 : y
当前字母 : t
当前字母 : h
当前字母 : o
当前字母 : n
当前字母 : banana
当前字母 : apple
当前字母 : mango
```

In [11]:

```
T = [(1,2), (3,4), (5,6)]
for (a,b) in T:
    print (a,b)
```

```
(1, 2)
(3, 4)
(5, 6)
```

In [12]:

```
D = {'a':1,'b':2,'c':3}
for key in D:
    print (key,'=>',D[key])
```

```
('a', '=>', 1)
('c', '=>', 3)
('b', '=>', 2)
```

In [13]:

```
D.items()
```

Out[13]:

```
[('a', 1), ('c', 3), ('b', 2)]
```

In [14]:

```
for (key,value) in D.items():
    print (key,'=>',value)
```

```
('a', '=>', 1)
('c', '=>', 3)
('b', '=>', 2)
```

嵌套的结构可以自动解包¶

In [15]:

```
for ((a,b),c) in [((1,2),3),((4,5),6)]: print (a,b,c)
```

```
(1, 2, 3)
(4, 5, 6)
```

2.3 break¶

In [16]:

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print 'Current Letter :', letter
```

```
Current Letter : P
Current Letter : y
Current Letter : t
```

2.4 continue ¶

In [17]:

```
for letter in 'Python':     # 第一个实例
   if letter == 'h':
      continue
   print '当前字母 :', letter
```

```
当前字母 : P
当前字母 : y
当前字母 : t
当前字母 : o
当前字母 : n
```

2.5 pass¶

In [18]:

```
for letter in 'Python':
   if letter == 'h':
      pass
      print '这是 pass 块'
   print '当前字母 :', letter
```

```
当前字母 : P
当前字母 : y
当前字母 : t
这是 pass 块
当前字母 : h
当前字母 : o
当前字母 : n
```

In [ ]:

```

```

Python 入门（6）- 函数

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的！

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了函数的调用、定义等用法

【Python 入门（6）】函数¶

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。函数能提高应用的模块性，和代码的重复利用率。

一、调用函数¶

Python内置了很多有用的函数，我们可以直接调用。

要调用一个函数，需要知道函数的名称和参数，比如求绝对值的函数`abs`，只有一个参数。可以直接从Python的官方网站查看文档：

http://docs.python.org/2/library/functions.html#abs

也可以在交互式命令行通过`help(abs)`查看`abs`函数的帮助信息。

调用`abs`函数：

In [2]:

```
abs(-100)
```

Out[2]:

```
100
```

调用函数的时候，如果传入的参数数量不对，会报`TypeError`的错误，并且Python会明确地告诉你：abs()有且仅有1个参数，但给出了两个：

In [3]:

```
abs(1, 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-05b55862d84c> in <module>()
----> 1 abs(1, 2)

TypeError: abs() takes exactly one argument (2 given)
```

如果传入的参数数量是对的，但参数类型不能被函数所接受，也会报`TypeError`的错误，并且给出错误信息：`str`是错误的参数类型：

In [4]:

```
abs('a')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-3b9a69fe3abb> in <module>()
----> 1 abs('a')

TypeError: bad operand type for abs(): 'str'
```

二、定义函数¶

在Python中，定义一个函数要使用`def`语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用`return`语句返回。

我们以自定义一个求绝对值的`my_abs`函数为例：

In [5]:

```
def my_abs(x):
    if x >= 0:
        return x
    else:
        return -x
```

测试并调用my_abs看看返回结果是否正确:

In [6]:

```
my_abs(-100)
```

Out[6]:

```
100
```

请注意，函数体内部的语句在执行时，一旦执行到`return`时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有`return`语句，函数执行完毕后也会返回结果，只是结果为`None`。

`return None`可以简写为`return`。

三、函数的参数¶

定义函数的时候，我们把参数的名字和位置确定下来，函数的接口定义就完成了。对于函数的调用者来说，只需要知道如何传递正确的参数，以及函数将返回什么样的值就够了，函数内部的复杂逻辑被封装起来，调用者无需了解。

Python的函数定义非常简单，但灵活度却非常大。除了正常定义的必选参数外，还可以使用默认参数、可变参数和关键字参数，使得函数定义出来的接口，不但能处理复杂的参数，还可以简化调用者的代码。

3.1 默认参数¶

我们仍以具体的例子来说明如何定义函数的默认参数。先写一个计算x^2的函数：

In [7]:

```
def power(x):
    return x * x
```

当我们调用 `power` 函数时，必须传入有且仅有的一个参数 `x`：

In [8]:

```
power(5)
```

Out[8]:

```
25
```

现在，如果我们要计算 x^3 怎么办？可以再定义一个power3函数，但是如果要计算 x^4、x^5……怎么办？我们不可能定义无限多个函数。

你也许想到了，可以把 power(x) 修改为 power(x, n)，用来计算 xn ，说干就干：

In [9]:

```
def power(x, n):
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

对于这个修改后的 power 函数，可以计算任意 n 次方：

In [10]:

```
power(5, 3)
```

Out[10]:

```
125
```

但是，旧的调用代码失败了，原因是我们增加了一个参数，导致旧的代码无法正常调用：

In [11]:

```
power(5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-1fcd865a69f9> in <module>()
----> 1 power(5)

TypeError: power() takes exactly 2 arguments (1 given)
```

这个时候，默认参数就排上用场了。由于我们经常计算 x^2，所以，完全可以把第二个参数 n 的默认值设定为 2：

In [12]:

```
def power(x, n=2):
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

这样，当我们调用 `power(5)` 时，相当于调用 `power(5, 2)`：

In [13]:

```
power(5)
```

Out[13]:

```
25
```

In [14]:

```
power(5, 2)
```

Out[14]:

```
25
```

而对于 `n > 2` 的其他情况，就必须明确地传入 `n`，比如 `power(5, 3)`。

从上面的例子可以看出，默认参数可以简化函数的调用。设置默认参数时，有几点要注意：

一是必选参数在前，默认参数在后，否则Python 的解释器会报错（思考一下为什么默认参数不能放在必选参数前面）；

二是如何设置默认参数。

当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面。变化小的参数就可以作为默认参数。

使用默认参数有什么好处？最大的好处是能降低调用函数的难度。

3.2 可变参数¶

在Python函数中，还可以定义可变参数。顾名思义，可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。

我们以数学题为例子，给定一组数字a，b，c……，请计算 $a^2 + b^2 + c^2 + \ldots\ldots$。

要定义出这个函数，我们必须确定输入的参数。由于参数个数不确定，我们首先想到可以把a，b，c……作为一个 list 或 tuple 传进来，这样，函数可以定义如下：

In [15]:

```
def calc(numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

但是调用的时候，需要先组装出一个 list 或 tuple：

In [16]:

```
calc([1, 2, 3])
```

Out[16]:

```
14
```

In [17]:

```
calc((1, 3, 5, 7))
```

Out[17]:

```
84
```

如果利用可变参数，我们把函数的参数改为可变参数:

In [19]:

```
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

调用函数的方式可以简化成这样：

In [20]:

```
calc(1, 2, 3)
```

Out[20]:

```
14
```

In [21]:

```
calc(1, 3, 5, 7)
```

Out[21]:

```
84
```

定义可变参数和定义 list 或 tuple 参数相比，仅仅在参数前面加了一个 `*` 号。在函数内部，参数 numbers 接收到的是一个 tuple，因此，函数代码完全不变。但是，调用该函数时，可以传入任意个参数，包括 0 个参数：

In [22]:

```
calc()
```

Out[22]:

```
0
```

如果已经有一个 list 或者 tuple，要调用一个可变参数怎么办？ Python允许你在 list 或 tuple 前面加一个 `*` 号，把 list 或 tuple 的元素变成可变参数传进去,可以这样做：

In [23]:

```
nums = [1, 2, 3]
calc(*nums)
```

Out[23]:

```
14
```

这种写法相当有用，而且很常见。

3.3 关键字参数¶

可变参数允许你传入 0 个或任意个参数，这些可变参数在函数调用时自动组装为一个 tuple。而关键字参数允许你传入 0 个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个 dict。请看示例：

In [24]:

```
def person(name, age, **kw):
    print 'name:', name, 'age:', age, 'other:', kw
```

函数person除了必选参数 `name` 和 `age` 外，还接受关键字参数 `kw`。在调用该函数时，可以只传入必选参数：

In [25]:

```
person('Michael', 30)
```

```
name: Michael age: 30 other: {}
```

也可以传入任意个数的关键字参数：

In [26]:

```
person('Adam', 45, gender='M', job='Engineer')
```

```
name: Adam age: 45 other: {'gender': 'M', 'job': 'Engineer'}
```

关键字参数有什么用？它可以扩展函数的功能。比如，在person函数里，我们保证能接收到name和age这两个参数，但是，如果调用者愿意提供更多的参数，我们也能收到。试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，利用关键字参数来定义这个函数就能满足注册的需求。

3.4 混合参数¶

在Python中定义函数，可以用必选参数、默认参数、可变参数和关键字参数，这4种参数都可以一起使用，或者只用其中某些，但是请注意，参数定义的顺序必须是：必选参数、默认参数、可变参数和关键字参数。

比如定义一个函数，包含上述4种参数：

In [27]:

```
def func(a, b, c=0, *args, **kw):
    print 'a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw
```

在函数调用的时候，Python解释器自动按照参数位置和参数名把对应的参数传进去。

In [28]:

```
func(1, 2, 3, 'a', 'b', x=99)
```

```
a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}
```

小结

Python的函数具有非常灵活的参数形态，既可以实现简单的调用，又可以传入非常复杂的参数。

默认参数一定要用不可变对象，如果是可变对象，运行会有逻辑错误！

要注意定义可变参数和关键字参数的语法：

*args是可变参数，args接收的是一个tuple；

**kw是关键字参数，kw接收的是一个dict。

以及调用函数时如何传入可变参数和关键字参数的语法：

可变参数既可以直接传入：func(1, 2, 3)，又可以先组装list或tuple，再通过*args传入：func(*(1, 2, 3))；

关键字参数既可以直接传入：func(a=1, b=2)，又可以先组装dict，再通过**kw传入：func(**{'a': 1, 'b': 2})。

使用*args和**kw是Python的习惯写法，当然也可以用其他参数名，但最好使用习惯用法。

Python 入门（7）- 函数【选修篇】

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的！

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了函数一些扩展公能，比如定义全局变量、编写匿名函数、编写递归函数等等

【Python 入门（7）】函数【选修篇】¶

这一节会讲解一些函数的其他功能，如全局变量、递归函数、匿名函数lambda。他可以很大效率的提升你的工作效率！

一、global 声明了一个全局变量¶

- 全局变量是位于模块文件内部的顶层的变量名
- 全局变量如果是在函数内被赋值的话，必须经过声明
- 全局变量名在函数的内部不经过声明也可以被引用

在JoinQuant平台运行回测时，除了可以使用global定义全局变量，也可以中使用全局对象g，https://www.joinquant.com/api#g

这是为了方便用户使用，我们自己定义的方法！

在模拟盘中，如果中途进程中断，我们会使用pickle.dumps序列化所有的g下面的变量内容, 保存到磁盘中，再启动的时候模拟盘就不会有任何数据影响。如果没有用g声明，会出现模拟盘重启后，变量数据丢失的问题。

Python 查找名字的规则是LGB规则：

- 大多数名字引用在三个作用域中查找：先局部(Local)，次之全局(Global)，再次之内置(Build-in)。

In [1]:

```
x = 88
def func():
    global x
    x = 99
func()
print x
```

```
99
```

In [8]:

```
y,z = 1,2
def all_global():
    global x
    x = y + z
all_global()
print x
```

```
3
```

二、匿名函数 lambda ¶

python 使用 lambda 来创建匿名函数。

- lambda只是一个表达式，函数体比def简单很多。
- lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
- lambda函数拥有自己的名字空间，且不能访问自有参数列表之外或全局名字空间里的参数。
- 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

In [2]:

```
import time
start = time.clock()
fib=lambda n,x=0,y=1:x if not n else fib(n-1,y,x+y)
print fib(20)
end = time.clock()
print "read: %f s" % (end - start)
```

```
6765
read: 0.000425 s
```

In [3]:

```
start = time.clock()
fib=lambda n:1 if n<=2 else fib(n-1)+fib(n-2)
print fib(20)
end = time.clock()
print "read: %f s" % (end - start)
```

```
6765
read: 0.004816 s
```

三、Python 函数中的多态¶

一个操作的意义取决于被操作对象的类型

In [1]:

```
def times(x,y):
    return x*y
times(2,4)
```

Out[1]:

```
8
```

In [2]:

```
# 传递了与上不同的数据类型
times('Python',4)
```

Out[2]:

```
'PythonPythonPythonPython'
```

In [4]:

```
def intersect(s1,s2):
    return [x for x in s1 if x in s2]
s1 = 'Python'
s2 = 'python'
intersect(s1,s2)
```

Out[4]:

```
['y', 't', 'h', 'o', 'n']
```

In [5]:

```
# 传递了不同的数据类型
intersect([1,2,3],(1,4))
```

Out[5]:

```
[1]
```

四、递归¶

（1）递归就是在过程或函数里调用自身；

（2）在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

递归算法一般用于解决三类问题：（1）数据的定义是按递归定义的。（比如Fibonacci函数）（2）问题解法按递归算法实现。（回溯）（3）数据的结构形式是按递归定义的。（比如树的遍历，图的搜索）

递归的缺点：递归算法解题的运行效率较低。在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。

示例：斐波那契数列

斐波那契数列由十三世纪意大利数学家斐波那契发现。数列中的一系列数字常被人们称之为神奇数奇异数。具体数列为：0，1，1，2，3，5，8，13，21，34，55，89，144，233等，从该数列的第三项数字开始，每个数字等于前两个相邻数字之和。而斐波那契数列中相邻两项之商就接近黄金分割数0.618，与这一数字相关的0.191、0.382、0.5和0.809等数字就构成了股市中关于市场时间和空间计算的重要数字。

在金融市场的分析方法中，斐波那契数字频频出现。例如，在波浪理论中，一轮牛市行情可以用1个上升浪来表示，也可以用5个低一个层次的小浪来表示，还可继续细分为21个或89个小浪；在空间分析体系中，反弹行情的高度通常是前方下降趋势幅度的0.382、0.5、0.618；回调行情通常是前方上升趋势的0.382、0.5和0.618。

In [2]:

```
def fib(num):
    result=[0,1]
    for i in range(num-2):
        result.append(result[-2]+result[-1])
    return result
print fib(15)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

In [4]:

```
import time
start = time.clock()
def fib(n):
    if n<=2:return 1
    else:
        return fib(n-1)+fib(n-2)
print fib(20)
end = time.clock()
print "read: %f s" % (end - start)
```

```
6765
read: 0.005082 s
```

In [5]:

```
start = time.clock()
def fib(n):
    return n<=2 and 1 or fib(n-1)+fib(n-2)
print fib(20)
end = time.clock()
print "read: %f s" % (end - start)
```

```
6765
read: 0.004045 s
```

In [ ]:

```

```

Python 提高（1）- 时间

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的！

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了 Python 中时间的用法

【Python 提高】时间¶

时间格式¶

在Python中，通常有这几种方式来表示时间：1）时间戳； 2）格式化的时间字符串； 3）元组（struct_time）共九个元素。

1）时间戳（timestamp）的方式：

通常来说，时间戳表示的是从1970年1月1日00:00:00开始按秒计算的偏移量。我们运行“type(time.time())”，返回的是float类型。返回时间戳方式的函数主要有time()，clock()等。

2）格式化的时间字符串，比如"%Y-%m-%d %H:%M:%S"

3）元组（struct_time）方式：

struct_time元组共有9个元素，返回struct_time的函数主要有gmtime()，localtime()，strptime()。下面列出这种方式元组中的几个元素：

struct_time(tm_year=2016, tm_mon=2, tm_mday=2, tm_hour=11, tm_min=16, tm_sec=43, tm_wday=1, tm_yday=33, tm_isdst=0)

年，月，日，时，分，秒，星期几（0-6），年的第几天，是否夏令时（默认为-1）

接着介绍time模块中常用的几个函数：¶

1）time.localtime([secs])：将一个时间戳转换为当前时区的struct_time。secs参数未提供，则以当前时间为准。

2）time.gmtime([secs])：和localtime()方法类似，gmtime()方法是将一个时间戳转换为UTC时区（0时区）的struct_time。

3）time.time()：返回当前时间的时间戳。

4）time.mktime(t)：将一个struct_time转化为时间戳。

5）time.sleep(secs)：线程推迟指定的时间运行。单位为秒。

6）time.clock()：这个需要注意，在不同的系统上含义不同。在UNIX系统上，它返回的是“进程时间”，它是用秒表示的浮点数（时间戳）。而在WINDOWS中，第一次调用，返回的是进程运行的实际时间。而第二次之后的调用是自第一次调用以后到现在的运行时间。（实际上是以WIN32上QueryPerformanceCounter()为基础，它比毫秒表示更为精确）

7）time.asctime([t])：把一个表示时间的元组或者struct_time表示为这种形式：'Sun Jun 20 23:21:05 1993'。如果没有参数，将会将time.localtime()作为参数传入。

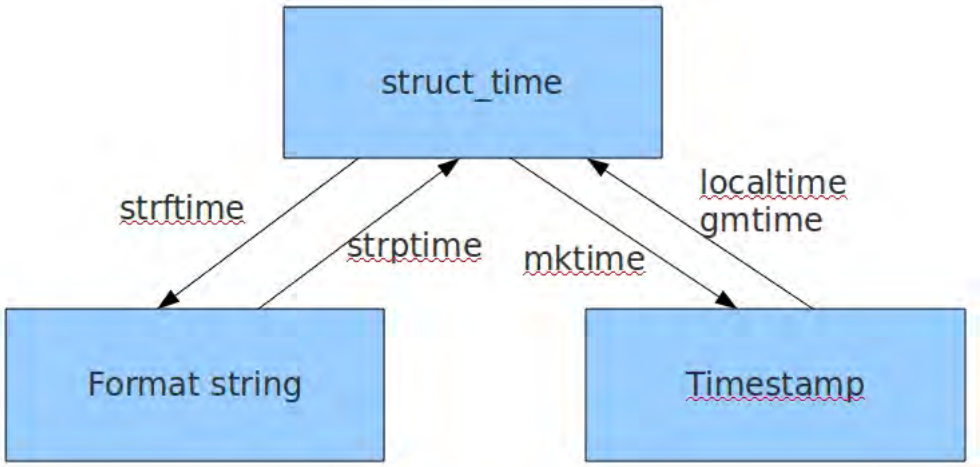
8）time.ctime([secs])：把一个时间戳（按秒计算的浮点数）转化为time.asctime()的形式。如果参数未给或者为None的时候，将会默认time.time()为参数。它的作用相当于time.asctime(time.localtime(secs))。

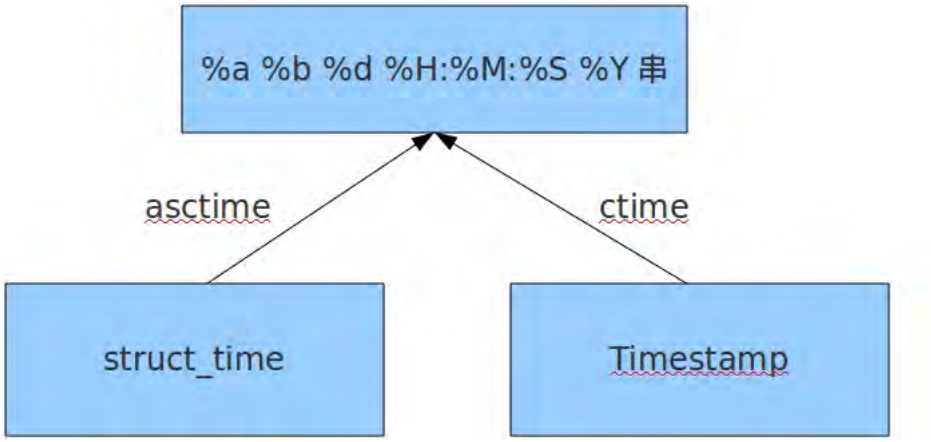
9）time.strftime(format[, t])：把一个代表时间的元组或者struct_time（如由time.localtime()和time.gmtime()返回）转化为格式化的时间字符串。如果t未指定，将传入time.localtime()。如果元组中任何一个元素越界，ValueError的错误将会被抛出。

10）time.strptime(string[, format])：把一个格式化时间字符串转化为struct_time。实际上它和strftime()是逆操作。在这个函数中，format默认为："%a %b %d %H:%M:%S %Y"。

最后，对time模块进行一个总结。根据之前描述，在Python中共有三种表达方式：1）timestamp 2）tuple或者struct_time 3）格式化字符串。

它们之间的转化如图所示：





1、使用datetime 模块来获取当前的日期和时间¶

In [1]:

```
import datetime
i = datetime.datetime.now()
print ("当前的日期和时间是 %s" % i)
print ("ISO格式的日期和时间是 %s" % i.isoformat() )
print ("当前的年份是 %s" %i.year)
print ("当前的月份是 %s" %i.month)
print ("当前的日期是  %s" %i.day)
print ("dd/mm/yyyy 格式是  %s/%s/%s" % (i.day, i.month, i.year) )
print ("当前小时是 %s" %i.hour)
print ("当前分钟是 %s" %i.minute)
print ("当前秒是  %s" %i.second)
```

```
当前的日期和时间是 2016-02-01 17:19:16.102793
ISO格式的日期和时间是 2016-02-01T17:19:16.102793
当前的年份是 2016
当前的月份是 2
当前的日期是  1
```

```
dd/mm/yyyy 格式是  1/2/2016
当前小时是 17
当前分钟是 19
当前秒是  16
```

2、使用time 模块来获取当前的时间和日期¶

In [6]:

```
import time
print (time.strftime("%H:%M:%S"))
## 12 hour format ##
print (time.strftime("%I:%M:%S"))
# 日期
print (time.strftime("%d/%m/%Y"))
print time.localtime()
```

```
17:39:03
05:39:03
01/02/2016
time.struct_time(tm_year=2016, tm_mon=2, tm_mday=1, tm_hour=17, tm_min=39, tm_sec=3, tm_wday=0, tm_yday=32, tm_isdst=0)
```

3、获取当前时间并转换为指定日期格式¶

In [7]:

```
import datetime
#获得当前时间
now = datetime.datetime.now() # ->这是时间数组格式
print now
#转换为指定的格式:
otherStyleTime = now.strftime("%Y-%m-%d %H:%M:%S")
print otherStyleTime
print now.strftime("%Y-%m-%d")
```

```
2016-02-01 17:39:42.062818
2016-02-01 17:39:42
2016-02-01
```

4、字符串格式转化¶

如a = "2013-10-10 23:40:00"，想改为 a = "2013/10/10 23:40:00"

方法：先转换为时间数组，然后转换为其他格式

In [5]:

```
a = "2013-10-10 23:40:00"
timeArray = time.strptime(a, "%Y-%m-%d %H:%M:%S")
otherStyleTime = time.strftime("%Y/%m/%d %H:%M:%S", timeArray)
otherStyleTime
```

Out[5]:

```
'2013/10/10 23:40:00'
```

5、时间和时间戳之间的相互转化¶

In [7]:

```
#日期到时间戳上的转换:
import datetime
import time
t = datetime.datetime(2014,12, 6, 12, 10, 10)
timestamp = time.mktime(t.timetuple())
print timestamp
```

```
1417839010.0
```

In [8]:

```
#时间戳到时间日期的转换:
import datetime
import time
t = time.localtime(timestamp)
timeStr = time.strftime('%Y-%m-%d %H:%M:%S', t)
print timeStr
```

```
2014-12-06 12:10:10
```

6、时间戳转换为指定格式日期¶

利用localtime()转换为时间数组,然后格式化为需要的格式,如:

In [10]:

```
timeStamp = 1381419600
timeArray = time.localtime(timeStamp)
otherStyleTime = time.strftime("%Y-%m-%d %H:%M:%S", timeArray)
otherStyleTime
```

Out[10]:

```
'2013-10-10 23:40:00'
```

In [12]:

```
timeStamp = 1381419600
dateArray = datetime.datetime.utcfromtimestamp(timeStamp)
otherStyleTime = dateArray.strftime("%Y-%m-%d %H:%M:%S")
otherStyleTime
```

Out[12]:

```
'2013-10-10 15:40:00'
```

7、获得三天前的时间的方法¶

In [13]:

```
import time
import datetime
#先获得时间数组格式的日期
threeDayAgo = (datetime.datetime.now() - datetime.timedelta(days = 3))
#转换为时间戳:
timeStamp = int(time.mktime(threeDayAgo.timetuple()))
#转换为其他字符串格式:
otherStyleTime = threeDayAgo.strftime("%Y-%m-%d %H:%M:%S")
otherStyleTime
```

Out[13]:

```
'2016-01-30 12:25:08'
```

8、给定时间戳,计算该时间的几天前时间¶

In [14]:

```
timeStamp = 1381419600
#先转换为datetime
import datetime
import time
dateArray = datetime.datetime.utcfromtimestamp(timeStamp)
threeDayAgo = dateArray - datetime.timedelta(days = 3)
threeDayAgo
```

Out[14]:

```
datetime.datetime(2013, 10, 7, 15, 40)
```

9、计算昨天和明天的日期¶

In [15]:

```
import datetime #导入日期时间模块
today = datetime.date.today() #获得今天的日期
print today #输出今天日期
yesterday = today - datetime.timedelta(days=1) #用今天日期减掉时间差，参数为1天，获得昨天的日期
print yesterday
tomorrow = today + datetime.timedelta(days=1) #用今天日期加上时间差，参数为1天，获得明天的日期
print tomorrow
print "昨天:%s， 今天:%s， 明天: %s" % (yesterday, today, tomorrow) #字符串拼接在一起输出，这3天的日期
```

```
2016-02-02
2016-02-01
2016-02-03
昨天:2016-02-01， 今天:2016-02-02， 明天：2016-02-03
```

10、获取历史交易日¶

历史交易日往往因为法定节假日的不固定而导致交易日不固定，给我们的回测带来了很大不便，这里提供一个简单方法，通过获取某只股票历史的交易日获得。

In [16]:

```
import time
initial = '2005-01-05'
tradingdays = get_price('000423.XSHE', fields='price', start_date=initial,
                        end_date=time.strftime('%Y-%m-%d', time.localtime()))
tradingdays.index
```

Out[16]:

```
DatetimeIndex(['2005-01-05', '2005-01-06', '2005-01-07', '2005-01-10',
               '2005-01-11', '2005-01-12', '2005-01-13', '2005-01-14',
               '2005-01-17', '2005-01-18',
               ...
               '2016-01-19', '2016-01-20', '2016-01-21', '2016-01-22',
               '2016-01-25', '2016-01-26', '2016-01-27', '2016-01-28',
               '2016-01-29', '2016-02-01'],
              dtype='datetime64[ns]', length=2691, freq=None, tz=None)
```

In [17]:

```
start = '2015-06-01'
end = '2015-08-01'
days = tradingdays[start:end].index
days
```

Out[17]:

```
DatetimeIndex(['2015-06-01', '2015-06-02', '2015-06-03', '2015-06-04',
               '2015-06-05', '2015-06-08', '2015-06-09', '2015-06-10',
               '2015-06-11', '2015-06-12', '2015-06-15', '2015-06-16',
               '2015-06-17', '2015-06-18', '2015-06-19', '2015-06-23',
               '2015-06-24', '2015-06-25', '2015-06-26', '2015-06-29',
               '2015-06-30', '2015-07-01', '2015-07-02', '2015-07-03',
               '2015-07-06', '2015-07-07', '2015-07-08', '2015-07-09',
               '2015-07-10', '2015-07-13', '2015-07-14', '2015-07-15',
               '2015-07-16', '2015-07-17', '2015-07-20', '2015-07-21',
               '2015-07-22', '2015-07-23', '2015-07-24', '2015-07-27',
               '2015-07-28', '2015-07-29', '2015-07-30', '2015-07-31'],
              dtype='datetime64[ns]', freq=None, tz=None)
```

In [ ]:

```

```

Python 提高（2）- 函数式编程和列表生成式

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的!

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了函数式编程和列表生成式， 这些功能可以极大的提高你的工作效率，一定要学习哦~

【Python 提高（2）】函数式编程和列表生成式¶

迭代器和解析是一种神奇的东西，学会他们可以大大加快我们的工作速度，下面开始

一、常见的几种迭代器¶

迭代器在python中是以C语言的速度运行的，而while循环版本则是通过Python虚拟机运行Python字节码的。

1. range
2. zip 可以让我们使用for循环来并行使用多个序列，zip会取得一个或多个序列为参数，然后返回元组的列表，将这些序列中的并排的元素配成对。

3. enumerate 可以获得元素和元素的偏移值

4. map map会对一个序列对象中的每一个元素应用被传入的函数，并且返回一个包含所有函数调用结果的一个列表。

5. filter 基于某一测试函数过滤出一些元素

6. reduce 对每对元素都应用函数并运行到最后结果

1、 range¶

In [2]:

```
X = 'spam'
for i in range(len(X)):
    print (X[i])
```

```
s
p
a
m
```

In [3]:

```
S = 'abcdefghijk'
for i in range(0,len(S),2):
    print S[i]
```

```
a
c
e
g
i
k
```

下面方法更容易理解，但是会复制一个字符串，如果字符串很大，占用内存较大¶

In [4]:

```
S = 'abcdefghijk'
for c in S[::2]:
    print c
```

```
a
c
e
g
i
k
```

2、 zip¶

结合zip实现并行遍历

In [20]:

```
L1 = [1,2,3,4]
L2 = [5,6,7,8]
for (x,y) in zip(L1,L2):
    print (x,y,x+y)
```

```
(1, 5, 6)
(2, 6, 8)
(3, 7, 10)
(4, 8, 12)
```

使用zip构造字典

In [3]:

```
keys = ['a','b','c']
vals = [1,3,5]
D2 = {}
for (k,v) in zip(keys,vals): D2[k]=v
D2
```

Out[3]:

```
{'a': 1, 'b': 3, 'c': 5}
```

3、 enumerate ¶

可以获得元素和元素的偏移值

In [1]:

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
dict(enumerate(seasons, start=3))
```

Out[1]:

```
{3: 'Spring', 4: 'Summer', 5: 'Fall', 6: 'Winter'}
```

4、 map¶

map会对一个序列对象中的每一个元素应用被传入的函数，并且返回一个包含所有函数调用结果的一个列表。

map对每一个元素都应用了函数调用而不是任意的表达式，所以不太通用，但是在某些情况下，map比列表解析运行起来速度更快

In [1]:

```
S1 = 'abc'
S2 = 'xyz123'
map(None,S1,S2)
```

Out[1]:

```
[('a', 'x'), ('b', 'y'), ('c', 'z'), (None, '1'), (None, '2'), (None, '3')]
```

In [7]:

```
map(pow,[1,2,3],[2,3,4])
```

Out[7]:

```
[1, 8, 81]
```

In [8]:

```
map((lambda x: x+3),[1,2,3,4])
```

Out[8]:

```
[4, 5, 6, 7]
```

In [10]:

```
def inc(x):
    return x+10
map(inc,[1,2,3])
```

Out[10]:

```
[11, 12, 13]
```

In [23]:

```
import math
list(map(math.sqrt,(x ** 2 for x in range(4))))
```

Out[23]:

```
[0.0, 1.0, 2.0, 3.0]
```

5、 filter ¶

基于某一测试函数过滤出一些元素

In [11]:

```
list(filter((lambda x:x>0),range(-5,5)))
```

Out[11]:

```
[1, 2, 3, 4]
```

6、reduce ¶

对每对元素都应用函数并运行到最后结果

In [12]:

```
reduce((lambda x,y:x+y),[1,2,3,4])
```

Out[12]:

```
10
```

In [13]:

```
import operator,functools
functools.reduce(operator.add,[2,4,6])
```

Out[13]:

```
12
```

二、列表推导式¶

列表推导式为从序列中创建列表提供了一个简单的方法。普通的程序通过将一些操作应用于序列的每个成员并通过返回的元素创建列表，或者通过满足特定条件的元素创建子序列。

例如，假设我们创建一个 squares 列表，可以像下面方式:

In [5]:

```
squares = []
for x in range(10):
     squares.append(x**2)
squares
```

Out[5]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

我们同样能够达到目的采用下面的方式:

In [6]:

```
squares = [x**2 for x in range(10)]
squares
```

Out[6]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

这也相当于 squares = map(lambda x: x**2, range(10))，但是上面的方式显得简洁以及具有可读性。

列表推导式由包含一个表达式的括号组成，表达式后面跟随一个 for 子句，之后可以有零或多个 for 或 if 子句。结果是一个列表，由表达式依据其后面的 for 和 if 子句上下文计算而来的结果构成。

例如，如下的列表推导式结合两个列表的元素，如果元素之间不相等的话:

In [7]:

```
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

Out[7]:

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

In [8]:

```
combs = []
for x in [1,2,3]:
    for y in [3,1,4]:
         if x != y:
            combs.append((x, y))
combs
```

Out[8]:

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

In [9]:

```
vec = [-4, -2, 0, 2, 4]
[x*2 for x in vec]
```

Out[9]:

```
[-8, -4, 0, 4, 8]
```

In [10]:

```
[x for x in vec if x >= 0]
```

Out[10]:

```
[0, 2, 4]
```

In [11]:

```
[abs(x) for x in vec]
```

Out[11]:

```
[4, 2, 0, 2, 4]
```

In [12]:

```
freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
[weapon.strip() for weapon in freshfruit]
```

Out[12]:

```
['banana', 'loganberry', 'passion fruit']
```

In [13]:

```
[(x, x**2) for x in range(6)]
```

Out[13]:

```
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

In [14]:

```
vec = [[1,2,3], [4,5,6], [7,8,9]]
[num for elem in vec for num in elem]
```

Out[14]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

列表推导式可使用复杂的表达式和嵌套函数:

In [15]:

```
from math import pi
[str(round(pi, i)) for i in range(1, 6)]
```

Out[15]:

```
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

三、嵌套的列表推导式¶

列表推导式可以嵌套。

考虑以下的 3x4 矩阵，一个列表中包含三个长度为4的列表:

In [16]:

```
matrix = [
     [1, 2, 3, 4],
     [5, 6, 7, 8],
     [9, 10, 11, 12],
 ]
```

现在，如果你想交换行和列，可以用嵌套的列表推导式:

In [17]:

```
[[row[i] for row in matrix] for i in range(4)]
```

Out[17]:

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

像前面看到的，嵌套的列表推导式是对 for 后面的内容进行求值，所以上例就等价于:

In [18]:

```
transposed = []
for i in range(4):
    transposed.append([row[i] for row in matrix])
transposed
```

Out[18]:

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

反过来说，如下也是一样的:

In [19]:

```
transposed = []
for i in range(4):
    transposed_row = []
    for row in matrix:
        transposed_row.append(row[i])
    transposed.append(transposed_row)
transposed
```

Out[19]:

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

在实际中，可以使用内置函数组成复杂流程语句。对此种情况 zip() 函数将会做的更好:

In [20]:

```
list(zip(*matrix))
```

Out[20]:

```
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

代码中使用的星号的说明，参数列表的分拆。 当要传递的参数已经是一个列表，但要调用的函数却只接受分开一个个的参数值。这时候需要把已有的列表拆开来。例如内建函数 range() 需要独立的 start ， stop 参数，可以在调用函数时加一个 * 操作符来自动把参数列表拆开

In [ ]:

```

```

Python 科学计算（1）- Numpy 库

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的!

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要讲解了 Numpy 库中的一些知识

Numpy本身并没有提供多么高级的数据分析功能，理解 Numpuy 数组以及面向数组的计算将有助于你更加高效的使用诸如 pandas 之类的工具，如果你是Python新手，而且只是想随意处理一下数据就行，那就跳过本节吧，没关系的。

【Python 科学计算（1）】 - Numpy 库¶

Numpy是高性能科学计算和数据分析的基础包。

Numpy本身并没有提供多么高级的数据分析功能，理解 Numpuy 数组以及面向数组的计算将有助于你更加高效的使用诸如 pandas 之类的工具，如果你是Python新手，而且只是想随意处理一下数据就行，那就跳过本节吧，没关系的。

一、ndarray 数组基础¶

Pyhton 中用列表保存一组值，可将列表当成是数组使用。此外，Python 有 array 模快，但他不支持多维数组，无论是列表还是 array 模块都没有科学运算函数，不适合做矩阵等科学计算。因此，Numpy没有使用 Python 本身的数组机制，而是提供了 ndarray 数组对象，该对象不断能方便的存取数组，而且拥有丰富的数组计算函数，比如向量的加法、减法、乘法等。

使用 ndarray 数组，首先需要导入 Numpy 函数库，也可以直接导入该函数库：

In [9]:

```
from numpy import *
```

或指定导入库的别名（在引入多个库的时候，推荐使用这个方法）。

In [10]:

```
import numpy as np
```

下面正式进入Numpy的数组世界。如果没有说明，所称数组均为 Numpy 数组对象，与 Python 的列表和 array 模块无关。

1.1 创建数组¶

创建数组是进行数组计算的先决条件，可以通过array()函数定义数组实例对象，其参数为Python 的序列对象（比如列表。）如果想定义多维数组，则传递多层嵌套的序列。例如下面这条语句定义了一个二维数组，其大小为（2,3），即有2行，3列。

In [8]:

```
a = np.array([[1,2,4.0],[3,6,9]])
a
```

Out[8]:

```
array([[ 1.,  2.,  4.],
       [ 3.,  6.,  9.]])
```

接着我们看下数组的一些属性：

In [5]:

```
# 查看行数
a.ndim
```

Out[5]:

```
2
```

In [6]:

```
# 查看数组的维数，返回(n,m)， 其中 n 为行数，m 为列数。
a.shape
```

Out[6]:

```
(2, 3)
```

In [9]:

```
# 查看元素的类型，比如 numpy.int32、numpy.float64
a.dtype
```

Out[9]:

```
dtype('float64')
```

1.2 特殊数组¶

Numpy的特殊数组主要有以下几种：

- zeros数组：全零数组，元素全为0；
- ones数组：全1数组，元素全为1；
- empty数组：空数组，元素全近似为0；

下面是全零、全1、空数组的创建方法：

In [10]:

```
np.zeros((2,3))
```

Out[10]:

```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

In [18]:

```
np.ones((3,4))
```

Out[18]:

```
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

In [26]:

```
np.empty((3,2))
```

Out[26]:

```
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

1.3 序列数组¶

arange函数：他与 Python 的 range 函数相似，但他属于Numpy 库，其参数依次为：`开始值、结束值、步长`。

In [22]:

```
np.arange(1,20,5)
```

Out[22]:

```
array([ 1,  6, 11, 16])
```

我们还可以使用 linspace 函数创建等差序列数组，其参数依次为：`开始值、结束值、元素数量`。

In [23]:

```
np.linspace(0,2,9)
```

Out[23]:

```
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
```

1.4 数组索引¶

Numpy 数组的每个元素、每行元素、每列元素都可以用索引访问，不过注意：索引是从 0 开始的。

其操作与列表基本相同。

In [27]:

```
a = np.array([[1,2,4.0],[3,6,9]])
```

In [31]:

```
# 取 a 的第一行元素
a[0]
```

Out[31]:

```
array([ 1.,  2.,  4.])
```

In [34]:

```
# 取 a 的第二列元素
a[:,1]
```

Out[34]:

```
array([ 2.,  6.])
```

In [36]:

```
# 取 a 的第一行的第三个元素
a[0,2]
```

Out[36]:

```
4.0
```

1.5 数组运算¶

In [40]:

```
a = np.array([1,2,3])
b = np.array([4.,5,6])
```

In [39]:

```
# 加法运算
a + b
```

Out[39]:

```
array([ 5.,  7.,  9.])
```

In [41]:

```
# 减法运算
a - b
```

Out[41]:

```
array([-3., -3., -3.])
```

In [42]:

```
# 乘法运算
a * b
```

Out[42]:

```
array([  4.,  10.,  18.])
```

In [43]:

```
# 乘方运算：a的2次方
a ** 2
```

Out[43]:

```
array([1, 4, 9])
```

In [44]:

```
# 除法运算
a/b
```

Out[44]:

```
array([ 0.25,  0.4 ,  0.5 ])
```

In [47]:

```
# 数组点乘
np.dot(a,b)
```

Out[47]:

```
32.0
```

In [49]:

```
# 判断大小，返回 bool 值
a >= 2
```

Out[49]:

```
array([False,  True,  True], dtype=bool)
```

In [50]:

```
# a中最大的元素
a.max()
```

Out[50]:

```
3
```

In [51]:

```
# a中最小的元素
a.min()
```

Out[51]:

```
1
```

In [52]:

```
# a的和
a.sum()
```

Out[52]:

```
6
```

1.6 数组拷贝¶

数组的拷贝分为浅拷贝和深拷贝两种，浅拷贝通过数组变量的复制完成，深拷贝使用数组对象的copy方法完成。

浅拷贝只拷贝数组的引用，如果对拷贝对象修改。原数组也将修改。

下面的代码演示了浅拷贝的方法：

In [3]:

```
a = np.ones((2,3))
a
```

Out[3]:

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

In [5]:

```
# b 为 a 的浅拷贝
b = a
b
```

Out[5]:

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

In [2]:

```
# 对 b 进行修改，a 也会被修改
b[1,2] = 9
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-a2b0891fb16b> in <module>()
      1 # 对 b 进行修改，a 也会被修改
----> 2 b[1,2] = 9
```

```
      3 a

NameError: name 'b' is not defined
```

深拷贝会复制一份和原数组一样的数组，但他们在内存中是分开存放的，所以改变拷贝数组，原数组不会改变。

下面的代码演示了 b 使用 copy 方法从原数组 a 复制一份拷贝的情况。

In [7]:

```
a = np.ones((2,3))
a
```

Out[7]:

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

In [8]:

```
b = a.copy()
```

In [9]:

```
b[1,2] = 9
b
```

Out[9]:

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  9.]])
```

In [10]:

```
a
```

Out[10]:

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

二、矩阵¶

2.1 创建矩阵¶

Numpy的矩阵对象与数组对象相似，主要不同之处在于，矩阵对象的计算遵循矩阵数学运算规律。矩阵使用 matrix 函数创建，以（2,2）大小的矩阵为例（2行2列），定义方法如下：

In [2]:

```
A = np.matrix([[1.0,2.0],[3.0,4.0]])
A
```

Out[2]:

```
matrix([[ 1.,  2.],
        [ 3.,  4.]])
```

In [3]:

```
# 查看A的类型
type(A)
```

Out[3]:

```
numpy.matrixlib.defmatrix.matrix
```

2.2 矩阵运算¶

矩阵的常用数学运算有转置、乘法、求逆等。下面的代码演示了矩阵的基本运算。

In [12]:

```
# 转置
A.T
```

Out[12]:

```
matrix([[ 1.,  3.],
        [ 2.,  4.]])
```

In [13]:

```
B = np.matrix([[3.0],[5.0]])
B
```

Out[13]:

```
matrix([[ 3.],
        [ 5.]])
```

In [14]:

```
# 矩阵乘法
A * B
```

Out[14]:

```
matrix([[ 13.],
        [ 29.]])
```

In [15]:

```
# 逆矩阵
A.I
```

Out[15]:

```
matrix([[-2. ,  1. ],
        [ 1.5, -0.5]])
```

In [16]:

```
# 解线性方程组
solve(A, B)
```

Out[16]:

```
matrix([[-1.],
        [ 2.]])
```

有关 Numpy 的东西就讲到这里了，接下来让我们学习激动人心的 pandas ~~~

In [ ]:

```

```

Python 科学计算（2）- pandas 库之数据查看、选择

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的!

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了 pandas 库之数据查看、选择。平台获取的数据主要是 DataFrame 的形式，它便是 pandas 中的。

此节可是重中之重哦!

【Pyhton 科学计算（2）】 - pandas 库之数据查看、选择¶

pandas 是基于 Numpy 构建的，让以 Numpy 为中心的应用变得更加简单。

平台获取的数据主要是 DataFrame 的形式，它便是 pandas 中的。

除此之外，pandas 还包括 一位数组series 以及三维的Panel。

下面将进行详细介绍：

- Series：一维数组，与Numpy中的一维array类似。二者与Python基本的数据结构List也很相近，其区别是：List中的元素可以是不同的数据类型，而Array和Series中则只允许存储相同的数据类型，这样可以更有效的使用内存，提高运算效率。

- DataFrame：二维的表格型数据结构。很多功能与R中的data.frame类似。可以将DataFrame理解为Series的容器。以下的内容主要以DataFrame为主。
- Panel ： 三维的数组，可以理解为DataFrame的容器。

Pandas官网，更多功能请参考http://pandas-docs.github.io/pandas-docs-travis/index.html

本节主要讲解 DataFrame 的数据查看与选择

In [84]:

```
# 首先导入库
import pandas as pd
```

1. Series¶

由一组数据（各种Numpy数据类型），以及一组与之相关的标签数据（即索引）组成。仅由一组数据即可产生最简单的Series，可以通过传递一个list对象来创建一个Series，pandas会默认创建整型索引,更多series内容请参考官网 http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html

创建一个Series：

In [3]:

```
s = pd.Series([1,3,5,np.nan,6,8])
s
```

Out[3]:

```
0    1
1    3
2    5
3  NaN
4    6
5    8
dtype: float64
```

获取 Series 的索引:

In [4]:

```
s.index
```

Out[4]:

```
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
```

由于 Series 的用法在后续的 DataFrame 讲解及涉及，所以这了不做过多介绍，精彩内容请往下看。

2. DataFrame¶

DataFrame是一个表格型的数据结构，它含有一组有序的列，每一列的数据结构都是相同的，而不同的列之间则可以是不同的数据结构（数值、字符、布尔值等）。或者以数据库进行类比，DataFrame中的每一行是一个记录，名称为Index的一个元素，而每一列则为一个字段，是这个记录的一个属性。DataFrame既有行索引也有列索引，可以被看做由Series组成的字典（共用同一个索引）。

更多内容请参考：http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

2.1 创建一个DataFrame，包括一个numpy array，时间索引和列名字：¶

In [6]:

```
dates = pd.date_range('20130101',periods=6)
dates
```

Out[6]:

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D', tz=None)
```

In [7]:

```
df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
df
```

Out[7]:

	A	B	C	D
2013-01-01	0.970600	1.034030	-1.195059	-1.202971

	A	B	C	D
2013-01-02	0.514210	1.438718	0.983266	-0.561351
2013-01-03	0.374372	-0.995714	0.457419	0.801048
2013-01-04	-1.806843	-1.307637	0.162970	-1.051681
2013-01-05	-0.056926	-1.308821	-1.128257	-0.532996
2013-01-06	-0.606822	0.213184	-1.159009	-1.218311

In [13]:

```
df.<TAB>
```

```
  File "<ipython-input-13-8b5fccb86f33>", line 1
    df.<TAB>
       ^
SyntaxError: invalid syntax
```

2.2 查看数据 ¶

我们以平台获取的数据为例进行讲解：

In [77]:

```
# 获取平安银行近几个工作日的开盘价、最高价、最低价、收盘价。
df = get_price('000001.XSHE',start_date='2016-07-01', end_date='2016-07-20', frequency='daily', fields=['open','high','low','close'])
df
```

Out[77]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

查看前几条数据：

In [20]:

```
df.head()
```

Out[20]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78

查看后几条数据：

In [21]:

```
df.tail()
```

Out[21]:

	open	high	low	close
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

查看 DataFrame 的索引

In [22]:

```
df.index
```

Out[22]:

```
DatetimeIndex(['2016-07-01', '2016-07-04', '2016-07-05', '2016-07-06',
               '2016-07-07', '2016-07-08', '2016-07-11', '2016-07-12',
               '2016-07-13', '2016-07-14', '2016-07-15', '2016-07-18',
               '2016-07-19', '2016-07-20'],
              dtype='datetime64[ns]', freq=None, tz=None)
```

查看 DataFrame 的列名

In [23]:

```
df.columns
```

Out[23]:

```
Index([u'open', u'high', u'low', u'close'], dtype='object')
```

查看 DataFrame 的值

In [24]:

```
df.values
```

Out[24]:

```
array([[ 8.69,  8.73,  8.68,  8.71],
       [ 8.69,  8.86,  8.67,  8.81],
       [ 8.8 ,  8.83,  8.77,  8.81],
       [ 8.8 ,  8.82,  8.76,  8.79],
       [ 8.79,  8.8 ,  8.74,  8.78],
       [ 8.79,  8.79,  8.73,  8.74],
       [ 8.75,  8.79,  8.74,  8.75],
       [ 8.75,  8.89,  8.74,  8.88],
       [ 8.88,  9.05,  8.86,  8.99],
       [ 8.97,  9.  ,  8.91,  8.94],
       [ 8.95,  9.  ,  8.91,  8.99],
       [ 8.99,  9.08,  8.97,  9.04],
       [ 9.04,  9.05,  8.95,  8.97],
       [ 8.96,  8.99,  8.95,  8.96]])
```

使用 describe() 函数对于数据的快速统计汇总：

In [25]:

```
df.describe()
```

Out[25]:

	open	high	low	close
count	14.000000	14.000000	14.000000	14.000000

	open	high	low	close
mean	8.846429	8.905714	8.812857	8.868571
std	0.116461	0.118043	0.107018	0.110722
min	8.690000	8.730000	8.670000	8.710000
25%	8.760000	8.805000	8.740000	8.782500
50%	8.800000	8.875000	8.765000	8.845000
75%	8.957500	9.000000	8.910000	8.967500
max	9.040000	9.080000	8.970000	9.040000

对数据的转置：

In [26]:

```
df.T
```

Out[26]:

	2016-07-01 00:00:00	2016-07-04 00:00:00	2016-07-05 00:00:00	2016-07-06 00:00:00	2016-07-07 00:00:00	2016-07-08 00:00:00	2016-07-11 00:00:00	2016-07-12 00:00:00	2016-07-13 00:00:00	2016-07-14 00:00:00
open	8.69	8.69	8.80	8.80	8.79	8.79	8.75	8.75	8.88	8.97
high	8.73	8.86	8.83	8.82	8.80	8.79	8.79	8.89	9.05	9.00
low	8.68	8.67	8.77	8.76	8.74	8.73	8.74	8.74	8.86	8.91
close	8.71	8.81	8.81	8.79	8.78	8.74	8.75	8.88	8.99	8.94

按列对 DataFrame 进行排序

In [28]:

```
df.sort(columns='open')
```

Out[28]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-13	8.88	9.05	8.86	8.99
2016-07-15	8.95	9.00	8.91	8.99
2016-07-20	8.96	8.99	8.95	8.96
2016-07-14	8.97	9.00	8.91	8.94
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97

2.3 选择数据¶

2.3.1 通过下标选取数据:¶

df['open']，df.open

以上两个语句是等效的，都是返回 df 名称为 open 列的数据，返回的为一个 Series。

df[0:3], df['2016-07-05':'2016-07-08']

下标索引选取的是 DataFrame 的记录，与 List 相同 DataFrame 的下标也是从0开始，区间索引的话，为一个左闭右开的区间，即[0：3]选取的为0-2三条记录。

与此等价，还可以用起始的索引名称和结束索引名称选取数据,如：df['a':'b']。有一点需要注意的是使用起始索引名称和结束索引名称时，也会包含结束索引的数据。具体看下方示例：

以上两种方式返回的都是DataFrame。

选择一列数据：

In [37]:

```
df['open']
```

Out[37]:

```
2016-07-01    8.69
2016-07-04    8.69
2016-07-05    8.80
2016-07-06    8.80
2016-07-07    8.79
2016-07-08    8.79
2016-07-11    8.75
2016-07-12    8.75
2016-07-13    8.88
2016-07-14    8.97
2016-07-15    8.95
2016-07-18    8.99
2016-07-19    9.04
2016-07-20    8.96
Name: open, dtype: float64
```

选择多列数据：

In [31]:

```
df[['open','close']]
```

Out[31]:

	open	close
2016-07-01	8.69	8.71
2016-07-04	8.69	8.81
2016-07-05	8.80	8.81
2016-07-06	8.80	8.79
2016-07-07	8.79	8.78
2016-07-08	8.79	8.74
2016-07-11	8.75	8.75
2016-07-12	8.75	8.88
2016-07-13	8.88	8.99
2016-07-14	8.97	8.94
2016-07-15	8.95	8.99
2016-07-18	8.99	9.04
2016-07-19	9.04	8.97
2016-07-20	8.96	8.96

选择多行：

In [32]:

```
df[0:3]
```

Out[32]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71

	open	high	low	close
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81

或者按 index 选取:

In [42]:

```
df['2016-07-05':'2016-07-08']
```

Out[42]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74

2.3.2 使用标签选取数据：¶

df.loc[行标签,列标签]

df.loc['a':'b'] #选取 ab 两行数据

df.loc[:,'open'] #选取 open 列的数据

df.loc 的第一个参数是行标签，第二个参数为列标签（可选参数，默认为所有列标签），两个参数既可以是列表也可以是单个字符，如果两个参数都为列表则返回的是 DataFrame，否则，则为 Series。

PS：loc为location的缩写。

In [44]:

```
df.loc['2016-07-05','open']
```

Out[44]:

```
8.8000000000000007
```

In [46]:

```
df.loc['2016-07-05':'2016-07-08']
```

Out[46]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74

In [47]:

```
df.loc[:, 'open']
```

Out[47]:

```
2016-07-01    8.69
2016-07-04    8.69
2016-07-05    8.80
2016-07-06    8.80
2016-07-07    8.79
2016-07-08    8.79
2016-07-11    8.75
2016-07-12    8.75
2016-07-13    8.88
2016-07-14    8.97
2016-07-15    8.95
2016-07-18    8.99
2016-07-19    9.04
```

```
2016-07-20    8.96
Name: open, dtype: float64
```

In [48]:

```
df.loc['2016-07-05':'2016-07-08','open']
```

Out[48]:

```
2016-07-05    8.80
2016-07-06    8.80
2016-07-07    8.79
2016-07-08    8.79
Name: open, dtype: float64
```

2.3.3. 使用位置选取数据：¶

df.iloc[行位置,列位置]

df.iloc[1,1] #选取第二行，第二列的值，返回的为单个值

df.iloc[[0,2],:] #选取第一行及第三行的数据

df.iloc[0:2,:] #选取第一行到第三行（不包含）的数据

df.iloc[:,1] #选取所有记录的第二列的值，返回的为一个Series

df.iloc[1,:] #选取第一行数据，返回的为一个Series

PS：iloc 则为 integer & location 的缩写

In [64]:

```
df
```

Out[64]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

In [58]:

```
df.iloc[1,1] # 选取第二行，第二列的值，返回的为单个值
```

Out[58]:

```
8.8599999999999994
```

In [59]:

```
df.iloc[[0,2],:] # 选取第一行及第三行的数据
```

Out[59]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-05	8.80	8.83	8.77	8.81

In [60]:

```
df.iloc[0:2,:] # 选取第一行到第三行（不包含）的数据
```

Out[60]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81

In [61]:

```
df.iloc[:,1] # 选取所有记录的第一列的值，返回的为一个Series
```

Out[61]:

```
2016-07-01    8.73
2016-07-04    8.86
2016-07-05    8.83
2016-07-06    8.82
2016-07-07    8.80
2016-07-08    8.79
2016-07-11    8.79
2016-07-12    8.89
2016-07-13    9.05
2016-07-14    9.00
2016-07-15    9.00
2016-07-18    9.08
2016-07-19    9.05
2016-07-20    8.99
Name: high, dtype: float64
```

In [57]:

```
df.iloc[1,:] # 选取第一行数据，返回的为一个Series
```

Out[57]:

```
open     8.69
high     8.86
low      8.67
close    8.81
Name: 2016-07-04 00:00:00, dtype: float64
```

2.3.4. 更广义的切片方式是使用.ix，它自动根据给到的索引类型判断是使用位置还是标签进行切片¶

df.ix[1,1]

df.ix['a':'b']

In [63]:

```
df
```

Out[63]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75

	open	high	low	close
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

In [62]:

```
df.ix[1,1]
```

Out[62]:

```
8.8599999999999994
```

In [66]:

```
df.ix['2016-07-01':'2016-07-05']
```

Out[66]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81

In [67]:

```
df.ix['2016-07-05','open']
```

Out[67]:

```
8.8000000000000007
```

In [68]:

```
df.ix[1,'open']
```

Out[68]:

```
8.6899999999999995
```

In [70]:

```
df.ix['2016-07-01',0]
```

Out[70]:

```
8.6899999999999995
```

2.3.5 通过逻辑指针进行数据切片：¶

df[逻辑条件]

df[df.one >= 2] #单个逻辑条件

df[(df.one >=1) & (df.one < 3)] #多个逻辑条件组合

In [71]:

```
df
```

Out[71]:

	open	high	low	close

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

In [72]:

```
# 筛选出 open 大于 8.8 的数据
df[df.open > 8.8]
```

Out[72]:

	open	high	low	close
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	9.08	8.97	9.04
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

In [73]:

```
# 筛选出 open 大于 8.8 的数据，并且 close 小于 9.0 的数据
df[(df.open > 8.8) & (df.close < 9.0)]
```

Out[73]:

	open	high	low	close
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-19	9.04	9.05	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

使用 条件过来更改数据。

In [79]:

```
df[df>9.0]
```

Out[79]:

	open	high	low	close
2016-07-01	NaN	NaN	NaN	NaN
2016-07-04	NaN	NaN	NaN	NaN

	open	high	low	close
2016-07-05	NaN	NaN	NaN	NaN
2016-07-06	NaN	NaN	NaN	NaN
2016-07-07	NaN	NaN	NaN	NaN
2016-07-08	NaN	NaN	NaN	NaN
2016-07-11	NaN	NaN	NaN	NaN
2016-07-12	NaN	NaN	NaN	NaN
2016-07-13	NaN	9.05	NaN	NaN
2016-07-14	NaN	NaN	NaN	NaN
2016-07-15	NaN	NaN	NaN	NaN
2016-07-18	NaN	9.08	NaN	9.04
2016-07-19	9.04	9.05	NaN	NaN
2016-07-20	NaN	NaN	NaN	NaN

观察可以发现，df 中小于 9 的数都变为 NaN。

下面我们就把大于 9 的数赋值为0.

In [80]:

```
df[df > 9.0] = 0
```

In [81]:

```
df
```

Out[81]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	0.00	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	0.00	8.97	0.00
2016-07-19	0.00	0.00	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

发现大于 9 的数都被替换为了0.

接下来教大家使用isin()方法来过滤在指定列中的数据：

In [83]:

```
# 选取 high 列中数为 0 和 9 的数。
df[df['high'].isin([0.00,9.00])]
```

Out[83]:

	open	high	low	close
2016-07-13	8.88	0	8.86	8.99

	open	high	low	close
2016-07-14	8.97	9	8.91	8.94
2016-07-15	8.95	9	8.91	8.99
2016-07-18	8.99	0	8.97	0.00
2016-07-19	0.00	0	8.95	8.97

3. Panel¶

平台get_price，如果是多支股票, 则返回pandas.Panel对象。更多内容请参考http://pandas.pydata.org/pandas-docs/stable/api.html#panel

可通过 panel[列标,行标，股票代码] 获取数据.

In [1]:

```
panel = get_price(['000001.XSHE','000002.XSHE'],start_date='2016-07-12', end_date='2016-07-15', frequency='daily', fields=['open','hi
panel
```

Out[1]:

```
<class 'pandas.core.panel.Panel'>
Dimensions: 4 (items) x 4 (major_axis) x 2 (minor_axis)
Items axis: close to open
Major_axis axis: 2016-07-12 00:00:00 to 2016-07-15 00:00:00
Minor_axis axis: 000001.XSHE to 000002.XSHE
```

由打印的结果可以看出：

- 列标(Items axis: close to open)
- 行标(Major_axis axis: 2016-07-12 00:00:00 to 2016-07-15 00:00:00)
- 股票代码(Minor_axis axis: 000001.XSHE to 000002.XSHE)

In [2]:

```
# 取出 'open'数据
panel['open',:,:]
```

Out[2]:

	000001.XSHE	000002.XSHE
2016-07-12	8.75	17.53
2016-07-13	8.88	17.27
2016-07-14	8.97	17.41
2016-07-15	8.95	17.18

In [3]:

```
# 取出 '2016-07-15'数据
panel[:,'2016-07-15',:]
```

Out[3]:

	close	high	low	open
000001.XSHE	8.99	9.00	8.91	8.95
000002.XSHE	17.17	17.32	17.14	17.18

In [4]:

```
# 取出 000001 的 DataFrame 数据
panel[:,:,'000001.XSHE']
```

Out[4]:

	close	high	low	open
2016-07-12	8.88	8.89	8.74	8.75
2016-07-13	8.99	9.05	8.86	8.88
2016-07-14	8.94	9.00	8.91	8.97

	close	high	low	open
2016-07-15	8.99	9.00	8.91	8.95

Panel 操作与 DataFrame 基本相同，下节会为大家讲解 DataFrame 的数据处理与规整。

In [ ]:

```
```

Python 科学计算（3）- pandas 库之数据处理与规整

这是面向新用户的 Python 教程，并结合了 JoinQuant 获取到的数据进行了讲解。

如果你之前没有学过 Python， 或者对 Python 不熟，那不要再犹豫了，这个教程就是为你准备的!

更多内容请查看量化课堂 - Python编程板块。

本节概要： 主要介绍了 pandas 库之数据处理与规整。平台获取的数据主要是 DataFrame 的形式，它便是 pandas 中的。

此节可是重中之重哦!

【Pyhton 科学计算（3）】 - pandas 库之数据处理与规整¶

上一节我们介绍了 DataFrame 的数据查看与选择

本节主要讲解 DataFrame 的数据处理，包括缺失数据处理、函数的应用和映射、数据规整、分组等。

In [8]:

```
# 首先导入库
import pandas as pd
```

In [ ]:

```
# 获取平安银行近几个工作日的开盘价、最高价、最低价、收盘价。
df = get_price('000001.XSHE',start_date='2016-07-01', end_date='2016-07-20', frequency='daily', fields=['open','high','low','close'])
df[df > 9.0] = NaN
df
```

1 缺失数据处理¶

1. 1去掉包含缺失值的行：¶

In [4]:

```
df.dropna()
```

Out[4]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-20	8.96	8.99	8.95	8.96

1.2 对缺失值进行填充：¶

In [6]:

```
df.fillna(value=0)
```

Out[6]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	0.00	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	0.00	8.97	0.00
2016-07-19	0.00	0.00	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

1.3 判断数据是否为nan，并进行布尔填充：¶

In [7]:

```
pd.isnull(df)
```

Out[7]:

	open	high	low	close
2016-07-01	False	False	False	False
2016-07-04	False	False	False	False
2016-07-05	False	False	False	False
2016-07-06	False	False	False	False
2016-07-07	False	False	False	False
2016-07-08	False	False	False	False
2016-07-11	False	False	False	False
2016-07-12	False	False	False	False
2016-07-13	False	True	False	False
2016-07-14	False	False	False	False
2016-07-15	False	False	False	False
2016-07-18	False	True	False	True
2016-07-19	True	True	False	False
2016-07-20	False	False	False	False

2 函数的应用和映射¶

In [8]:

```
df.mean()#列计算平均值
```

Out[8]:

```
open     8.831538
high     8.863636
low      8.812857
close    8.855385
dtype: float64
```

In [9]:

```
df.mean(1)#行计算平均值
```

Out[9]:

```
2016-07-01    8.7025
2016-07-04    8.7575
2016-07-05    8.8025
2016-07-06    8.7925
2016-07-07    8.7775
2016-07-08    8.7625
2016-07-11    8.7575
2016-07-12    8.8150
2016-07-13    8.9100
2016-07-14    8.9550
2016-07-15    8.9625
2016-07-18    8.9800
2016-07-19    8.9600
2016-07-20    8.9650
dtype: float64
```

In [16]:

```
df.mean(axis = 1,skipna = False) # skipna参数默认是 True 表示排除缺失值
```

Out[16]:

```
2016-07-01    8.7025
2016-07-04    8.7575
2016-07-05    8.8025
2016-07-06    8.7925
2016-07-07    8.7775
2016-07-08    8.7625
2016-07-11    8.7575
2016-07-12    8.8150
2016-07-13       NaN
2016-07-14    8.9550
2016-07-15    8.9625
2016-07-18       NaN
2016-07-19       NaN
2016-07-20    8.9650
dtype: float64
```

In [17]:

```
df.sort_index()#行名字排序
```

Out[17]:

	open	high	low	close
2016-07-01	8.69	8.73	8.68	8.71
2016-07-04	8.69	8.86	8.67	8.81
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	NaN	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99
2016-07-18	8.99	NaN	8.97	NaN
2016-07-19	NaN	NaN	8.95	8.97
2016-07-20	8.96	8.99	8.95	8.96

In [18]:

```
df.sort_index(axis=1)#列名字排序
```

Out[18]:

	close	high	low	open
2016-07-01	8.71	8.73	8.68	8.69
2016-07-04	8.81	8.86	8.67	8.69
2016-07-05	8.81	8.83	8.77	8.80
2016-07-06	8.79	8.82	8.76	8.80
2016-07-07	8.78	8.80	8.74	8.79
2016-07-08	8.74	8.79	8.73	8.79
2016-07-11	8.75	8.79	8.74	8.75
2016-07-12	8.88	8.89	8.74	8.75
2016-07-13	8.99	NaN	8.86	8.88
2016-07-14	8.94	9.00	8.91	8.97
2016-07-15	8.99	9.00	8.91	8.95
2016-07-18	NaN	NaN	8.97	8.99
2016-07-19	8.97	NaN	8.95	NaN
2016-07-20	8.96	8.99	8.95	8.96

In [19]:

```
# 数据默认是按升序排序的，也可以降序排序
df.sort_index(axis=1,ascending = False)
```

Out[19]:

	open	low	high	close
2016-07-01	8.69	8.68	8.73	8.71
2016-07-04	8.69	8.67	8.86	8.81
2016-07-05	8.80	8.77	8.83	8.81
2016-07-06	8.80	8.76	8.82	8.79
2016-07-07	8.79	8.74	8.80	8.78
2016-07-08	8.79	8.73	8.79	8.74
2016-07-11	8.75	8.74	8.79	8.75
2016-07-12	8.75	8.74	8.89	8.88
2016-07-13	8.88	8.86	NaN	8.99
2016-07-14	8.97	8.91	9.00	8.94
2016-07-15	8.95	8.91	9.00	8.99
2016-07-18	8.99	8.97	NaN	NaN
2016-07-19	NaN	8.95	NaN	8.97
2016-07-20	8.96	8.95	8.99	8.96

常用的方法如上所介绍们，还要其他许多，可自行学习，下面罗列了一些，可供参考：

- count 非na值的数量
- describe 针对Series或个DataFrame列计算汇总统计
- min、max 计算最小值和最大值
- argmin、argmax 计算能够获取到最大值和最小值得索引位置（整数）
- idxmin、idxmax 计算能够获取到最大值和最小值得索引值
- quantile 计算样本的分位数（0到1）
- sum 值的总和

- mean 值得平均数
- median 值得算术中位数（50%分位数）
- mad 根据平均值计算平均绝对离差
- var 样本值的方差
- std 样本值的标准差
- skew 样本值得偏度（三阶矩）
- kurt 样本值得峰度（四阶矩）
- cumsum 样本值得累计和
- cummin，cummax 样本值得累计最大值和累计最小值
- cumprod 样本值得累计积
- diff 计算一阶差分（对时间序列很有用）
- pct_change 计算百分数变化

3 数据规整¶

Pandas提供了大量的方法能够轻松的对Series，DataFrame和Panel对象进行各种符合各种逻辑关系的合并操作

- concat 可以沿一条轴将多个对象堆叠到一起。
- append 将一行连接到一个DataFrame上
- duplicated 移除重复数据

3.1 concat ¶

In [6]:

```
df1 = get_price('000001.XSHE',start_date='2016-07-05', end_date='2016-07-08', frequency='daily', fields=['open','high','low','close'])
df1
```

Out[6]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74

In [7]:

```
df2 = get_price('000001.XSHE',start_date='2016-07-12', end_date='2016-07-15', frequency='daily', fields=['open','high','low','close'])
df2
```

Out[7]:

	open	high	low	close
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99

纵向拼接(默认)：

In [11]:

```
pd.concat([df1,df2],axis=0)
```

Out[11]:

	open	high	low	close

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-12	8.75	8.89	8.74	8.88
2016-07-13	8.88	9.05	8.86	8.99
2016-07-14	8.97	9.00	8.91	8.94
2016-07-15	8.95	9.00	8.91	8.99

横向拼接，index对不上的会用 NaN 填充:

In [13]:

```
pd.concat([df1,df2],axis=1)
```

Out[13]:

	open	high	low	close	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81	NaN	NaN	NaN	NaN
2016-07-06	8.80	8.82	8.76	8.79	NaN	NaN	NaN	NaN
2016-07-07	8.79	8.80	8.74	8.78	NaN	NaN	NaN	NaN
2016-07-08	8.79	8.79	8.73	8.74	NaN	NaN	NaN	NaN
2016-07-12	NaN	NaN	NaN	NaN	8.75	8.89	8.74	8.88
2016-07-13	NaN	NaN	NaN	NaN	8.88	9.05	8.86	8.99
2016-07-14	NaN	NaN	NaN	NaN	8.97	9.00	8.91	8.94
2016-07-15	NaN	NaN	NaN	NaN	8.95	9.00	8.91	8.99

下面演示一下index 可以对上情况的横向拼接结果：

In [19]:

```
df3 = get_price('000001.XSHE',start_date='2016-07-12', end_date='2016-07-15', frequency='daily', fields=['low','close'])
df4 = get_price('000001.XSHE',start_date='2016-07-12', end_date='2016-07-15', frequency='daily', fields=['open','high'])
```

In [20]:

```
pd.concat([df3,df4],axis=1)
```

Out[20]:

	low	close	open	high
2016-07-12	8.74	8.88	8.75	8.89
2016-07-13	8.86	8.99	8.88	9.05
2016-07-14	8.91	8.94	8.97	9.00
2016-07-15	8.91	8.99	8.95	9.00

3.2 append ¶

In [21]:

```
df1
```

Out[21]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78

	open	high	low	close
2016-07-08	8.79	8.79	8.73	8.74

In [25]:

```
s = df1.iloc[0]
s
```

Out[25]:

```
open     8.80
high     8.83
low      8.77
close    8.81
Name: 2016-07-05 00:00:00, dtype: float64
```

In [26]:

```
df1.append(s, ignore_index=False) # ignore_index=False 表示索引不变
```

Out[26]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-05	8.80	8.83	8.77	8.81

In [27]:

```
df1.append(s, ignore_index=True) # ignore_index=True 表示索引重置
```

Out[27]:

	open	high	low	close
0	8.80	8.83	8.77	8.81
1	8.80	8.82	8.76	8.79
2	8.79	8.80	8.74	8.78
3	8.79	8.79	8.73	8.74
4	8.80	8.83	8.77	8.81

3.3 移除重复数据duplicated ¶

In [29]:

```
z = df1.append(s, ignore_index=False)
z
```

Out[29]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74
2016-07-05	8.80	8.83	8.77	8.81

查看重复数据：

In [30]:

```
z.duplicated()
```

Out[30]:

```
2016-07-05    False
2016-07-06    False
2016-07-07    False
2016-07-08    False
2016-07-05     True
dtype: bool
```

移除重复数据：

In [32]:

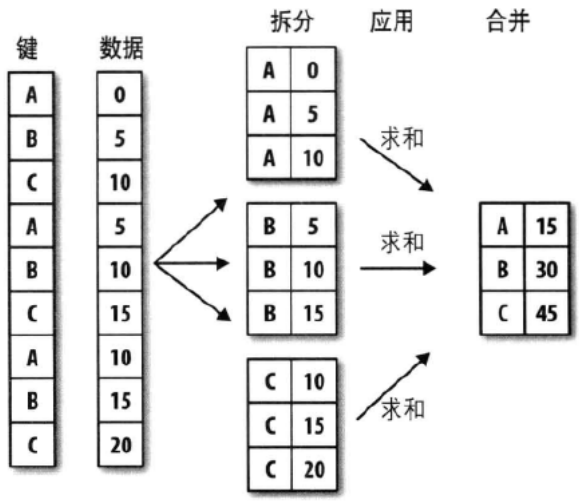
```
z.drop_duplicates()
```

Out[32]:

	open	high	low	close
2016-07-05	8.80	8.83	8.77	8.81
2016-07-06	8.80	8.82	8.76	8.79
2016-07-07	8.79	8.80	8.74	8.78
2016-07-08	8.79	8.79	8.73	8.74

可以看到'2016-07-05'的重复行被删除了

4 分组¶



In [37]:

```
z.groupby('open').sum()
```

Out[37]:

	high	low	close
open			
8.79	17.59	17.47	17.52
8.80	26.48	26.30	26.41

In [39]:

```
z.groupby(['open','close']).sum()
```

Out[39]:

		high	low
open	close		
8.79	8.74	8.79	8.73
	8.78	8.80	8.74

		high	low
open	close		
8.80	8.79	8.82	8.76
	8.81	17.66	17.54

In [ ]:

```
```

In [2]:

```
df9 = get_price(['000001.XSHE','000002.XSHE'],start_date='2016-07-12', end_date='2016-07-15', frequency='daily', fields=['open','high
df9
```

Out[2]:

```
<class 'pandas.core.panel.Panel'>
Dimensions: 4 (items) x 4 (major_axis) x 2 (minor_axis)
Items axis: close to open
Major_axis axis: 2016-07-12 00:00:00 to 2016-07-15 00:00:00
Minor_axis axis: 000001.XSHE to 000002.XSHE
```

In [26]:

```
df9[:,0,:]
```

Out[26]:

	close	high	low	open
000001.XSHE	8.88	8.89	8.74	8.75
000002.XSHE	17.39	17.69	16.85	17.53

In [27]:

```
df9[:,:,0]
```

Out[27]:

	close	high	low	open
2016-07-12	8.88	8.89	8.74	8.75
2016-07-13	8.99	9.05	8.86	8.88
2016-07-14	8.94	9.00	8.91	8.97
2016-07-15	8.99	9.00	8.91	8.95

In [30]:

```
df9[:,:,1]
```

Out[30]:

	close	high	low	open
2016-07-12	17.39	17.69	16.85	17.53
2016-07-13	17.58	17.74	17.13	17.27
2016-07-14	17.24	17.45	17.20	17.41
2016-07-15	17.17	17.32	17.14	17.18

In [33]:

```
df9[0,:,:]
```

Out[33]:

	000001.XSHE	000002.XSHE

	000001.XSHE	000002.XSHE
2016-07-12	8.88	17.39
2016-07-13	8.99	17.58
2016-07-14	8.94	17.24
2016-07-15	8.99	17.17

In [36]:

```
df9.ix[:,0]
```

Out[36]:

	close	high	low	open
000001.XSHE	8.88	8.89	8.74	8.75
000002.XSHE	17.39	17.69	16.85	17.53

In [ ]:

```

```

【量化课堂】Statsmodels 统计包之 OLS 回归

Statsmodels 是 Python 中一个强大的统计分析包，包含了回归分析、时间序列分析、假设检
验等等的功能。Statsmodels 在计量的简便性上是远远不及 Stata 等软件的，但它的优点在于可以与 Python 的其他的任务（如 NumPy、Pandas）有效结合，提高工作效率。在本文中，我们重点介绍最回归分析中最常用的 OLS（ordinary least square）功能。

当你需要在 Python 中进行回归分析时……
import statsmodels.api as sm！！！

```
本文由JoinQuant量化课堂推出，难度为入门，深度为level-0。
阅读本文需要掌握回归分析（level-0）的知识。

作者：Haozun、肖睿
编辑：宏观经济算命师
```

在一切开始之前

上帝导入了 NumPy（大家都叫它囊派？我叫它囊辟），

```
import numpy as np
```

便有了时间。

上帝导入了 matplotlib，

```
import matplotlib.pyplot as plt
```

便有了空间。

上帝导入了 Statsmodels，

```
import statsmodels.api as sm
```

世界开始了。

简单 OLS 回归

假设我们有回归模型

$$Y = \beta < em > 0 + \beta_1 X_1 + \cdots + \beta_n X_n + \varepsilon,$$

并且有 k 组数据 $(y(t), x_1(t), \ldots, x_n(t)) < /em > t = 1^k$。OLS 回归用于计算回归系数 $\beta < em > i$ 的估值 $b_0, b_1, \ldots, b_n$，使误差平方

$$\sum < /em > t = 1^k (y(t) - b_0 - b_1 x_1(t) - \ldots - b_n x_n(t))^2$$

最小化。

statsmodels.OLS 的输入有 (endog, exog, missing, hasconst) 四个，我们现在只考虑前两个。第一个输入 endog 是回归中的反应变量（也称因变量），是上面模型中的 $y(t)$, 输入是一个长度为 k 的 array。第二个输入 exog 则是回归变量（也称自变量）的值，即模型中的 $x_1(t), \ldots, x_n(t)$。但是要注意，statsmodels.OLS 不会假设回归模型有常数项，所以我们应该假设模型是

$$Y = \beta_0 X_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

并且在数据中，对于所有 $t = 1, \ldots, k$, 设置 $x_0(t) = 1$。因此，exog的输入是一个 $k \times (n+1)$ 的 array，其中最左一列的数值全为 1。往往输入的数据中，没有专门的数值全为1的一列，Statmodels 有直接解决这个问题的函数：sm.add_constant()。它会在一个 array 左侧加上一列 1。（本文中所有输入 array 的情况也可以使用同等的 list、pd.Series 或 pd.DataFrame。）

确切地说，statsmodels.OLS 是 statsmodels.regression.linear_model 里的一个函数（从这个命名也能看出，statsmodel 有很多很多功能，其中的一项叫回归）。它的输出结果是一个 statsmodels.regression.linear_model.OLS，只是一个类，并没有进行任何运算。在 OLS 的模型之上调用拟合函数 fit()，才进行回归运算，并且得到 statsmodels.regression.linear_model.RegressionResultsWrapper，它包含了这组数据进行回归拟合的结果摘要。调用 params 可以查看计算出的回归系数 $b_0, b_1, \ldots, b_n$。

简单的线性回归

上面的介绍绕了一个大圈圈，现在我们来看一个例子，假设回归公式是:

$$Y = 1 + 10 \cdot X.$$

我们从最简单的一元模型开始，虚构一组数据。首先设定数据量，也就是上面的 k 值。

```
nsample = 100
```

然后创建一个 array，是上面的 x_1 的数据。这里，我们想要 x_1 的值从 0 到 10 等差排列。

```
x = np.linspace(0, 10, nsample)
```

使用 sm.add_constant() 在 array 上加入一列常项1。

```
X = sm.add_constant(x)
```

然后设置模型里的 β_0, β_1，这里要设置成 $1, 10$。

```
beta = np.array([1, 10])
```

然后还要在数据中加上误差项，所以生成一个长度为k的正态分布样本。

```
e = np.random.normal(size=nsample)
```

由此，我们生成反应项 $y(t)$。

```
y = np.dot(X, beta) + e
```

好嘞，在反应变量和回归变量上使用 OLS() 函数。

```
model = sm.OLS(y,X)
```

然后获取拟合结果。

```
results = model.fit()
```

再调取计算出的回归系数。

```
print(results.params)
```

得到

```
[ 1.04510666, 9.97239799]
```

和实际的回归系数非常接近。

当然，也可以将回归拟合的摘要全部打印出来。

```
print(results.summary())
```

得到

OLS Regression Results

Dep. Variable:	y	R-squared:	0.999
Model:	OLS	Adj. R-squared:	0.999
Method:	Least Squares	F-statistic:	8.249e+04
Date:	Tue, 12 Jul 2016	Prob (F-statistic):	3.52e-145
Time:	19:25:24	Log-Likelihood:	-142.12
No. Observations:	100	AIC:	288.2
Df Residuals:	98	BIC:	293.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	1.0451	0.201	5.200	0.000	0.646	1.444
x1	9.9724	0.035	287.213	0.000	9.903	10.041

Omnibus:	0.993	Durbin-Watson:	2.048
Prob(Omnibus):	0.609	Jarque-Bera (JB):	0.943
Skew:	0.015	Prob(JB):	0.624
Kurtosis:	2.525	Cond. No.	11.7

中间偏下的 coef 列就是计算出的回归系数。

我们还可以将拟合结果画出来。先调用拟合结果的 fittedvalues 得到拟合的 y 值。

```
y_fitted = results.fittedvalues
```

然后使用 matplotlib.pyploft 画图。首先设定图轴，图片大小为 8×6。

```
fig, ax = plt.subplots(figsize=(8,6))
```

画出原数据，图像为圆点，默认颜色为蓝。

```
ax.plot(x, y, 'o', label='data')
```

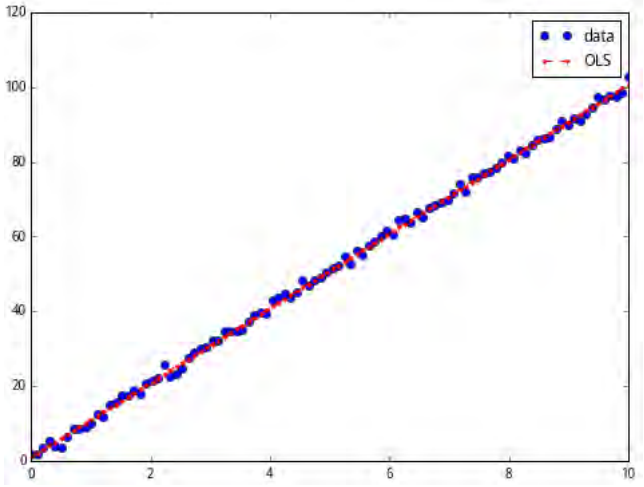
画出拟合数据，图像为红色带点间断线。

```
ax.plot(x, y_fitted, 'r--.',label='OLS')
```

放置注解。

```
ax.legend(loc='best')
```

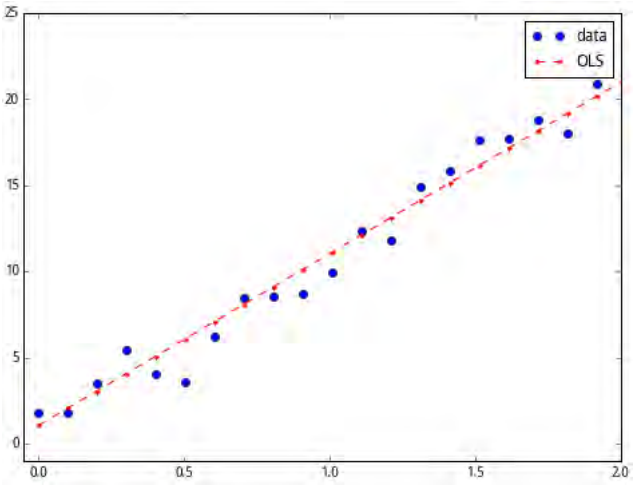
得到



在大图中看不清细节，我们在 0 到 2 的区间放大一下，可以见数据和拟合的关系。

加入改变坐标轴区间的指令

```
ax.axis((-0.05, 2, -1, 25))
```



高次模型的回归

假设反应变量 Y 和回归变量 X 的关系是高次的多项式，即

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_n X^n,$$

我们依然可以使用 OLS 进行线性回归。但前提条件是，我们必须知道 X 在这个关系中的所有次方数；比如，如果这个公式里有一个 $X^2.5$ 项，但我们对此并不知道，那么用线性回归的方法就不能得到准确的拟合。

虽然 X 和 Y 的关系不是线性的，但是 Y 和 $X, X^2, \ldots, X^n$ 的关系是高元线性的。也就是说，只要我们把高次项当做其他的自变量，即设 $X_1 = X, X_2 = X^2, \ldots, X_n = X^n$。那么，对于线性公式

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

可以进行常规的 OLS 回归，估测出每一个回归系数 β_i。可以理解为把一元非线性的问题映射到高元，从而变成一个线性关系。

下面以

$$Y = 1 + 0.1 \cdot X + 10 \cdot X^2$$

为例做一次演示。

首先设定数据量，也就是上面的 k 值。

```
nsample = 100
```

然后创建一个 array，是上面的 x_1 的数据。这里，我们想要 x_1 的值从 0 到 10 等差排列。

```
x = np.linspace(0, 10, nsample)
```

再创建一个 $k \times 2$ 的 array，两列分别为 x_1 和 x_2。我们需要 x_2 为 x_1 的平方。

```
X = np.column_stack((x, x**2))
```

使用 sm.add_constant() 在 array 上加入一列常项 1。

```
X = sm.add_constant(X)
```

然后设置模型里的 $\beta_0, \beta_1, \beta_2$，我们想设置成 $1, 0.1, 10$。

```
beta = np.array([1, 0.1, 10])
```

然后还要在数据中加上误差项，所以生成一个长度为k的正态分布样本。

```
e = np.random.normal(size=nsample)
```

由此，我们生成反应项 $y(t)$，它与 $x_1(t)$ 是二次多项式关系。

```
y = np.dot(X, beta) + e
```

在反应变量和回归变量上使用 OLS() 函数。

```
model = sm.OLS(y,X)
```

然后获取拟合结果。

```
results = model.fit()
```

再调取计算出的回归系数。

```
print(results.params)
```

得到

```
[ 0.95119465, 0.10235581, 9.9998477]
```

获取全部摘要

```
print(results.summary())
```

得到

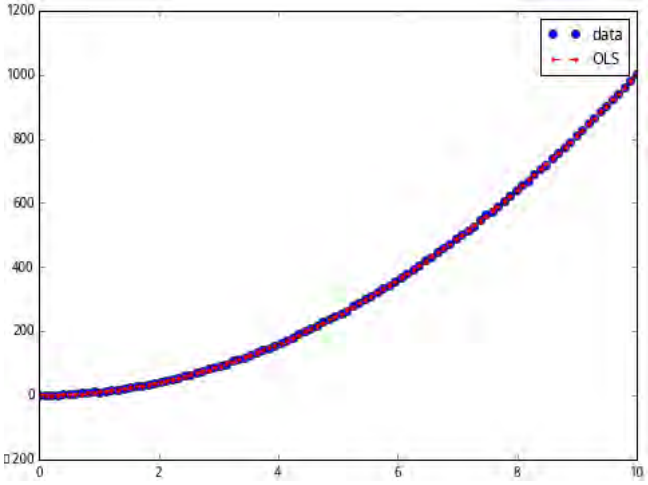
OLS Regression Results

Dep. Variable:	y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	4.335e+06
Date:	Fri, 08 Jul 2016	Prob (F-statistic):	7.29e-241
Time:	12:57:41	Log-Likelihood:	-142.77
No. Observations:	100	AIC:	291.5
Df Residuals:	97	BIC:	299.4
Df Model:	2		
Covariance Type:	nonrobust		

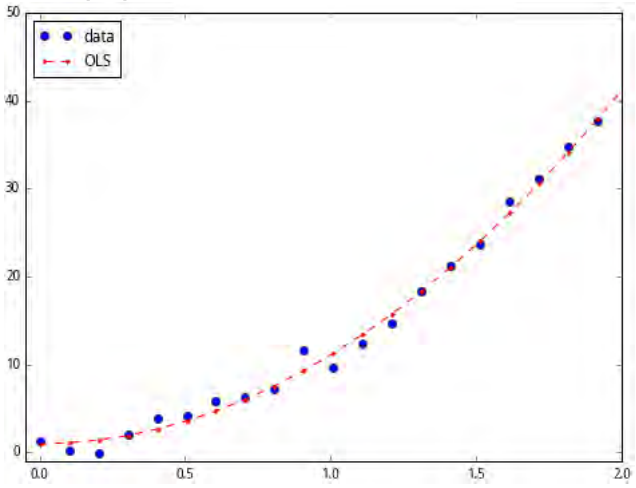
	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	0.9512	0.301	3.158	0.002	0.353	1.549
x1	0.1024	0.139	0.735	0.464	-0.174	0.379
x2	9.9998	0.013	742.263	0.000	9.973	10.027

Omnibus:	0.858	Durbin-Watson:	2.127
Prob(Omnibus):	0.651	Jarque-Bera (JB):	0.397
Skew:	0.007	Prob(JB):	0.820
Kurtosis:	3.308	Cond. No.	144.

拟合结果图如下



在横轴的 $[0, 2]$ 区间放大，可以看到



哑变量

一般而言，有连续取值的变量叫做连续变量，它们的取值可以是任何的实数，或者是某一区间里的任何实数，比如股价、时间、身高。但有些性质不是连续的，只有有限个取值的可能性，一般是用于分辨类别，比如性别、婚姻情况、股票所属行业，表达这些变量叫做分类变量。在回归分析中，我们需要将分类变量转化为哑变量(dummy variable)。

如果我们想表达一个有 d 种取值的分类变量，那么它所对应的哑变量的取值是一个 d 元组（可以看成一个长度为 d 的向量），其中有一个元素为 1，其他都是 0。元素呈现出 1 的位置就是变量所取的类别。比如说，某个分类变量的取值是 {a,b,c,d}，那么类别 a 对应的哑变量是$(1,0,0,0)$，b 对应 $(0,1,0,0)$，c 对应 $(0,0,1,0)$，d 对应 $(0,0,0,1)$。这么做的用处是，假如 a、b、c、d 四种情况分别对应四个系数 $\beta_0,\beta_1,\beta_2,\beta_3$，设 (x_0,x_1,x_2,x_3) 是一个取值所对应的哑变量，那么

$$\beta_0 x_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3$$

可以直接得出相应的系数。可以理解为，分类变量的取值本身只是分类，无法构成任何线性关系，但是若映射到高元的 $0,1$ 点上，便可以用线性关系表达，从而进行回归。

Statsmodels 里有一个函数 categorical() 可以直接把类别 {0,1,…,d-1} 转换成所对应的元组。确切地说，sm.categorical() 的输入有 (data, col, dictnames, drop) 四个。其中，data 是一个 $k\times 1$ 或 $k\times 2$ 的 array，其中记录每一个样本的分类变量取值。drop 是一个 Bool值，意义为是否在输出中丢掉样本变量的值。中间两个输入可以不用在意。这个函数的输出是一个 $k\times d$ 的 array（如果 drop=False，则是$k\times(d+1)$），其中每一行是所对应的样本的哑变量；这里 d 是 data 中分类变量的类别总数。

我们来举一个例子。这里假设一个反应变量 Y 对应连续自变量 X 和一个分类变量 Z。常项系数为 10，X 的系数为 1；Z 有 {a,b,c}三个种类，其中 a 类有系数 1，b 类有系数 3，c 类有系数 8。也就是说，将 Z 转换为哑变量 (Z_1,Z_2,Z_3)，其中 Z_i 取值于 $0,1$，有线性公式

$$Y=10+X+Z_1+3\cdot Z_2+8\cdot Z_3.$$

可以用常规的方法进行 OLS 回归。

我们按照这个关系生成一组数据来做一次演示。先定义样本数量为 50。

```
nsample = 50
```

设定分类变量的 array。前 20 个样本分类为 a。

```
groups = np.zeros(nsample, int)
```

之后的 20 个样本分类为 b。

```
groups[20:40] = 1
```

最后 10 个是 c 类。

```
groups[40:] = 2
```

转变成哑变量。

```
dummy = sm.categorical(groups, drop=True)
```

创建一组连续变量，是 50 个从 0 到 20 递增的值。

```
x = np.linspace(0, 20, nsample)
```

将连续变量和哑变量的 array 合并，并加上一列常项。

```
X = np.column_stack((x, dummy))
X = sm.add_constant(X)
```

定义回归系数。我们想设定常项系数为 10，唯一的连续变量的系数为 1，并且分类变量的三种分类 a、b、c 的系数分别为 $1,3,8$。

```
beta = [10, 1, 1, 3, 8]
```

再生成一个正态分布的噪音样本。

```
e = np.random.normal(size=nsample)
```

最后，生成反映变量。

```
y = np.dot(X, beta) + e
```

得到了虚构数据后，放入 OLS 模型并进行拟合运算。

```
result = sm.OLS(y,X).fit()
print(result.summary())
```

得到

OLS Regression Results

Dep. Variable:	y	R-squared:	0.991
Model:	OLS	Adj. R-squared:	0.990
Method:	Least Squares	F-statistic:	1628.
Date:	Fri, 08 Jul 2016	Prob (F-statistic):	1.10e-46
Time:	16:25:07	Log-Likelihood:	-65.273
No. Observations:	50	AIC:	138.5
Df Residuals:	46	BIC:	146.2
Df Model:	3		
Covariance Type:	nonrobust		

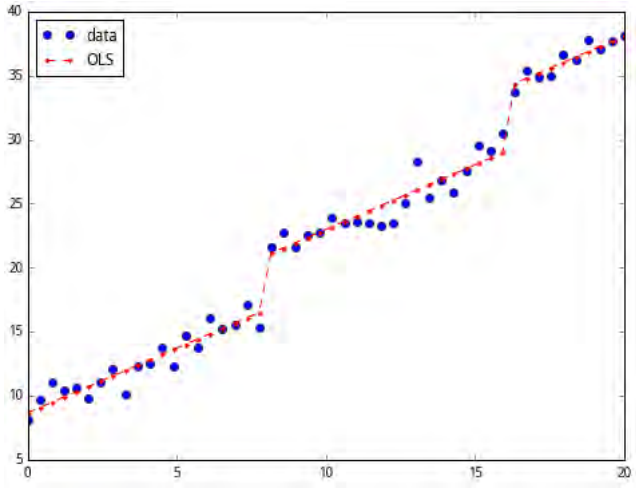
	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	9.8569	0.527	18.692	0.000	8.795	10.918
x1	1.0112	0.061	16.666	0.000	0.889	1.133
x2	-1.2012	0.334	-3.592	0.001	-1.874	-0.528
x3	3.0207	0.279	10.808	0.000	2.458	3.583
x4	8.0374	0.630	12.765	0.000	6.770	9.305

Omnibus:	0.015	Durbin-Watson:	2.138
Prob(Omnibus):	0.992	Jarque-Bera (JB):	0.129
Skew:	0.038	Prob(JB):	0.937
Kurtosis:	2.763	Cond. No.	2.12e+17

再画图出来

```
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="data")
ax.plot(x, result.fittedvalues, 'r--.', label="OLS")
ax.legend(loc='best')
```

得出



这里要指出，哑变量是和其他自变量并行的影响因素，也就是说，哑变量和原先的 x 同时影响了回归的结果。初学者往往会误解这一点，认为哑变量是一个选择变量：也就是说，上图中给出的回归结果，是在只做了一次回归的情况下完成的，而不是分成 3 段进行 3 次回归。哑变量的取值藏在其他的三个维度中。可以理解成：上图其实是将高元的回归结果映射到平面上之后得到的图。

简单应用

我们来做一个非常简单的实际应用。设 x 为上证指数的日收益率，y 为深证成指的日收益率。通过对股票市场的认知，我们认为 x 和 y 有很强的线性关系。因此可以假设模型

$$y = \beta_0 + \beta_1 x$$

并使用 Statsmodels 包进行 OLS 回归分析。

我们取上证指数和深证成指一年中的收盘价。

```
data = get_price(['000001.XSHG', '399001.XSHE'], start_date='2015-01-01', end_date='2016-01-01', frequency='daily', fields=['close'])['close']
x_price = data['000001.XSHG'].values
y_price = data['399001.XSHE'].values
```

计算两个指数一年内的日收益率，记载于 x_pct 和 y_pct 两个 list 中。

```
x_pct, y_pct = [], []
for i in range(1, len(x_price)):
    x_pct.append(x_price[i]/x_price[i-1]-1)
for i in range(1, len(y_price)):
    y_pct.append(y_price[i]/y_price[i-1]-1)
```

将数据转化为 array 的形式；不要忘记添加常数项。

```
x = np.array(x_pct)
X = sm.add_constant(x)
y = np.array(y_pct)
```

上吧，λu.λv.(sm.OLS(u,v).fit())！全靠你了！

```
results = sm.OLS(y, X).fit()
print(results.summary())
```

得到

OLS Regression Results

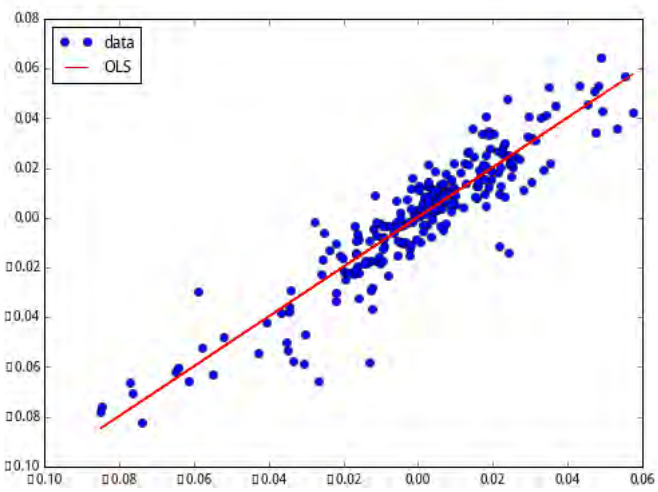
Dep. Variable:	y	R-squared:	0.846
Model:	OLS	Adj. R-squared:	0.845
Method:	Least Squares	F-statistic:	1325.
Date:	Fri, 08 Jul 2016	Prob (F-statistic):	6.56e-100
Time:	17:12:33	Log-Likelihood:	765.13
No. Observations:	243	AIC:	-1526.
Df Residuals:	241	BIC:	-1519.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	0.0002	0.001	0.327	0.744	-0.001	0.002
x1	0.9991	0.027	36.396	0.000	0.945	1.053

Omnibus:	41.392	Durbin-Watson:	2.013
Prob(Omnibus):	0.000	Jarque-Bera (JB):	95.086
Skew:	-0.802	Prob(JB):	2.25e-21
Kurtosis:	5.611	Cond. No.	41.0

恩，$y = 0.002 + 0.9991x$，合情合理，或者干脆直接四舍五入到 $y = x$。最后，画出数据和拟合线。

```
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="data")
ax.plot(x, results.fittedvalues, 'r--', label="OLS")
ax.legend(loc='best')
```



结语

本篇文章中，我们介绍了 Statsmodels 中很常用 OLS 回归功能，并展示了一些使用方法。线性回归的应用场景非常广泛。在我们量化课堂应用类的内容中，也有相当多的策略内容采用线性回归的内容。我们会将应用类文章中涉及线性回归的部分加上链接，链接到本篇文章中来，形成体系。量化课堂在未来还会介绍 Statsmodel 包其他的一些功能，敬请期待。

文章更迭记录：
v1.2，2016-09-09，修正公式，感谢 深蓝lhsfcboy 指出
v1.1，2016-07-18，修正文字
v1.0，2016-07-13，文章上线

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

简单线性回归¶

In [13]:

```
nsample = 100
x = np.linspace(0, 10, nsample)
X = sm.add_constant(x)
beta = np.array([1, 10])
e = np.random.normal(size=nsample)
y = np.dot(X, beta) + e
```

In [14]:

```
model = sm.OLS(y,X)
results = model.fit()
```

In [15]:

```
print(results.params)
```

```
[ 1.15699288  9.98710951]
```

In [16]:

```
print(results.summary())
```

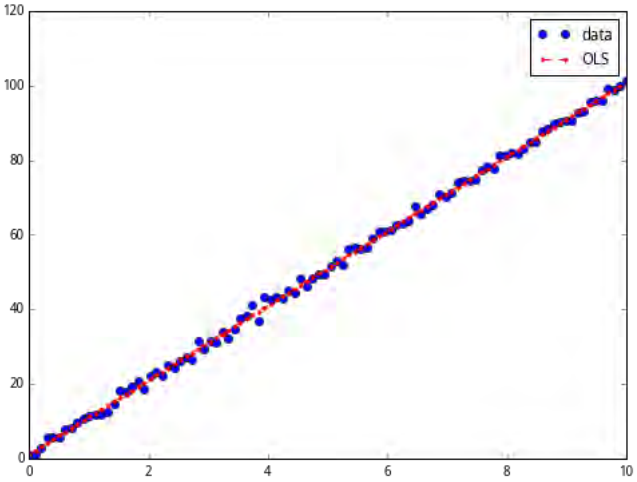
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.999
Model:                            OLS   Adj. R-squared:                  0.999
Method:                 Least Squares   F-statistic:                 7.396e+04
Date:                Wed, 13 Jul 2016   Prob (F-statistic):          7.35e-143
Time:                        14:45:48   Log-Likelihood:                -147.72
No. Observations:                 100   AIC:                             299.4
Df Residuals:                      98   BIC:                             304.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          1.1570      0.213      5.443      0.000         0.735     1.579
x1             9.9871      0.037    271.957      0.000         9.914    10.060
==============================================================================
Omnibus:                        0.573   Durbin-Watson:                   2.369
Prob(Omnibus):                  0.751   Jarque-Bera (JB):                0.717
Skew:                           0.137   Prob(JB):                        0.699
Kurtosis:                       2.688   Cond. No.                         11.7
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [17]:

```
y_fitted = results.fittedvalues
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label='data')
ax.plot(x, y_fitted, 'r--.',label='OLS')
ax.legend(loc='best')
```

Out[17]:

```
<matplotlib.legend.Legend at 0x7fc4814a9438>
```

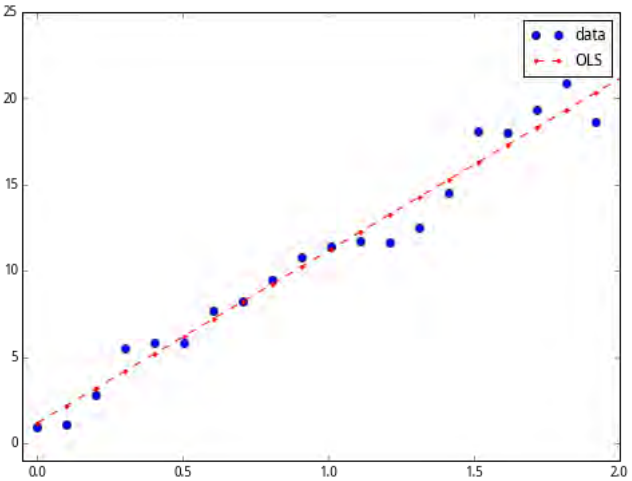


In [22]:

```
y_fitted = results.fittedvalues
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label='data')
ax.plot(x, y_fitted, 'r--.',label='OLS')
ax.legend(loc='best')
ax.axis((-0.05, 2, -1, 25))
```

Out[22]:

```
(-0.05, 2, -1, 25)
```



高次模型的回归¶

In [23]:

```
nsample = 100
x = np.linspace(0, 10, nsample)
X = np.column_stack((x, x**2))
X = sm.add_constant(X)
beta = np.array([1, 0.1, 10])
e = np.random.normal(size=nsample)
y = np.dot(X, beta) + e
```

In [24]:

```
model = sm.OLS(y,X)
results = model.fit()
```

In [25]:

```
print(results.params)
```

```
[ 0.6864524   0.20010325  9.9915112 ]
```

In [26]:

```
print(results.summary())
```

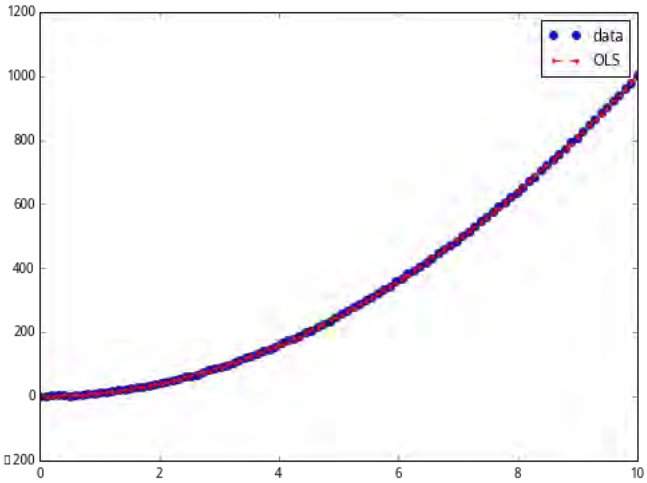
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 4.614e+06
Date:                Wed, 13 Jul 2016   Prob (F-statistic):          3.57e-242
Time:                        14:47:19   Log-Likelihood:                -139.67
No. Observations:                 100   AIC:                             285.3
Df Residuals:                      97   BIC:                             293.2
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          0.6865      0.292      2.351      0.021         0.107     1.266
x1             0.2001      0.135      1.483      0.141        -0.068     0.468
x2             9.9915      0.013    765.007      0.000         9.966    10.017
==============================================================================
Omnibus:                        3.918   Durbin-Watson:                   2.046
Prob(Omnibus):                  0.141   Jarque-Bera (JB):                2.354
Skew:                          -0.145   Prob(JB):                        0.308
Kurtosis:                       2.306   Cond. No.                         144.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [27]:

```
y_fitted = results.fittedvalues
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label='data')
ax.plot(x, y_fitted, 'r--.',label='OLS')
ax.legend(loc='best')
```

Out[27]:

```
<matplotlib.legend.Legend at 0x7fc481325208>
```

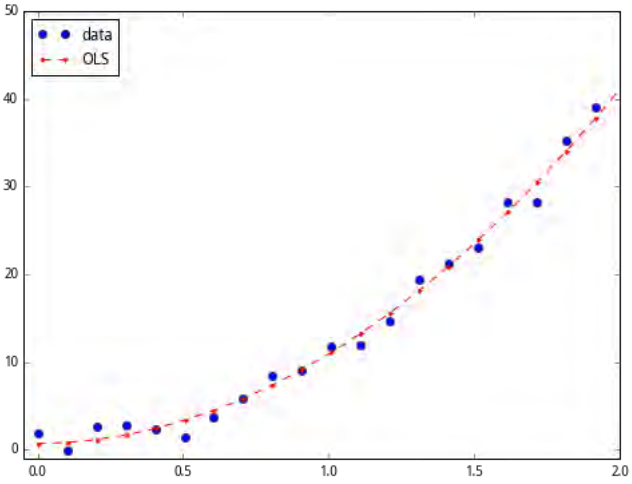


In [28]:

```
y_fitted = results.fittedvalues
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label='data')
ax.plot(x, y_fitted, 'r--.',label='OLS')
ax.legend(loc='best')
ax.axis((-0.05, 2, -1, 50))
```

Out[28]:

```
(-0.05, 2, -1, 50)
```



哑变量¶

In [62]:

```
nsample = 50
groups = np.zeros(nsample, int)
groups[20:40] = 1
groups[40:] = 2
dummy = sm.categorical(groups, drop=True)
x = np.linspace(0, 20, nsample)
X = np.column_stack((x, dummy))
X = sm.add_constant(X)
beta = [10, 1, 1, 3, 8]
e = np.random.normal(size=nsample)
y = np.dot(X, beta) + e
```

In [63]:

```
result = sm.OLS(y,X).fit()
print(result.summary())
```

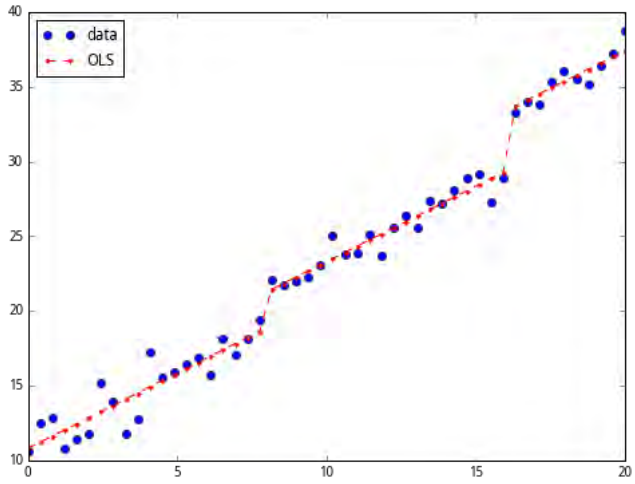
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.987
Model:                            OLS   Adj. R-squared:                  0.986
Method:                 Least Squares   F-statistic:                     1180.
Date:                Wed, 13 Jul 2016   Prob (F-statistic):           1.68e-43
Time:                        14:53:16   Log-Likelihood:                -67.128
No. Observations:                  50   AIC:                             142.3
Df Residuals:                      46   BIC:                             149.9
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         10.3005      0.547     18.822      0.000         9.199    11.402
x1             1.0102      0.063     16.043      0.000         0.883     1.137
x2             0.4935      0.347      1.422      0.162        -0.205     1.192
x3             2.8904      0.290      9.966      0.000         2.307     3.474
x4             6.9166      0.653     10.585      0.000         5.601     8.232
==============================================================================
Omnibus:                        0.377   Durbin-Watson:                   2.017
Prob(Omnibus):                  0.828   Jarque-Bera (JB):                0.031
Skew:                           0.000   Prob(JB):                        0.985
Kurtosis:                       3.122   Cond. No.                     2.12e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.51e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [64]:

```
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="data")
ax.plot(x, result.fittedvalues, 'r--.', label="OLS")
ax.legend(loc='best')
```

Out[64]:

```
<matplotlib.legend.Legend at 0x7fc4810369e8>
```



简单应用¶

In [65]:

```
data = get_price(['000001.XSHG', '399001.XSHE'], start_date='2015-01-01', end_date='2016-01-01', frequency='daily', fields=['close'])
x_price = data['000001.XSHG'].values
y_price = data['399001.XSHE'].values
```

In [67]:

```
x_pct, y_pct = [], []
for i in range(1, len(x_price)):
    x_pct.append(x_price[i]/x_price[i-1]-1)
for i in range(1, len(y_price)):
    y_pct.append(y_price[i]/y_price[i-1]-1)

x = np.array(x_pct)
X = sm.add_constant(x)
y = np.array(y_pct)
```

In [68]:

```
results = sm.OLS(y, X).fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.846
Model:                            OLS   Adj. R-squared:                  0.845
Method:                 Least Squares   F-statistic:                     1325.
Date:                Wed, 13 Jul 2016   Prob (F-statistic):          6.56e-100
Time:                        14:54:37   Log-Likelihood:                765.13
No. Observations:                 243   AIC:                            -1526.
Df Residuals:                     241   BIC:                            -1519.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          0.0002      0.001      0.327      0.744        -0.001     0.002
x1             0.9991      0.027     36.396      0.000         0.945     1.053
==============================================================================
Omnibus:                       41.392   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               95.086
Skew:                          -0.802   Prob(JB):                     2.25e-21
Kurtosis:                       5.611   Cond. No.                         41.0
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

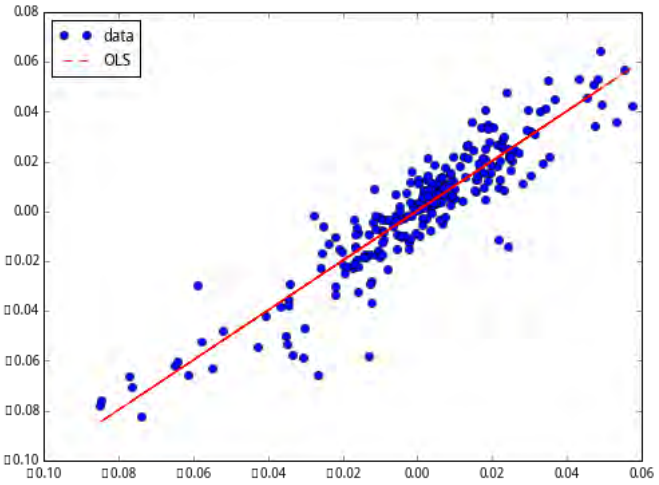
In [69]:

```
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="data")
```

```
ax.plot(x, results.fittedvalues, 'r--', label="OLS")
ax.legend(loc='best')
```

Out[69]:

```
<matplotlib.legend.Legend at 0x7fc481303c50>
```



【量化课堂】scikit-learn 之 kNN 分类

导语：scikit-learn是Python中一个功能非常齐全的机器学习库，本篇文章将介绍如何用scikit-learn来进行kNN分类计算。

```
作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，本文的难度属于进阶（上），深度为level-1。
```

阅读本文之前请掌握 kNN（level-1）的知识。
建议读者掌握 kd树（level-1）的知识。
不费话，

```
from sklearn import neighbors
```

开始吧。

功能详解

本篇中，我们讲解的是 scikit-learn 库中的 neighbors.KNeighborsClassifier，翻译为 k 最近邻分类功能，也就是我们常说的 kNN，k-nearest neighbors。首先进行这个类初始化：

```
neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n-
```

好多啊参数呀，真是的。来，一个一个讲。

n_neighbors 就是 kNN 里的 k，就是在做分类时，我们选取问题点最近的多少个最近邻。

weights 是在进行分类判断时给最近邻附上的加权，默认的 'uniform' 是等权加权，还有 'distance' 选项是按照距离的倒数进行加权，也可以使用用户自己设置的其他加权方法。举个例子：假如距离询问点最近的三个数据点中，有 1 个 A 类和 2 个 B 类，并且假设 A 类离询问点非常近，而两个 B 类距离则稍远。在等权加权中，3NN 会判断问题点为 B 类；而如果使用距离加权，那么 A 类有更高的权重（因为更近），如果它的权重高于两个 B 类的权重的总和，那么算法会判断问题点为 A 类。权重功能的选项应该视应用的场景而定。

algorithm 是分类时采取的算法，有 'brute'、'kd_tree' 和 'ball_tree'。kd_tree 的算法在 kd 树文章中有详细介绍，而 ball_tree 是另一种基于树状结构的 kNN 算法，brute 则是最直接的蛮力计算。根据样本量的大小和特征的维度数量，不同的算法有各自的优势。默认的 'auto' 选项会在学习时自动选择最合适的算法，所以一般来讲选择 auto 就可以。

leaf_size 是 kd_tree 或 ball_tree 生成的树的树叶（树叶就是二叉树中没有分枝的节点）的大小。在 kd 树文章中我们所有的二叉树的叶子中都只有一个数据点，但实际上树叶中可以有多于一个的数据点，算法在达到叶子时在其中执行蛮力计算即可。对于很多使用场景来说，叶子的大小并不是很重要，我们设 leaf_size=1 就好。

metric 和 p，是我们在 kNN 入门文章中介绍过的距离函数的选项，如果 metric ='minkowski' 并且 p=p 的话，计算两点之间的距离就是

$$d((x < em > 1, \ldots, x_n), (y_1, \ldots, y_n)) = \left(\sum < /em > i = 1^n |x_i - y_i|^p \right)^{1/p}$$

一般来讲，默认的 metric='minkowski'（默认）和 p=2（默认）就可以满足大部分需求。其他的 metric 选项可见说明文档。metric_params 是一些特殊 metric 选项需要的特定参数，默认是 None。

n_jobs 是并行计算的线程数量，默认是 1，输入 -1 则设为 CPU 的内核数。

在创建了一个 KNeighborsClassifier 类之后，我们需要给它数据来进行学习。这时需要使用 fit() 拟合功能。

```
neighbors.KNeighborsClassifier.fit(X,y)
```

在这里：

X 是一个 list 或 array 的数据，每一组数据可以是 tuple 也可以是 list 或者一维 array，但要注意所有数据的长度必须一样（等同于特征的数量）。当然，也可以把 X 理解为一个矩阵，其中每一横行是一个样本的特征数据。

y 是一个和 X 长度相同的 list 或 array，其中每个元素是 X 中相对应的数据的分类标签。

KNeighborsClassifier 类在对训练数据执行 fit() 后会根据原先 algorithm 的选项，依据训练数据生成一个 kd_tree 或者 ball_tree。如果输入是 algorithm='brute'，则什么都不做。这些信息都会被保存在一个类中，我们可以用它进行预测和计算。几个常用的功能有：
k 最近邻

```
neighbors.KNeighborsClassifier.kneighbors(X=None, n_neighbors=None, return_distance= True)
```

这里 X 是一 list 或 array 的坐标，如果不提供，则默认输入训练时的样本数据。

n_neighbors 是指定搜寻最近的样本数据的数量，如果不提供，则以初始化 kNeighborsClassifier 时的 n_neighbors 为准。

这个功能输出的结果是 (dist=array[array[float]], index=array[array[int]])。index 的长度和 X 相同，index[i] 是长度为 n_neighbors 的一 array 的整数；假设训练数据是 fit(X_train, y_train)，那么 X_train(index[i][j]) 是在训练数据（X_train）中离 X[i] 第 j 近的元素，并且 dist[i][j] 是它们之间的距离。

输入的 return_distance 是是否输出距离，如果选择 False，那么功能的输出会只有 index 而没有 dist。
预测

```
neighbors.kNeighborsClassifier.predict(X)
```

也许是最常用的预测功能。输入 X 是一 list 或 array 的坐标，输出y是一个长度相同的 array，y[i] 是通过 kNN 分类对 X[i] 所预测的分类标签。
概率预测

```
neighbors.kNeighborsClassifier.predict_proba(X)
```

输入和上面的相同，输出 p 是 array[array[float]]，p[i][j] 是通过概率 kNN 判断 X[i] 属于第 j 类的概率。这里类别的排序是按照词典排序；举例来说，如果训练用的分类标签里有 (1,'1','a')三种，那么1就是第0类，'1' 是第1类，'a' 是第2类，因为在 Python 中 1<'1'<'a'。
正确率打分

```
neighbors.KNeighborsClassifier.score(X, y, sample_weight=None)
```

这是用来评估一次 kNN 学习的准确率的方法。很多可能会因为样本特征的选择不当或者 k 值得选择不当而出现过拟合或者偏差过大的问题。为了保证训练方法的准确性，一般我们会将已经带有分类标签的样本数据分成两组，一组进行学习，一组进行测试。这个 score() 就是在学习之后进行测试的功能。同 fit() 一样，这里的 X 是特征坐标，y 是样本的分类标签；sample_weight 是对样本的加权，长度等于 sample 的数量。返回的是正确率的百分比。

实际例子

好，举例子了。

除了sklearn.neighbors，还需要导入numpy和matplotlib画图。

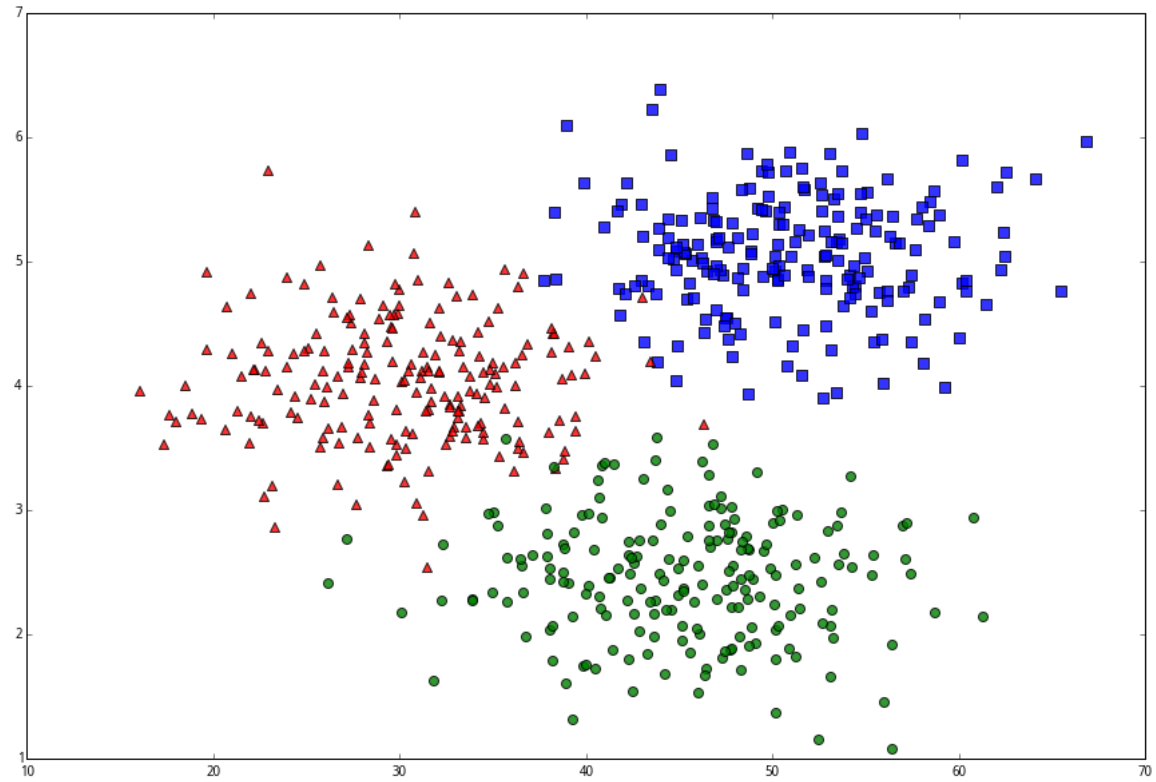
```
import random
from sklearn import neighbors
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

我们随机生成 6 组 200 个的正态分布

```
x1 = random.normal(50, 6, 200)
y1 = random.normal(5, 0.5, 200)

x2 = random.normal(30,6,200)
y2 = random.normal(4,0.5,200)

x3 = random.normal(45,6,200)
y3 = random.normal(2.5, 0.5, 200)
```

x1、x2、x3 作为 x 坐标，y1、y2、y3 作为 y 坐标，两两配对。(x1,y1) 标为 1 类，(x2, y2) 标为 2 类，(x3, y3)是 3 类。将它们画出得到下图，1 类是蓝色，2 类红色，3 类绿色。

```
plt.scatter(x1,y1,c='b',marker='s',s=50,alpha=0.8)
plt.scatter(x2,y2,c='r', marker='^', s=50, alpha=0.8)
plt.scatter(x3,y3, c='g', s=50, alpha=0.8)
```



我们把所有的 x 坐标和 y 坐标放在一起

```
x_val = np.concatenate((x1,x2,x3))
y_val = np.concatenate((y1,y2,y3))
```

记得计算距离的归一化问题吗？我们求出 x 值的最大差还有 y 值的最大差。

```
x_diff = max(x_val)-min(x_val)
y_diff = max(y_val)-min(y_val)
```

将坐标除以这个差以归一化，再将 x 和 y 值两两配对。

```
x_normalized = [x/(x_diff) for x in x_val]
y_normalized = [y/(y_diff) for y in y_val]
xy_normalized = zip(x_normalized,y_normalized)
```

训练使用的特征数据已经准备好了，还需要生成相应的分类标签。生成一个长度600的list，前200个是1，中间200个是2，最后200个是3，对应三种标签。

```
labels = [1]*200+[2]*200+[3]*200
```

然后，就要生成 sklearn 的最近 k 邻分类功能了。参数中，n_neighbors 设为 30，其他的都使用默认值即可。

```
clf = neighbors.KNeighborsClassifier(30)
```

（注意我们是从sklearn里导入了neighbors。如果是直接导入了sklearn，应该输入sklearn.neighbors.KNeighborsClassifier()）

下面就要进行拟合了。归一化的数据是 xy_normalized，分类标签是 labels，

```
clf.fit(xy_normalized, labels)
```

就这么简单。下面我们来实现一些功能。
k 最近邻

首先，我们想知道 (50,5) 和 (30,3) 两个点附近最近的 5 个样本分别都是什么。啊，坐标别忘了除以 x_diff 和 y_diff 来归一化。

```
nearests = clf.kneighbors([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)], 10, False)
nearests
```

得到

```
array([[ 97, 134, 177, 144,  10], [278, 569, 242, 324, 504]])
```

也就是说训练数据中的第 97、134、177、144、10 个离 (50,5) 最近，第 278、569、242、324、504 个离 (30,3) 最近。
预测

还是上面那两个点，我们通过 30NN 来判断它们属于什么类别。

```
prediction = clf.predict([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)])
prediction
```

得到

```
array([1, 2])
```

也就是说 (50,5) 判断为 1 类，而 (30,3) 是 2 类。
概率预测

那么这两个点的分类的概率都是多少呢?

```
prediction_proba = clf.predict_proba([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)])
prediction_proba
```

得到

```
array([[ 1. ,  0. ,  0. ], [ 0. ,  0.8,  0.2]])
```

告诉我们，(50, 5) 有 100% 的可能性是 1 类，而 (30,3) 有 80% 是 2 类，20% 是3类。
准确率打分

我们再用同样的均值和标准差生成一些正态分布点，以此检测预测的准确性。

```
x1_test = random.normal(50, 6, 100)
y1_test = random.normal(5, 0.5, 100)

x2_test = random.normal(30,6,100)
y2_test = random.normal(4,0.5,100)

x3_test = random.normal(45,6,100)
y3_test = random.normal(2.5, 0.5, 100)

xy_test_normalized = zip(np.concatenate((x1_test,x2_test,x3_test))/x_diff,\
                         np.concatenate((y1_test,y2_test,y3_test))/y_diff)

labels_test = [1]*100+[2]*100+[3]*100
```

测试数据生成完毕，下面进行测试

```
score = clf.score(xy_test_normalized, labels_test)
score
```

得到预测的正确率是 97% 还是很不错的。

再看一下，如果使用 1NN 分类，会出现过拟合的现象，那么准确率的平分就变为…

clf1 = neighbors.KNeighborsClassifier(1)
clf1.fit(xy_normalized, labels)
clf1.score(xy_test_normalized, labels_test)

95%，的确是降低了。我们还应该注意，这里的预测准确率很高是因为训练和测试的数据都是人为按照正态分布生成的，在实际使用的很多场景中（比如，涨跌预测）是很难达到这个精度的。

生成些漂亮的图

说到 kNN 那当然离不开分类图，不过这一般是为了教学用的，毕竟只能展示两个维度的数据，超过三个特征的话就画不出来了。所以这部分内容只是本篇的附加部分，有兴趣的读者可以向下阅读。

首先我们需要生成一个区域里大量的坐标点。这要用到 np.meshgrid() 函数。给定两个 array，比如 x=[1,2,3] 和 y=[4,5]，np.meshgrid(x,y) 会输出两个矩阵

Undefined control sequence \<

和

Undefined control sequence \<

这两个叠加到一起得到六个坐标，

Undefined control sequence \<

就是以 [1,2,3] 为横轴，[4,5] 为竖轴所得到的长方形区间内的所有坐标点。

好，我们现在要生成 [1,80]x[1,7] 的区间里的坐标点，横轴要每 0.1 一跳，竖轴每 0.01 一跳。于是

```
xx,yy = np.meshgrid(np.arange(1,70.1,0.1), np.arange(1,7.01,0.01))
```

于是 xx 和 yy 都是 601 乘 691 的矩阵。还有，不要忘了除以 x_diff 和 y_diff 来将坐标归一化。

```
xx_normalized = xx/x_diff
yy_normalized = yy/y_diff
```

下面，np.ndarray.ravel() 功能可以把一个矩阵抻直成一个一维 array，把

Undefined control sequence \<

变成

$$[1 \quad amp;2 \quad amp;3 \quad amp;1 \quad amp;2 \quad amp;3]$$

np.c_() 又把两个 array 粘起来（类似于 zip），输入

$$[1 \quad amp;2 \quad amp;3 \quad amp;1 \quad amp;2 \quad amp;3]$$

和

$$[4 \quad amp;4 \quad amp;4 \quad amp;5 \quad amp;5 \quad amp;5]$$

输出

Undefined control sequence \<

或者理解为

$$(1,4),(2,4),(3,4),(1,5),(2,5),(3,5)$$

于是

```
coords = np.c_[xx_normalized.ravel(), yy_normalized.ravel()]
```

得到一个 array 的坐标。下面就可以进行预测

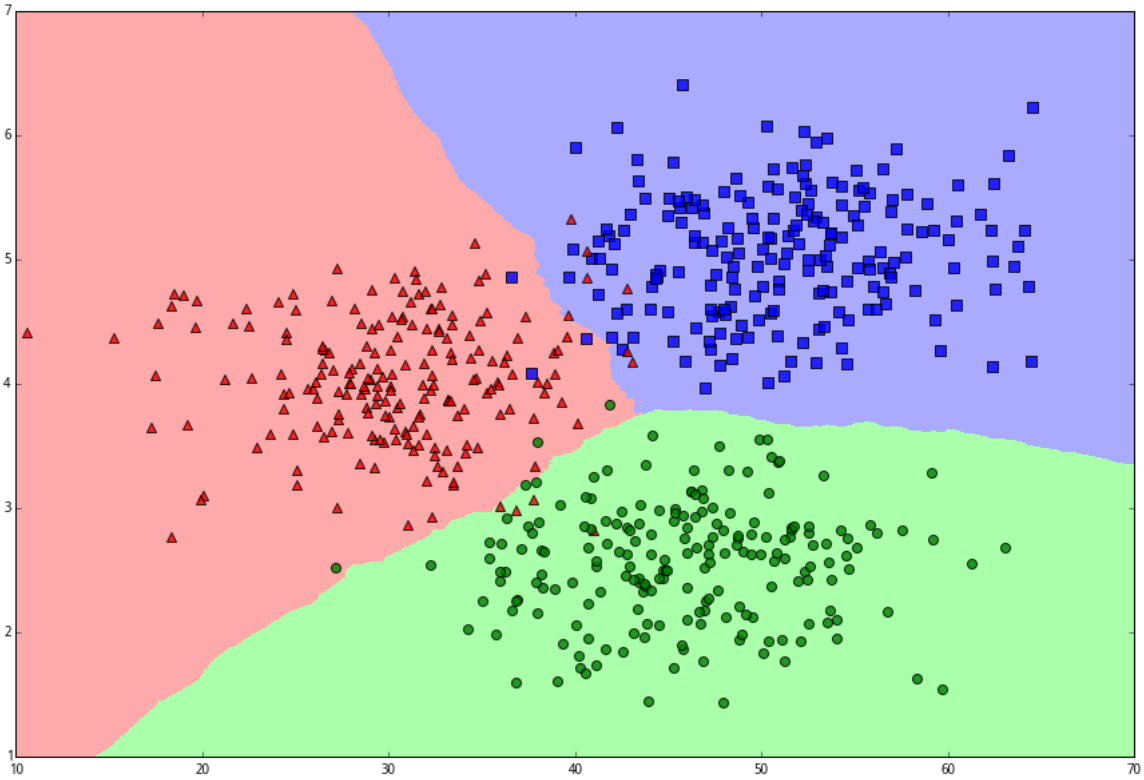
```
Z = clf.predict(coords)
```

当然，Z 是一个一维 array，为了和 xx 还有 yy 相对应，要把Z的形状再转换回矩阵

```
Z = Z.reshape(xx.shape)
```

下面用 pcolormesh 画出背景颜色。这里，ListedColormap 是自己生成 colormap 的功能，#rrggbb 颜色的 rgb 代码。pcolormesh 会根据 Z 的值（1、2、3）选择 colormap 里相对应的颜色。pcolormesh 和 ListedColormap 的具体使用方法会在未来关于画图的文章中细讲。

```
light_rgb = ListedColormap([ '#AAAAFF', '#FFAAAA','#AAFFAA'])
plt.pcolormesh(xx, yy,Z, cmap=light_rgb)
plt.scatter(x1,y1,c='b',marker='s',s=50,alpha=0.8)
plt.scatter(x2,y2,c='r', marker='^', s=50, alpha=0.8)
plt.scatter(x3,y3, c='g', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
```



下面再进行概率预测，使用

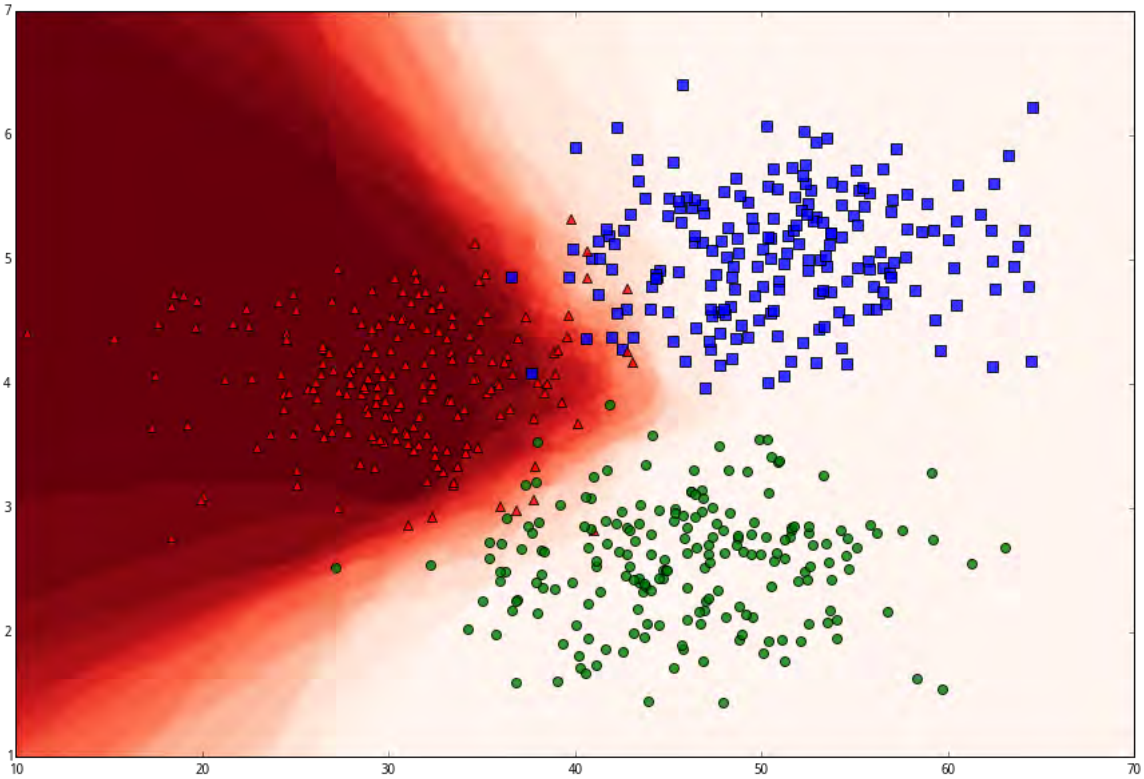
```
Z_proba = clf.predict_proba(coords)
```

得到每个坐标点的分类概率值。假设我们想画出红色的概率，那么提取所有坐标的 2 类概率，转换成矩阵形状

```
Z_proba_reds = Z_proba[:,1].reshape(xx.shape)
```

再选一个预设好的红色调 cmap 画出来

```
plt.pcolormesh(xx, yy,Z_proba_reds, cmap='Reds')
plt.scatter(x1,y1,c='b',marker='s',s=50,alpha=0.8)
plt.scatter(x2,y2,c='r', marker='^', s=50, alpha=0.8)
plt.scatter(x3,y3, c='g', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
```



结语

scikit-learn 包的功能非常齐全，使用 kNN 分类进行预测也简单易懂。使用的难点在于将数据整理成函数可以处理的格式的过程偏于繁琐，从输出中读取结论可能也有些麻烦。本文细致地讲解了包中函数的输入、输出以及处理方法，希望读者可以轻松地将这些功能运用在实际应用中。

v1.2，2016-11-30，修改错字，感谢 东皇太后 指出。
v1.1，2016-11-11，修补漏字，感谢 沙漠蒲公英 指出。
v1.0，2016-10-13，文章上线

In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import neighbors
from matplotlib.colors import ListedColormap
```

In [4]:

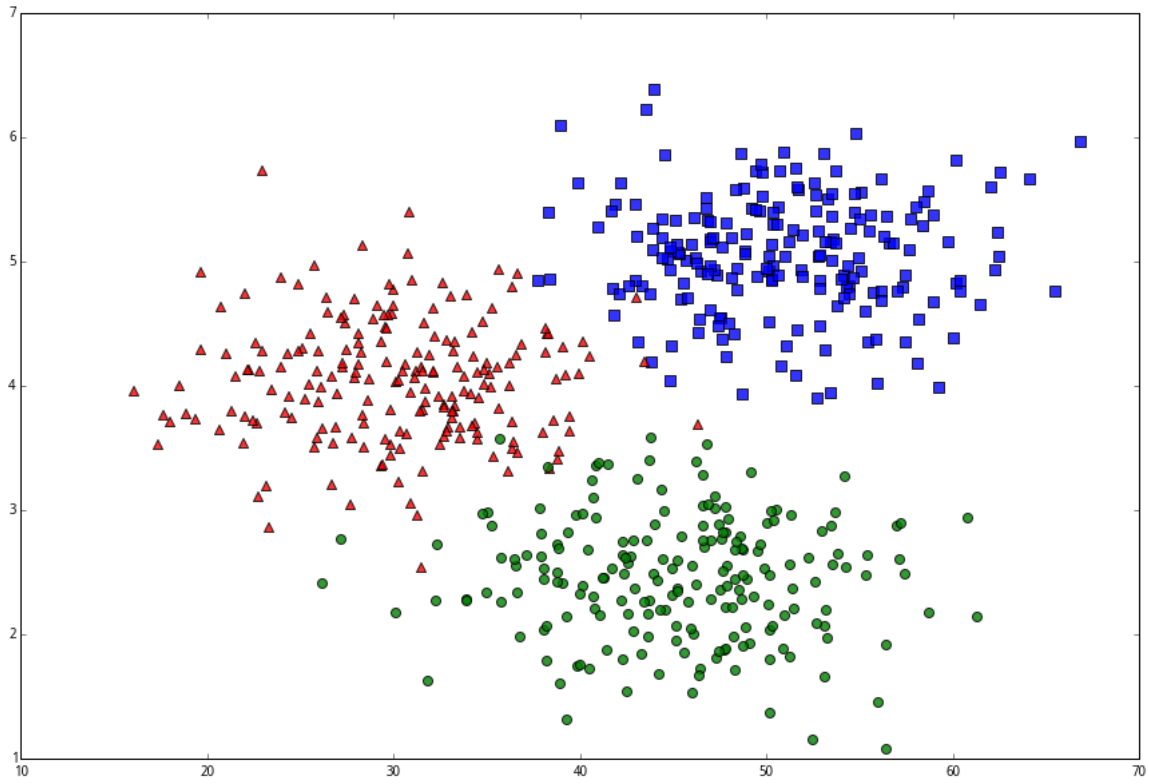
```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 10
```

In [5]:

```
x1 = random.normal(50, 6, 200)
y1 = random.normal(5, 0.5, 200)
plt.scatter(x1,y1,c='b',marker='s',s=50,alpha=0.8)

x2 = random.normal(30,6,200)
y2 = random.normal(4,0.5,200)
plt.scatter(x2,y2,c='r', marker='^', s=50, alpha=0.8)

x3 = random.normal(45,6,200)
y3 = random.normal(2.5, 0.5, 200)
plt.scatter(x3,y3, c='g', s=50, alpha=0.8)

plt.axis((10, 70,1,7))
```

Out[5]:

```
(10, 70, 1, 7)
```



In [5]:

```
x_val = np.concatenate((x1,x2,x3))
y_val = np.concatenate((y1,y2,y3))
```

In [6]:

```
x_diff = max(x_val)-min(x_val)
y_diff = max(y_val)-min(y_val)
```

In [7]:

```
x_normalized = x_val/x_diff
y_normalized = y_val/y_diff
xy_normalized = zip(x_normalized,y_normalized)
```

In [8]:

```
labels = [1]*200+[2]*200+[3]*200
```

In [9]:

```
clf = neighbors.KNeighborsClassifier(30)
```

In [10]:

```
clf.fit(xy_normalized, labels)
```

Out[10]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=30, p=2,
           weights='uniform')
```

In [11]:

```
nearests = clf.kneighbors([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)], 5, False)
nearests
```

Out[11]:

```
array([[131, 144,  75,  61, 115],
       [281, 236, 375, 291, 234]])
```

In [12]:

```
prediction = clf.predict([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)])
prediction
```

Out[12]:

```
array([1, 2])
```

In [13]:

```
prediction_proba = clf.predict_proba([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)])
prediction_proba
```

Out[13]:

```
array([[ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.93333333,  0.06666667]])
```

In [16]:

```
x1_test = random.normal(50, 6, 100)
y1_test = random.normal(5, 0.5, 100)

x2_test = random.normal(30,6,100)
y2_test = random.normal(4,0.5,100)

x3_test = random.normal(45,6,100)
y3_test = random.normal(2.5, 0.5, 100)

xy_test_normalized = zip(np.concatenate((x1_test,x2_test,x3_test))/x_diff,\
                         np.concatenate((y1_test,y2_test,y3_test))/y_diff)

labels_test = [1]*100+[2]*100+[3]*100
```

In [17]:

```
score = clf.score(xy_test_normalized, labels_test)
score
```

Out[17]:

```
0.93666666666666665
```

In [18]:

```
clf1 = neighbors.KNeighborsClassifier(1)
clf1.fit(xy_normalized, labels)
clf1.score(xy_test_normalized, labels_test)
```

Out[18]:

```
0.93333333333333335
```

In [ ]:

```

```

In [19]:

```
xx,yy = np.meshgrid(np.arange(1,70.1,0.1), np.arange(1,7.01,0.01))
```

In [20]:

```
xx_normalized = xx/x_diff
yy_normalized = yy/y_diff
```

In [21]:

```
coords = np.c_[xx_normalized.ravel(), yy_normalized.ravel()]
```

In [22]:

```
Z = clf.predict(coords)
```

In [23]:

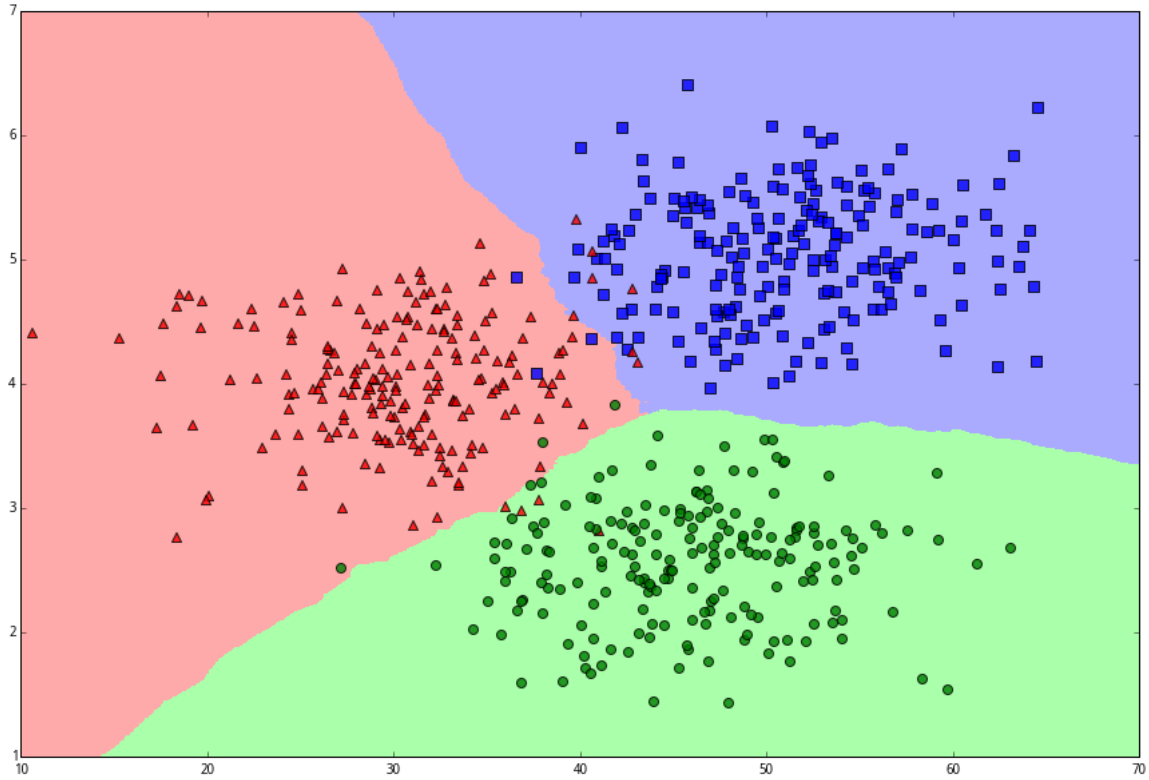
```
Z = Z.reshape(xx.shape)
```

In [24]:

```
light_rgb = ListedColormap([ '#AAAAFF', '#FFAAAA','#AAFFAA'])
plt.pcolormesh(xx, yy,Z, cmap=light_rgb)
plt.scatter(x1,y1,c='b',marker='s',s=50,alpha=0.8)
plt.scatter(x2,y2,c='r', marker='^', s=50, alpha=0.8)
plt.scatter(x3,y3, c='g', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
```

Out[24]:

```
(10, 70, 1, 7)
```



In [ ]:

```
```

In [27]:

```
Z_proba = clf.predict_proba(coords)
```

In [29]:

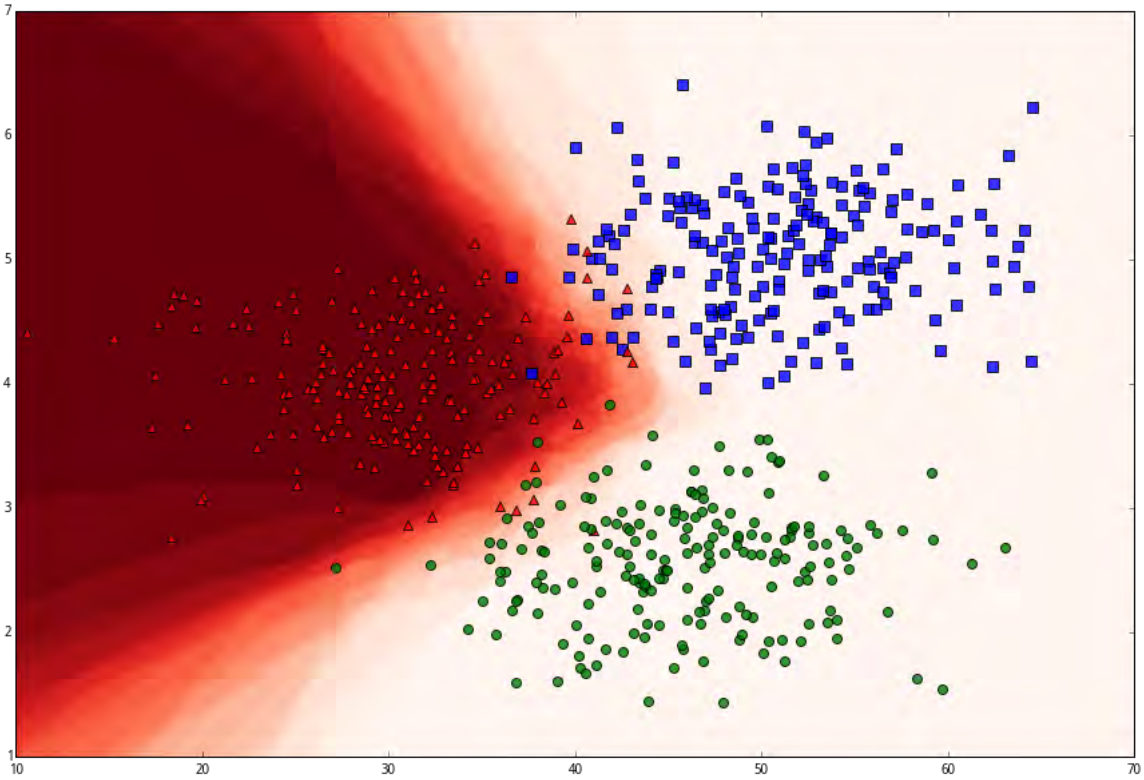
```
Z_proba_reds = Z_proba[:,1].reshape(xx.shape)
```

In [30]:

```
plt.pcolormesh(xx, yy,Z_proba_reds, cmap='Reds')
plt.scatter(x1,y1,c='b',marker='s',s=50,alpha=0.8)
plt.scatter(x2,y2,c='r', marker='^', s=50, alpha=0.8)
plt.scatter(x3,y3, c='g', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
```

Out[30]:

```
(10, 70, 1, 7)
```



In [ ]:

```
```

【量化课堂】多回测运行和参数分析框架

导语：本文提供了在研究中自动根据指定参数进行回测，并将一组回测结果可视化的框架。

```
作者：Sunx
编辑：肖睿

本文由JoinQuant量化课堂推出。难度为入门，深度为level-0。
```

引言

在因子分析系列中我们遇到了一个比较明显的需求，就是把全部股票按照各个分位进行回测，并且分析每个分位的收益情况。但其实有这个需求的不仅限于因子的分析，有时我们在策略中还会有一些其他的参数需要进行调整并分析调整后的结果。本篇文章对于这个需求设立了一个参数分析框架，集合运行回测和各种分析结果的功能于一体。也希望读者多提意见，我们会对其进行改进。

整个框架写在了一个parameter_analysis 类里面，附在文章结尾的研究模块。下面讲功能和使用方法。

初始化回测

如果想进行分析的回测还没有被运行过，我们需要运行回测并保存回测结果。首先，

```
parameter_analysis(algorithm_id=None)
```

将这个类初始化。输入中的 algorithm_id 是想要回测使用的策略编码，获取方式是策略编辑页 url 的尾部，如下

https://www.joinquant.com/algorithm/index/edit?algorithmId=9f668b395694629f[redacted]1a13c3f0f

parameter_analysis.run_backtest() 函数可以运行回测，并且 parameter_analysis.organize_backtest_results() 函数可以以回测结果分析一些指标和数据。但这两个函数都已经打包在 parameter_analysis.get_backtest_data() 中，所以我们只讲最后的这个函数。调用这个函数的输入较多，有

```
parameter_analysis.get_backtest_data(algorithm_id=None,
                                     benchmark_id_None,
                                     file_name='results.pkl',
                                     running_max=10,
                                     start_date='2006-01-01',
                                     end_date='2016-11-30',
                                     frequency='day',
                                     initial_cash='1000000',
                                     param_names=[],
                                     param_values=[]
                                    )
```

我们一个一个来讲。

- algorithm_id 正如其名是策略的 id 编码，和之前提到的一样。如果在创建 parameter_analysis 时已经提供了的话这里就不用再提供了。
- benchmark_id 是自定义基准的回测编码，注意不是策略而是回测。函数会以这个回测的收益率曲线作为基准曲线，要求这个回测的起始和结束时间和参数中的 start_date 与 end_date 分别吻合。如果不提供 benchmark_id，则策略会自动用 algorithm_id 对应的策略中指定的基准作为基准。
- file_name 是储存数据的文件名，因为是以 pickle 进行存储，文件名结尾必须是 .pkl。回测运行完毕后的 pickle 文件将被存储于和研究的 .ipyny 相同的文件夹，默认文件名为'results.pkl'。
- running_max 是同时可运行的回测的上限。由于在平台上同时运行的回测数量上限是 10 个，固 running_max 默认设为 10。但是，如果需要在这个框架运行的时候留出一些空位来运行其他回测，可以将 running_max 设为 9 或者更小的数字。当这个函数被要求运行超过 running_max 数量的回测时，会自动将它们进行排队，在之前的完成后再运行后面的。
- start_date 和 end_date 顾名思义就是回测起始和结束的日期，默认数值如上所述。
- frequency 是回测频率，按需求使用 'day' 或者 'minute'，默认为'day'。
- initial_cash 是回测的起始资金，默认一百万。
- 值得细讲的是 param_names 和 param_values 两个输入。这里 param_names 是一个 list 的 strings，它对应着策略中所有需要调整的全局变量的名字。比如说，我们需要将策略中的 g.abc 和 g.x_y_z 两个参数更改数值进行多次回测，那么 param_names 应该设为 ['abc', 'x_y_z']。而param_values 是一个 list 的 list，要求 len(param_values)==len(param_names)。其中 param_values[i] 是 param_names[i] 的名字所对应的调参值。举例来说，我们需要把 g.abc 和 g.x_y_z 分别赋予 ['a','b', 'c'] 和 [1,2] 的值进行回测，那么就应该输入 param_values=[['a','b','c'], [1,2]]。函数会自动列举所有这些参数选项的组合进行回测，在上面的例子中就会产生六个回测，分别对应参数值 (g.abc, g.x_y_z) 等于 ('a',1)、('a',2)、('b',1)、('b',2)、('c',1)、('c',2)。

在所有参数对应的回测运行完成之后，函数会整理数据并存储 pickle 文件。

数据内容和读取文件

当我们运行完 parameter_analysis.get_backtest_data() 之后，除了保存在 pickle 文件，还会直接保存在这个类下面。我们逐一介绍这些项目：

- parameter_analysis.algorithm_id 顾名思义就是我们输入的策略编码。
- parameter_analysis.params_df 是包含了所有回测使用的参数的 DataFrame。其中，横行 index 是回测的编号，代表它是第几个被运行的回测，从 0 开始计数。竖列 columns 是之前输入的 param_names 的名字。df 中的内容是回测相对应的参数数值。举例来说，param_names = ['abc', 'x_y_z']，param_values=[['a','b','c'], [1,2]] 时，生成的 Data Frame 如下：.

	abc	x_y_z
0	a	1
1	a	2
2	b	1
3	b	2
4	c	1
5	c	2

- parameter_analysis.evaluations 是一个 dict 的 dict。如果 i 是一个回测的编号（对应params_df 里的 index），那么 parameter_analysis.evaluations[i] 就是这个回测的各项指标，比如收益率、夏普比率、最大回撤。


```
{0: {u'algorithm_return': 11.411854404999987,
  u'algorithm_volatility': 0.32121429839499344,
  u'alpha': 0.18073140375063612,
  u'annual_algo_return': 0.27236964842838485,
  u'annual_bm_return': 0.09083216417665851,
  u'benchmark_return': 1.4819725192878503,
  u'benchmark_volatility': 0.2809114990509962,
  u'beta': 1.0158576860565844,
  u'excess_return': 10.982155774863001,
  u'information': 1.1284098427858311,
  u'max_drawdown': 0.651666430265925,
  u'max_drawdown_period': [u'2008-01-15', u'2008-11-04'],
  u'max_leverage': 0.0,
  u'period_label': u'2016-11',
  u'sharpe': 0.7234100399311696,
  u'sortino': 0.8579392188420932,
  u'trading_days': 2614,
  u'treasury_return': 0.42969863013698634},
```


- 当然，为了可以更直观地查看数据，还有 DataFrame 版的 parameter_analysis.evaluations_df。它是 evaluations 的 df 版，并且还带有 params_df 中的参数数据。

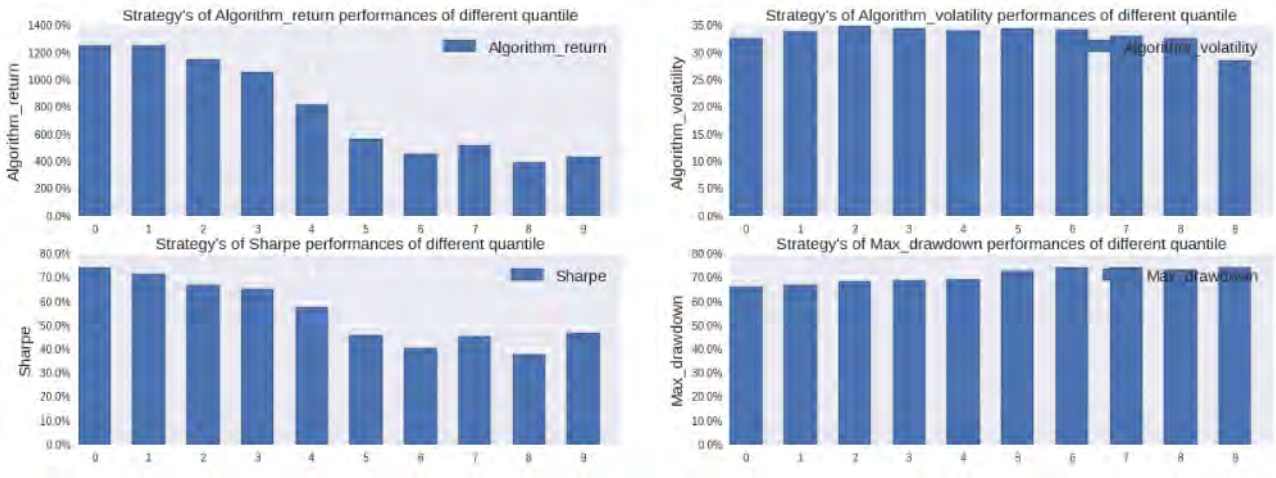
	factor	quantile	algorithm_return	algorithm_volatility	alpha	annual_algo_return	annual_bm_return	benchmark_return	I
0	BP	(0, 10)	11.41185	0.3212143	0.1807314	0.2723696	0.09083216	1.481973	(
1	BP	(10, 20)	11.73249	0.3407085	0.1806987	0.2754771	0.09083216	1.481973	(
2	BP	(20, 30)	15.57672	0.3464588	0.2126366	0.3080715	0.09083216	1.481973	(
3	BP	(30, 40)	11.00186	0.3464812	0.1730495	0.2682887	0.09083216	1.481973	(
4	BP	(40, 50)	7.429355	0.347341	0.1311858	0.2261453	0.09083216	1.481973	(
5	BP	(50, 60)	5.206482	0.3475327	0.09645008	0.1907672	0.09083216	1.481973	(
6	BP	(60, 70)	4.178264	0.3458848	0.07650819	0.1703177	0.09083216	1.481973	(
7	BP	(70, 80)	5.591166	0.3367506	0.10557	0.1976354	0.09083216	1.481973	(
8	BP	(80, 90)	4.303324	0.3306247	0.08225757	0.1729918	0.09083216	1.481973	(
9	BP	(90, 100)	2.040364	0.3125721	0.0252338	0.1122091	0.09083216	1.481973	(

- parameter_analysis.backtest_ids 是一个 dict 的 strings，keys 依旧是回测的编号，对应的 内容是该回测的 url 编码。
- parameter_analysis.benchmark_id 是自定义基准回测的编码，如果没提供则是空。
- parameter_analysis.dates 是一个 list 的 strings，是按序排列的回测中的每一个交易日。
- parameter_analysis.benchmark_returns 是一 list 的 float，是基准的收益率数据，对应着 dates 中相同位置的日期。
- parameter_analysis.returns 和 excess_returns 和 log_returns 和 log_excess_returns 都是一 dict 的 list，keys 是各个回测的编号，而相对应的 list 分别是该策略的收益率和超额收益率和对数收益率和对数超额收益率。其中，超额收益率是用回测净值除以基准净值计算而出，对数超额收益率是超额收益率的对数。和 benchmark_returns 一样，list 中的数据对应着 dates 中相同位置的日期。
- 最后两个特别计算的指标是 self.excess_max_drawdown 和 self.excess_annual_return。这两个都是 dict，keys 是回测的编号，内容分别是策略超额收益曲线的最大回撤和策略超额收益曲线的年化收益率。这两个指标的意义在于，有时我们想看策略相对于基准的强弱，所以在排除掉基准的影响之后可以进行一些有意义的分析。
- 如果我们通过 parameter_analysis.get_backtest_data() 运行了回测并且获取了各种数据，但是关闭了研究模块，那么在再次打开研究继续分析时不需要再重新把回测都运行一遍，可以使用读取 pickle 文件的函数把已保存的数据读取。
- 使用 parameter_analysis.read_backtest_data(file_name='results.pkl') 即可读取数据。这里 file_name 是被读取的文件名，如果不是默认的 results.pkl 的话则需要另行输入。在读取之后，parameter_analysis 中的每个项目会变成上面所讲的内容，可以进行调取或者使用接下来的介绍的功能。

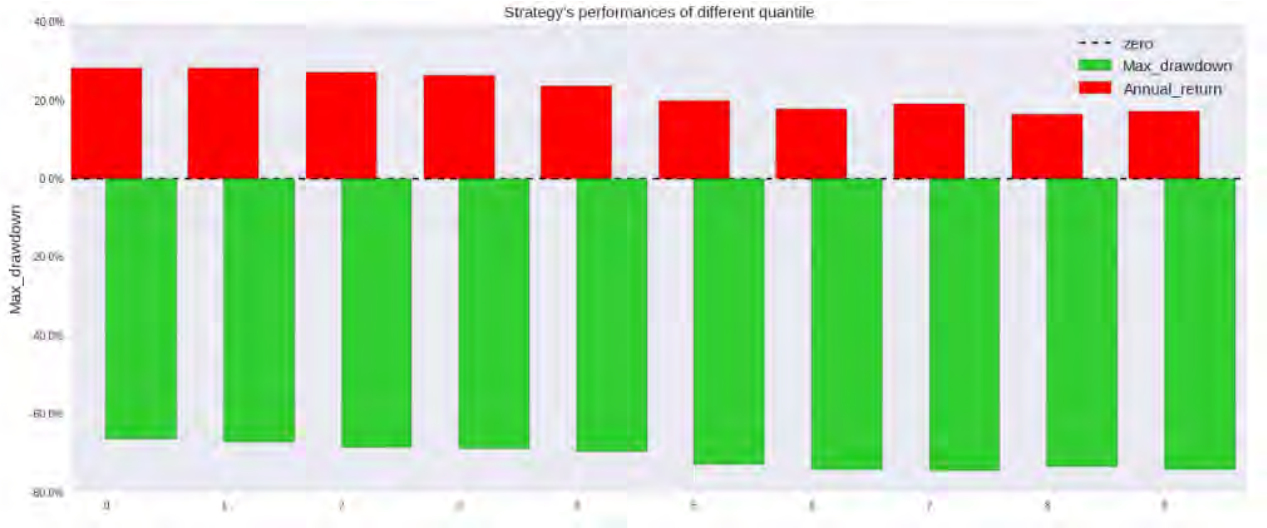
一些可视化功能

在获取数据之后，我们可以把一些指标或者曲线画出来，便于进行观察和分析。

首先是 parameter_analysis.get_eval4_bar(sort_by=[]) 函数，这个函数会以 bar 图的形式画出回测的收益率、最大回撤、夏普比率和波动率四个图表。每个图表上从左到右是每一个回测，默认是按照回测编号进行排列，但是也可以输入 sort_by 进行自定义排列。sort_by 是一个 list 的 strings，它是参数名称 param_names 的一个子集，意义为按照这些变量进行排序。举例来说，如果 sort_by=['abc']，那就是按照 'abc' 参数从小到大排列回测，然后划出四张图表；如果 sort_by=['abc', 'x_y_z']，那就是先按照 'abc' 进行排列，然后再按照 'x_y_z' 进行排列（以 'x_y_z' 排序后，组内按照 'abc' 排序），排好之后画图。下图是按照 BP 指标大小排列出来的， get_eval4_bar() 函数使用默认值既可。



接下来是 parameter_analysis.get_eval(sort_by=[]) 和 parameter_analysis.get_excess_eval(sort_by=[])。第一个函数会画出各个回测年化收益率和最大回撤，而第二个函数会画出超额年化收益率和超额最大回撤，sort_by 的功能和之前解释的一样。这两个函数的效果还是直接举例画出来最直接。下图是以 BP 单因子策略十分位结果的 paramter_analysis.get_eval() 得出的图。



最后是画出回测的收益曲线、超额收益曲线、对数收益曲线以及超额对数收益曲线的函数，分别是 parameter_analysis.plot_returns() 、plot_excess_returns()、plot_log_returns() 和 plot_log_excess_returns()。这些函数没有输入，直接用就行。图是以 BP 单因子策略十分位结果的 paramter_analysis.plot_returns() 得出的图。

深入研究举例

作为示例，这里我们以 BP 因子为例将策略分为 (0,10)，(10,20)，…，(90,100) 十个分位区间进行回测并分析。

在回测之前我们需要制定一个自定义基准。在三篇因子研究分析中，我们发现很多因子不论是最大 5% 还是最小 5% 的分位都很轻松地跑赢了上证指数。经过一些分析，可以发现原因在于上证指数的指数构成偏大盘股，并且按照市值进行加权，所以该指数的小市值暴露度很低；然而我们回测的策略是把所有分位内的股票进行等权分配，小市值暴露度比上证指数要大，所以从这点来看可比性不高。为了剔除市值影响造成的策略和基准之间的差异，我们构建一个“等权全指”作为自定义基准，就是每月初将资金等权分配于所有二十一个交易日没停牌的股票之间，的到回测如下，下图基准是上证指数。

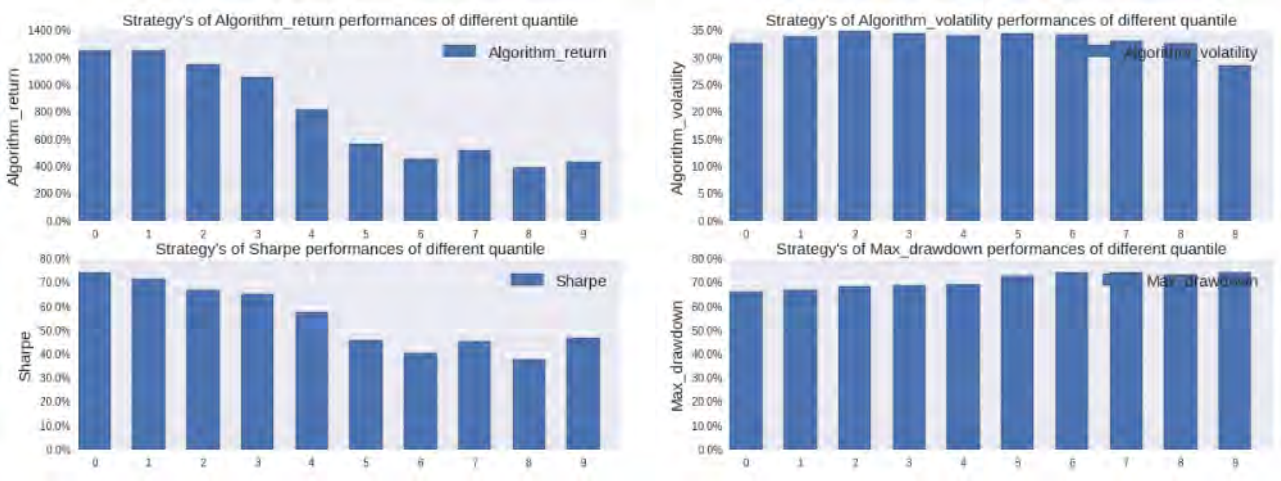
对应我们调用的指令如下，初始化 parameter_analysis 类并且启动【量化课堂】因子研究系列之一 -- 估值和资本结构因子中的 BP 因子回测，按 10 个百分点进行分位（共十个回测），并使用上述的等权全指作为基准：

```
# 初始化 parameter_analysis 类，设定回测策略 id
pa = parameter_analysis('bce2e5c55b3b631f91985c9bf113414f')[这个怎么表述？可以定义简化名称调用class？]
```

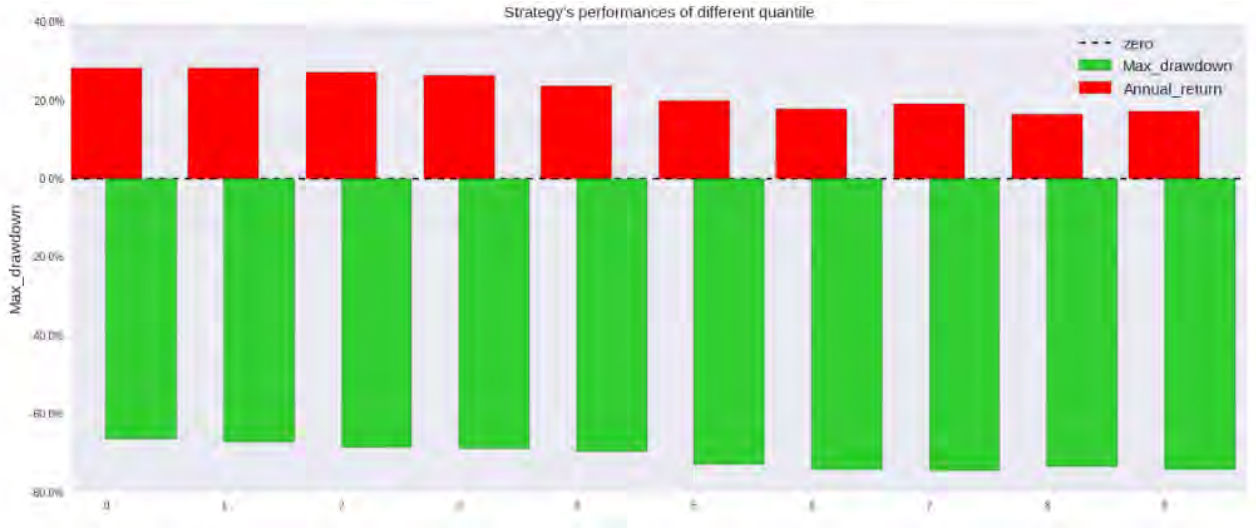
```
# 运行回测
pa.get_backtest_data(file_name = 'results.pkl',
                         running_max = 10,
                         benchmark_id = 'ae0684d86e9e7128b1ab9c7d77893029',
                         start_date = '2006-02-01',
                         end_date = '2016-11-01',
                         frequency = 'day',
                         initial_cash = '2000000',
                         param_names = ['factor', 'quantile'],
                         param_values = [['BP'], tuple(zip(range(0,100,10), range(10,101,10)))]
                         )
```

我们先进行回测，然后画出收益图。首先是收益曲线 parameter_analysis.plot_returns()，这里回测编号从 0 到 9 是 BP 值从小到大的回测。可以看出这个因子是具有单调性质的，也就是说 BP 值更大的股票表现一般比 BP 小的股票表现更好。但是，BP 指标的单调性并不是绝对的，比如 (10,20) 分位的股票的收益率是最高的 。

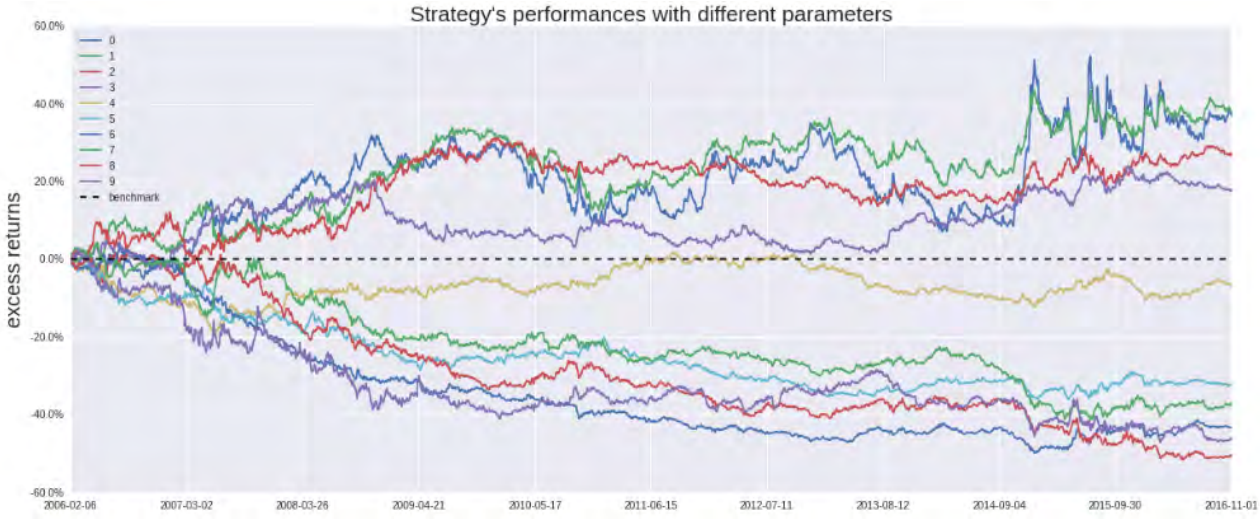
再用 parameter_analysis.get_eval4_bar() 画出策略的年化收益、最大回撤、夏普率和波动率，可以看出 BP 策略各分位表现出了收益不单调，对应的其他指标也并不单调。能够简单总结 BP 较小还是体现公司的股票回报率会好于 BP 较高的股票，伴随的是收益较高反而最大回撤较小。夏普率的变动与回报率较为一致，但是波动率变动在不同分位间差距不明显。



我们使用正的年化收益与负的最大回撤构建柱状图，parameter_analysis.get_eval():



加入新的基准后可以计算对于新基准回报率的超额收益率。进一步分析，我们使用 parameter_analysis.plot_excess_returns() 画出几个回测的超额收益曲线。这个图给我们一些有用的信息：BP 值最小的 10% 的股票的波动率极大，并且收益并没有超出基准太多，说明选这个区间的股票进行投资也许并不是很好。第二现象是，几个分位的回测在 06 年和 10 之间的超额收益有相互拉开，但在 10 年之后不同分位并没有明显的超额收益区别，说明这个因子在 10 年之后基本已经失效。



如【更新说明】新超额收益 和 对数轴的思路，我们也生成对数轴回报率，使用函数 parameter_analysis.plot_log_returns() 获取。如对数轴说明文章中介绍的，这样的图形在收益膨胀了现值后仍然可以明晰波动的相对大小。

结合超额收益与对数轴，可以通过 parameter_analysis.plot_log_excess_returns() 获得超额收益的对数轴图。如果是超额收益始终持续放大的过程，超额收益的对数轴图形将会是斜线上升的，然而下图中收益持平说明在相当长的一段时间中没有变动。

最后，可以通过 parameter_analysis.get_excess_eval() 获得超额回报和超额回报最大回撤的柱状图。可以看出，超额回报要明显小于策略原始回报，因为剔除了大盘整体的收益。同时，最大回撤也拉开了距离；按照策略净值算的最大回撤由于经历过同样的股灾，所以最大回撤都是百分之七十多，但在剔除大盘影响之后就能比较清晰地对比超额收益的回撤。整体来说，按 BP 选股的各个分为超额收益都不高，并且相对于基准的超额回撤都不小。

但这并不说明 BP 因子是一个无用的指标，只是说我们不能单单使用 BP 作为衡量标准并希望能得到超额收益，也许分行业分别计算 BP 或者结合其他的一些方法依然可以获得较好的收益。

小结

本文提供了一个使用研究模块调用回测结果进行调参并且研究分析的代码框架，其具有较好的兼容性和拓展性，细节之处还需要各位测试了解。在接下来的文章里，我们将使用次框架对因子系列文章 （【量化课堂】因子研究系列之一 -- 估值和资本结构因子、【量化课堂】因子研究系列之二 -- 成长因子和【量化课堂】因子研究系列之三 -- 技术因子）中的因子按分位排列进行回测并深入地分析这些因子的收益效果，敬请期待。

本文参考了多篇社区优秀帖子，这里一并感谢：
研究调用回测：【重磅更新】研究模块调用回测功能
回测排队运行：多个回测同时优化，改改列表和算法ID就行 by zhao
结果数据保存：JQ平台如何把DataFrame对象pickle或者json出来?

v1.1，2017-05-04，修改错别字，感谢 czdleaf 指出
v1.0，2017-01-03，文章上线

In [1]:

```
#1 先导入所需要的程序包
import datetime
import numpy as np
import pandas as pd
import time
from jqdata import *
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import copy
```

```
import pickle

# 定义类'参数分析'
class parameter_analysis(object):

    # 定义函数中不同的变量
    def __init__(self, algorithm_id=None):
        self.algorithm_id = algorithm_id            # 回测id

        self.params_df = pd.DataFrame()             # 回测中所有调参备选值的内容，列名字为对应修改面两名称，对应回测中的 g.XXXX
        self.results = {}                           # 回测结果的回报率，key 为 params_df 的行序号，value 为
        self.evaluations = {}                       # 回测结果的各项指标，key 为 params_df 的行序号，value 为一个 dataframe
        self.backtest_ids = {}                      # 回测结果的 id

        # 新加入的基准的回测结果 id，可以默认为空 ''，则使用回测中设定的基准
        self.benchmark_id = 'ae0684d86e9e7128b1ab9c7d77893029'

        self.benchmark_returns = []                 # 新加入的基准的回测回报率
        self.returns = {}                           # 记录所有回报率
        self.excess_returns = {}                    # 记录超额收益率
        self.log_returns = {}                       # 记录收益率的 log 值
        self.log_excess_returns = {}                # 记录超额收益的 log 值
        self.dates = []                             # 回测对应的所有日期
        self.excess_max_drawdown = {}               # 计算超额收益的最大回撤
        self.excess_annual_return = {}              # 计算超额收益率的年化指标
        self.evaluations_df = pd.DataFrame()        # 记录各项回测指标，除日回报率外

    # 定义排队运行多参数回测函数
    def run_backtest(self,                          #
                     algorithm_id=None,             # 回测策略id
                     running_max=10,                # 回测中同时巡行最大回测数量
                     start_date='2006-01-01',       # 回测的起始日期
                     end_date='2016-11-30',         # 回测的结束日期
                     frequency='day',               # 回测的运行频率
                     initial_cash='1000000',        # 回测的初始持仓金额
                     param_names=[],                # 回测中调整参数涉及的变量
                     param_values=[]                # 回测中每个变量的备选参数值
                     ):
        # 当此处回测策略的 id 没有给出时，调用类输入的策略 id
        if algorithm_id == None: algorithm_id=self.algorithm_id

        # 生成所有参数组合并加载到 df 中
        # 包含了不同参数具体备选值的排列组合中一组参数的 tuple 的 list
        param_combinations = list(itertools.product(*param_values))
        # 生成一个 dataframe， 对应的列为每个调参的变量，每个值为调参对应的备选值
        to_run_df = pd.DataFrame(param_combinations)
        # 修改列名称为调参变量的名字
        to_run_df.columns = param_names

        # 设定运行起始时间和保存格式
        start = time.time()
        # 记录结束的运行回测
        finished_backtests = {}
        # 记录运行中的回测
        running_backtests = {}
        # 计数器
        pointer = 0
        # 总运行回测数目，等于排列组合中的元素个数
        total_backtest_num = len(param_combinations)
        # 记录回测结果的回报率
        all_results = {}
        # 记录回测结果的各项指标
        all_evaluations = {}

        # 在运行开始时显示
        print '【运行中|已完成|待运行】:',
        # 当运行回测开始后，如果没有全部运行完全的话：
        while len(finished_backtests)<total_backtest_num:
            # 显示运行、完成和待运行的回测个数
            print('[%s|%s|%s].' % (len(finished_backtests),
                                   len(running_backtests),
                                   (total_backtest_num-len(finished_backtests)-len(running_backtests)) )),
            # 记录当前运行中的空位数量
            to_run = min(running_max-len(running_backtests), total_backtest_num-len(running_backtests)-len(finished_backtests))
            # 把可用的空位进行跑回测
            for i in range(pointer, pointer+to_run):
                # 备选的参数排列组合的 df 中第 i 行变成 dict，每个 key 为列名字，value 为 df 中对应的值
                params = to_run_df.ix[i].to_dict()
                # 记录策略回测结果的 id，调整参数 extras 使用 params 的内容
                backtest = create_backtest(algorithm_id = algorithm_id,
                                           start_date = start_date,
                                           end_date = end_date,
```

```
                                        frequency = frequency,
                                        initial_cash = initial_cash,
                                        extras = params,
                                        # 再回测中把改参数的结果起一个名字，包含了所有涉及的变量参数值
                                        name = str(params)
                                        )
                # 记录运行中 i 回测的回测 id
                running_backtests[i] = backtest
            # 计数器计数运行完的数量
            pointer = pointer+to_run

            # 获取回测结果
            failed = []
            finished = []
            # 对于运行中的回测，key 为 to_run_df 中所有排列组合中的序数
            for key in running_backtests.keys():
                # 研究调用回测的结果，running_backtests[key] 为运行中保存的结果 id
                bt = get_backtest(running_backtests[key])
                # 获得运行回测结果的状态，成功和失败都需要运行结束后返回，如果没有返回则运行没有结束
                status = bt.get_status()
                # 当运行回测失败
                if status == 'failed':
                    # 失败 list 中记录对应的回测结果 id
                    failed.append(key)
                # 当运行回测成功时
                elif status == 'done':
                    # 成功 list 记录对应的回测结果 id，finish 仅记录运行成功的
                    finished.append(key)
                    # 回测回报率记录对应回测的回报率 dict， key to_run_df 中所有排列组合中的序数， value 为回报率的 dict
                    # 每个 value 一个 list 每个对象为一个包含时间、日回报率和基准回报率的 dict
                    all_results[key] = bt.get_results()
                    # 回测回报率记录对应回测结果指标 dict， key to_run_df 中所有排列组合中的序数， value 为回测结果指标的 dataframe
                    all_evaluations[key] = bt.get_risk()
            # 记录运行中回测结果 id 的 list 中删除失败的运行
            for key in failed:
                running_backtests.pop(key)
            # 在结束回测结果 dict 中记录运行成功的回测结果 id，同时在运行中的记录中删除该回测
            for key in finished:
                finished_backtests[key] = running_backtests.pop(key)
            # 当一组同时运行的回测结束时报告时间
            if len(finished_backtests) != 0 and len(finished_backtests) % running_max == 0 and to_run !=0:
                # 记录当时时间
                middle = time.time()
                # 计算剩余时间，假设没工作量时间相等的话
                remain_time = (middle - start) * (total_backtest_num - len(finished_backtests)) / len(finished_backtests)
                # print 当前运行时间
                print('[已用%s时,尚余%s时,请不要关闭浏览器].' % (str(round((middle - start) / 60.0 / 60.0,3)),
                                            str(round(remain_time / 60.0 / 60.0,3)))),
            # 5秒钟后再跑一下
            time.sleep(5)
        # 记录结束时间
        end = time.time()
        print ''
        print('【回测完成】总用时：%s秒(即%s小时)。' % (str(int(end-start)),
                                            str(round((end-start)/60.0/60.0,2)))),
        # 对应修改类内部对应
        self.params_df = to_run_df
        self.results = all_results
        self.evaluations = all_evaluations
        self.backtest_ids = finished_backtests


#7 最大回撤计算方法
def find_max_drawdown(self, returns):
    # 定义最大回撤的变量
    result = 0
    # 记录最高的回报率点
    historical_return = 0
    # 遍历所有日期
    for i in range(len(returns)):
        # 最高回报率记录
        historical_return = max(historical_return, returns[i])
        # 最大回撤记录
        drawdown = 1-(returns[i] + 1) / (historical_return + 1)
        # 记录最大回撤
        result = max(drawdown, result)
    # 返回最大回撤值
    return result

# log 收益、新基准下超额收益和相对与新基准的最大回撤
def organize_backtest_results(self, benchmark_id=None):
    # 若新基准的回测结果 id 没给出
```

```
        if benchmark_id==None:
            # 使用默认的基准回报率，默认的基准在回测策略中设定
            self.benchmark_returns = [x['benchmark_returns'] for x in self.results[0]]
        # 当新基准指标给出后
        else:
            # 基准使用新加入的基准回测结果
            self.benchmark_returns = [x['returns'] for x in get_backtest(benchmark_id).get_results()]
        # 回测日期为结果中记录的第一项对应的日期
        self.dates = [x['time'] for x in self.results[0]]

        # 对应每个回测在所有备选回测中的顺序 （key），生成新数据
        # 由 {key: {u'benchmark_returns': 0.022480100091729405,
        #           u'returns': 0.03184566700000002,
        #           u'time': u'2006-02-14'}} 格式转化为：
        # {key: []} 格式，其中 list 为对应 date 的一个回报率 list
        for key in self.results.keys():
            self.returns[key] = [x['returns'] for x in self.results[key]]
        # 生成对于基准（或新基准）的超额收益率
        for key in self.results.keys():
            self.excess_returns[key] = [(x+1)/(y+1)-1 for (x,y) in zip(self.returns[key], self.benchmark_returns)]
        # 生成 log 形式的收益率
        for key in self.results.keys():
            self.log_returns[key] = [log(x+1) for x in self.returns[key]]
        # 生成超额收益率的 log 形式
        for key in self.results.keys():
            self.log_excess_returns[key] = [log(x+1) for x in self.excess_returns[key]]
        # 生成超额收益率的最大回撤
        for key in self.results.keys():
            self.excess_max_drawdown[key] = self.find_max_drawdown(self.excess_returns[key])
        # 生成年化超额收益率
        for key in self.results.keys():
            self.excess_annual_return[key] = (self.excess_returns[key][-1]+1)**(252./float(len(self.dates)))-1
        # 把调参数据中的参数组合 df 与对应结果的 df 进行合并
        self.evaluations_df = pd.concat([self.params_df, pd.DataFrame(self.evaluations).T], axis=1)
#         self.evaluations_df =

    # 获取最总分析数据，调用排队回测函数和数据整理的函数
    def get_backtest_data(self,
                          algorithm_id=None,                        # 回测策略id
                          benchmark_id=None,                        # 新基准回测结果id
                          file_name='results.pkl',                  # 保存结果的 pickle 文件名字
                          running_max=10,                           # 最大同时运行回测数量
                          start_date='2006-01-01',                  # 回测开始时间
                          end_date='2016-11-30',                    # 回测结束日期
                          frequency='day',                          # 回测的运行频率
                          initial_cash='1000000',                   # 回测初始持仓资金
                          param_names=[],                           # 回测需要测试的变量
                          param_values=[]                           # 对应每个变量的备选参数
                          ):
        # 调运排队回测函数，传递对应参数
        self.run_backtest(algorithm_id=algorithm_id,
                          running_max=running_max,
                          start_date=start_date,
                          end_date=end_date,
                          frequency=frequency,
                          initial_cash=initial_cash,
                          param_names=param_names,
                          param_values=param_values
                          )
        # 回测结果指标中加入 log 收益率和超额收益率等指标
        self.organize_backtest_results(benchmark_id)
        # 生成 dict 保存所有结果。
        results = {'returns':self.returns,
                   'excess_returns':self.excess_returns,
                   'log_returns':self.log_returns,
                   'log_excess_returns':self.log_excess_returns,
                   'dates':self.dates,
                   'benchmark_returns':self.benchmark_returns,
                   'evaluations':self.evaluations,
                   'params_df':self.params_df,
                   'backtest_ids':self.backtest_ids,
                   'excess_max_drawdown':self.excess_max_drawdown,
                   'excess_annual_return':self.excess_annual_return,
                   'evaluations_df':self.evaluations_df}
        # 保存 pickle 文件
        pickle_file = open(file_name, 'wb')
        pickle.dump(results, pickle_file)
        pickle_file.close()

    # 读取保存的 pickle 文件，赋予类中的对象名对应的保存内容
    def read_backtest_data(self, file_name='results.pkl'):
        pickle_file = open(file_name, 'rb')
```

```
        results = pickle.load(pickle_file)
        self.returns = results['returns']
        self.excess_returns = results['excess_returns']
        self.log_returns = results['log_returns']
        self.log_excess_returns = results['log_excess_returns']
        self.dates = results['dates']
        self.benchmark_returns = results['benchmark_returns']
        self.evaluations = results['evaluations']
        self.params_df = results['params_df']
        self.backtest_ids = results['backtest_ids']
        self.excess_max_drawdown = results['excess_max_drawdown']
        self.excess_annual_return = results['excess_annual_return']
        self.evaluations_df = results['evaluations_df']

    # 回报率折线图
    def plot_returns(self):
        # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；
        fig = plt.figure(figsize=(20,8))
        ax = fig.add_subplot(110)
        # 作图
        for key in self.returns.keys():
            ax.plot(range(len(self.returns[key])), self.returns[key], label=key)
        # 设定benchmark曲线并标记
        ax.plot(range(len(self.benchmark_returns)), self.benchmark_returns, label='benchmark', c='k', linestyle='--')
        ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]
        plt.xticks(ticks, [self.dates[i] for i in ticks])
        # 设置图例样式
        ax.legend(loc = 2, fontsize = 10)
        # 设置y标签样式
        ax.set_ylabel('returns',fontsize=20)
        # 设置x标签样式
        ax.set_yticklabels([str(x*100)+'% 'for x in ax.get_yticks()])
        # 设置图片标题样式
        ax.set_title("Strategy's performances with different parameters", fontsize=21)
        plt.xlim(0, len(self.returns[0]))

    # 超额收益率图
    def plot_excess_returns(self):
        # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；
        fig = plt.figure(figsize=(20,8))
        ax = fig.add_subplot(110)
        # 作图
        for key in self.returns.keys():
            ax.plot(range(len(self.excess_returns[key])), self.excess_returns[key], label=key)
        # 设定benchmark曲线并标记
        ax.plot(range(len(self.benchmark_returns)), [0]*len(self.benchmark_returns), label='benchmark', c='k', linestyle='--')
        ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]
        plt.xticks(ticks, [self.dates[i] for i in ticks])
        # 设置图例样式
        ax.legend(loc = 2, fontsize = 10)
        # 设置y标签样式
        ax.set_ylabel('excess returns',fontsize=20)
        # 设置x标签样式
        ax.set_yticklabels([str(x*100)+'% 'for x in ax.get_yticks()])
        # 设置图片标题样式
        ax.set_title("Strategy's performances with different parameters", fontsize=21)
        plt.xlim(0, len(self.excess_returns[0]))

    # log回报率图
    def plot_log_returns(self):
        # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；
        fig = plt.figure(figsize=(20,8))
        ax = fig.add_subplot(110)
        # 作图
        for key in self.returns.keys():
            ax.plot(range(len(self.log_returns[key])), self.log_returns[key], label=key)
        # 设定benchmark曲线并标记
        ax.plot(range(len(self.benchmark_returns)), [log(x+1) for x in self.benchmark_returns], label='benchmark', c='k', linestyle='
        ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]
        plt.xticks(ticks, [self.dates[i] for i in ticks])
        # 设置图例样式
        ax.legend(loc = 2, fontsize = 10)
        # 设置y标签样式
        ax.set_ylabel('log returns',fontsize=20)
        # 设置图片标题样式
        ax.set_title("Strategy's performances with different parameters", fontsize=21)
        plt.xlim(0, len(self.log_returns[0]))

    # 超额收益率的 log 图
    def plot_log_excess_returns(self):
        # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；
        fig = plt.figure(figsize=(20,8))
```

```
    ax = fig.add_subplot(110)
    # 作图
    for key in self.returns.keys():
        ax.plot(range(len(self.log_excess_returns[key])), self.log_excess_returns[key], label=key)
    # 设定benchmark曲线并标记
    ax.plot(range(len(self.benchmark_returns)), [0]*len(self.benchmark_returns), label='benchmark', c='k', linestyle='--')
    ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]
    plt.xticks(ticks, [self.dates[i] for i in ticks])
    # 设置图例样式
    ax.legend(loc = 2, fontsize = 10)
    # 设置y标签样式
    ax.set_ylabel('log excess returns',fontsize=20)
    # 设置图片标题样式
    ax.set_title("Strategy's performances with different parameters", fontsize=21)
    plt.xlim(0, len(self.log_excess_returns[0]))


# 回测的4个主要指标，包括总回报率、最大回撤夏普率和波动
def get_eval4_bar(self, sort_by=[]):

    sorted_params = self.params_df
    for by in sort_by:
        sorted_params = sorted_params.sort(by)
    indices = sorted_params.index

    fig = plt.figure(figsize=(20,7))

    # 定义位置
    ax1 = fig.add_subplot(221)
    # 设定横轴为对应分位，纵轴为对应指标
    ax1.bar(range(len(indices)),
            [self.evaluations[x]['algorithm_return'] for x in indices], 0.6, label = 'Algorithm_return')
    plt.xticks([x+0.3 for x in range(len(indices))], indices)
    # 设置图例样式
    ax1.legend(loc='best',fontsize=15)
    # 设置y标签样式
    ax1.set_ylabel('Algorithm_return', fontsize=15)
    # 设置y标签样式
    ax1.set_yticklabels([str(x*100)+'% 'for x in ax1.get_yticks()])
    # 设置图片标题样式
    ax1.set_title("Strategy's of Algorithm_return performances of different quantile", fontsize=15)
    # x轴范围
    plt.xlim(0, len(indices))

    # 定义位置
    ax2 = fig.add_subplot(224)
    # 设定横轴为对应分位，纵轴为对应指标
    ax2.bar(range(len(indices)),
            [self.evaluations[x]['max_drawdown'] for x in indices], 0.6, label = 'Max_drawdown')
    plt.xticks([x+0.3 for x in range(len(indices))], indices)
    # 设置图例样式
    ax2.legend(loc='best',fontsize=15)
    # 设置y标签样式
    ax2.set_ylabel('Max_drawdown', fontsize=15)
    # 设置x标签样式
    ax2.set_yticklabels([str(x*100)+'% 'for x in ax2.get_yticks()])
    # 设置图片标题样式
    ax2.set_title("Strategy's of Max_drawdown performances of different quantile", fontsize=15)
    # x轴范围
    plt.xlim(0, len(indices))

    # 定义位置
    ax3 = fig.add_subplot(223)
    # 设定横轴为对应分位，纵轴为对应指标
    ax3.bar(range(len(indices)),
            [self.evaluations[x]['sharpe'] for x in indices], 0.6, label = 'Sharpe')
    plt.xticks([x+0.3 for x in range(len(indices))], indices)
    # 设置图例样式
    ax3.legend(loc='best',fontsize=15)
    # 设置y标签样式
    ax3.set_ylabel('Sharpe', fontsize=15)
    # 设置x标签样式
    ax3.set_yticklabels([str(x*100)+'% 'for x in ax3.get_yticks()])
    # 设置图片标题样式
    ax3.set_title("Strategy's of Sharpe performances of different quantile", fontsize=15)
    # x轴范围
    plt.xlim(0, len(indices))

    # 定义位置
    ax4 = fig.add_subplot(222)
    # 设定横轴为对应分位，纵轴为对应指标
    ax4.bar(range(len(indices)),
```

```
            [self.evaluations[x]['algorithm_volatility'] for x in indices], 0.6, label = 'Algorithm_volatility')
    plt.xticks([x+0.3 for x in range(len(indices))], indices)
    # 设置图例样式
    ax4.legend(loc='best',fontsize=15)
    # 设置y标签样式
    ax4.set_ylabel('Algorithm_volatility', fontsize=15)
    # 设置x标签样式
    ax4.set_yticklabels([str(x*100)+'% 'for x in ax4.get_yticks()])
    # 设置图片标题样式
    ax4.set_title("Strategy's of Algorithm_volatility performances of different quantile", fontsize=15)
    # x轴范围
    plt.xlim(0, len(indices))

#14 年化回报和最大回撤，正负双色表示
def get_eval(self, sort_by=[]):

    sorted_params = self.params_df
    for by in sort_by:
        sorted_params = sorted_params.sort(by)
    indices = sorted_params.index

    # 大小
    fig = plt.figure(figsize = (20, 8))
    # 图1位置
    ax = fig.add_subplot(110)
    # 生成图超额收益率的最大回撤
    ax.bar([x+0.3 for x in range(len(indices))],
           [-self.evaluations[x]['max_drawdown'] for x in indices], color = '#32CD32',
                 width = 0.6, label = 'Max_drawdown', zorder=10)
    # 图年化超额收益
    ax.bar([x for x in range(len(indices))],
           [self.evaluations[x]['annual_algo_return'] for x in indices], color = 'r',
                 width = 0.6, label = 'Annual_return')
    plt.xticks([x+0.3 for x in range(len(indices))], indices)
    # 设置图例样式
    ax.legend(loc='best',fontsize=15)
    # 基准线
    plt.plot([0, len(indices)], [0, 0], c='k',
             linestyle='--', label='zero')
    # 设置图例样式
    ax.legend(loc='best',fontsize=15)
    # 设置y标签样式
    ax.set_ylabel('Max_drawdown', fontsize=15)
    # 设置x标签样式
    ax.set_yticklabels([str(x*100)+'% 'for x in ax.get_yticks()])
    # 设置图片标题样式
    ax.set_title("Strategy's performances of different quantile", fontsize=15)
    #   设定x轴长度
    plt.xlim(0, len(indices))


#14 超额收益的年化回报和最大回撤
# 加入新的benchmark后超额收益和
def get_excess_eval(self, sort_by=[]):

    sorted_params = self.params_df
    for by in sort_by:
        sorted_params = sorted_params.sort(by)
    indices = sorted_params.index

    # 大小
    fig = plt.figure(figsize = (20, 8))
    # 图1位置
    ax = fig.add_subplot(110)
    # 生成图超额收益率的最大回撤
    ax.bar([x+0.3 for x in range(len(indices))],
           [-self.excess_max_drawdown[x] for x in indices], color = '#32CD32',
                 width = 0.6, label = 'Excess_max_drawdown')
    # 图年化超额收益
    ax.bar([x for x in range(len(indices))],
           [self.excess_annual_return[x] for x in indices], color = 'r',
                 width = 0.6, label = 'Excess_annual_return')
    plt.xticks([x+0.3 for x in range(len(indices))], indices)
    # 设置图例样式
    ax.legend(loc='best',fontsize=15)
    # 基准线
    plt.plot([0, len(indices)], [0, 0], c='k',
             linestyle='--', label='zero')
    # 设置图例样式
    ax.legend(loc='best',fontsize=15)
    # 设置y标签样式
    ax.set_ylabel('Max_drawdown', fontsize=15)
```

```
        # 设置x标签样式
        ax.set_yticklabels([str(x*100)+'% 'for x in ax.get_yticks()])
        # 设置图片标题样式
        ax.set_title("Strategy's performances of different quantile", fontsize=15)
        #   设定x轴长度
        plt.xlim(0, len(indices))
```

In [2]:

```
#2 设定回测策略 id
pa = parameter_analysis('bce2e5c55b3b631f91985c9bf113414f')
```

In [3]:

```
#3 运行回测
pa.get_backtest_data(file_name = 'results.pkl',
                           running_max = 10,
                           benchmark_id = 'ae0684d86e9e7128b1ab9c7d77893029',
                           start_date = '2006-02-01',
                           end_date = '2016-11-01',
                           frequency = 'day',
                           initial_cash = '2000000',
                           param_names = ['factor', 'quantile'],
                           param_values = [['BP'], tuple(zip(range(0,100,10), range(10,101,10)))]
                           )
```

```
【运行中|已完成|待运行】: [0|0|10]. [0|10|0]. [0|10|0]. [0|10|0]. [0|10|0]. [0|10|0]. [0|10|0]. [0|10|0]. [0|10|0]. [0|10|0].
【回测完成】总用时：559秒(即0.16小时)。
```

In [137]:

```
#4 数据读取
pa.read_backtest_data('results.pkl')
```

In [95]:

```
#5 回测参数的 Dataframe
pa.params_df
```

Out[95]:

	factor	quantile
0	BP	(0, 10)
1	BP	(10, 20)
2	BP	(20, 30)
3	BP	(30, 40)
4	BP	(40, 50)
5	BP	(50, 60)
6	BP	(60, 70)
7	BP	(70, 80)
8	BP	(80, 90)
9	BP	(90, 100)

In [4]:

```
#6 查看回测结果指标
pa.evaluations_df
```

Out[4]:

	factor	quantile	algorithm_return	algorithm_volatility	alpha	annual_algo_return	annual_bm_return	
0	BP	(0, 10)	12.48958	0.3259261	0.1894443	0.2825425	0.09083216	
1	BP	(10, 20)	12.54216	0.3388973	0.188028	0.2830198	0.09083216	

	factor	quantile	algorithm_return	algorithm_volatility	alpha	annual_algo_return	annual_bm_return
2	BP	(20, 30)	11.48453	0.3480029	0.1768537	0.2730803	0.09083216
3	BP	(30, 40)	10.57775	0.3441174	0.168524	0.2639323	0.09083216
4	BP	(40, 50)	8.154993	0.3412167	0.1413735	0.2358674	0.09083216
5	BP	(50, 60)	5.654552	0.3444933	0.1041304	0.1987321	0.09083216
6	BP	(60, 70)	4.575096	0.341814	0.08477608	0.1786116	0.09083216
7	BP	(70, 80)	5.207267	0.3307248	0.0987676	0.1907816	0.09083216
8	BP	(80, 90)	3.888208	0.326185	0.07321037	0.1638834	0.09083216
9	BP	(90, 100)	4.311805	0.2854319	0.09056309	0.173171	0.09083216

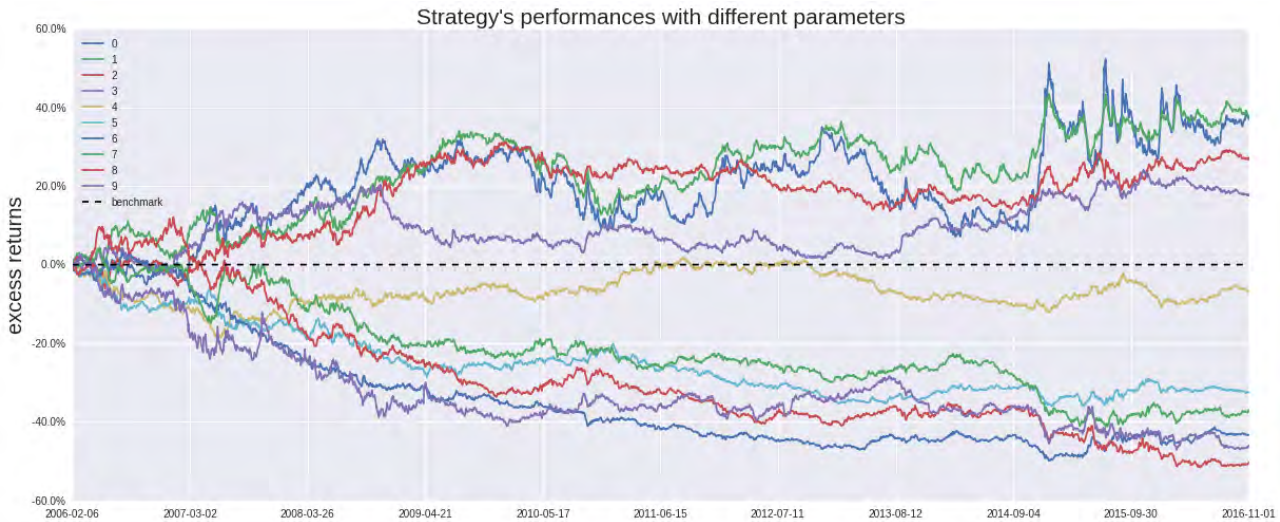
In [5]:

```
#7 回报率折线图
pa.plot_returns()
```

```
/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/axes/_subplots.py:69: MatplotlibDeprecationWarning: The use of 0 (whic
  mplDeprecation)
```

In [6]:

```
#8 超额收益率图
pa.plot_excess_returns()
```



In [7]:

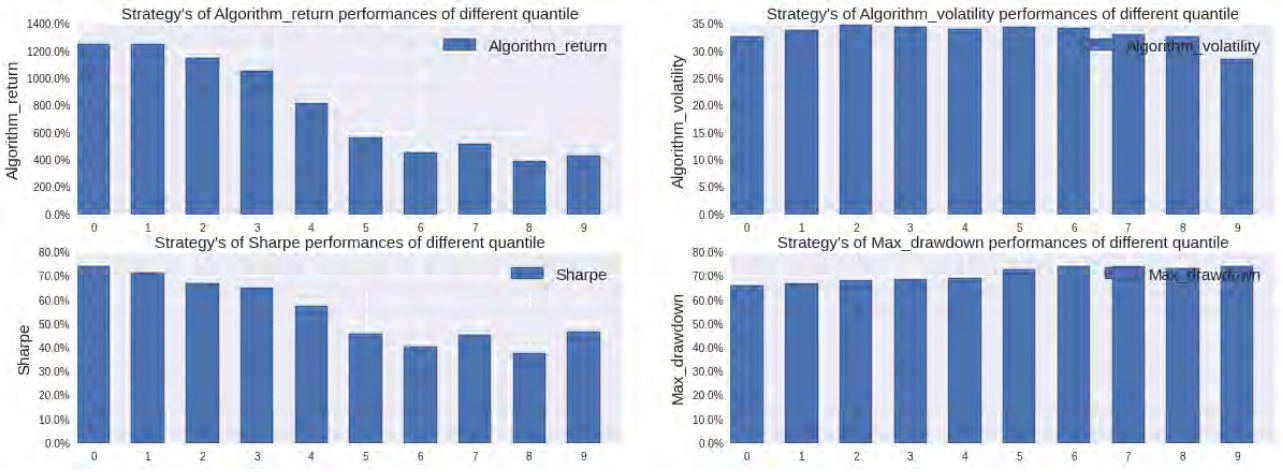
```
#9 log回报率图
pa.plot_log_returns()
```

In [8]:

```
#10 超额收益率的 log 图
pa.plot_log_excess_returns()
```

In [9]:

```
#11 回测的4个主要指标，包括总回报率、最大回撤夏普率和波动
# get_eval4_bar(self, sort_by=[])
pa.get_eval4_bar()
```



In [10]:

```
#12 年化回报和最大回撤，正负双色显示
# get_eval(self, sort_by=[])
pa.get_eval()
```

In [11]:

```
#13 超额收益的年化回报和最大回撤
# 加入新的benchmark后超额收益和
# get_excess_eval(self, sort_by=[])
pa.get_excess_eval()
```

In [166]:

```
# test 测试最后bar图中的sort_by对应内容
param_names=['abc','x_y_z']
param_values=[['a','b','c'], [1,2]]
param_combinations = list(itertools.product(*param_values))
to_run_df = pd.DataFrame(param_combinations)
to_run_df.columns = param_names
# to_run_df.ix[1].to_dict()
to_run_df

# sort_by = ['abc']
sort_by = ['abc', 'x_y_z']
# sort_by = ['x_y_z']
# sort_by = ['x_y_z','abc']
sorted_params = to_run_df
for by in sort_by:
    sorted_params = sorted_params.sort(by)
indices = sorted_params.index
sorted_params
```

Out[166]:

	abc	x_y_z
0	a	1
2	b	1
4	c	1
1	a	2

	abc	x_y_z
3	b	2
5	c	2

In [ ]:

```
```

一键即得标准化、规范化、二值化等多种机器学习数据预处理方式

预处理数据

数据预处理在众多深度学习算法中都起着重要作用，实际上，对数据进行适当处理后，很多算法能够发挥最佳效果。然而面对各种各样的数据，很多时候我们不知道怎么样才能针对性进行处理。本文介绍了Python 下的机器学习工具scikit-learn 。其中，“sklearn.preprocessing ”模块提供了几种常见的函数和转换类，把原始的特征向量变得更适合估计器使用。

1. 标准化，即减去平均值再用方差缩放比例¶

在scikit-learn中，数据的标准化是很多机器学习估计器的常见要求；如果单个特征看起来不符合标准正态分布（平均值为0，方差为1）的话，数据之后可能会有很差的表现。

实际上我们通常忽略分布的具体形态，数据转换仅指，减去每个特征的平均值，再除以他们的标准差。

例如，学习算法中目标函数的很多成分都假设，所有的特征都是围绕着0的，并且拥有相同算数级别的方差（比如SVM中的RBF核，以及线性模型中的l1,l2正则化）。如果一个特征的方差级别高于其他的特征，它会在目标函数中占据主导地位，并使得估计器不能按照预期很好地从其他特征中学习。

scale函数就提供了一个快速且简便的方法，对一个数组型数据集执行这个操作：

In [1]:

```
from sklearn import preprocessing
import numpy as np
X = np.array([[ 1., -1.,  2.],
              [ 2.,  0.,  0.],
              [ 0.,  1., -1.]])
X_scaled = preprocessing.scale(X)

X_scaled
```

Out[1]:

```
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

调整后的数据平均值为0，方差为1：

In [2]:

```
X_scaled.mean(axis=0)
```

Out[2]:

```
array([ 0.,  0.,  0.])
```

In [3]:

```
X_scaled.std(axis=0)
```

Out[3]:

```
array([ 1.,  1.,  1.])
```

preprocessing模块还提供了一个类"StandardScaler"，它能计算训练集的平均值和标准差，以便之后对测试集进行相同的转换。因此，这个类适合用于sklearn.pipeline.Pipeline的前几个步骤：

In [4]:

```
scaler = preprocessing.StandardScaler().fit(X)
scaler
```

Out[4]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [5]:

```
scaler.mean_
```

Out[5]:

```
array([ 1.        ,  0.        ,  0.33333333])
```

In [6]:

```
scaler.scale_
```

Out[6]:

```
array([ 0.81649658,  0.81649658,  1.24721913])
```

In [8]:

```
scaler.transform(X)
```

Out[8]:

```
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

这个scaler之后能对新的数据进行，跟先前对训练集一样的操作：

In [9]:

```
scaler.transform([[-1.,  1., 0.]])
```

Out[9]:

```
array([[-2.44948974,  1.22474487, -0.26726124]])
```

此外，也可以通过在创建StandardScaler时增加with_mean=False或者with_std=False语句，来阻止集中化或缩放比例。

1.1 把特征缩放到一个范围内¶

另一个标准化的操作，是把特征缩放到一个最小值与最大值之间（通常是0到1），或者是把每个特征的最大绝对值变到1。这分别可以通过MinMaxScaler或者MaxAbsScaler实现。

使用这种转换方式是为了增加强健性，来解决特征的标准差非常小的问题，以及在稀疏数据中保留0元素。

以下是一个把数据矩阵缩放到[0,1]范围内的一个例子：

In [7]:

```
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_train_minmax
```

Out[7]:

```
array([[ 0.5       ,  0.        ,  1.        ],
       [ 1.        ,  0.5       ,  0.33333333],
       [ 0.        ,  1.        ,  0.        ]])
```

相同的转换器可以用到新的测试集上：相同的缩放、平移操作会与之前对训练数据的操作保持一致：

In [8]:

```
X_test = np.array([[ -3., -1.,  4.]])
X_test_minmax = min_max_scaler.transform(X_test)
X_test_minmax
```

Out[8]:

```
array([[-1.5       ,  0.        ,  1.66666667]])
```

我们也可以找出从训练数据中学到的转换的具体特性：

In [9]:

```
min_max_scaler.scale_
```

Out[9]:

```
array([ 0.5       ,  0.5       ,  0.33333333])
```

In [10]:

```
min_max_scaler.min_
```

Out[10]:

```
array([ 0.        ,  0.5       ,  0.33333333])
```

如果MinMaxScaler被给予一个明确的feature_range=(min,max)，完整的公式是：

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std / (max - min) + min

MaxAbsScaler的功能很类似，但是它把训练数据缩放到了[-1,1]范围内。这对已经围绕着0的数据或者稀疏数据来说是很有意义的。

这里用了这个scaler把之前例子的数据进行了转换：

In [11]:

```
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
max_abs_scaler = preprocessing.MaxAbsScaler()
X_train_maxabs = max_abs_scaler.fit_transform(X_train)
X_train_maxabs                # doctest +NORMALIZE_WHITESPACE^
```

Out[11]:

```
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])
```

In [12]:

```
X_test = np.array([[ -3., -1.,  4.]])
X_test_maxabs = max_abs_scaler.transform(X_test)
X_test_maxabs
```

Out[12]:

```
array([[-1.5, -1. ,  2. ]])
```

In [13]:

```
max_abs_scaler.scale_
```

Out[13]:

```
array([ 2.,  1.,  2.])
```

与scale一样，这个模块也提供了比较简便的函数minmax_scale以及maxabs_scale，如果你不想创建一个对象。

1.2 转换稀疏数据¶

把稀疏数据集中化会破坏数据中的稀疏性结构，因此不是一个理想的做法。但是，对稀疏的输入转换测度是有道理的，特别是当特征具有不同的测度的时候。

MaxAbsScaler以及maxabs_scale是特别为转换稀疏数据设计的，并且我们建议使用他们。然而，scale和StandardScaler可以接受scipy.sparse矩阵作为输入，只要在创建时说明with_mean=False。否则会产生ValueError，因为默认的集中化会破坏稀疏性，并且会分配过多的内存从而导致运行崩溃。RobustScaler不能用于稀疏输入，但你可以对稀疏输入使用transform方法。

注意，scalers同时接受Compressed Sparse Rows以及Compressed Sparse Columns形式。其他类型的稀疏输入会被转换为Compressed Sparse Rows 的形式。为了避免不必要的内存复制，建议选择CSR或者CSC的表达形式。

最后，如果集中化后的数据预期非常小，使用toarray方法把稀疏输入转换为数组是另一个选择。

1.3 转换具有异常值的数据¶

如果你的数据有很多异常值，使用平均值和方差来进行转换可能表现不会很好。在这些情况下，你可以使用robust_scale以及RobustScaler。他们对数据的中心和范围采用更健壮的估计。

参考资料：¶

关于集中化和缩放比例重要性更多的讨论：Should I normalize/standardize/resclae the data?.

1.4 集中化核矩阵¶

如果你有一个核矩阵来计算特征空间的点积，KernelCenterer可以转换核矩阵，使其包含特征空间的内积，移去空间中的平均值。

2. 规范化¶

规范化是指，对单个样本缩放比例，使其范数为1。如果你计划使用平方的形式，例如点积或者其他核来量化样本之间的相似性，这个过程是有用的。

这个假设是Vector Space Model的基础，而Vector Space Model经常用在文本分类和集群环境下。

normalize函数提供了一个快速且简便的方法，对单个数组型数据集实现这个操作，使用了了l1或者l2范数：

In [14]:

```
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
X_normalized = preprocessing.normalize(X, norm='l2')
X_normalized
```

Out[14]:

```
array([[ 0.40824829, -0.40824829,  0.81649658],
       [ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.70710678, -0.70710678]])
```

Preprocessing模块还提供了一个类Normalizer来执行相同的操作，它使用了Transformer API（即使fit方法在这个情况下是没有用的：这个类没有状态，因为这个操作对样本独立进行处理）。

因此，这个类适用于sklearn.pipeline.Pipeline的前几个步骤：

In [15]:

```
normalizer = preprocessing.Normalizer().fit(X)  # fit does nothing
normalizer
```

Out[15]:

```
Normalizer(copy=True, norm='l2')
```

这个normalizer和其他transformer一样，可以用到样本向量中。

In [16]:

```
normalizer.transform(X)
```

Out[16]:

```
array([[ 0.40824829, -0.40824829,  0.81649658],
       [ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.70710678, -0.70710678]])
```

In [17]:

```
normalizer.transform([[-1.,  1., 0.]])
```

Out[17]:

```
array([[-0.70710678,  0.70710678,  0.        ]])
```

稀疏输入¶

normalize和normalizer同时接受scipy.sparse中密集的数组型，以及稀疏的矩阵作为输入。

对于稀疏输入，数据会被转换为Compressed Sparse Rows表示形式。为了避免不必要的内存复制，建议使用CSR表示形式。

3. 二值化¶

3.1 特征二值化¶

特征二值化是指，通过设置阈值，把数值的特征转换为布尔值。这对于下游概率的估计器来说是很有用的，它假设输入的数据按照多元Bernoulli分布。例如，sklearn.neural_network.BernoulliRBM。

在加工文本过程中使用布尔型特征值是很常见的（也许是为了简化概率推理），即使规范化计数，或者TF-IDF评价的特征经常在实际中表现的更好。

至于Normalizer，Binarizer类在sklearn.pipeline.Pipeline的前几个步骤会被用到。fit方法没有什么用，因为每个样本都被独立处理：

In [18]:

```
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
binarizer = preprocessing.Binarizer().fit(X)  # fit does nothing
binarizer
```

Out[18]:

```
Binarizer(copy=True, threshold=0.0)
```

In [19]:

```
binarizer.transform(X)
```

Out[19]:

```
array([[ 1.,  0.,  1.],
       [ 1.,  0.,  0.],
       [ 0.,  1.,  0.]])
```

binarizer的阈值可以进行调整：

In [20]:

```
binarizer = preprocessing.Binarizer(threshold=1.1)
binarizer.transform(X)
```

Out[20]:

```
array([[ 0.,  0.,  1.],
       [ 1.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

至于StandardScaler以及Normalizer类，预处理模块提供了一个伴随函数binarize，它在transformer API不必要时会被使用。

稀疏输入¶

binarize和Binarizer同时接受scipy.sparse中密集的数组型，以及稀疏的矩阵作为输入。

对于稀疏输入，数据会被转换为Compressed Sparse Rows表示形式。为了避免不必要的内存复制，建议使用CSR表示形式。

4. 编码类型特征¶

经常，特征不是通过连续值表达的，而是通过类型。例如，一个人可以拥有特征["男人", "女人"], ["来自欧洲", "来自美国", "来自亚洲"], ["使用 Firefox", "使用 Chrome", "使用 Safari", "使用 Internet Explorer"]. 这些特征可以被有效编码为整数，例如，["男人", "来自美国", "使用 Internet Explorer"]可以表示为[0, 1, 3]，["女人", "来自亚国", "使用 Chrome"]可以表示为[1, 2, 1]。

这样的整数表达方式不能直接被用于scikit-learn估计器中，因为估计器会把它们当做连续的输入，把这些类别解释为有顺序的，而这并不是我们所期望的。

一种把类型特征转换为可以被scikit-learn估计器使用的方式是，使用一种one-of-K，或one-hot编码，它们在OneHotEncoder中被用到。这个估计器把每个，有m个可能值的类型特征转换为m个布尔值，只有一个值为1.

继续之前的例子：

In [24]:

```
enc = preprocessing.OneHotEncoder()
enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
```

Out[24]:

```
OneHotEncoder(categorical_features='all', dtype=<type 'numpy.float64'>,
       handle_unknown='error', n_values='auto', sparse=True)
```

In [25]:

```
enc.transform([[0, 1, 3]]).toarray()
```

Out[25]:

```
array([[ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.]])
```

默认地，每个特征能取多少值会自动地从数据集中推测出来。也可以通过n_values明确的说明这个参数。在这个例子中，我们的数据集有两个性别，三个州，以及四个网页浏览器。之后，我们用估计器来学习，转换每个数据点。结果是，前两个数字是性别的编码，之后三个数字是州的编码，最后四个数字是网页浏览器的编码。

注意，如果训练数据有可能丢失了一些类型特征，我们必须明确设置n_values。例如，

In [22]:

```
enc = preprocessing.OneHotEncoder(n_values=[2, 3, 4])
# 注意第二个和第三个类型值有缺失值
# 特征
enc.fit([[1, 2, 3], [0, 2, 0]])
```

Out[22]:

```
OneHotEncoder(categorical_features='all', dtype=<type 'numpy.float64'>,
       handle_unknown='error', n_values=[2, 3, 4], sparse=True)
```

In [23]:

```
enc.transform([[1, 0, 0]]).toarray()
```

Out[23]:

```
array([[ 0.,  1.,  1.,  0.,  0.,  1.,  0.,  0.,  0.]])
```

用字典表示，而不是用整数表示的类型特征，可以参照Loading features from dicts.

5. 缺失值的处理¶

由于各种各样的原因，很多现实生活中的数据集都会有缺失值，经常会被编码为空白，NaNs或其他占位符。然而，这些数据集与scikit-learn估计器并不兼容，因为估计器假设数组中的所有值都是数值型的，并且都有意义。处理不兼容的数据集一个基本的方法是，丢弃包含缺失值的整行或整列。但是，一些有价值的数据可能会因此丢失，尽管它们不兼容。一个更好的办法是，对缺失值进行其他处理，从数据的已有部分来推测缺失值。

Imputer类提供了处理缺失值的一些基本方法，它把缺失值用它所在行或列的平均值，中位数或众数来代替。这个类同时也允许不同的缺失值编码方式。

下面这个例子把缺失值（编码为np.nan)用它所在列的平均值代替：

In [34]:

```
import numpy as np
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit([[1, 2], [np.nan, 3], [7, 6]])
Imputer(axis=0, copy=True, missing_values='NaN', strategy='mean', verbose=0)
X = [[np.nan, 2], [6, np.nan], [7, 6]]
print(imp.transform(X))
```

```
[[ 4.          2.        ]
 [ 6.          3.66666667]
 [ 7.          6.        ]]
```

Imputer也支持稀疏矩阵：

In [35]:

```
import scipy.sparse as sp
X = sp.csc_matrix([[1, 2], [0, 3], [7, 6]])
imp = Imputer(missing_values=0, strategy='mean', axis=0)
imp.fit(X)
```

Out[35]:

```
Imputer(axis=0, copy=True, missing_values=0, strategy='mean', verbose=0)
```

In [37]:

```
X_test = sp.csc_matrix([[0, 2], [6, 0], [7, 6]])
print(imp.transform(X_test))
```

```
[[ 4.          2.        ]
 [ 6.          3.66666667]
 [ 7.          6.        ]]
```

注意，在这里，缺失值被编码为0，所以被隐性地储存在了矩阵中。当缺失值个数比观察值多很多的时候，这个形式适用。

Imputer能被用到Pipeline中，用来建立一个支持处理缺失值的复合估计器。参考Imputing missing values before builiding an estimator.

6. 生成多项式特征¶

经常，通过考虑输入数据的非线性特征来增加模型的复杂性是有用的。一个简单且常用的方法是使用多项式特征，它能得到特征的高次项以及交互项，通过PolynomialFeatures实现：

In [38]:

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(6).reshape(3, 2)
X
```

Out[38]:

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

In [39]:

```
poly = PolynomialFeatures(2)
poly.fit_transform(X)
```

Out[39]:

```
array([[  1.,   0.,   1.,   0.,   0.,   1.],
       [  1.,   2.,   3.,   4.,   6.,   9.],
       [  1.,   4.,   5.,  16.,  20.,  25.]])
```

在这里，X的特征从（X1,X2)被转换为了(1, X_1, X_2, X_1^2, X_1X_2, X_2^2).

有时候，只有特征之间的交互项是被要求的，它可以通过设置interaction_only=True来实现：

In [40]:

```
X = np.arange(9).reshape(3, 3)
X
```

Out[40]:

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [41]:

```
poly = PolynomialFeatures(degree=3, interaction_only=True)
poly.fit_transform(X)
```

Out[41]:

```
array([[   1.,    0.,    1.,    2.,    0.,    0.,    2.,    0.],
       [   1.,    3.,    4.,    5.,   12.,   15.,   20.,   60.],
       [   1.,    6.,    7.,    8.,   42.,   48.,   56.,  336.]])
```

在这里，X的特征从(X_1, X_2, X_3)被转换为了(1, X_1, X_2, X_3, X_1X_2, X_1X_3, X_2X_3, X_1X_2X_3).

注意，在核方法（如sklearn.svm.SVC,sklearn.decomposition.KernelPCA)中使用多项式核函数时，多项式特征被隐性使用。

7. 定制转换器¶

经常，你想要把一个已经存在的Python函数转换为一个转换器，用来协助数据清理或处理。你可以通过FunctionTransformer来实现。例如，为了生成一个可以进行log变化的转换器，可以：

In [42]:

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer
transformer = FunctionTransformer(np.log1p)
X = np.array([[0, 1], [2, 3]])
transformer.transform(X)
```

Out[42]:

```
array([[ 0.        ,  0.69314718],
       [ 1.09861229,  1.38629436]])
```

使用FunctionTransformer来定制特征选择的完整示例代码，可见Using FunctionTransformer to select columns.

pandas.dataframe 专题使用指南

取用数据的时候有时候会取用dataframe的格式，很多人都不熟，本篇重点整理了相关的的内容，以便大家查阅和学习。

蓝字是文章链接，大家都知道吧

选取数据

- 选取行名、列名、值
- 以标签（行、列的名字）为索引选择数据—— x.loc[行标签,列标签]
- 以位置（第几行、第几列）为索引选择数据—— x.iloc[行位置,列位置]
- 同时根据标签和位置选择数据——x.ix[行,列]
- 选择连续的多行多列——切片
- 选择不连续的某几行或某几列
- 简便地获取行或列
- 如何返回一个dataframe的单列或单行
- 按条件选取数据——df[逻辑条件]

转置、排序

- 转置 df.T
- 按行名或列名排序——df.sort_index
- 按值排序——df.sort_index

增删行或列

- 增加一列
- 增加一行
- 删除行或列——df.drop

连接多个dataframe

- 横向连接
- 纵向连接
- 按索引链接

组建dataframe

- 组建方法——pd.DataFrame
- 用字典型数据组建——pd.DataFrame
- 简便地获得聚宽数据中的时间索引

缺失值处理

- 去掉缺失值——df.dropna
- 对缺失值进行填充——df.fillna
- 判断数据是否为缺失——df.isnull

常用统计函数

- 常用统计函数
  - describe 针对Series或个DataFrame列计算汇总统计
  - count 非na值的数量
  - min、max 计算最小值和最大值
  - idxmin、idxmax 计算能够获取到最大值和最小值得索引值
  - quantile 计算样本的分位数（0到1）
  - sum 值的总和
  - mean 值得平均数
  - median 值得算术中位数（50%分位数）
  - mad 根据平均值计算平均绝对离差
  - var 样本值的方差
  - std 样本值的标准差
  - skew 样本值得偏度（三阶矩）
  - kurt 样本值得峰度（四阶矩）
  - cumsum 样本值得累计和
  - cummin，cummax 样本值得累计最大值和累计最小值
  - cumprod 样本值得累计积

    - diff 计算一阶差分
    - pct_change 计算百分数变化
- 查看函数的详细信息
- 更多的函数

panel 类型数据分解成dataframe

- panel类型数据分解成dataframe方法
- 更多panel操作指路

研究内存取dataframe

- 把dataframe存成csv文件——df.to_csv()
- 读取被存成csv文件的dataframe——pd.read_csv()

欢迎反馈建议：）

【重磅更新】研究模块调用回测功能

为了进一步打通研究模块与回测模块，我们增加了在研究中调用回测的新功能。

主要功能如下：

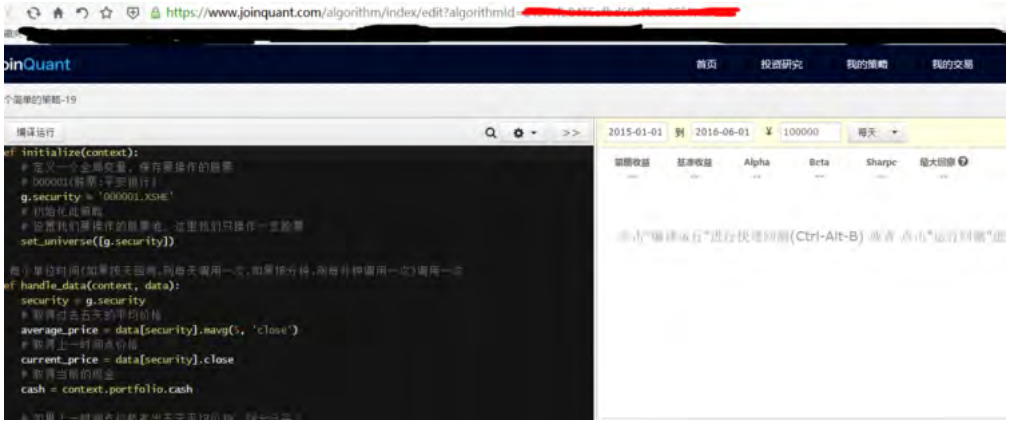
1. 可以在研究中创建回测
2. 设定初始持仓
3. 设定全局变量值
4. 获取回测状态
5. 获得回测参数
6. 获得收益曲线
7. 获得持仓详情
8. 获得交易详情
9. 获得所有 record 记录
10. 获得总的风险指标
11. 获得分月计算的风险指标

具体功能参看API - 研究中创建回测函数

具体用法见下面的研究内容。

研究模块调用回测功能¶

首先打开策略编译页面，找到algorithmId， 如下所示(上方红色隐去的部分)：



测试所使用的代码如下所示：

```
def initialize(context):
    # 定义一个全局变量, 保存要操作的股票
    # 000001(股票:平安银行)
    g.security = '000001.XSHE'
    # 初始化此策略
    # 设置我们要操作的股票池, 这里我们只操作一支股票
    set_universe([g.security])
    # 设置均线长度
    g.ma_long = 5
    g.m = 1.01

# 每个单位时间(如果按天回测,则每天调用一次,如果按分钟,则每分钟调用一次)调用一次
def handle_data(context, data):
    # 取得过去五天的平均价格
    average_price = data[security].mavg(g.ma_long, 'close')
```

```
    # 取得上一时间点价格
    current_price = data[security].close
    # 取得当前的现金
    cash = context.portfolio.cash

    # 如果上一时间点价格高出五天平均价1%, 则全仓买入
    if current_price > g.m*average_price:
        # 用所有 cash 买入股票
        order_value(security, cash)
        # 记录这次买入
        log.info("Buying %s" % (security))
    # 如果上一时间点价格低于五天平均价, 则空仓卖出
    elif current_price < average_price and context.portfolio.positions[security].sellable_amount > 0:
        # 卖出所有股票,使这只股票的最终持有量为0
        order_target(security, 0)
        # 记录这次卖出
        log.info("Selling %s" % (security))
    # 画出上一时间点价格
    record(stock_price=current_price)
```

接下来设定初始化仓位，设定g.全局变量的值，当然也可以选择不设置：

In [1]:

```
# 设定初始仓位
InitialPositions = [
    {
        'security': '000001.XSHE',
        'amount': '1000',  # 100股
        'avg_cost': '10.0' # 持仓均价10元
    }
]

# 设定g.全局变量的值
extra_vars = {'ma_long': 5, 'm': 1.01}
```

传入create_backtest 函数的algorithm_id 切记要改成自己的algorithmId ，否则会提示错误：¶

```
Exception: can not find algorithm
```

获取 algorithmId 文章初始所示。¶

In [3]:

```
# 记得此处传入你自己的 algorithmId
out_algorithm_id = create_backtest('bfdde744e43517e4de41d8bc0ffd59d7','2015-01-01', '2016-06-01', \
                   frequency="day",initial_cash=100000, initial_positions=InitialPositions, extras=extra_vars)
out_algorithm_id
```

Out[3]:

```
u'849c256bca210cd95e2d2bc9219239b3'
```

In [3]:

```
out_algorithm_id = '456bbf8cd483f21218528c45335cb229'
```

In [12]:

```
gt = get_backtest(out_algorithm_id)
```

In [14]:

```
gt.get_status()        # 获取回测状态
```

Out[14]:

```
u'running'
```

In [5]:

```
gt.get_params()        # 获取回测参数
```

Out[5]:

```
{u'algorithm_id': u'bfdde744e43517e4de41d8bc0ffd59d7',
 u'end_date': u'2016-06-01 23:59:59',
```

```
 u'extras': {u'm': 1.01, u'ma_long': 5},
 u'frequency': u'day',
 u'initial_cash': u'100000',
 u'initial_positions': [{u'amount': u'1000',
   u'avg_cost': u'10.0',
   u'security': u'000001.XSHE'}],
 u'start_date': u'2015-01-01 00:00:00'}
```

In [29]:

```
gt.get_results()       # 获取收益曲线
```

Out[29]:

```
[{u'benchmark_returns': 0.030516412660366532,
  u'returns': 0.10932029999999981,
  u'time': u'2015-01-05'},
 {u'benchmark_returns': 0.03038001191384132,
  u'returns': 0.0919802999999999,
  u'time': u'2015-01-06'},
 {u'benchmark_returns': 0.031152855147783987,
  u'returns': 0.07158029999999993,
  u'time': u'2015-01-07'},
 {u'benchmark_returns': 0.007231503478643653,
  u'returns': 0.07018800000000014,
  u'time': u'2015-01-08'},
 {u'benchmark_returns': 0.003683952112584432,
  u'returns': 0.07018800000000014,
  u'time': u'2015-01-09'},
 {u'benchmark_returns': -0.005696287607482753,
  u'returns': 0.07018800000000014,
  u'time': u'2015-01-12'},
 {u'benchmark_returns': -0.005564980664769736,
  u'returns': 0.07018800000000014,
  u'time': u'2015-01-13'},
 {u'benchmark_returns': -0.008852465047308744,
  u'returns': 0.07018800000000014,
  u'time': u'2015-01-14'},
 {u'benchmark_returns': 0.01992696051311582,
  u'returns': 0.07018800000000014,
  u'time': u'2015-01-15'},
 {u'benchmark_returns': 0.028706697361551292,
  u'returns': 0.06680789999999992,
  u'time': u'2015-01-16'},
 {u'benchmark_returns': -0.050527703925483225,
  u'returns': -0.04029209999999994,
  u'time': u'2015-01-19'},
 {u'benchmark_returns': -0.03890619052807176,
  u'returns': -0.04255460000000011,
  u'time': u'2015-01-20'},
 {u'benchmark_returns': 0.0042957745482432586,
  u'returns': -0.04255460000000011,
  u'time': u'2015-01-21'},
 {u'benchmark_returns': 0.009595594425680698,
  u'returns': -0.04255460000000011,
  u'time': u'2015-01-22'},
 {u'benchmark_returns': 0.010761226531360135,
  u'returns': -0.04255460000000011,
  u'time': u'2015-01-23'},
 {u'benchmark_returns': 0.02102043039812318,
  u'returns': -0.044801300000000044,
  u'time': u'2015-01-26'},
 {u'benchmark_returns': 0.011666508664418762,
  u'returns': -0.06832130000000003,
  u'time': u'2015-01-27'},
 {u'benchmark_returns': -0.0023720146418561017,
  u'returns': -0.07833889999999999,
  u'time': u'2015-01-28'},
 {u'benchmark_returns': -0.014689115248726292,
  u'returns': -0.07833889999999999,
  u'time': u'2015-01-29'},
 {u'benchmark_returns': -0.028105062533516523,
  u'returns': -0.07833889999999999,
  u'time': u'2015-01-30'},
 {u'benchmark_returns': -0.050865875900789614,
  u'returns': -0.07833889999999999,
  u'time': u'2015-02-02'},
 {u'benchmark_returns': -0.027240530830955012,
  u'returns': -0.07833889999999999,
  u'time': u'2015-02-03'},
 {u'benchmark_returns': -0.03733673297572937,
```

```
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-04'},
{u'benchmark_returns': -0.04719097943942685,
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-05'},
{u'benchmark_returns': -0.06262124314282036,
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-06'},
{u'benchmark_returns': -0.0531408252811143,
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-09'},
{u'benchmark_returns': -0.03587226437973734,
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-10'},
{u'benchmark_returns': -0.02818033763429606,
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-11'},
{u'benchmark_returns': -0.025704183003391612,
 u'returns': -0.07833889999999999,
 u'time': u'2015-02-12'},
{u'benchmark_returns': -0.018076494783803354,
 u'returns': -0.07861419999999986,
 u'time': u'2015-02-13'},
{u'benchmark_returns': -0.00968445300329246,
 u'returns': -0.08055420000000002,
 u'time': u'2015-02-16'},
{u'benchmark_returns': -0.0032212649329810272,
 u'returns': -0.0757042,
 u'time': u'2015-02-17'},
{u'benchmark_returns': -0.015557608798697165,
 u'returns': -0.08831420000000001,
 u'time': u'2015-02-25'},
{u'benchmark_returns': 0.00922261479099129,
 u'returns': -0.09046320000000008,
 u'time': u'2015-02-26'},
{u'benchmark_returns': 0.01107562742220991,
 u'returns': -0.09046320000000008,
 u'time': u'2015-02-27'},
{u'benchmark_returns': 0.01911874364158872,
 u'returns': -0.09046320000000008,
 u'time': u'2015-03-02'},
{u'benchmark_returns': -0.007302533742912898,
 u'returns': -0.09046320000000008,
 u'time': u'2015-03-03'},
{u'benchmark_returns': -0.000815857577245338,
 u'returns': -0.09046320000000008,
 u'time': u'2015-03-04'},
{u'benchmark_returns': -0.010572755790310695,
 u'returns': -0.09046320000000008,
 u'time': u'2015-03-05'},
{u'benchmark_returns': -0.015616753520738103,
 u'returns': -0.09046320000000008,
 u'time': u'2015-03-06'},
{u'benchmark_returns': 0.0011446909122294624,
 u'returns': -0.09046320000000008,
 u'time': u'2015-03-09'},
{u'benchmark_returns': -0.0037054592842357126,
 u'returns': -0.10688419999999998,
 u'time': u'2015-03-10'},
{u'benchmark_returns': -0.002561617339308264,
 u'returns': -0.10118419999999995,
 u'time': u'2015-03-11'},
{u'benchmark_returns': 0.016735692424806325,
 u'returns': -0.05368419999999996,
 u'time': u'2015-03-12'},
{u'benchmark_returns': 0.023757500979849855,
 u'returns': -0.02803420000000001,
 u'time': u'2015-03-13'},
{u'benchmark_returns': 0.04866450368664044,
 u'returns': -0.001434199999999941,
 u'time': u'2015-03-16'},
{u'benchmark_returns': 0.06322457590545905,
 u'returns': -0.001434199999999941,
 u'time': u'2015-03-17'},
{u'benchmark_returns': 0.08839164559576984,
 u'returns': 0.005215799999999993,
 u'time': u'2015-03-18'},
{u'benchmark_returns': 0.086604286435908,
 u'returns': -0.015684200000000037,
 u'time': u'2015-03-19'},
{u'benchmark_returns': 0.10155601557006033,
 u'returns': -0.01790499999999995,
```

```
 u'time': u'2015-03-20'},
{u'benchmark_returns': 0.12404997021539721,
 u'returns': -0.01790499999999995,
 u'time': u'2015-03-23'},
{u'benchmark_returns': 0.12432871447956173,
 u'returns': -0.01790499999999995,
 u'time': u'2015-03-24'},
{u'benchmark_returns': 0.11509364816814083,
 u'returns': -0.01790499999999995,
 u'time': u'2015-03-25'},
{u'benchmark_returns': 0.11780666467630985,
 u'returns': -0.01790499999999995,
 u'time': u'2015-03-26'},
{u'benchmark_returns': 0.12394696218275159,
 u'returns': -0.01790499999999995,
 u'time': u'2015-03-27'},
{u'benchmark_returns': 0.1569098156184514,
 u'returns': -0.01790499999999995,
 u'time': u'2015-03-30'},
{u'benchmark_returns': 0.14644657661010196,
 u'returns': -0.03709899999999999,
 u'time': u'2015-03-31'},
{u'benchmark_returns': 0.167017337327253,
 u'returns': -0.023598999999999926,
 u'time': u'2015-04-01'},
{u'benchmark_returns': 0.1672666507249474,
 u'returns': -0.033498999999999945,
 u'time': u'2015-04-02'},
{u'benchmark_returns': 0.18021679795002687,
 u'returns': -0.03079899999999991,
 u'time': u'2015-04-03'},
{u'benchmark_returns': 0.20554602039502456,
 u'returns': 0.027709999999999976,
 u'time': u'2015-04-07'},
{u'benchmark_returns': 0.21566542764605412,
 u'returns': 0.09610099999999999,
 u'time': u'2015-04-08'},
{u'benchmark_returns': 0.20613859957183744,
 u'returns': 0.10060100000000016,
 u'time': u'2015-04-09'},
{u'benchmark_returns': 0.2294223767971577,
 u'returns': 0.21040100000000006,
 u'time': u'2015-04-10'},
{u'benchmark_returns': 0.251115472287585,
 u'returns': 0.22390100000000013,
 u'time': u'2015-04-13'},
{u'benchmark_returns': 0.25595769878923114,
 u'returns': 0.2059010000000001,
 u'time': u'2015-04-14'},
{u'benchmark_returns': 0.23963743436421536,
 u'returns': 0.23200100000000012,
 u'time': u'2015-04-15'},
{u'benchmark_returns': 0.27728432339428455,
 u'returns': 0.25810100000000014,
 u'time': u'2015-04-16'},
{u'benchmark_returns': 0.3006563932190154,
 u'returns': 0.25270100000000006,
 u'time': u'2015-04-17'},
{u'benchmark_returns': 0.2796537911342345,
 u'returns': 0.2059010000000001,
 u'time': u'2015-04-20'},
{u'benchmark_returns': 0.30717306622935414,
 u'returns': 0.2043355,
 u'time': u'2015-04-21'},
{u'benchmark_returns': 0.3413157012257675,
 u'returns': 0.2043355,
 u'time': u'2015-04-22'},
{u'benchmark_returns': 0.34162076347629466,
 u'returns': 0.1798953000000001,
 u'time': u'2015-04-23'},
{u'benchmark_returns': 0.3307961473863834,
 u'returns': 0.1508817,
 u'time': u'2015-04-24'},
{u'benchmark_returns': 0.36049613649130285,
 u'returns': 0.1508817,
 u'time': u'2015-04-27'},
{u'benchmark_returns': 0.3418949799148485,
 u'returns': 0.1508817,
 u'time': u'2015-04-28'},
{u'benchmark_returns': 0.3510822210682556,
 u'returns': 0.1508817,
 u'time': u'2015-04-29'},
```

```
{u'benchmark_returns': 0.34416596744776395,
 u'returns': 0.1508817,
 u'time': u'2015-04-30'},
{u'benchmark_returns': 0.3548785198538078,
 u'returns': 0.13642979999999993,
 u'time': u'2015-05-04'},
{u'benchmark_returns': 0.3008547685785883,
 u'returns': 0.12584850000000003,
 u'time': u'2015-05-05'},
{u'benchmark_returns': 0.2885419128082283,
 u'returns': 0.12584850000000003,
 u'time': u'2015-05-06'},
{u'benchmark_returns': 0.2649864660462602,
 u'returns': 0.12584850000000003,
 u'time': u'2015-05-07'},
{u'benchmark_returns': 0.28997864847235433,
 u'returns': 0.12584850000000003,
 u'time': u'2015-05-08'},
{u'benchmark_returns': 0.32736943236631255,
 u'returns': 0.12584850000000003,
 u'time': u'2015-05-11'},
{u'benchmark_returns': 0.3434683993146006,
 u'returns': 0.12296339999999994,
 u'time': u'2015-05-12'},
{u'benchmark_returns': 0.33526596023154154,
 u'returns': 0.11701340000000005,
 u'time': u'2015-05-13'},
{u'benchmark_returns': 0.3302689386918263,
 u'returns': 0.11786340000000006,
 u'time': u'2015-05-14'},
{u'benchmark_returns': 0.3066936826928113,
 u'returns': 0.10793240000000015,
 u'time': u'2015-05-15'},
{u'benchmark_returns': 0.29471475406124736,
 u'returns': 0.10793240000000015,
 u'time': u'2015-05-18'},
{u'benchmark_returns': 0.3388842588727694,
 u'returns': 0.10793240000000015,
 u'time': u'2015-05-19'},
{u'benchmark_returns': 0.34559025159145995,
 u'returns': 0.10793240000000015,
 u'time': u'2015-05-20'},
{u'benchmark_returns': 0.36994344462823014,
 u'returns': 0.10590119999999992,
 u'time': u'2015-05-21'},
{u'benchmark_returns': 0.4011738387895991,
 u'returns': 0.13905119999999993,
 u'time': u'2015-05-22'},
{u'benchmark_returns': 0.44310150394557546,
 u'returns': 0.16200119999999996,
 u'time': u'2015-05-25'},
{u'benchmark_returns': 0.4712371293019648,
 u'returns': 0.1637012,
 u'time': u'2015-05-26'},
{u'benchmark_returns': 0.46628934786576703,
 u'returns': 0.1535012,
 u'time': u'2015-05-27'},
{u'benchmark_returns': 0.3679712935856274,
 u'returns': 0.08550119999999994,
 u'time': u'2015-05-28'},
{u'benchmark_returns': 0.3699018452304308,
 u'returns': 0.08324559999999992,
 u'time': u'2015-05-29'},
{u'benchmark_returns': 0.43650333007424225,
 u'returns': 0.08324559999999992,
 u'time': u'2015-06-01'},
{u'benchmark_returns': 0.4607529491001654,
 u'returns': 0.08324559999999992,
 u'time': u'2015-06-02'},
{u'benchmark_returns': 0.4555799083398304,
 u'returns': 0.08324559999999992,
 u'time': u'2015-06-03'},
{u'benchmark_returns': 0.46628425406195495,
 u'returns': 0.11695119999999992,
 u'time': u'2015-06-04'},
{u'benchmark_returns': 0.48018920651271113,
 u'returns': 0.11197119999999994,
 u'time': u'2015-06-05'},
{u'benchmark_returns': 0.5150531807267444,
 u'returns': 0.18003119999999995,
 u'time': u'2015-06-08'},
{u'benchmark_returns': 0.5047835062632564,
```

```
 u'returns': 0.15845120000000001,
 u'time': u'2015-06-09'},
{u'benchmark_returns': 0.5024208302617226,
 u'returns': 0.13853119999999985,
 u'time': u'2015-06-10'},
{u'benchmark_returns': 0.5017071317498207,
 u'returns': 0.1353951,
 u'time': u'2015-06-11'},
{u'benchmark_returns': 0.5097793958465688,
 u'returns': 0.1353951,
 u'time': u'2015-06-12'},
{u'benchmark_returns': 0.4775333538028783,
 u'returns': 0.1353951,
 u'time': u'2015-06-15'},
{u'benchmark_returns': 0.43328913986877793,
 u'returns': 0.1353951,
 u'time': u'2015-06-16'},
{u'benchmark_returns': 0.45423316320971896,
 u'returns': 0.1353951,
 u'time': u'2015-06-17'},
{u'benchmark_returns': 0.39529162734297296,
 u'returns': 0.1353951,
 u'time': u'2015-06-18'},
{u'benchmark_returns': 0.3122351752622248,
 u'returns': 0.1353951,
 u'time': u'2015-06-19'},
{u'benchmark_returns': 0.35441158783769455,
 u'returns': 0.1353951,
 u'time': u'2015-06-23'},
{u'benchmark_returns': 0.3810224679196481,
 u'returns': 0.1353951,
 u'time': u'2015-06-24'},
{u'benchmark_returns': 0.3318927301514982,
 u'returns': 0.1353951,
 u'time': u'2015-06-25'},
{u'benchmark_returns': 0.2270959234005101,
 u'returns': 0.1353951,
 u'time': u'2015-06-26'},
{u'benchmark_returns': 0.18616268194430496,
 u'returns': 0.1353951,
 u'time': u'2015-06-29'},
{u'benchmark_returns': 0.26580968134012317,
 u'returns': 0.1353951,
 u'time': u'2015-06-30'},
{u'benchmark_returns': 0.20355858794098536,
 u'returns': 0.10049550000000007,
 u'time': u'2015-07-01'},
{u'benchmark_returns': 0.16251809361562453,
 u'returns': 0.09619280000000008,
 u'time': u'2015-07-02'},
{u'benchmark_returns': 0.09967215712686817,
 u'returns': 0.09619280000000008,
 u'time': u'2015-07-03'},
{u'benchmark_returns': 0.13154238964486287,
 u'returns': 0.09619280000000008,
 u'time': u'2015-07-06'},
{u'benchmark_returns': 0.11158203641786746,
 u'returns': 0.09619280000000008,
 u'time': u'2015-07-07'},
{u'benchmark_returns': 0.03659982935757222,
 u'returns': 0.045945899999999984,
 u'time': u'2015-07-08'},
{u'benchmark_returns': 0.10298567650666945,
 u'returns': 0.0436356,
 u'time': u'2015-07-09'},
{u'benchmark_returns': 0.1621105893106527,
 u'returns': 0.11794490000000013,
 u'time': u'2015-07-10'},
{u'benchmark_returns': 0.19189689009127808,
 u'returns': 0.08700490000000016,
 u'time': u'2015-07-13'},
{u'benchmark_returns': 0.16369334735072694,
 u'returns': 0.044234899999999966,
 u'time': u'2015-07-14'},
{u'benchmark_returns': 0.12254956200361944,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-15'},
{u'benchmark_returns': 0.13120817951696595,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-16'},
{u'benchmark_returns': 0.17482811949497767,
 u'returns': 0.02652880000000013,
```

```
 u'time': u'2015-07-17'},
{u'benchmark_returns': 0.17740784813672916,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-20'},
{u'benchmark_returns': 0.178934291345769,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-21'},
{u'benchmark_returns': 0.17643096976119965,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-22'},
{u'benchmark_returns': 0.20293148409389028,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-23'},
{u'benchmark_returns': 0.18184143837700106,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-24'},
{u'benchmark_returns': 0.08065925140893215,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-27'},
{u'benchmark_returns': 0.07849778065797786,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-28'},
{u'benchmark_returns': 0.11225441852107076,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-29'},
{u'benchmark_returns': 0.0797200105837923,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-30'},
{u'benchmark_returns': 0.08008421755636075,
 u'returns': 0.02652880000000013,
 u'time': u'2015-07-31'},
{u'benchmark_returns': 0.08363176892241997,
 u'returns': 0.02652880000000013,
 u'time': u'2015-08-03'},
{u'benchmark_returns': 0.11728483277466584,
 u'returns': 0.0320433,
 u'time': u'2015-08-04'},
{u'benchmark_returns': 0.0942905533993359,
 u'returns': 0.010703300000000082,
 u'time': u'2015-08-05'},
{u'benchmark_returns': 0.08437235139888588,
 u'returns': 0.005853299999999839,
 u'time': u'2015-08-06'},
{u'benchmark_returns': 0.10562285193585774,
 u'returns': 0.009399399999999947,
 u'time': u'2015-08-07'},
{u'benchmark_returns': 0.1558307781775785,
 u'returns': 0.009399399999999947,
 u'time': u'2015-08-10'},
{u'benchmark_returns': 0.15082300305203744,
 u'returns': 0.002446999999999866,
 u'time': u'2015-08-11'},
{u'benchmark_returns': 0.1365198849366318,
 u'returns': -0.018453000000000053,
 u'time': u'2015-08-12'},
{u'benchmark_returns': 0.1533116092033715,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-13'},
{u'benchmark_returns': 0.15276742116277386,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-14'},
{u'benchmark_returns': 0.15399332994689718,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-17'},
{u'benchmark_returns': 0.08254990159054021,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-18'},
{u'benchmark_returns': 0.09973413173991608,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-19'},
{u'benchmark_returns': 0.06445048468958214,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-20'},
{u'benchmark_returns': 0.015799564479774153,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-21'},
{u'benchmark_returns': -0.07305957911031047,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-24'},
{u'benchmark_returns': -0.13888482485097076,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-25'},
```

```
{u'benchmark_returns': -0.14376214200110082,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-26'},
{u'benchmark_returns': -0.0928393852910755,
 u'returns': -0.025420099999999946,
 u'time': u'2015-08-27'},
{u'benchmark_returns': -0.05416949066206711,
 u'returns': -0.03225279999999997,
 u'time': u'2015-08-28'},
{u'benchmark_returns': -0.0473070049707035,
 u'returns': -0.01045280000000004,
 u'time': u'2015-08-31'},
{u'benchmark_returns': -0.048567155436008314,
 u'returns': 0.03423719999999997,
 u'time': u'2015-09-01'},
{u'benchmark_returns': -0.04750622929757864,
 u'returns': 0.05821720000000008,
 u'time': u'2015-09-02'},
{u'benchmark_returns': -0.08014817309311328,
 u'returns': -0.026802800000000016,
 u'time': u'2015-09-07'},
{u'benchmark_returns': -0.056507829600942916,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-08'},
{u'benchmark_returns': -0.038033735130691415,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-09'},
{u'benchmark_returns': -0.04984683214925967,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-10'},
{u'benchmark_returns': -0.05278227809055924,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-11'},
{u'benchmark_returns': -0.07147540612473302,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-14'},
{u'benchmark_returns': -0.10795270120171319,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-15'},
{u'benchmark_returns': -0.06351888457016075,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-16'},
{u'benchmark_returns': -0.08396456410481357,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-17'},
{u'benchmark_returns': -0.07992517768178153,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-18'},
{u'benchmark_returns': -0.06380215665993627,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-21'},
{u'benchmark_returns': -0.055090903173864225,
 u'returns': -0.030246799999999907,
 u'time': u'2015-09-22'},
{u'benchmark_returns': -0.07659864080334955,
 u'returns': -0.0456561000000002,
 u'time': u'2015-09-23'},
{u'benchmark_returns': -0.07038108727242376,
 u'returns': -0.04257580000000005,
 u'time': u'2015-09-24'},
{u'benchmark_returns': -0.08539309308501986,
 u'returns': -0.04257580000000005,
 u'time': u'2015-09-25'},
{u'benchmark_returns': -0.08233652780863143,
 u'returns': -0.04257580000000005,
 u'time': u'2015-09-28'},
{u'benchmark_returns': -0.1004189653635491,
 u'returns': -0.04257580000000005,
 u'time': u'2015-09-29'},
{u'benchmark_returns': -0.09360062597189067,
 u'returns': -0.04257580000000005,
 u'time': u'2015-09-30'},
{u'benchmark_returns': -0.06713152342937512,
 u'returns': -0.04257580000000005,
 u'time': u'2015-10-08'},
{u'benchmark_returns': -0.05478357701053138,
 u'returns': -0.030982200000000182,
 u'time': u'2015-10-09'},
{u'benchmark_returns': -0.024340458527239 76,
 u'returns': -0.001822199999999996,
 u'time': u'2015-10-12'},
{u'benchmark_returns': -0.025090379644027938,
```

```
 u'returns': -0.010462199999999977,
 u'time': u'2015-10-13'},
{u'benchmark_returns': -0.036107428322398105,
 u'returns': -0.02342220000000006,
 u'time': u'2015-10-14'},
{u'benchmark_returns': -0.01326935893064074,
 u'returns': -0.029003,
 u'time': u'2015-10-15'},
{u'benchmark_returns': 0.00010187607624301265,
 u'returns': -0.029003,
 u'time': u'2015-10-16'},
{u'benchmark_returns': 0.00013498580102178082,
 u'returns': -0.029003,
 u'time': u'2015-10-19'},
{u'benchmark_returns': 0.012450954451489249,
 u'returns': -0.029003,
 u'time': u'2015-10-20'},
{u'benchmark_returns': -0.017106974124891594,
 u'returns': -0.0365724999999999,
 u'time': u'2015-10-21'},
{u'benchmark_returns': -0.002595293042288449,
 u'returns': -0.035740399999999894,
 u'time': u'2015-10-22'},
{u'benchmark_returns': 0.010622278882928882,
 u'returns': -0.035740399999999894,
 u'time': u'2015-10-23'},
{u'benchmark_returns': 0.015721742477088574,
 u'returns': -0.0441094999999998,
 u'time': u'2015-10-26'},
{u'benchmark_returns': 0.016745597043329896,
 u'returns': -0.0471395,
 u'time': u'2015-10-27'},
{u'benchmark_returns': -0.0024860592494280187,
 u'returns': -0.06531949999999997,
 u'time': u'2015-10-28'},
{u'benchmark_returns': -0.00011262966206859737,
 u'returns': -0.06451669999999998,
 u'time': u'2015-10-29'},
{u'benchmark_returns': 0.00010583792365248534,
 u'returns': -0.06451669999999998,
 u'time': u'2015-10-30'},
{u'benchmark_returns': -0.01634063964026422,
 u'returns': -0.06451669999999998,
 u'time': u'2015-11-02'},
{u'benchmark_returns': -0.019303252535228577,
 u'returns': -0.06451669999999998,
 u'time': u'2015-11-03'},
{u'benchmark_returns': 0.02683670538429217,
 u'returns': -0.06451669999999998,
 u'time': u'2015-11-04'},
{u'benchmark_returns': 0.04874911742774213,
 u'returns': -0.02853720000000004,
 u'time': u'2015-11-05'},
{u'benchmark_returns': 0.07348349678312127,
 u'returns': -0.003057199999999982,
 u'time': u'2015-11-06'},
{u'benchmark_returns': 0.0867786077219237,
 u'returns': 0.04202280000000003,
 u'time': u'2015-11-09'},
{u'benchmark_returns': 0.08476457429242101,
 u'returns': 0.02634279999999989,
 u'time': u'2015-11-10'},
{u'benchmark_returns': 0.08488003384549647,
 u'returns': 0.010662799999999972,
 u'time': u'2015-11-11'},
{u'benchmark_returns': 0.07403532552943726,
 u'returns': -0.0020772000000000013,
 u'time': u'2015-11-12'},
{u'benchmark_returns': 0.06014565449011733,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-13'},
{u'benchmark_returns': 0.06520663156658535,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-16'},
{u'benchmark_returns': 0.06358255711781258,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-17'},
{u'benchmark_returns': 0.051468642685226884,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-18'},
{u'benchmark_returns': 0.06827508238520208,
 u'returns': -0.01903369999999993,
```

```
 u'time': u'2015-11-19'},
{u'benchmark_returns': 0.06810925077220653,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-20'},
{u'benchmark_returns': 0.062152896181203676,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-23'},
{u'benchmark_returns': 0.062309672142977446,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-24'},
{u'benchmark_returns': 0.07015356403548112,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-25'},
{u'benchmark_returns': 0.06387743176071581,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-26'},
{u'benchmark_returns': 0.006589967187413892,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-27'},
{u'benchmark_returns': 0.00925572451577028,
 u'returns': -0.01903369999999993,
 u'time': u'2015-11-30'},
{u'benchmark_returns': 0.016410820937231563,
 u'returns': -0.01903369999999993,
 u'time': u'2015-12-01'},
{u'benchmark_returns': 0.053272698202028845,
 u'returns': -0.01903369999999993,
 u'time': u'2015-12-02'},
{u'benchmark_returns': 0.061010752170880256,
 u'returns': -0.01548689999999997,
 u'time': u'2015-12-03'},
{u'benchmark_returns': 0.04071845272879315,
 u'returns': -0.041406900000000024,
 u'time': u'2015-12-04'},
{u'benchmark_returns': 0.04355258857205113,
 u'returns': -0.03852690000000003,
 u'time': u'2015-12-07'},
{u'benchmark_returns': 0.025275171526768725,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-08'},
{u'benchmark_returns': 0.028930824729285654,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-09'},
{u'benchmark_returns': 0.02529328282921184,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-10'},
{u'benchmark_returns': 0.021041371591573155,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-11'},
{u'benchmark_returns': 0.05026367509455376,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-14'},
{u'benchmark_returns': 0.04547182065282751,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-15'},
{u'benchmark_returns': 0.042939351190888875,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-16'},
{u'benchmark_returns': 0.06287536735522625,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-17'},
{u'benchmark_returns': 0.06627831129084072,
 u'returns': -0.04744240000000011,
 u'time': u'2015-12-18'},
{u'benchmark_returns': 0.09402595859020502,
 u'returns': -0.020176799999999884,
 u'time': u'2015-12-21'},
{u'benchmark_returns': 0.09707318522627117,
 u'returns': -0.02587679999999981,
 u'time': u'2015-12-22'},
{u'benchmark_returns': 0.0941436820560857,
 u'returns': -0.022076799999999897,
 u'time': u'2015-12-23'},
{u'benchmark_returns': 0.0836790281022326,
 u'returns': -0.03347679999999975,
 u'time': u'2015-12-24'},
{u'benchmark_returns': 0.08616904919907009,
 u'returns': -0.031881600000000065,
 u'time': u'2015-12-25'},
{u'benchmark_returns': 0.05487922732655948,
 u'returns': -0.031881600000000065,
 u'time': u'2015-12-28'},
```

```
{u'benchmark_returns': 0.06456962310096626,
 u'returns': -0.031881600000000065,
 u'time': u'2015-12-29'},
{u'benchmark_returns': 0.0655034871331932,
 u'returns': -0.031881600000000065,
 u'time': u'2015-12-30'},
{u'benchmark_returns': 0.05583374956313558,
 u'returns': -0.031881600000000065,
 u'time': u'2015-12-31'},
{u'benchmark_returns': -0.018292132478517575,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-04'},
{u'benchmark_returns': -0.015543176354562593,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-05'},
{u'benchmark_returns': 0.001727082481418174,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-06'},
{u'benchmark_returns': -0.06772523456259083,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-07'},
{u'benchmark_returns': -0.04871430976835922,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-08'},
{u'benchmark_returns': -0.09657144555077468,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-11'},
{u'benchmark_returns': -0.08998911906907903,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-12'},
{u'benchmark_returns': -0.10692063995155232,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-13'},
{u'benchmark_returns': -0.08833051995002417,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-14'},
{u'benchmark_returns': -0.11743340205251995,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-15'},
{u'benchmark_returns': -0.11403784413243323,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-18'},
{u'benchmark_returns': -0.08789072658866537,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-19'},
{u'benchmark_returns': -0.10168559627925933,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-20'},
{u'benchmark_returns': -0.12801283638560657,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-21'},
{u'benchmark_returns': -0.11892401884141435,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-22'},
{u'benchmark_returns': -0.11455936474606676,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-25'},
{u'benchmark_returns': -0.16786814405843165,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-26'},
{u'benchmark_returns': -0.17074232285943514,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-27'},
{u'benchmark_returns': -0.19241809941690102,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-28'},
{u'benchmark_returns': -0.16628858379519507,
 u'returns': -0.031881600000000065,
 u'time': u'2016-01-29'},
{u'benchmark_returns': -0.17903512036233915,
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-01'},
{u'benchmark_returns': -0.161974839439059,
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-02'},
{u'benchmark_returns': -0.1655674143710354,
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-03'},
{u'benchmark_returns': -0.15534550846774142,
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-04'},
{u'benchmark_returns': -0.16127990310453189,
```

```
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-05'},
{u'benchmark_returns': -0.16611414931353918,
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-15'},
{u'benchmark_returns': -0.14055211173541649,
 u'returns': -0.031881600000000065,
 u'time': u'2016-02-16'},
{u'benchmark_returns': -0.13311385641982,
 u'returns': -0.02281119999999992,
 u'time': u'2016-02-17'},
{u'benchmark_returns': -0.1358366077530524,
 u'returns': -0.02749120000000005,
 u'time': u'2016-02-18'},
{u'benchmark_returns': -0.13643470521732848,
 u'returns': -0.03334119999999996,
 u'time': u'2016-02-19'},
{u'benchmark_returns': -0.11739449105117716,
 u'returns': -0.00877120000000009,
 u'time': u'2016-02-22'},
{u'benchmark_returns': -0.1257438863742163,
 u'returns': -0.02515119999999993,
 u'time': u'2016-02-23'},
{u'benchmark_returns': -0.12003254374657757,
 u'returns': -0.03342600000000007,
 u'time': u'2016-02-24'},
{u'benchmark_returns': -0.17402553410655386,
 u'returns': -0.03342600000000007,
 u'time': u'2016-02-25'},
{u'benchmark_returns': -0.16573947174424575,
 u'returns': -0.03342600000000007,
 u'time': u'2016-02-26'},
{u'benchmark_returns': -0.1857083146442614,
 u'returns': -0.03342600000000007,
 u'time': u'2016-02-29'},
{u'benchmark_returns': -0.17064562548373452,
 u'returns': -0.03342600000000007,
 u'time': u'2016-03-01'},
{u'benchmark_returns': -0.13650610336742885,
 u'returns': -0.03342600000000007,
 u'time': u'2016-03-02'},
{u'benchmark_returns': -0.13449993703492513,
 u'returns': -0.03371479999999982,
 u'time': u'2016-03-03'},
{u'benchmark_returns': -0.12446299280783191,
 u'returns': -0.005874800000000069,
 u'time': u'2016-03-04'},
{u'benchmark_returns': -0.12136420555762295,
 u'returns': -0.011674799999999985,
 u'time': u'2016-03-07'},
{u'benchmark_returns': -0.12056249743541125,
 u'returns': -0.017474800000000013,
 u'time': u'2016-03-08'},
{u'benchmark_returns': -0.13068380071341545,
 u'returns': -0.027914799999999906,
 u'time': u'2016-03-09'},
{u'benchmark_returns': -0.14731260815489688,
 u'returns': -0.023381499999999944,
 u'time': u'2016-03-10'},
{u'benchmark_returns': -0.14585841206325934,
 u'returns': -0.023381499999999944,
 u'time': u'2016-03-11'},
{u'benchmark_returns': -0.1324430307566704,
 u'returns': -0.023381499999999944,
 u'time': u'2016-03-14'},
{u'benchmark_returns': -0.12987088622281717,
 u'returns': -0.023381499999999944,
 u'time': u'2016-03-15'},
{u'benchmark_returns': -0.12555439687240444,
 u'returns': -0.016772700000000085,
 u'time': u'2016-03-16'},
{u'benchmark_returns': -0.11588434801433622,
 u'returns': -0.009872700000000068,
 u'time': u'2016-03-17'},
{u'benchmark_returns': -0.10236904325629892,
 u'returns': 0.0004773000000000138,
 u'time': u'2016-03-18'},
{u'benchmark_returns': -0.08044468907280033,
 u'returns': 0.025777299999999892,
 u'time': u'2016-03-21'},
{u'benchmark_returns': -0.08713650403754691,
 u'returns': 0.0177273,
```

```
 u'time': u'2016-03-22'},
{u'benchmark_returns': -0.0842211220234852,
 u'returns': 0.016577299999999795,
 u'time': u'2016-03-23'},
{u'benchmark_returns': -0.09957093192555688,
 u'returns': -0.0006727000000000816,
 u'time': u'2016-03-24'},
{u'benchmark_returns': -0.09505267134636308,
 u'returns': -0.004261399999999971,
 u'time': u'2016-03-25'},
{u'benchmark_returns': -0.10300143899957681,
 u'returns': -0.004261399999999971,
 u'time': u'2016-03-28'},
{u'benchmark_returns': -0.11271320045108468,
 u'returns': -0.004261399999999971,
 u'time': u'2016-03-29'},
{u'benchmark_returns': -0.0898291453304676,
 u'returns': -0.004261399999999971,
 u'time': u'2016-03-30'},
{u'benchmark_returns': -0.08931619928658441,
 u'returns': -0.011339700000000064,
 u'time': u'2016-03-31'},
{u'benchmark_returns': -0.08823888807922564,
 u'returns': -0.010209699999999988,
 u'time': u'2016-04-01'},
{u'benchmark_returns': -0.07618609929238573,
 u'returns': -0.005689700000000131,
 u'time': u'2016-04-05'},
{u'benchmark_returns': -0.0781551940527011,
 u'returns': -0.004559699999999833,
 u'time': u'2016-04-06'},
{u'benchmark_returns': -0.0918058524975911,
 u'returns': -0.015859700000000143,
 u'time': u'2016-04-07'},
{u'benchmark_returns': -0.09847432086153196,
 u'returns': -0.022780399999999923,
 u'time': u'2016-04-08'},
{u'benchmark_returns': -0.08591812276350175,
 u'returns': -0.022780399999999923,
 u'time': u'2016-04-11'},
{u'benchmark_returns': -0.0892138138299603,
 u'returns': -0.022780399999999923,
 u'time': u'2016-04-12'},
{u'benchmark_returns': -0.07706412957504938,
 u'returns': -0.022780399999999923,
 u'time': u'2016-04-13'},
{u'benchmark_returns': -0.07297522005939938,
 u'returns': -0.027432700000000088,
 u'time': u'2016-04-14'},
{u'benchmark_returns': -0.07400156492972665,
 u'returns': -0.024162700000000092,
 u'time': u'2016-04-15'},
{u'benchmark_returns': -0.08638296066027007,
 u'returns': -0.03724270000000007,
 u'time': u'2016-04-18'},
{u'benchmark_returns': -0.08359557461644362,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-19'},
{u'benchmark_returns': -0.099804199841243,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-20'},
{u'benchmark_returns': -0.10558371454323434,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-21'},
{u'benchmark_returns': -0.10153756468069641,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-22'},
{u'benchmark_returns': -0.10517909672708947,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-25'},
{u'benchmark_returns': -0.10033222920419216,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-26'},
{u'benchmark_returns': -0.10408039154371973,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-27'},
{u'benchmark_returns': -0.10559005349908945,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-28'},
{u'benchmark_returns': -0.10667554309145777,
 u'returns': -0.03631390000000001,
 u'time': u'2016-04-29'},
```

```
 {u'benchmark_returns': -0.09060329031427361,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-03'},
 {u'benchmark_returns': -0.09175740476355543,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-04'},
 {u'benchmark_returns': -0.090495697858197,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-05'},
 {u'benchmark_returns': -0.1141438235506359,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-06'},
 {u'benchmark_returns': -0.13246425493922098,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-09'},
 {u'benchmark_returns': -0.13147464205416126,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-10'},
 {u'benchmark_returns': -0.12759876673349924,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-11'},
 {u'benchmark_returns': -0.1255248528102939,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-12'},
 {u'benchmark_returns': -0.1298268814176622,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-13'},
 {u'benchmark_returns': -0.1240614312739744,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-16'},
 {u'benchmark_returns': -0.12668858888899892,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-17'},
 {u'benchmark_returns': -0.13177930811994776,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-18'},
 {u'benchmark_returns': -0.1333458225856431,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-19'},
 {u'benchmark_returns': -0.12889777160232674,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-20'},
 {u'benchmark_returns': -0.12634968114203082,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-23'},
 {u'benchmark_returns': -0.1330471558887909,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-24'},
 {u'benchmark_returns': -0.1342724137979826,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-25'},
 {u'benchmark_returns': -0.13286165653329862,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-26'},
 {u'benchmark_returns': -0.1333460772758337,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-27'},
 {u'benchmark_returns': -0.1321546365641727,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-30'},
 {u'benchmark_returns': -0.10304912266304067,
  u'returns': -0.03631390000000001,
  u'time': u'2016-05-31'},
 {u'benchmark_returns': -0.10559964682960232,
  u'returns': -0.03993160000000007,
  u'time': u'2016-06-01'}]
```

In [30]:

```
gt.get_positions()     # 获取所有持仓列表
```

Out[30]:

```
[{u'amount': 10200,
  u'avg_cost': 10.775686274509804,
  u'closeable_amount': 10200,
  u'price': 10.87,
  u'security': u'000001.XSHE',
  u'security_name': u'\u5e73\u5b89\u94f6\u884c',
  u'time': u'2015-01-05'},
 {u'amount': 10200,
```

```
 u'avg_cost': 10.775686274509804,
 u'closeable_amount': 10200,
 u'price': 10.7,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-06'},
{u'amount': 10200,
 u'avg_cost': 10.775686274509804,
 u'closeable_amount': 10200,
 u'price': 10.5,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-07'},
{u'amount': 10200,
 u'avg_cost': 10.46,
 u'closeable_amount': 10200,
 u'price': 10.43,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-16'},
{u'amount': 10200,
 u'avg_cost': 10.46,
 u'closeable_amount': 10200,
 u'price': 9.38,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-19'},
{u'amount': 9800,
 u'avg_cost': 9.75,
 u'closeable_amount': 9800,
 u'price': 9.73,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-26'},
{u'amount': 9800,
 u'avg_cost': 9.75,
 u'closeable_amount': 9800,
 u'price': 9.49,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-27'},
{u'amount': 9700,
 u'avg_cost': 9.46,
 u'closeable_amount': 9700,
 u'price': 9.46,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-02-13'},
{u'amount': 9700,
 u'avg_cost': 9.46,
 u'closeable_amount': 9700,
 u'price': 9.44,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-02-16'},
{u'amount': 9700,
 u'avg_cost': 9.46,
 u'closeable_amount': 9700,
 u'price': 9.49,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-02-17'},
{u'amount': 9700,
 u'avg_cost': 9.46,
 u'closeable_amount': 9700,
 u'price': 9.36,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-02-25'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 9.34,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-10'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 9.4,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
```

```
 u'time': u'2015-03-11'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 9.9,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-12'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 10.17,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-13'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 10.45,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-16'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 10.45,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-17'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 10.52,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-18'},
{u'amount': 9500,
 u'avg_cost': 9.51,
 u'closeable_amount': 9500,
 u'price': 10.3,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-19'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 10.68,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-31'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 10.83,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-01'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 10.72,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-02'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 10.75,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-03'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 11.4,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-07'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 12.16,
```

```
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-08'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 12.21,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-09'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.43,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-10'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.58,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-13'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.38,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-14'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.67,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-15'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.96,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-16'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.9,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-17'},
{u'amount': 9000,
 u'avg_cost': 10.89,
 u'closeable_amount': 9000,
 u'price': 13.38,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-20'},
{u'amount': 8600,
 u'avg_cost': 13.96,
 u'closeable_amount': 8600,
 u'price': 13.68,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-23'},
{u'amount': 8300,
 u'avg_cost': 13.73,
 u'closeable_amount': 8300,
 u'price': 13.56,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-04'},
{u'amount': 8500,
 u'avg_cost': 13.14,
 u'closeable_amount': 8500,
 u'price': 13.11,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-12'},
{u'amount': 8500,
 u'avg_cost': 13.14,
```

```
 u'closeable_amount': 8500,
 u'price': 13.04,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-13'},
{u'amount': 8500,
 u'avg_cost': 13.14,
 u'closeable_amount': 8500,
 u'price': 13.05,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-14'},
{u'amount': 8500,
 u'avg_cost': 12.99,
 u'closeable_amount': 8500,
 u'price': 12.97,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-21'},
{u'amount': 8500,
 u'avg_cost': 12.99,
 u'closeable_amount': 8500,
 u'price': 13.36,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-22'},
{u'amount': 8500,
 u'avg_cost': 12.99,
 u'closeable_amount': 8500,
 u'price': 13.63,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-25'},
{u'amount': 8500,
 u'avg_cost': 12.99,
 u'closeable_amount': 8500,
 u'price': 13.65,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-26'},
{u'amount': 8500,
 u'avg_cost': 12.99,
 u'closeable_amount': 8500,
 u'price': 13.53,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-27'},
{u'amount': 8500,
 u'avg_cost': 12.99,
 u'closeable_amount': 8500,
 u'price': 12.73,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-28'},
{u'amount': 8300,
 u'avg_cost': 13.03,
 u'closeable_amount': 8300,
 u'price': 13.44,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-04'},
{u'amount': 8300,
 u'avg_cost': 13.03,
 u'closeable_amount': 8300,
 u'price': 13.38,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-05'},
{u'amount': 8300,
 u'avg_cost': 13.03,
 u'closeable_amount': 8300,
 u'price': 14.2,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-08'},
{u'amount': 8300,
 u'avg_cost': 13.03,
 u'closeable_amount': 8300,
 u'price': 13.94,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-09'},
```

```
{u'amount': 8300,
 u'avg_cost': 13.03,
 u'closeable_amount': 8300,
 u'price': 13.7,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-10'},
{u'amount': 9600,
 u'avg_cost': 11.79,
 u'closeable_amount': 9600,
 u'price': 11.43,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-01'},
{u'amount': 9600,
 u'avg_cost': 11.35,
 u'closeable_amount': 9600,
 u'price': 10.83,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-08'},
{u'amount': 9100,
 u'avg_cost': 11.38,
 u'closeable_amount': 9100,
 u'price': 12.2,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-10'},
{u'amount': 9100,
 u'avg_cost': 11.38,
 u'closeable_amount': 9100,
 u'price': 11.86,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-13'},
{u'amount': 9100,
 u'avg_cost': 11.38,
 u'closeable_amount': 9100,
 u'price': 11.39,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-14'},
{u'amount': 9700,
 u'avg_cost': 10.5,
 u'closeable_amount': 9700,
 u'price': 10.56,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-04'},
{u'amount': 9700,
 u'avg_cost': 10.5,
 u'closeable_amount': 9700,
 u'price': 10.34,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-05'},
{u'amount': 9700,
 u'avg_cost': 10.5,
 u'closeable_amount': 9700,
 u'price': 10.29,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-06'},
{u'amount': 9500,
 u'avg_cost': 10.61,
 u'closeable_amount': 9500,
 u'price': 10.54,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-11'},
{u'amount': 9500,
 u'avg_cost': 10.61,
 u'closeable_amount': 9500,
 u'price': 10.32,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-12'},
{u'amount': 10900,
 u'avg_cost': 8.95,
 u'closeable_amount': 10900,
 u'price': 8.89,
 u'security': u'000001.XSHE',
```

```
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-28'},
{u'amount': 10900,
 u'avg_cost': 8.95,
 u'closeable_amount': 10900,
 u'price': 9.09,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-31'},
{u'amount': 10900,
 u'avg_cost': 8.95,
 u'closeable_amount': 10900,
 u'price': 9.5,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-01'},
{u'amount': 10900,
 u'avg_cost': 8.95,
 u'closeable_amount': 10900,
 u'price': 9.72,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-02'},
{u'amount': 10900,
 u'avg_cost': 8.95,
 u'closeable_amount': 10900,
 u'price': 8.94,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-07'},
{u'amount': 10800,
 u'avg_cost': 8.93,
 u'closeable_amount': 10800,
 u'price': 8.79,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-23'},
{u'amount': 10800,
 u'avg_cost': 8.84,
 u'closeable_amount': 10800,
 u'price': 8.95,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-09'},
{u'amount': 10800,
 u'avg_cost': 8.84,
 u'closeable_amount': 10800,
 u'price': 9.22,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-12'},
{u'amount': 10800,
 u'avg_cost': 8.84,
 u'closeable_amount': 10800,
 u'price': 9.14,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-13'},
{u'amount': 10800,
 u'avg_cost': 8.84,
 u'closeable_amount': 10800,
 u'price': 9.02,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-14'},
{u'amount': 10400,
 u'avg_cost': 9.28,
 u'closeable_amount': 10400,
 u'price': 9.21,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-21'},
{u'amount': 10100,
 u'avg_cost': 9.54,
 u'closeable_amount': 10100,
 u'price': 9.46,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-26'},
{u'amount': 10100,
 u'avg_cost': 9.54,
 u'closeable_amount': 10100,
```

```
 u'price': 9.43,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-27'},
{u'amount': 10100,
 u'avg_cost': 9.54,
 u'closeable_amount': 10100,
 u'price': 9.25,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-28'},
{u'amount': 9800,
 u'avg_cost': 9.54,
 u'closeable_amount': 9800,
 u'price': 9.91,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-05'},
{u'amount': 9800,
 u'avg_cost': 9.54,
 u'closeable_amount': 9800,
 u'price': 10.17,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-06'},
{u'amount': 9800,
 u'avg_cost': 9.54,
 u'closeable_amount': 9800,
 u'price': 10.63,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-09'},
{u'amount': 9800,
 u'avg_cost': 9.54,
 u'closeable_amount': 9800,
 u'price': 10.47,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-10'},
{u'amount': 9800,
 u'avg_cost': 9.54,
 u'closeable_amount': 9800,
 u'price': 10.31,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-11'},
{u'amount': 9800,
 u'avg_cost': 9.54,
 u'closeable_amount': 9800,
 u'price': 10.18,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-12'},
{u'amount': 9600,
 u'avg_cost': 10.18,
 u'closeable_amount': 9600,
 u'price': 10.22,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-03'},
{u'amount': 9600,
 u'avg_cost': 10.18,
 u'closeable_amount': 9600,
 u'price': 9.95,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-04'},
{u'amount': 9600,
 u'avg_cost': 10.18,
 u'closeable_amount': 9600,
 u'price': 9.98,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-07'},
{u'amount': 9500,
 u'avg_cost': 9.98,
 u'closeable_amount': 9500,
 u'price': 10.27,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-21'},
{u'amount': 9500,
```

```
 u'avg_cost': 9.98,
 u'closeable_amount': 9500,
 u'price': 10.21,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-22'},
{u'amount': 9500,
 u'avg_cost': 9.98,
 u'closeable_amount': 9500,
 u'price': 10.25,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-23'},
{u'amount': 9500,
 u'avg_cost': 9.98,
 u'closeable_amount': 9500,
 u'price': 10.13,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-24'},
{u'amount': 11700,
 u'avg_cost': 8.25,
 u'closeable_amount': 11700,
 u'price': 8.33,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-17'},
{u'amount': 11700,
 u'avg_cost': 8.25,
 u'closeable_amount': 11700,
 u'price': 8.29,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-18'},
{u'amount': 11700,
 u'avg_cost': 8.25,
 u'closeable_amount': 11700,
 u'price': 8.24,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-19'},
{u'amount': 11700,
 u'avg_cost': 8.25,
 u'closeable_amount': 11700,
 u'price': 8.45,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-22'},
{u'amount': 11700,
 u'avg_cost': 8.25,
 u'closeable_amount': 11700,
 u'price': 8.31,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-23'},
{u'amount': 11600,
 u'avg_cost': 8.3,
 u'closeable_amount': 11600,
 u'price': 8.3,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-03'},
{u'amount': 11600,
 u'avg_cost': 8.3,
 u'closeable_amount': 11600,
 u'price': 8.54,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-04'},
{u'amount': 11600,
 u'avg_cost': 8.3,
 u'closeable_amount': 11600,
 u'price': 8.49,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-07'},
{u'amount': 11600,
 u'avg_cost': 8.3,
 u'closeable_amount': 11600,
 u'price': 8.44,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
```

```
 u'time': u'2016-03-08'},
{u'amount': 11600,
 u'avg_cost': 8.3,
 u'closeable_amount': 11600,
 u'price': 8.35,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-09'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.5,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-16'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.56,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-17'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.65,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-18'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.87,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-21'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.8,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-22'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.79,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-23'},
{u'amount': 11500,
 u'avg_cost': 8.44,
 u'closeable_amount': 11500,
 u'price': 8.64,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-24'},
{u'amount': 11300,
 u'avg_cost': 8.8,
 u'closeable_amount': 11300,
 u'price': 8.74,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-31'},
{u'amount': 11300,
 u'avg_cost': 8.8,
 u'closeable_amount': 11300,
 u'price': 8.75,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-01'},
{u'amount': 11300,
 u'avg_cost': 8.8,
 u'closeable_amount': 11300,
 u'price': 8.79,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-05'},
{u'amount': 11300,
 u'avg_cost': 8.8,
 u'closeable_amount': 11300,
 u'price': 8.8,
```

```
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-06'},
{u'amount': 11300,
 u'avg_cost': 8.8,
 u'closeable_amount': 11300,
 u'price': 8.7,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-07'},
{u'amount': 10900,
 u'avg_cost': 8.94,
 u'closeable_amount': 10900,
 u'price': 8.9,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-14'},
{u'amount': 10900,
 u'avg_cost': 8.94,
 u'closeable_amount': 10900,
 u'price': 8.93,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-15'},
{u'amount': 10900,
 u'avg_cost': 8.94,
 u'closeable_amount': 10900,
 u'price': 8.81,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-18'},
{u'amount': 11100,
 u'avg_cost': 8.64,
 u'closeable_amount': 11100,
 u'price': 8.61,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-06-01'}]
```

In [31]:

```
gt.get_orders()        # 获取交易列表
```

Out[31]:

```
[{u'action': u'open',
  u'amount': 9200,
  u'commission': 29.97,
  u'filled': 9200,
  u'price': 10.86,
  u'security': u'000001.XSHE',
  u'security_name': u'\u5e73\u5b89\u94f6\u884c',
  u'time': u'2015-01-05 09:30:00'},
 {u'action': u'close',
  u'amount': 10200,
  u'commission': 139.23,
  u'filled': 10200,
  u'price': 10.5,
  u'security': u'000001.XSHE',
  u'security_name': u'\u5e73\u5b89\u94f6\u884c',
  u'time': u'2015-01-08 09:30:00'},
 {u'action': u'open',
  u'amount': 10200,
  u'commission': 32.01,
  u'filled': 10200,
  u'price': 10.46,
  u'security': u'000001.XSHE',
  u'security_name': u'\u5e73\u5b89\u94f6\u884c',
  u'time': u'2015-01-16 09:30:00'},
 {u'action': u'close',
  u'amount': 10200,
  u'commission': 124.25,
  u'filled': 10200,
  u'price': 9.37,
  u'security': u'000001.XSHE',
  u'security_name': u'\u5e73\u5b89\u94f6\u884c',
  u'time': u'2015-01-20 09:30:00'},
 {u'action': u'open',
  u'amount': 9800,
  u'commission': 28.67,
  u'filled': 9800,
```

```
 u'price': 9.75,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-26 09:30:00'},
{u'action': u'close',
 u'amount': 9800,
 u'commission': 119.76,
 u'filled': 9800,
 u'price': 9.4,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-01-28 09:30:00'},
{u'action': u'open',
 u'amount': 9700,
 u'commission': 27.53,
 u'filled': 9700,
 u'price': 9.46,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-02-13 09:30:00'},
{u'action': u'close',
 u'amount': 9700,
 u'commission': 117.9,
 u'filled': 9700,
 u'price': 9.35,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-02-26 09:30:00'},
{u'action': u'open',
 u'amount': 9500,
 u'commission': 27.1,
 u'filled': 9500,
 u'price': 9.51,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-10 09:30:00'},
{u'action': u'close',
 u'amount': 9500,
 u'commission': 127.08,
 u'filled': 9500,
 u'price': 10.29,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-20 09:30:00'},
{u'action': u'open',
 u'amount': 9000,
 u'commission': 29.4,
 u'filled': 9000,
 u'price': 10.89,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-03-31 09:30:00'},
{u'action': u'close',
 u'amount': 9000,
 u'commission': 156.55,
 u'filled': 9000,
 u'price': 13.38,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-21 09:30:00'},
{u'action': u'open',
 u'amount': 8600,
 u'commission': 36.02,
 u'filled': 8600,
 u'price': 13.96,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-23 09:30:00'},
{u'action': u'close',
 u'amount': 8600,
 u'commission': 149.36,
 u'filled': 8600,
 u'price': 13.36,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-04-24 09:30:00'},
{u'action': u'open',
 u'amount': 8300,
 u'commission': 34.19,
 u'filled': 8300,
 u'price': 13.73,
 u'security': u'000001.XSHE',
```

```
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-04 09:30:00'},
{u'action': u'close',
 u'amount': 8300,
 u'commission': 145.13,
 u'filled': 8300,
 u'price': 13.45,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-05 09:30:00'},
{u'action': u'open',
 u'amount': 8500,
 u'commission': 33.51,
 u'filled': 8500,
 u'price': 13.14,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-12 09:30:00'},
{u'action': u'close',
 u'amount': 8500,
 u'commission': 143.1,
 u'filled': 8500,
 u'price': 12.95,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-15 09:30:00'},
{u'action': u'open',
 u'amount': 8500,
 u'commission': 33.12,
 u'filled': 8500,
 u'price': 12.99,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-21 09:30:00'},
{u'action': u'close',
 u'amount': 8500,
 u'commission': 140.56,
 u'filled': 8500,
 u'price': 12.72,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-05-29 09:30:00'},
{u'action': u'open',
 u'amount': 8300,
 u'commission': 32.44,
 u'filled': 8300,
 u'price': 13.03,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-04 09:30:00'},
{u'action': u'close',
 u'amount': 8300,
 u'commission': 147.61,
 u'filled': 8300,
 u'price': 13.68,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-06-11 09:30:00'},
{u'action': u'open',
 u'amount': 9600,
 u'commission': 33.96,
 u'filled': 9600,
 u'price': 11.79,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-01 09:30:00'},
{u'action': u'close',
 u'amount': 9600,
 u'commission': 142.27,
 u'filled': 9600,
 u'price': 11.4,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-02 09:30:00'},
{u'action': u'open',
 u'amount': 9600,
 u'commission': 32.69,
 u'filled': 9600,
 u'price': 11.35,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-08 09:30:00'},
```

```
{u'action': u'close',
 u'amount': 9600,
 u'commission': 135.03,
 u'filled': 9600,
 u'price': 10.82,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-09 09:30:00'},
{u'action': u'open',
 u'amount': 9100,
 u'commission': 31.07,
 u'filled': 9100,
 u'price': 11.38,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-10 09:30:00'},
{u'action': u'close',
 u'amount': 9100,
 u'commission': 132.61,
 u'filled': 9100,
 u'price': 11.21,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-07-15 09:30:00'},
{u'action': u'open',
 u'amount': 9700,
 u'commission': 30.55,
 u'filled': 9700,
 u'price': 10.5,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-04 09:30:00'},
{u'action': u'close',
 u'amount': 9700,
 u'commission': 130.39,
 u'filled': 9700,
 u'price': 10.34,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-07 09:30:00'},
{u'action': u'open',
 u'amount': 9500,
 u'commission': 30.24,
 u'filled': 9500,
 u'price': 10.61,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-11 09:30:00'},
{u'action': u'close',
 u'amount': 9500,
 u'commission': 126.71,
 u'filled': 9500,
 u'price': 10.26,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-13 09:30:00'},
{u'action': u'open',
 u'amount': 10900,
 u'commission': 29.27,
 u'filled': 10900,
 u'price': 8.95,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-08-28 09:30:00'},
{u'action': u'close',
 u'amount': 10900,
 u'commission': 126.4,
 u'filled': 10900,
 u'price': 8.92,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-08 09:30:00'},
{u'action': u'open',
 u'amount': 10800,
 u'commission': 28.93,
 u'filled': 10800,
 u'price': 8.93,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-23 09:30:00'},
{u'action': u'close',
 u'amount': 10800,
```

```
 u'commission': 123.97,
 u'filled': 10800,
 u'price': 8.83,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-09-24 09:30:00'},
{u'action': u'open',
 u'amount': 10800,
 u'commission': 28.64,
 u'filled': 10800,
 u'price': 8.84,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-09 09:30:00'},
{u'action': u'close',
 u'amount': 10800,
 u'commission': 126.08,
 u'filled': 10800,
 u'price': 8.98,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-15 09:30:00'},
{u'action': u'open',
 u'amount': 10400,
 u'commission': 28.95,
 u'filled': 10400,
 u'price': 9.28,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-21 09:30:00'},
{u'action': u'close',
 u'amount': 10400,
 u'commission': 124.79,
 u'filled': 10400,
 u'price': 9.23,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-22 09:30:00'},
{u'action': u'open',
 u'amount': 10100,
 u'commission': 28.91,
 u'filled': 10100,
 u'price': 9.54,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-26 09:30:00'},
{u'action': u'close',
 u'amount': 10100,
 u'commission': 121.72,
 u'filled': 10100,
 u'price': 9.27,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-10-29 09:30:00'},
{u'action': u'open',
 u'amount': 9800,
 u'commission': 28.05,
 u'filled': 9800,
 u'price': 9.54,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-05 09:30:00'},
{u'action': u'close',
 u'amount': 9800,
 u'commission': 127.65,
 u'filled': 9800,
 u'price': 10.02,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-11-13 09:30:00'},
{u'action': u'open',
 u'amount': 9600,
 u'commission': 29.32,
 u'filled': 9600,
 u'price': 10.18,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-03 09:30:00'},
{u'action': u'close',
 u'amount': 9600,
 u'commission': 123.55,
 u'filled': 9600,
```

```
 u'price': 9.9,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-08 09:30:00'},
{u'action': u'open',
 u'amount': 9500,
 u'commission': 28.44,
 u'filled': 9500,
 u'price': 9.98,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-21 09:30:00'},
{u'action': u'close',
 u'amount': 9500,
 u'commission': 125.48,
 u'filled': 9500,
 u'price': 10.16,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2015-12-25 09:30:00'},
{u'action': u'open',
 u'amount': 11700,
 u'commission': 28.96,
 u'filled': 11700,
 u'price': 8.25,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-17 09:30:00'},
{u'action': u'close',
 u'amount': 11700,
 u'commission': 125.48,
 u'filled': 11700,
 u'price': 8.25,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-02-24 09:30:00'},
{u'action': u'open',
 u'amount': 11600,
 u'commission': 28.88,
 u'filled': 11600,
 u'price': 8.3,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-03 09:30:00'},
{u'action': u'close',
 u'amount': 11600,
 u'commission': 126.67,
 u'filled': 11600,
 u'price': 8.4,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-10 09:30:00'},
{u'action': u'open',
 u'amount': 11500,
 u'commission': 29.12,
 u'filled': 11500,
 u'price': 8.44,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-16 09:30:00'},
{u'action': u'close',
 u'amount': 11500,
 u'commission': 128.87,
 u'filled': 11500,
 u'price': 8.62,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-25 09:30:00'},
{u'action': u'open',
 u'amount': 11300,
 u'commission': 29.83,
 u'filled': 11300,
 u'price': 8.8,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-03-31 09:30:00'},
{u'action': u'close',
 u'amount': 11300,
 u'commission': 127.07,
 u'filled': 11300,
 u'price': 8.65,
 u'security': u'000001.XSHE',
```

```
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-08 09:30:00'},
{u'action': u'open',
 u'amount': 10900,
 u'commission': 29.23,
 u'filled': 10900,
 u'price': 8.94,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-14 09:30:00'},
{u'action': u'close',
 u'amount': 10900,
 u'commission': 125.12,
 u'filled': 10900,
 u'price': 8.83,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-04-19 09:30:00'},
{u'action': u'open',
 u'amount': 11100,
 u'commission': 28.77,
 u'filled': 11100,
 u'price': 8.64,
 u'security': u'000001.XSHE',
 u'security_name': u'\u5e73\u5b89\u94f6\u884c',
 u'time': u'2016-06-01 09:30:00'}]
```

In [32]:

```
gt.get_records()       # 获取所有record()记录
```

Out[32]:

```
[{u'stock_price': 10.74, u'time': u'2015-01-05'},
 {u'stock_price': 10.87, u'time': u'2015-01-06'},
 {u'stock_price': 10.7, u'time': u'2015-01-07'},
 {u'stock_price': 10.5, u'time': u'2015-01-08'},
 {u'stock_price': 10.15, u'time': u'2015-01-09'},
 {u'stock_price': 10.23, u'time': u'2015-01-12'},
 {u'stock_price': 10.02, u'time': u'2015-01-13'},
 {u'stock_price': 9.96, u'time': u'2015-01-14'},
 {u'stock_price': 10.05, u'time': u'2015-01-15'},
 {u'stock_price': 10.41, u'time': u'2015-01-16'},
 {u'stock_price': 10.43, u'time': u'2015-01-19'},
 {u'stock_price': 9.38, u'time': u'2015-01-20'},
 {u'stock_price': 9.38, u'time': u'2015-01-21'},
 {u'stock_price': 9.78, u'time': u'2015-01-22'},
 {u'stock_price': 9.7, u'time': u'2015-01-23'},
 {u'stock_price': 9.77, u'time': u'2015-01-26'},
 {u'stock_price': 9.73, u'time': u'2015-01-27'},
 {u'stock_price': 9.49, u'time': u'2015-01-28'},
 {u'stock_price': 9.54, u'time': u'2015-01-29'},
 {u'stock_price': 9.43, u'time': u'2015-01-30'},
 {u'stock_price': 9.45, u'time': u'2015-02-02'},
 {u'stock_price': 9.25, u'time': u'2015-02-03'},
 {u'stock_price': 9.46, u'time': u'2015-02-04'},
 {u'stock_price': 9.3, u'time': u'2015-02-05'},
 {u'stock_price': 9.35, u'time': u'2015-02-06'},
 {u'stock_price': 9.16, u'time': u'2015-02-09'},
 {u'stock_price': 9.17, u'time': u'2015-02-10'},
 {u'stock_price': 9.34, u'time': u'2015-02-11'},
 {u'stock_price': 9.31, u'time': u'2015-02-12'},
 {u'stock_price': 9.4, u'time': u'2015-02-13'},
 {u'stock_price': 9.46, u'time': u'2015-02-16'},
 {u'stock_price': 9.44, u'time': u'2015-02-17'},
 {u'stock_price': 9.49, u'time': u'2015-02-25'},
 {u'stock_price': 9.36, u'time': u'2015-02-26'},
 {u'stock_price': 9.54, u'time': u'2015-02-27'},
 {u'stock_price': 9.49, u'time': u'2015-03-02'},
 {u'stock_price': 9.52, u'time': u'2015-03-03'},
 {u'stock_price': 9.23, u'time': u'2015-03-04'},
 {u'stock_price': 9.2, u'time': u'2015-03-05'},
 {u'stock_price': 9.08, u'time': u'2015-03-06'},
 {u'stock_price': 9.12, u'time': u'2015-03-09'},
 {u'stock_price': 9.6, u'time': u'2015-03-10'},
 {u'stock_price': 9.34, u'time': u'2015-03-11'},
 {u'stock_price': 9.4, u'time': u'2015-03-12'},
 {u'stock_price': 9.9, u'time': u'2015-03-13'},
 {u'stock_price': 10.17, u'time': u'2015-03-16'},
 {u'stock_price': 10.45, u'time': u'2015-03-17'},
 {u'stock_price': 10.45, u'time': u'2015-03-18'},
```

```
{u'stock_price': 10.52, u'time': u'2015-03-19'},
{u'stock_price': 10.3, u'time': u'2015-03-20'},
{u'stock_price': 10.39, u'time': u'2015-03-23'},
{u'stock_price': 10.47, u'time': u'2015-03-24'},
{u'stock_price': 10.41, u'time': u'2015-03-25'},
{u'stock_price': 10.11, u'time': u'2015-03-26'},
{u'stock_price': 10.25, u'time': u'2015-03-27'},
{u'stock_price': 10.23, u'time': u'2015-03-30'},
{u'stock_price': 10.62, u'time': u'2015-03-31'},
{u'stock_price': 10.68, u'time': u'2015-04-01'},
{u'stock_price': 10.83, u'time': u'2015-04-02'},
{u'stock_price': 10.72, u'time': u'2015-04-03'},
{u'stock_price': 10.75, u'time': u'2015-04-07'},
{u'stock_price': 11.4, u'time': u'2015-04-08'},
{u'stock_price': 12.16, u'time': u'2015-04-09'},
{u'stock_price': 12.21, u'time': u'2015-04-10'},
{u'stock_price': 13.43, u'time': u'2015-04-13'},
{u'stock_price': 13.58, u'time': u'2015-04-14'},
{u'stock_price': 13.38, u'time': u'2015-04-15'},
{u'stock_price': 13.67, u'time': u'2015-04-16'},
{u'stock_price': 13.96, u'time': u'2015-04-17'},
{u'stock_price': 13.9, u'time': u'2015-04-20'},
{u'stock_price': 13.38, u'time': u'2015-04-21'},
{u'stock_price': 13.6, u'time': u'2015-04-22'},
{u'stock_price': 13.92, u'time': u'2015-04-23'},
{u'stock_price': 13.68, u'time': u'2015-04-24'},
{u'stock_price': 13.25, u'time': u'2015-04-27'},
{u'stock_price': 13.48, u'time': u'2015-04-28'},
{u'stock_price': 13.71, u'time': u'2015-04-29'},
{u'stock_price': 13.67, u'time': u'2015-04-30'},
{u'stock_price': 13.71, u'time': u'2015-05-04'},
{u'stock_price': 13.56, u'time': u'2015-05-05'},
{u'stock_price': 13.02, u'time': u'2015-05-06'},
{u'stock_price': 12.91, u'time': u'2015-05-07'},
{u'stock_price': 12.87, u'time': u'2015-05-08'},
{u'stock_price': 13.01, u'time': u'2015-05-11'},
{u'stock_price': 13.16, u'time': u'2015-05-12'},
{u'stock_price': 13.11, u'time': u'2015-05-13'},
{u'stock_price': 13.04, u'time': u'2015-05-14'},
{u'stock_price': 13.05, u'time': u'2015-05-15'},
{u'stock_price': 12.69, u'time': u'2015-05-18'},
{u'stock_price': 12.46, u'time': u'2015-05-19'},
{u'stock_price': 12.8, u'time': u'2015-05-20'},
{u'stock_price': 12.92, u'time': u'2015-05-21'},
{u'stock_price': 12.97, u'time': u'2015-05-22'},
{u'stock_price': 13.36, u'time': u'2015-05-25'},
{u'stock_price': 13.63, u'time': u'2015-05-26'},
{u'stock_price': 13.65, u'time': u'2015-05-27'},
{u'stock_price': 13.53, u'time': u'2015-05-28'},
{u'stock_price': 12.73, u'time': u'2015-05-29'},
{u'stock_price': 12.58, u'time': u'2015-06-01'},
{u'stock_price': 13.06, u'time': u'2015-06-02'},
{u'stock_price': 12.95, u'time': u'2015-06-03'},
{u'stock_price': 13.0, u'time': u'2015-06-04'},
{u'stock_price': 13.44, u'time': u'2015-06-05'},
{u'stock_price': 13.38, u'time': u'2015-06-08'},
{u'stock_price': 14.2, u'time': u'2015-06-09'},
{u'stock_price': 13.94, u'time': u'2015-06-10'},
{u'stock_price': 13.7, u'time': u'2015-06-11'},
{u'stock_price': 13.53, u'time': u'2015-06-12'},
{u'stock_price': 13.56, u'time': u'2015-06-15'},
{u'stock_price': 13.07, u'time': u'2015-06-16'},
{u'stock_price': 12.84, u'time': u'2015-06-17'},
{u'stock_price': 12.92, u'time': u'2015-06-18'},
{u'stock_price': 12.63, u'time': u'2015-06-19'},
{u'stock_price': 12.01, u'time': u'2015-06-23'},
{u'stock_price': 12.28, u'time': u'2015-06-24'},
{u'stock_price': 12.42, u'time': u'2015-06-25'},
{u'stock_price': 12.21, u'time': u'2015-06-26'},
{u'stock_price': 11.31, u'time': u'2015-06-29'},
{u'stock_price': 11.13, u'time': u'2015-06-30'},
{u'stock_price': 11.94, u'time': u'2015-07-01'},
{u'stock_price': 11.43, u'time': u'2015-07-02'},
{u'stock_price': 11.29, u'time': u'2015-07-03'},
{u'stock_price': 10.73, u'time': u'2015-07-06'},
{u'stock_price': 11.4, u'time': u'2015-07-07'},
{u'stock_price': 12.03, u'time': u'2015-07-08'},
{u'stock_price': 10.83, u'time': u'2015-07-09'},
{u'stock_price': 11.71, u'time': u'2015-07-10'},
{u'stock_price': 12.2, u'time': u'2015-07-13'},
{u'stock_price': 11.86, u'time': u'2015-07-14'},
{u'stock_price': 11.39, u'time': u'2015-07-15'},
```

```
{u'stock_price': 11.15, u'time': u'2015-07-16'},
{u'stock_price': 11.17, u'time': u'2015-07-17'},
{u'stock_price': 11.35, u'time': u'2015-07-20'},
{u'stock_price': 11.17, u'time': u'2015-07-21'},
{u'stock_price': 11.14, u'time': u'2015-07-22'},
{u'stock_price': 11.1, u'time': u'2015-07-23'},
{u'stock_price': 11.22, u'time': u'2015-07-24'},
{u'stock_price': 10.99, u'time': u'2015-07-27'},
{u'stock_price': 10.21, u'time': u'2015-07-28'},
{u'stock_price': 10.36, u'time': u'2015-07-29'},
{u'stock_price': 10.35, u'time': u'2015-07-30'},
{u'stock_price': 10.07, u'time': u'2015-07-31'},
{u'stock_price': 10.15, u'time': u'2015-08-03'},
{u'stock_price': 10.53, u'time': u'2015-08-04'},
{u'stock_price': 10.56, u'time': u'2015-08-05'},
{u'stock_price': 10.34, u'time': u'2015-08-06'},
{u'stock_price': 10.29, u'time': u'2015-08-07'},
{u'stock_price': 10.35, u'time': u'2015-08-10'},
{u'stock_price': 10.61, u'time': u'2015-08-11'},
{u'stock_price': 10.54, u'time': u'2015-08-12'},
{u'stock_price': 10.32, u'time': u'2015-08-13'},
{u'stock_price': 10.32, u'time': u'2015-08-14'},
{u'stock_price': 10.38, u'time': u'2015-08-17'},
{u'stock_price': 10.3, u'time': u'2015-08-18'},
{u'stock_price': 10.08, u'time': u'2015-08-19'},
{u'stock_price': 10.04, u'time': u'2015-08-20'},
{u'stock_price': 9.89, u'time': u'2015-08-21'},
{u'stock_price': 9.44, u'time': u'2015-08-24'},
{u'stock_price': 8.5, u'time': u'2015-08-25'},
{u'stock_price': 7.77, u'time': u'2015-08-26'},
{u'stock_price': 8.14, u'time': u'2015-08-27'},
{u'stock_price': 8.87, u'time': u'2015-08-28'},
{u'stock_price': 8.89, u'time': u'2015-08-31'},
{u'stock_price': 9.09, u'time': u'2015-09-01'},
{u'stock_price': 9.5, u'time': u'2015-09-02'},
{u'stock_price': 9.72, u'time': u'2015-09-07'},
{u'stock_price': 8.94, u'time': u'2015-09-08'},
{u'stock_price': 9.03, u'time': u'2015-09-09'},
{u'stock_price': 9.11, u'time': u'2015-09-10'},
{u'stock_price': 9.07, u'time': u'2015-09-11'},
{u'stock_price': 9.0, u'time': u'2015-09-14'},
{u'stock_price': 8.93, u'time': u'2015-09-15'},
{u'stock_price': 8.68, u'time': u'2015-09-16'},
{u'stock_price': 8.95, u'time': u'2015-09-17'},
{u'stock_price': 8.84, u'time': u'2015-09-18'},
{u'stock_price': 8.88, u'time': u'2015-09-21'},
{u'stock_price': 8.88, u'time': u'2015-09-22'},
{u'stock_price': 9.0, u'time': u'2015-09-23'},
{u'stock_price': 8.79, u'time': u'2015-09-24'},
{u'stock_price': 8.79, u'time': u'2015-09-25'},
{u'stock_price': 8.66, u'time': u'2015-09-28'},
{u'stock_price': 8.65, u'time': u'2015-09-29'},
{u'stock_price': 8.52, u'time': u'2015-09-30'},
{u'stock_price': 8.61, u'time': u'2015-10-08'},
{u'stock_price': 8.79, u'time': u'2015-10-09'},
{u'stock_price': 8.95, u'time': u'2015-10-12'},
{u'stock_price': 9.22, u'time': u'2015-10-13'},
{u'stock_price': 9.14, u'time': u'2015-10-14'},
{u'stock_price': 9.02, u'time': u'2015-10-15'},
{u'stock_price': 9.17, u'time': u'2015-10-16'},
{u'stock_price': 9.22, u'time': u'2015-10-19'},
{u'stock_price': 9.25, u'time': u'2015-10-20'},
{u'stock_price': 9.3, u'time': u'2015-10-21'},
{u'stock_price': 9.21, u'time': u'2015-10-22'},
{u'stock_price': 9.3, u'time': u'2015-10-23'},
{u'stock_price': 9.42, u'time': u'2015-10-26'},
{u'stock_price': 9.46, u'time': u'2015-10-27'},
{u'stock_price': 9.43, u'time': u'2015-10-28'},
{u'stock_price': 9.25, u'time': u'2015-10-29'},
{u'stock_price': 9.25, u'time': u'2015-10-30'},
{u'stock_price': 9.33, u'time': u'2015-11-02'},
{u'stock_price': 9.15, u'time': u'2015-11-03'},
{u'stock_price': 9.07, u'time': u'2015-11-04'},
{u'stock_price': 9.6, u'time': u'2015-11-05'},
{u'stock_price': 9.91, u'time': u'2015-11-06'},
{u'stock_price': 10.17, u'time': u'2015-11-09'},
{u'stock_price': 10.63, u'time': u'2015-11-10'},
{u'stock_price': 10.47, u'time': u'2015-11-11'},
{u'stock_price': 10.31, u'time': u'2015-11-12'},
{u'stock_price': 10.18, u'time': u'2015-11-13'},
{u'stock_price': 10.05, u'time': u'2015-11-16'},
{u'stock_price': 10.13, u'time': u'2015-11-17'},
```

```
{u'stock_price': 10.26, u'time': u'2015-11-18'},
{u'stock_price': 10.2, u'time': u'2015-11-19'},
{u'stock_price': 10.26, u'time': u'2015-11-20'},
{u'stock_price': 10.31, u'time': u'2015-11-23'},
{u'stock_price': 10.22, u'time': u'2015-11-24'},
{u'stock_price': 10.08, u'time': u'2015-11-25'},
{u'stock_price': 10.12, u'time': u'2015-11-26'},
{u'stock_price': 10.04, u'time': u'2015-11-27'},
{u'stock_price': 9.63, u'time': u'2015-11-30'},
{u'stock_price': 9.64, u'time': u'2015-12-01'},
{u'stock_price': 9.65, u'time': u'2015-12-02'},
{u'stock_price': 10.27, u'time': u'2015-12-03'},
{u'stock_price': 10.22, u'time': u'2015-12-04'},
{u'stock_price': 9.95, u'time': u'2015-12-07'},
{u'stock_price': 9.98, u'time': u'2015-12-08'},
{u'stock_price': 9.82, u'time': u'2015-12-09'},
{u'stock_price': 9.85, u'time': u'2015-12-10'},
{u'stock_price': 9.82, u'time': u'2015-12-11'},
{u'stock_price': 9.71, u'time': u'2015-12-14'},
{u'stock_price': 9.91, u'time': u'2015-12-15'},
{u'stock_price': 9.79, u'time': u'2015-12-16'},
{u'stock_price': 9.76, u'time': u'2015-12-17'},
{u'stock_price': 9.91, u'time': u'2015-12-18'},
{u'stock_price': 10.04, u'time': u'2015-12-21'},
{u'stock_price': 10.27, u'time': u'2015-12-22'},
{u'stock_price': 10.21, u'time': u'2015-12-23'},
{u'stock_price': 10.25, u'time': u'2015-12-24'},
{u'stock_price': 10.13, u'time': u'2015-12-25'},
{u'stock_price': 10.19, u'time': u'2015-12-28'},
{u'stock_price': 9.84, u'time': u'2015-12-29'},
{u'stock_price': 9.93, u'time': u'2015-12-30'},
{u'stock_price': 9.94, u'time': u'2015-12-31'},
{u'stock_price': 9.85, u'time': u'2016-01-04'},
{u'stock_price': 9.3, u'time': u'2016-01-05'},
{u'stock_price': 9.36, u'time': u'2016-01-06'},
{u'stock_price': 9.47, u'time': u'2016-01-07'},
{u'stock_price': 8.98, u'time': u'2016-01-08'},
{u'stock_price': 9.13, u'time': u'2016-01-11'},
{u'stock_price': 8.84, u'time': u'2016-01-12'},
{u'stock_price': 8.88, u'time': u'2016-01-13'},
{u'stock_price': 8.79, u'time': u'2016-01-14'},
{u'stock_price': 8.84, u'time': u'2016-01-15'},
{u'stock_price': 8.59, u'time': u'2016-01-18'},
{u'stock_price': 8.55, u'time': u'2016-01-19'},
{u'stock_price': 8.79, u'time': u'2016-01-20'},
{u'stock_price': 8.65, u'time': u'2016-01-21'},
{u'stock_price': 8.47, u'time': u'2016-01-22'},
{u'stock_price': 8.54, u'time': u'2016-01-25'},
{u'stock_price': 8.52, u'time': u'2016-01-26'},
{u'stock_price': 8.1, u'time': u'2016-01-27'},
{u'stock_price': 8.11, u'time': u'2016-01-28'},
{u'stock_price': 7.96, u'time': u'2016-01-29'},
{u'stock_price': 8.21, u'time': u'2016-02-01'},
{u'stock_price': 8.05, u'time': u'2016-02-02'},
{u'stock_price': 8.17, u'time': u'2016-02-03'},
{u'stock_price': 8.09, u'time': u'2016-02-04'},
{u'stock_price': 8.17, u'time': u'2016-02-05'},
{u'stock_price': 8.15, u'time': u'2016-02-15'},
{u'stock_price': 8.04, u'time': u'2016-02-16'},
{u'stock_price': 8.22, u'time': u'2016-02-17'},
{u'stock_price': 8.33, u'time': u'2016-02-18'},
{u'stock_price': 8.29, u'time': u'2016-02-19'},
{u'stock_price': 8.24, u'time': u'2016-02-22'},
{u'stock_price': 8.45, u'time': u'2016-02-23'},
{u'stock_price': 8.31, u'time': u'2016-02-24'},
{u'stock_price': 8.33, u'time': u'2016-02-25'},
{u'stock_price': 7.94, u'time': u'2016-02-26'},
{u'stock_price': 8.04, u'time': u'2016-02-29'},
{u'stock_price': 7.85, u'time': u'2016-03-01'},
{u'stock_price': 7.96, u'time': u'2016-03-02'},
{u'stock_price': 8.29, u'time': u'2016-03-03'},
{u'stock_price': 8.3, u'time': u'2016-03-04'},
{u'stock_price': 8.54, u'time': u'2016-03-07'},
{u'stock_price': 8.49, u'time': u'2016-03-08'},
{u'stock_price': 8.44, u'time': u'2016-03-09'},
{u'stock_price': 8.35, u'time': u'2016-03-10'},
{u'stock_price': 8.33, u'time': u'2016-03-11'},
{u'stock_price': 8.34, u'time': u'2016-03-14'},
{u'stock_price': 8.42, u'time': u'2016-03-15'},
{u'stock_price': 8.47, u'time': u'2016-03-16'},
{u'stock_price': 8.5, u'time': u'2016-03-17'},
{u'stock_price': 8.56, u'time': u'2016-03-18'},
```

```
 {u'stock_price': 8.65, u'time': u'2016-03-21'},
 {u'stock_price': 8.87, u'time': u'2016-03-22'},
 {u'stock_price': 8.8, u'time': u'2016-03-23'},
 {u'stock_price': 8.79, u'time': u'2016-03-24'},
 {u'stock_price': 8.64, u'time': u'2016-03-25'},
 {u'stock_price': 8.7, u'time': u'2016-03-28'},
 {u'stock_price': 8.61, u'time': u'2016-03-29'},
 {u'stock_price': 8.56, u'time': u'2016-03-30'},
 {u'stock_price': 8.79, u'time': u'2016-03-31'},
 {u'stock_price': 8.74, u'time': u'2016-04-01'},
 {u'stock_price': 8.75, u'time': u'2016-04-05'},
 {u'stock_price': 8.79, u'time': u'2016-04-06'},
 {u'stock_price': 8.8, u'time': u'2016-04-07'},
 {u'stock_price': 8.7, u'time': u'2016-04-08'},
 {u'stock_price': 8.68, u'time': u'2016-04-11'},
 {u'stock_price': 8.8, u'time': u'2016-04-12'},
 {u'stock_price': 8.76, u'time': u'2016-04-13'},
 {u'stock_price': 8.88, u'time': u'2016-04-14'},
 {u'stock_price': 8.9, u'time': u'2016-04-15'},
 {u'stock_price': 8.93, u'time': u'2016-04-18'},
 {u'stock_price': 8.81, u'time': u'2016-04-19'},
 {u'stock_price': 8.84, u'time': u'2016-04-20'},
 {u'stock_price': 8.64, u'time': u'2016-04-21'},
 {u'stock_price': 8.63, u'time': u'2016-04-22'},
 {u'stock_price': 8.66, u'time': u'2016-04-25'},
 {u'stock_price': 8.62, u'time': u'2016-04-26'},
 {u'stock_price': 8.7, u'time': u'2016-04-27'},
 {u'stock_price': 8.68, u'time': u'2016-04-28'},
 {u'stock_price': 8.75, u'time': u'2016-04-29'},
 {u'stock_price': 8.68, u'time': u'2016-05-03'},
 {u'stock_price': 8.77, u'time': u'2016-05-04'},
 {u'stock_price': 8.8, u'time': u'2016-05-05'},
 {u'stock_price': 8.8, u'time': u'2016-05-06'},
 {u'stock_price': 8.64, u'time': u'2016-05-09'},
 {u'stock_price': 8.5, u'time': u'2016-05-10'},
 {u'stock_price': 8.44, u'time': u'2016-05-11'},
 {u'stock_price': 8.52, u'time': u'2016-05-12'},
 {u'stock_price': 8.52, u'time': u'2016-05-13'},
 {u'stock_price': 8.5, u'time': u'2016-05-16'},
 {u'stock_price': 8.51, u'time': u'2016-05-17'},
 {u'stock_price': 8.46, u'time': u'2016-05-18'},
 {u'stock_price': 8.46, u'time': u'2016-05-19'},
 {u'stock_price': 8.42, u'time': u'2016-05-20'},
 {u'stock_price': 8.46, u'time': u'2016-05-23'},
 {u'stock_price': 8.44, u'time': u'2016-05-24'},
 {u'stock_price': 8.38, u'time': u'2016-05-25'},
 {u'stock_price': 8.4, u'time': u'2016-05-26'},
 {u'stock_price': 8.39, u'time': u'2016-05-27'},
 {u'stock_price': 8.43, u'time': u'2016-05-30'},
 {u'stock_price': 8.44, u'time': u'2016-05-31'},
 {u'stock_price': 8.66, u'time': u'2016-06-01'}]
```

In [6]:

```
gt.get_risk()          # 获取总的风险指标
```

Out[6]:

```
{u'algorithm_return': -0.03993160000000073,
 u'algorithm_volatility': 0.26412380390523005,
 u'alpha': -0.03924882545523693,
 u'annual_algo_return': -0.029097790469759577,
 u'annual_bm_return': -0.07768720537202145,
 u'benchmark_return': -0.10559964682959888,
 u'benchmark_volatility': 0.3703428719597582,
 u'beta': 0.25362965260469034,
 u'excess_return': -0.09615077808219252,
 u'information': 0.04426724611143751,
 u'max_drawdown': 0.2570703782923631,
 u'max_drawdown_period': [u'2015-04-16', u'2015-10-28'],
 u'max_leverage': 0.0,
 u'period_label': u'2016-06',
 u'sharpe': -0.26161137106200577,
 u'sortino': -0.3579121975894534,
 u'trading_days': 345,
 u'treasury_return': 0.05621917808219178}
```

In [13]:

```
    k = gt.get_risk()
k['max_drawdown_period'] = [(u'2015-04-16', u'2015-10-28')]
```

```
import pandas as pd
pd.DataFrame(k)
```

Out[13]:

	algorithm_return	algorithm_volatility	alpha	annual_algo_return	annual_bm_return	benchmark_return	b
0	-0.039932	0.264124	-0.039249	-0.029098	-0.077687	-0.1056	0

In [34]:

```
gt.get_period_risks()  # 获取分月计算的风险指标
```

Out[34]:

```
{'algorithm_return':          one_month  three_month  six_month  twelve_month
 2015-01  -0.078339          NaN        NaN           NaN
 2015-02  -0.013155          NaN        NaN           NaN
 2015-03   0.058672    -0.037099        NaN           NaN
 2015-04   0.195223     0.248704        NaN           NaN
 2015-05  -0.058769     0.190986        NaN           NaN
 2015-06   0.048142     0.179140   0.135395           NaN
 2015-07  -0.095884    -0.108050   0.113781           NaN
 2015-08  -0.036026    -0.086498   0.087968           NaN
 2015-09  -0.032462    -0.156748  -0.005688           NaN
 2015-10  -0.022917    -0.088693  -0.187159           NaN
 2015-11   0.048620    -0.008672  -0.094419           NaN
 2015-12  -0.013097     0.011170  -0.147329    -0.031882
 2016-01   0.000000     0.034886  -0.056901     0.050406
 2016-02  -0.001595    -0.014672  -0.023216     0.062710
 2016-03   0.022850     0.021218   0.032625     0.026752
 2016-04  -0.025261    -0.004578   0.030148    -0.162654
 2016-05   0.000000    -0.002988  -0.017615    -0.110372
 2016-06  -0.003754    -0.028920  -0.008315    -0.154419,
 'algorithm_volatility':          one_month  three_month  six_month  twelve_month
 2015-01   0.551591          NaN        NaN           NaN
 2015-02   0.061606          NaN        NaN           NaN
 2015-03   0.252247     0.360363        NaN           NaN
 2015-04   0.504242     0.342942        NaN           NaN
 2015-05   0.257960     0.360300        NaN           NaN
 2015-06   0.256289     0.362769   0.361047           NaN
 2015-07   0.342802     0.289815   0.318062           NaN
 2015-08   0.137823     0.262110   0.314885           NaN
 2015-09   0.349207     0.290239   0.329422           NaN
 2015-10   0.168346     0.235108   0.264144           NaN
 2015-11   0.250009     0.267131   0.263587           NaN
 2015-12   0.143085     0.191121   0.246877     0.307872
 2016-01   0.000000     0.165536   0.201439     0.265925
 2016-02   0.137720     0.112635   0.203496     0.267121
 2016-03   0.160498     0.121426   0.160014     0.260393
 2016-04   0.071094     0.128748   0.148512     0.214105
 2016-05   0.000000     0.103656   0.107621     0.200957
 2016-06        NaN     0.050013   0.098227     0.194811,
 'alpha':          one_month  three_month  six_month  twelve_month
 2015-01  -0.340114          NaN        NaN           NaN
 2015-02  -0.285462          NaN        NaN           NaN
 2015-03  -0.705164    -0.771542        NaN           NaN
 2015-04   2.222741    -0.120284        NaN           NaN
 2015-05  -0.683038    -0.435415        NaN           NaN
 2015-06   0.761667     0.769600   0.005202           NaN
 2015-07  -0.475122    -0.254308   0.148066           NaN
 2015-08  -0.378048    -0.229071   0.173228           NaN
 2015-09  -0.210559    -0.392439   0.042345           NaN
 2015-10  -1.756624    -0.329792  -0.290167           NaN
 2015-11   0.696051    -0.158699  -0.135661           NaN
 2015-12  -0.356621    -0.268037  -0.241502    -0.077842
 2016-01  -0.040000     0.167151  -0.099187     0.050571
 2016-02  -0.026688    -0.058275  -0.033985     0.073793
 2016-03  -0.488207     0.087901   0.033778     0.037806
 2016-04  -0.242546    -0.120085   0.056010    -0.137978
 2016-05  -0.040000    -0.141177  -0.050841    -0.093103
 2016-06  -0.649478    -0.185857  -0.033635    -0.149710,
 'benchmark_return':          one_month  three_month  six_month  twelve_month
 2015-01  -0.028105          NaN        NaN           NaN
 2015-02   0.040314          NaN        NaN           NaN
 2015-03   0.133888     0.146447        NaN           NaN
 2015-04   0.172463     0.383036        NaN           NaN
 2015-05   0.019146     0.354896        NaN           NaN
 2015-06  -0.075985     0.104116   0.265810           NaN
 2015-07  -0.146725    -0.196465   0.111318           NaN
```

```
2015-08  -0.117946    -0.304554  -0.057743           NaN
2015-09  -0.048592    -0.283937  -0.209384           NaN
2015-10   0.103383    -0.074048  -0.255966           NaN
2015-11   0.009149     0.059371  -0.263264           NaN
2015-12   0.046151     0.164866  -0.165883      0.055834
2016-01  -0.210376    -0.166377  -0.228105     -0.142179
2016-02  -0.023293    -0.193176  -0.145274     -0.194628
2016-03   0.118375    -0.137474   0.004727     -0.205647
2016-04  -0.019062     0.071503  -0.106770     -0.335406
2016-05   0.004059     0.101511  -0.111275     -0.345244
2016-06  -0.002844    -0.017880  -0.152897     -0.293416,
'benchmark_volatility':          one_month  three_month  six_month  twelve_month
2015-01   0.385965          NaN        NaN           NaN
2015-02   0.231550          NaN        NaN           NaN
2015-03   0.215239     0.289107        NaN           NaN
2015-04   0.227623     0.222182        NaN           NaN
2015-05   0.392263     0.285261        NaN           NaN
2015-06   0.553131     0.412760   0.357441           NaN
2015-07   0.605976     0.523553   0.413022           NaN
2015-08   0.602886     0.579502   0.463818           NaN
2015-09   0.341508     0.528661   0.476164           NaN
2015-10   0.254009     0.435778   0.482086           NaN
2015-11   0.303270     0.303330   0.470614           NaN
2015-12   0.239572     0.264695   0.423372      0.392687
2016-01   0.510408     0.369562   0.400915      0.406872
2016-02   0.340911     0.381768   0.345499      0.410844
2016-03   0.228227     0.383871   0.329590      0.410629
2016-04   0.139199     0.241611   0.314981      0.406144
2016-05   0.186952     0.191895   0.299716      0.393849
2016-06        NaN     0.161722   0.310476      0.376342,
'beta':          one_month  three_month  six_month  twelve_month
2015-01   0.998299          NaN        NaN           NaN
2015-02   0.053142          NaN        NaN           NaN
2015-03   0.503765     0.740981        NaN           NaN
2015-04   0.908467     0.560545        NaN           NaN
2015-05   0.492564     0.607708        NaN           NaN
2015-06   0.079165     0.296824   0.433632           NaN
2015-07   0.267527     0.237490   0.293042           NaN
2015-08   0.019433     0.132218   0.228646           NaN
2015-09   0.332594     0.174039   0.226857           NaN
2015-10   0.444858     0.124459   0.193814           NaN
2015-11   0.318151     0.337919   0.172912           NaN
2015-12   0.308802     0.331235   0.207554      0.300930
2016-01   0.000000     0.116289   0.119401      0.209774
2016-02   0.109030     0.066582   0.169153      0.208931
2016-03   0.311315     0.068936   0.152924      0.203582
2016-04   0.280393     0.202790   0.137841      0.177735
2016-05   0.000000     0.213803   0.097306      0.151277
2016-06   0.000000     0.101655   0.071174      0.166119,
'information':          one_month  three_month  six_month  twelve_month
2015-01  -1.495108          NaN        NaN           NaN
2015-02  -3.972598          NaN        NaN           NaN
2015-03  -3.082445    -2.486074        NaN           NaN
2015-04   0.701547    -1.229573        NaN           NaN
2015-05  -3.945548    -1.467470        NaN           NaN
2015-06   2.432314     0.544099  -0.597592           NaN
2015-07   0.937668     0.651784  -0.077393           NaN
2015-08   1.459703     1.621478   0.474841           NaN
2015-09   0.533801     1.043315   0.816343           NaN
2015-10  -9.605852    -0.304668   0.214521           NaN
2015-11   1.428907    -0.930650   0.738407           NaN
2015-12  -3.164017    -2.435357  -0.038820     -0.303545
2016-01   5.502363     2.169026   0.865370      0.388355
2016-02   0.894252     2.087511   0.708246      0.564849
2016-03  -4.625734     1.721183   0.047084      0.509923
2016-04  -0.737979    -1.470819   0.817772      0.448668
2016-05  -0.346719    -2.255747   0.566012      0.656193
2016-06   0.000000    -0.515542   1.138215      0.400542,
'max_drawdown':          one_month  three_month  six_month  twelve_month
2015-01   0.169166          NaN        NaN           NaN
2015-02   0.015968          NaN        NaN           NaN
2015-03   0.042095     0.194898        NaN           NaN
2015-04   0.085223     0.085223        NaN           NaN
2015-05   0.069138     0.138984        NaN           NaN
2015-06   0.037826     0.138984   0.194898           NaN
2015-07   0.095884     0.130083   0.184065           NaN
2015-08   0.062300     0.179897   0.230787           NaN
2015-09   0.098159     0.159461   0.241441           NaN
2015-10   0.063613     0.116740   0.207919           NaN
2015-11   0.058594     0.116740   0.207919           NaN
2015-12   0.032458     0.085857   0.176780      0.257070
2016-01   0.000000     0.085857   0.116740      0.257070
```

```
 2016-02   0.024873     0.032458   0.116740      0.257070
 2016-03   0.036184     0.036184   0.085857      0.257070
 2016-04   0.032833     0.061436   0.085857      0.207919
 2016-05   0.000000     0.061436   0.061436      0.207919
 2016-06   0.003754     0.035534   0.064058      0.176780,
 'sharpe':          one_month  three_month  six_month  twelve_month
 2015-01  -1.231537          NaN        NaN           NaN
 2015-02  -3.863970          NaN        NaN           NaN
 2015-03   3.455035    -0.534996        NaN           NaN
 2015-04  14.509434     4.562859        NaN           NaN
 2015-05  -2.213395     2.666836        NaN           NaN
 2015-06   2.771223     2.490390   0.736012           NaN
 2015-07  -2.058534    -1.381017   0.651133           NaN
 2015-08  -2.857995    -1.273788   0.441451           NaN
 2015-09  -1.082495    -1.813080  -0.155587           NaN
 2015-10  -1.953666    -1.573303  -1.461285           NaN
 2015-11   2.878934    -0.287666  -0.844368           NaN
 2015-12  -1.212611     0.034408  -1.267645     -0.235983
 2016-01   0.000000     0.624247  -0.760145      0.043911
 2016-02  -0.469338    -0.894105  -0.437123      0.089959
 2016-03   1.485103     0.436854   0.182304     -0.049185
 2016-04  -4.412679    -0.460251   0.149685     -0.960658
 2016-05   0.000000    -0.497998  -0.701335     -0.756690
 2016-06   0.000000    -4.004414  -0.615471     -1.074621,
 'sortino':          one_month  three_month  six_month  twelve_month
 2015-01  -1.547878          NaN        NaN           NaN
 2015-02  -2.573161          NaN        NaN           NaN
 2015-03   6.793850    -0.509806        NaN           NaN
 2015-04  31.717828     9.242985        NaN           NaN
 2015-05  -1.411676     5.001742        NaN           NaN
 2015-06   5.781571     4.389607   1.004910           NaN
 2015-07  -3.993221    -1.452989   1.124913           NaN
 2015-08  -3.616613    -1.745249   0.777726           NaN
 2015-09  -0.647030    -1.576591  -0.232070           NaN
 2015-10  -3.097906    -1.234436  -1.366953           NaN
 2015-11   4.714085    -0.270438  -0.969428           NaN
 2015-12  -1.477629     0.056091  -1.301615     -0.316775
 2016-01   0.000000     0.912570  -0.624453      0.068803
 2016-02  -0.746075    -1.053971  -0.376419      0.142102
 2016-03   2.739812     0.621290   0.307398     -0.071815
 2016-04  -4.509301    -0.698224   0.228798     -0.921254
 2016-05   0.000000    -0.604989  -0.855342     -0.766371
 2016-06   0.000000    -3.077574  -0.728925     -1.049463}
```

In [35]:

```
gt.get_period_risks()['alpha'] # 获取分月计算的风险指标中的alpha
```

Out[35]:

	one_month	three_month	six_month	twelve_month
2015-01	-0.340114	NaN	NaN	NaN
2015-02	-0.285462	NaN	NaN	NaN
2015-03	-0.705164	-0.771542	NaN	NaN
2015-04	2.222741	-0.120284	NaN	NaN
2015-05	-0.683038	-0.435415	NaN	NaN
2015-06	0.761667	0.769600	0.005202	NaN
2015-07	-0.475122	-0.254308	0.148066	NaN
2015-08	-0.378048	-0.229071	0.173228	NaN
2015-09	-0.210559	-0.392439	0.042345	NaN
2015-10	-1.756624	-0.329792	-0.290167	NaN
2015-11	0.696051	-0.158699	-0.135661	NaN
2015-12	-0.356621	-0.268037	-0.241502	-0.077842
2016-01	-0.040000	0.167151	-0.099187	0.050571
2016-02	-0.026688	-0.058275	-0.033985	0.073793
2016-03	-0.488207	0.087901	0.033778	0.037806
2016-04	-0.242546	-0.120085	0.056010	-0.137978

	one_month	three_month	six_month	twelve_month
2016-05	-0.040000	-0.141177	-0.050841	-0.093103
2016-06	-0.649478	-0.185857	-0.033635	-0.149710

In [ ]:

```
```

祝大家使用愉快~~~

【教程】如何在JoinQuant 的回测及研究中发送邮件!

社区中很多朋友除了想要接受微信信息之外，还想收取邮件，现就为大家介绍一下如何在JoinQuant的回测及研究中发送邮件!

（注：记得请先开启邮箱的SMTP服务哦，下面以163邮箱为例）

163邮箱函数形式如下:

```
def send_163_email(subject,message):
    import smtplib
    from email.mime.text import MIMEText
    from email.header import Header

    '''
    记得请先开启邮箱的SMTP服务
    '''
    ## 发送邮件
    sender = '量化模拟盘<XX@163.com>' #发送的邮箱
    receiver = 'XX@163.com' #要接受的邮箱（注:测试中发送其他邮箱会提示错误）
    smtpserver = 'smtp.163.com'
    username = 'XX@163.com' #你的邮箱账号
    password = '**********' #你的邮箱密码

    msg = MIMEText(str(message),'plain','utf-8') #中文需参数‘utf-8'，单字节字符不需要
    msg['Subject'] = Header(subject, 'utf-8') #邮件主题
    msg['to'] = receiver
    msg['from'] = sender    #自己的邮件地址

    smtp = smtplib.SMTP()
    try :
        smtp.connect('smtp.163.com') # 链接
        smtp.login(username, password) # 登陆
        smtp.sendmail(sender, receiver, msg.as_string()) #发送
        print '邮件发送成功'
    except:
        print '邮件发送失败'
    smtp.quit() # 结束
```

```
df = get_price('000001.XSHE', start_date='2015-06-01', end_date='2015-06-05', frequency='daily', fields=['close','low'])
z = df['close']
send_163_email(subject='Hi, JoinQuant! ',message=z)
```

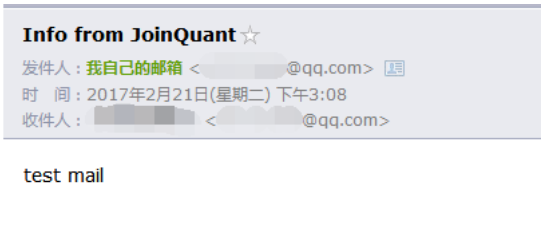
结果如下:


```
2015-06-01    15.90
2015-06-02    15.77
2015-06-03    15.83
2015-06-04    16.37
2015-06-05    16.30
Name: close, dtype: float64
```

由于QQ邮箱的smtp 服务用了ssl 认证，发往QQ邮箱函数形式如下:

```
def send_qq_email(subject,message):
    import smtplib
    from email.mime.text import MIMEText
    from email.header import Header
    ## 发送邮件
    sender = '量化模拟盘<12345678@qq.com>' #发送的邮箱
    receiver = '12345678@qq.com' #要接受的邮箱（注:测试中发送其他邮箱会提示错误）
    smtpserver = 'smtp.qq.com'
    username = '12345678@qq.com' #你的邮箱账号
    password = 'abcdefghijklmnop' #你的邮箱授权码。一个16位字符串

    msg = MIMEText(str(message),'plain','utf-8') #中文需参数‘utf-8’，单字节字符不需要
    msg['Subject'] = Header(subject, 'utf-8') #邮件主题
    msg['to'] = receiver
    msg['from'] = sender    #自己的邮件地址

    server = smtplib.SMTP_SSL('smtp.qq.com')
    try :
        #server.connect() # ssl无需这条
        server.login(username, password) # 登陆
        server.sendmail(sender, receiver, msg.as_string()) #发送
        print '邮件发送成功'
    except:
        print '邮件发送失败'
    server.quit() # 结束
```

```
send_qq_email(subject='Info from JoinQuant', message="test mail")
```



希望对你有用，祝好!

【教程】 PrettyTable介绍 - 让你的日志数据更美观

Python通过prettytable模块可以将输出内容如表格方式整齐的输出。

使用该模块，用户可以在日志中输出清晰的日志文档。

PrettyTable介绍¶

Python通过prettytable模块可以将输出内容如表格方式整齐的输出。

使用该模块，用户可以在日志中输出清晰的日志文档。

常用方法如下：¶

```
sortby - name of field to sort rows by
reversesort - True or False to sort in descending or ascending order
int_format - controls formatting of integer data
float_format - controls formatting of floating point data

add_row(row)
    """Add a row to the table

    Arguments:

    row - row of data, should be a list with as many elements as the table
    has fields"""

del_row(row_index)
    """Delete a row to the table

    Arguments:

    row_index - The index of the row you want to delete.  Indexing starts at 0."""

add_column(fieldname, column, align="c", valign="t")
    """Add a column to the table.

    Arguments:
```

```
    fieldname - name of the field to contain the new column of data
    column - column of data, should be a list with as many elements as the
    table has rows
    align - desired alignment for this column - "l" for left, "c" for centre and "r" for right
    valign - desired vertical alignment for new columns - "t" for top, "m" for middle and "b" for bottom"""

clear_rows()
    """Delete all rows from the table but keep the current field names"""

clear()
    """Delete all rows and field names from the table, maintaining nothing but styling options"""
```

示例如下：

In [1]:

```
from prettytable import *
```

In [75]:

```
x = PrettyTable(["City name", "Area", "Population", "Annual Rainfall"])
x.sortby = "Population"
x.reversesort = True
x.int_format["Area"] = "04d"
x.float_format = "6.1f"
x.align["City name"] = "l" # Left align city names
x.add_row(["Adelaide", 1295, 1158259, 600.5])
x.add_row(["Brisbane", 5905, 1857594, 1146.4])
x.add_row(["Darwin", 112, 120900, 1714.7])
x.add_row(["Hobart", 1357, 205556, 619.5])
x.add_row(["Sydney", 2058, 4336374, 1214.8])
x.add_row(["Melbourne", 1566, 3806092, 646.9])
x.add_row(["Perth", 5386, 1554769, 869.4])
print x
```

```
+-----------+-------+------------+-----------------+
| City name |  Area | Population | Annual Rainfall |
+-----------+-------+------------+-----------------+
| Sydney    | 2058d |  4336374   |     1214.8f     |
| Melbourne | 1566d |  3806092   |      646.9f     |
| Brisbane  | 5905d |  1857594   |     1146.4f     |
| Perth     | 5386d |  1554769   |      869.4f     |
| Adelaide  | 1295d |  1158259   |      600.5f     |
| Hobart    | 1357d |   205556   |      619.5f     |
| Darwin    | 0112d |   120900   |     1714.7f     |
+-----------+-------+------------+-----------------+
```

In [76]:

```
# 通过行添加
row = PrettyTable()
row.field_names = ["City name", "Area", "Population", "Annual Rainfall"]
row.add_row(["Adelaide",1295, 1158259, 600.5])
row.add_row(["Brisbane",5905, 1857594, 1146.4])
row.add_row(["Darwin", 112, 120900, 1714.7])
row.add_row(["Hobart", 1357, 205556, 619.5])
row.add_row(["Sydney", 2058, 4336374, 1214.8])
row.add_row(["Melbourne", 1566, 3806092, 646.9])
row.add_row(["Perth", 5386, 1554769, 869.4])
print row
```

```
+-----------+------+------------+-----------------+
| City name | Area | Population | Annual Rainfall |
+-----------+------+------------+-----------------+
|  Adelaide | 1295 |  1158259   |      600.5      |
|  Brisbane | 5905 |  1857594   |      1146.4     |
|   Darwin  | 112  |   120900   |      1714.7     |
|   Hobart  | 1357 |   205556   |      619.5      |
|   Sydney  | 2058 |  4336374   |      1214.8     |
| Melbourne | 1566 |  3806092   |      646.9      |
|   Perth   | 5386 |  1554769   |      869.4      |
+-----------+------+------------+-----------------+
```

In [77]:

```
# 通过列添加
col = PrettyTable()
col.add_column("City name",["Adelaide","Brisbane","Darwin","Hobart","Sydney","Melbourne","Perth"],align="l",valign="t")
col.add_column("Area", [1295, 5905, 112, 1357, 2058, 1566, 5386])
```

```
col.add_column("Population", [1158259, 1857594, 120900, 205556, 4336374, 3806092, 1554769])
col.add_column("Annual Rainfall",[600.5, 1146.4, 1714.7, 619.5, 1214.8, 646.9, 869.4])
print col
```

```
+-----------+------+------------+-----------------+
| City name | Area | Population | Annual Rainfall |
+-----------+------+------------+-----------------+
| Adelaide  | 1295 |  1158259   |      600.5      |
| Brisbane  | 5905 |  1857594   |      1146.4     |
| Darwin    | 112  |   120900   |      1714.7     |
| Hobart    | 1357 |   205556   |      619.5      |
| Sydney    | 2058 |  4336374   |      1214.8     |
| Melbourne | 1566 |  3806092   |      646.9      |
| Perth     | 5386 |  1554769   |      869.4      |
+-----------+------+------------+-----------------+
```

In [87]:

```
# 行列混输
mix = PrettyTable()
mix.field_names = ["City name", "Area"]
mix.add_row(["Adelaide",1295])
mix.add_row(["Brisbane",5905])
mix.add_row(["Darwin", 112])
mix.add_row(["Hobart", 1357])
mix.add_row(["Sydney", 2058])
mix.add_row(["Melbourne", 1566])
mix.add_row(["Perth", 5386])
mix.add_column("Population", [1158259, 1857594, 120900, 205556, 4336374, 3806092, 1554769])
mix.add_column("Annual Rainfall",[600.5, 1146.4, 1714.7, 619.5, 1214.8, 646.9, 869.4])
print mix
```

```
+-----------+------+------------+-----------------+
| City name | Area | Population | Annual Rainfall |
+-----------+------+------------+-----------------+
|  Adelaide | 1295 |  1158259   |      600.5      |
|  Brisbane | 5905 |  1857594   |      1146.4     |
|   Darwin  | 112  |   120900   |      1714.7     |
|   Hobart  | 1357 |   205556   |      619.5      |
|   Sydney  | 2058 |  4336374   |      1214.8     |
| Melbourne | 1566 |  3806092   |      646.9      |
|   Perth   | 5386 |  1554769   |      869.4      |
+-----------+------+------------+-----------------+
```

In [81]:

```
# 清空内容
mix.clear()
print mix
```

```
++
||
++
++
```

In [88]:

```
# 清空行内容
mix.clear_rows()
print mix
```

```
+-----------+------+------------+-----------------+
| City name | Area | Population | Annual Rainfall |
+-----------+------+------------+-----------------+
+-----------+------+------------+-----------------+
```

In [13]:

```
# 加入转义字符
t = PrettyTable(['Field 1', 'Field 2'])
t.add_row(['value 1', 'value2\nsecond line'])
t.add_row(['value 3\n\nother line', 'value4\n\n\nvalue5'])
print t.get_string(hrules=ALL)
```

```
+------------+-------------+
|  Field 1   |   Field 2   |
+------------+-------------+
|  value 1   |    value2   |
```

```
|            | second line |
+------------+-------------+
|  value 3   |    value4   |
|            |             |
| other line |             |
|            |    value5   |
+------------+-------------+
```

In [14]:

```
# 测试多种语言的支持情况
x = PrettyTable(["Kanji", "Hiragana", "English"])
x.add_row(["神戸", "こうべ", "Kobe"])
x.add_row(["京都", "きょうと", "Kyoto"])
x.add_row(["長崎", "ながさき", "Nagasaki"])
x.add_row(["名古屋", "なごや", "Nagoya"])
x.add_row(["大阪", "おおさか", "Osaka"])
x.add_row(["札幌", "さっぽろ", "Sapporo"])
x.add_row(["東京", "とうきょう", "Tokyo"])
x.add_row(["横浜", "よこはま", "Yokohama"])
print x
```

```
+--------+------------+----------+
| Kanji  |  Hiragana  | English  |
+--------+------------+----------+
|  神戸  |   こうべ   |   Kobe   |
|  京都  |  きょうと  |  Kyoto   |
|  長崎  |  ながさき  | Nagasaki |
| 名古屋 |   なごや   |  Nagoya  |
|  大阪  |  おおさか  |  Osaka   |
|  札幌  |  さっぽろ  | Sapporo  |
|  東京  | とうきょう |  Tokyo   |
|  横浜  |  よこはま  | Yokohama |
+--------+------------+----------+
```

In [ ]:

```

```

【翻译搬运】SciPy-Python 科学算法库

SciPy库提供了大量有用的函数和类，用来解决各种专业领域的问题。本文翻译自Jupyter nbviewer中的第三讲。首先，介绍了一些特殊函数，如贝塞尔函数，这对物理学问题的计算提供了方便；之后是各种数值积分问题，常微分方程求解问题以及傅里叶变换，这些也可以描述并求解一些诸如复摆运动、阻尼震荡等复杂的物理过程；同时，该库还可以高效处理线性代数问题，如矩阵的运算、特征值和特征向量以及稀疏矩阵的存储和运算；最优化问题，即寻找函数极值和零点的问题，和插值问题，即用多项式拟合曲线的问题，在SciPy库中也可以找到相应的函数；最后介绍了统计上的应用，包括各种分布的密度函数、分布函数及其图像绘制，以及一些统计检验问题。

作者：J.R. Johansson (邮箱：jrjohansson@gmail.com)

译者：一路向上

最新版本的用法介绍见网站http://github.com/jrjohansson/scientific-python-lectures. 其他相关介绍见http://jrjohansson.github.io.

In [2]:

```
# 调用绘图函数
%matplotlib inline
import matplotlib.pyplot as plt
from IPython.display import Image
```

简介¶

SciPy框架建立于低一级的NumPy框架的多维数组之上，并且提供了大量的高级的科学算法。一些SciPy的应用如下：

1. 特殊函数 (scipy.special)
2. 积分 (scipy.integrate)
3. 优化 (scipy.optimize)
4. 插值 (scipy.interpolate)
5. 傅里叶变换 (scipy.fftpack)
6. 信号处理 (scipy.signal)
7. 线型代数 (scipy.linalg)
8. 稀疏特征值问题 (scipy.sparse)
9. 统计 (scipy.stats)
10. 多维图像处理 (scipy.ndimage)
11. 文件输入输出 (scipy.io)

这些模块提供了大量的函数和类，可以用来解决各自领域的问题。 在本节中我们将看到如何使用这些子函数包。 我们首先导入scipy程序包：

In [3]:

```
from scipy import *
```

如果我们只需要用scipy框架中的一部分，我们也可以选择性的导入。例如，只导入线性代数函数包la，我们可以：

In [4]:

```
import scipy.linalg as la
```

特殊函数¶

大量的数学函数对许多物理问题的计算是非常重要的。SciPy提供了一系列非常广泛的特殊函数。详见 http://docs.scipy.org/doc/scipy/reference/special.html#module-scipy.special.

为了说明指定的特殊函数的用法，我们先看一下贝塞尔函数的使用细节：

In [5]:

```
# scipy.special 包含了大量的贝塞尔函数
# 我们使用函数jn 和 yn，它们是第一类和第二类实值贝塞尔函数
# jn_zeros 和 yn_zeros 代表了jn 和 yn 函数的零点
#
from scipy.special import jn, yn, jn_zeros, yn_zeros
```

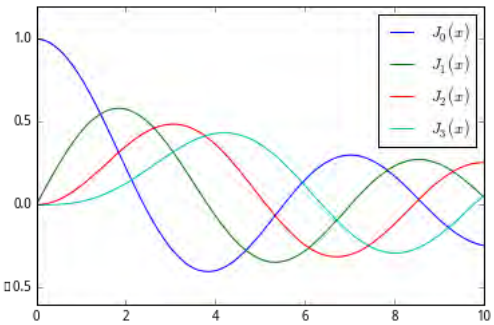
In [6]:

```
n = 0
x = 0.0

# 第一类贝塞尔函数
print "J_%d(%f) = %f" % (n, x, jn(n, x))

x = 1.0
# 第二类贝塞尔函数
print "Y_%d(%f) = %f" % (n, x, yn(n, x))
```

```
J_0(0.000000) = 1.000000
Y_0(1.000000) = 0.088257
```

In [7]:

```
x = linspace(0, 10, 100)

fig, ax = plt.subplots()
for n in range(4):
    ax.plot(x, jn(n, x), label=r"$J_%d(x)$" % n)
ax.legend();
```



In [8]:

```
# 贝塞尔函数的零点
n = 0
m = 4 # 计算的零点的个数
jn_zeros(n, m)
```

Out[8]:

```
array([  2.40482556,   5.52007811,   8.65372791,  11.79153444])
```

积分¶

数值积分：求积公式¶

对f(x)从a到b的积分叫做数值积分，也叫简单积分。SciPy提供了一系列计算不同数值积分的函数，包括quad,dblquad和tplquad，分别包含二重和三重积分。

In [9]:

```
from scipy.integrate import quad, dblquad, tplquad
```

quad有一系列的可供选择的参数，可以用来调节函数的各种行为（输入help(quad)获取更多信息）。

其基本用途如下：

In [10]:

```
# 定义一个简单函数作为被积函数
def f(x):
    return x
```

In [11]:

```
x_lower = 0 # x积分下限
x_upper = 1 # x积分上限

val, abserr = quad(f, x_lower, x_upper)

print "integral value =", val, ", absolute error =", abserr
```

```
integral value = 0.5 , absolute error = 5.55111512313e-15
```

如果我们需要添加更多对于被积函数的参数，可以使用args关键字参数：

In [12]:

```
def integrand(x, n):
    """
    Bessel function of first kind and order n.
    """
    return jn(n, x)


x_lower = 0  # x积分下限
x_upper = 10 # x积分上限

val, abserr = quad(integrand, x_lower, x_upper, args=(3,))

print val, abserr
```

```
0.736675137081 9.3891268825e-13
```

对于简单函数而言，对于被积函数，我们可以用λ函数（无名称的函数）来代替清晰定义的函数：

In [13]:

```
val, abserr = quad(lambda x: exp(-x ** 2), -Inf, Inf)

print "numerical  =", val, abserr

analytical = sqrt(pi)
print "analytical =", analytical
```

```
numerical  = 1.77245385091 1.42026367809e-08
analytical = 1.77245385091
```

如上所示，我们可以用'Inf'或者'-Inf'作为积分上下限。高维积分用法相同：

In [14]:

```
def integrand(x, y):
    return exp(-x**2-y**2)

x_lower = 0
x_upper = 10
y_lower = 0
y_upper = 10

val, abserr = dblquad(integrand, x_lower, x_upper, lambda x : y_lower, lambda x: y_upper)

print val, abserr
```

```
0.785398163397 1.63822994214e-13
```

注意，我们需要用λ函数对于y积分的极限，因为它们可以看做是x的函数。

常微分方程（ODE）¶

SciPy提供了两种不同的方法来解决常微分方程：函数odeint的API，和函数类ode面向对象的API。通常odeint比较容易上手，但是ode函数类能够更好的控制函数。

这里我们使用odeint函数，如需了解更多ode函数类的信息，请输入help(ode)。它和odeint很像，但却是面向对象的函数。

使用odeint之前，首先从scipy.integrate中调用它：

In [15]:

```
from scipy.integrate import odeint, ode
```

常微分方程系通常写作其一般形式：

```
y' = f(y, t)
```

其中

```
y = [y1(t), y2(t), ..., yn(t)]
```

f的微分是yi(t)。为了解决常微分方程，我们需要知道函数f和初始条件y(0).

高阶微分方程可以通过引进新的变量作为中间变量。

当我们定义了Python函数f和数组y_0(f和y(0)都是数学函数），我们调用odeint函数：

```
y_t = odeint(f, y_0, t)
```

t是解决ODE问题需要的时间坐标数组，y_t是对于给定点在时间t的一行数组，每一列代表在给定时间t所对应的一个解y_i(t)。我们下面将会看到如何设置f和y_0.

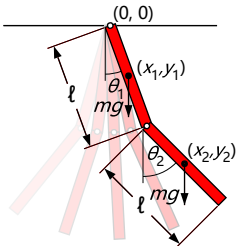
例：复摆¶

我们考虑一个物理问题：复摆。定义详见http://en.wikipedia.org/wiki/Double_pendulum.

In [16]:

```
Image(url='http://upload.wikimedia.org/wikipedia/commons/c/c9/Double-compound-pendulum-dimensioned.svg')
```

Out[16]:



θ1 = 6ml2 2pθ1 − 3cos(θ1 − θ2)pθ2 16 − 9cos2(θ1 − θ2) ⋅ θ2 = 6ml2 8pθ1 − 3cos(θ1 − θ2)pθ1 16 − 9cos2(θ1 − θ2) ⋅ pθ1 = − 12ml2 [⋅ θ1 ⋅ θ2 sin(θ1 − θ2) + 3g

lsinθ1] ⋅ pθ2 = − 12ml2 [− ⋅ θ1 ⋅ θ2 sin(θ1 − θ2) + gl sinθ2]

$$\dot{\theta}_1 = \frac{6}{ml^2}\frac{2p_{\theta_1} - 3cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9cos^2(\theta_1 - \theta_2)}$$
$$\dot{\theta}_2 = \frac{6}{ml^2}\frac{8p_{\theta_1} - 3cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9cos^2(\theta_1 - \theta_2)}$$
$$\dot{p_{\theta_1}} = -\frac{1}{2}ml^2\left[\dot{\theta}_1\dot{\theta}_2 sin(\theta_1 - \theta_2) + 3\frac{g}{l}sin\theta_1\right]$$
$$\dot{p_{\theta_2}} = -\frac{1}{2}ml^2\left[-\dot{\theta}_1\dot{\theta}_2 sin(\theta_1 - \theta_2) + \frac{g}{l}sin\theta_2\right]$$

为了让Python代码看起来更简洁，我们引进新的变量，并规定： = [θ1, θ2, pθ1, pθ2]

$$x = [\theta_1, \theta_2, p_{\theta_1}, p_{\theta_2}]$$

$x_1 = 6ml^2 2x_3 - 3\cos(x_1 - x_2)x_4 16 - 9\cos^2(x_1 - x_2)\cdot x_2 = 6ml^2 8x_4 - 3\cos(x_1 - x_2)x_3 16 - 9\cos^2(x_1 - x_2)\cdot x_3 = -12ml^2[\cdot x_1 \cdot x_2 \sin(x_1 - x_2) + 3g/\sin x_1]\cdot x_4 = -12ml^2[\cdot x_1 \cdot x_2 \sin(x_1 - x_2) + g/\sin x_2]$

$$\dot{x_1} = \frac{6}{ml^2}\frac{2x_3 - 3cos(x_1 - x_2)x_4}{16 - 9cos^2(x_1 - x_2)}$$
$$\dot{x_2} = \frac{6}{ml^2}\frac{8x_4 - 3cos(x_1 - x_2)x_3}{16 - 9cos^2(x_1 - x_2)}$$
$$\dot{x_3} = -\frac{1}{2}ml^2\left[\dot{x_1}\dot{x_2}sin(x_1 - x_2) + 3\frac{g}{l}sinx_1\right]$$
$$\dot{x_4} = -\frac{1}{2}ml^2\left[\dot{x_1}\dot{x_2}sin(x_1 - x_2) + \frac{g}{l}sinx_2\right]$$

In [17]:

```
g = 9.82
L = 0.5
m = 0.1

def dx(x, t):
    """
    The right-hand side of the pendulum ODE
    """
    x1, x2, x3, x4 = x[0], x[1], x[2], x[3]

    dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * cos(x1-x2) * x4)/(16 - 9 * cos(x1-x2)**2)
    dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * cos(x1-x2) * x3)/(16 - 9 * cos(x1-x2)**2)
    dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * sin(x1-x2) + 3 * (g/L) * sin(x1))
    dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * sin(x1-x2) + (g/L) * sin(x2))

    return [dx1, dx2, dx3, dx4]
```

In [18]:

```
# 确定初始状态
x0 = [pi/4, pi/2, 0, 0]
```

In [19]:

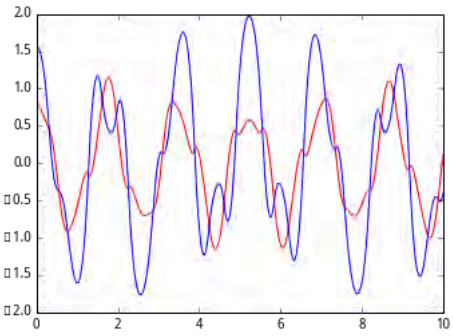
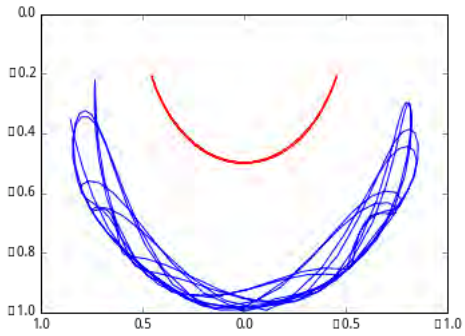
```
# 设定时间：从0s - 10s
t = linspace(0, 10, 250)
```

In [20]:

```
# 解常微分方程
x = odeint(dx, x0, t)
```

In [21]:

```
# 绘制角度关于时间的函数图像

fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].plot(t, x[:, 0], 'r', label="theta1")
axes[0].plot(t, x[:, 1], 'b', label="theta2")


x1 = + L * sin(x[:, 0])
y1 = - L * cos(x[:, 0])

x2 = x1 + L * sin(x[:, 1])
y2 = y1 - L * cos(x[:, 1])

axes[1].plot(x1, y1, 'r', label="pendulum1")
axes[1].plot(x2, y2, 'b', label="pendulum2")
axes[1].set_ylim([-1, 0])
axes[1].set_xlim([1, -1]);
```

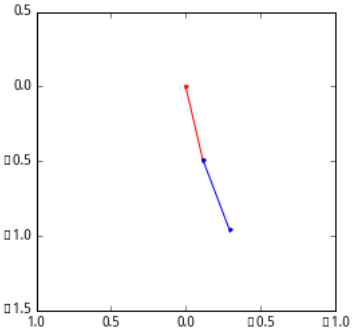
在matplotlib函数的应用中，会介绍如何绘制复摆运动的动图。

In [22]:

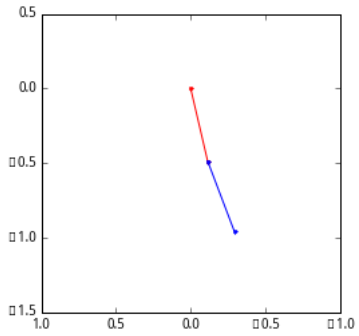
```
from IPython.display import display, clear_output
import time
```

In [23]:

```
fig, ax = plt.subplots(figsize=(4,4))

for t_idx, tt in enumerate(t[:200]):

    x1 = + L * sin(x[t_idx, 0])
    y1 = - L * cos(x[t_idx, 0])

    x2 = x1 + L * sin(x[t_idx, 1])
    y2 = y1 - L * cos(x[t_idx, 1])

    ax.cla()
    ax.plot([0, x1], [0, y1], 'r.-')
    ax.plot([x1, x2], [y1, y2], 'b.-')
    ax.set_ylim([-1.5, 0.5])
    ax.set_xlim([1, -1])

    clear_output()
    display(fig)

    time.sleep(0.1)
```



```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-23-469b60f59b8a> in <module>()
     18     display(fig)
     19
---> 20     time.sleep(0.1)

KeyboardInterrupt:
```



ps:这里的结果不是报错，是因为最后一行代码是每0.1s更新一次状态，为了后面的函数能够正常运行，我把它停掉了。

例： 阻尼谐波振荡器¶

常微分方程问题对计算物理非常重要，下面我们来看另一个例子：阻尼谐波振荡器。关于概念的解释详见 http://en.wikipedia.org/wiki/Damping.

阻尼谐波振荡器的方程为：

$$^{2}xdt^{2} + 2\zeta\omega_{0}^{dxdt} + \omega_{20}x = 0$$

在这个例子中，我们将为常微分方程等号右边的函数添加额外的参数，而不是像前面的例子那样使用全局变量。作为等号右边函数的额外参数的结果，我们需要将一个关键字参数args传递给odeint函数：

In [24]:

```
def dy(y, t, zeta, w0):
    """
    The right-hand side of the damped oscillator ODE
    """
    x, p = y[0], y[1]

    dx = p
    dp = -2 * zeta * w0 * p - w0**2 * x

    return [dx, dp]
```

In [25]:

```
# 初始状态:
y0 = [1.0, 0.0]
```

In [26]:

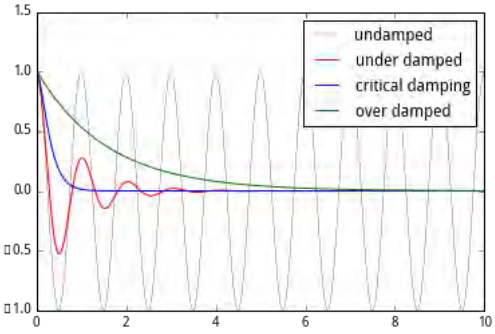
```
# 时间轴
t = linspace(0, 10, 1000)
w0 = 2*pi*1.0
```

In [27]:

```
# 在三种不同阻尼比的情况下解常微分方程

y1 = odeint(dy, y0, t, args=(0.0, w0)) # 无阻尼
y2 = odeint(dy, y0, t, args=(0.2, w0)) # 阻尼
y3 = odeint(dy, y0, t, args=(1.0, w0)) # 临界阻尼
y4 = odeint(dy, y0, t, args=(5.0, w0)) # 过阻尼
```

In [28]:

```
fig, ax = plt.subplots()
ax.plot(t, y1[:,0], 'k', label="undamped", linewidth=0.25)
ax.plot(t, y2[:,0], 'r', label="under damped")
ax.plot(t, y3[:,0], 'b', label=r"critical damping")
ax.plot(t, y4[:,0], 'g', label="over damped")
ax.legend();
```



傅里叶变换¶

傅里叶变换是计算物理中的一种通用工具，它在不同文章中都会反复出现。SciPy提供能够从NetLib中接入经典FFTPACK库的函数，它是由FORTRAN语言编写的一个行之有效的FFT库。SciPy API有一些额外的便利功能，但总的来说，API和原来的FORTRAN库密切相关。

为了调用fftpack，请输入：

In [29]:

```
from numpy.fft import fftfreq
from scipy.fftpack import *
```

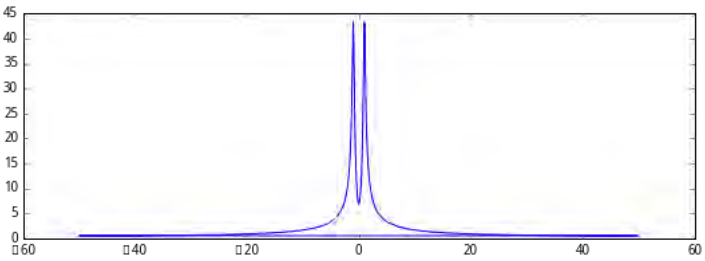
为演示如何用SciPy做一个快速傅里叶变换，让我们来看看用FFT如何解决之前讨论的阻尼震荡问题：

In [30]:

```
N = len(t)
dt = t[1]-t[0]

# 计算快速傅里叶变换
# y2 是前一节中阻尼震荡的解
F = fft(y2[:,0])

# 计算频率
w = fftfreq(N, dt)
```

In [31]:

```
fig, ax = plt.subplots(figsize=(9,3))
ax.plot(w, abs(F));
```



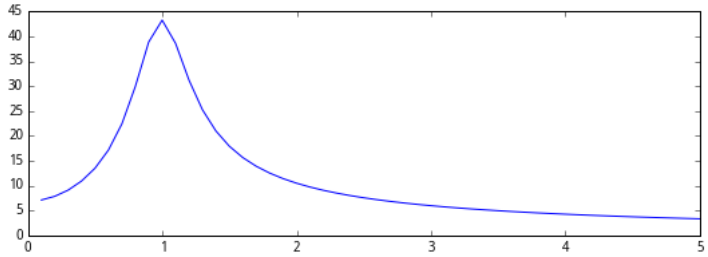
由于信号是实数，所以谱线图是对称的。我们因此只需要画出对应正频率的部分。为了提取w和F的部分，我们可以运用NumPy库：

In [32]:

```
indices = where(w > 0) # 只选择对应正频率的指数
w_pos = w[indices]
F_pos = F[indices]
```

In [33]:

```
fig, ax = plt.subplots(figsize=(9,3))
ax.plot(w_pos, abs(F_pos))
ax.set_xlim(0, 5);
```



和预期一样，我们看到谱线图在1处达到最高点，这正是在阻尼震荡这个例子中所采用的频率。

线性代数¶

线性代数部分包含了许多矩阵函数，包括线性方程的解，特征值的解，矩阵函数（例如矩阵指数函数），许多不同的分解（SVD，LU，cholesky）等等。详见：http://docs.scipy.org/doc/scipy/reference/linalg.html.

下面我们看看如何使用这些函数：

线性方程组¶

线性方程组的矩阵形式：

x = b

$$Ax = b$$

A是矩阵，x,b是向量。可以求解如下：

In [34]:

```
from scipy.linalg import *
```

In [35]:

```
A = array([[1,2,8], [3,7,6], [2,5,3]])
b = array([1,2,3])
```

In [36]:

```
x = solve(A, b)

x
```

Out[36]:

```
array([-14.6,   6.2,   0.4])
```

In [37]:

```
dot(A, x) - b
```

Out[37]:

```
array([  0.00000000e+00,   5.32907052e-15,   2.66453526e-15])
```

我们也可以：

X = B

$$AX = B$$

这里A,B,X都是矩阵：

In [38]:

```
A = rand(3,3)
B = rand(3,3)
```

In [39]:

```
X = solve(A, B)
```

In [40]:

```
X
```

Out[40]:

```
array([[ 1.02166987,  1.46979092,  0.64004897],
       [-2.1813796 ,  0.2648731 , -3.15222549],
       [ 1.54669347, -0.26698476,  1.98541742]])
```

In [41]:

```
# 检验
norm(dot(A, X) - B)
```

Out[41]:

```
4.677452743560217e-16
```

特征值和特征向量¶

矩阵A的特征值问题是：

$v_n = \lambda_n v_n$

In [42]:

```
evals = eigvals(A)
```

In [43]:

```
evals
```

Out[43]:

```
array([ 1.31586129+0.j,  0.58369033+0.j, -0.07474674+0.j])
```

In [44]:

```
evals, evecs = eig(A)
```

In [45]:

```
evals
```

Out[45]:

```
array([ 1.31586129+0.j,  0.58369033+0.j, -0.07474674+0.j])
```

In [46]:

```
evecs
```

Out[46]:

```
array([[-0.19090777, -0.88338693, -0.07776663],
       [-0.66739666,  0.11577427, -0.87511861],
       [-0.71981659,  0.45411876,  0.47761885]])
```

第n个特征值（储存在evals[n]中）对应的特征向量是evecs的第n列，也就是evecs[:,n]. 为了验证它，我们尝试把矩阵和特征向量相乘，并与特征向量和特征值的乘积做比较：

In [47]:

```
n = 1

norm(dot(A, evecs[:,n]) - evals[n] * evecs[:,n])
```

Out[47]:

```
2.1677797545656835e-16
```

还有许多其他的特殊的本征解，如埃尔米特矩阵（用eigh实现）

矩阵运算¶

In [48]:

```
# 矩阵的逆
inv(A)
```

Out[48]:

```
array([[ 1.90665827, -1.01212202,  0.6342907 ],
       [ 2.88091143, -9.0748278 ,  8.35451034],
       [-2.08791043,  5.44756731, -3.73714523]])
```

In [49]:

```
# 行列式
det(A)
```

Out[49]:

```
-0.05740964244539486
```

In [50]:

```
# 各阶范数
norm(A, ord=2), norm(A, ord=Inf)
```

Out[50]:

```
(1.4023823782071045, 1.5259518770408467)
```

稀疏矩阵¶

稀疏矩阵在处理巨型系统的数值模拟时非常有用，如果描述该问题的矩阵或向量大部分的元素为0。Scipy对于稀疏矩阵有很多处理方法，包括基础的线性代数处理（包括解方程，计算特征值等等）

许多方法都能有效存储稀疏矩阵，一些常用的方法包括坐标形式（COO）,列表的列表形式（LIL），压缩稀疏列（CSC）和压缩稀疏行（CSR）。每种方法都有优势和不足。大多数的计算算法（解方程，矩阵和矩阵相乘等等）都能用CSR或者CSC形式处理，但是它们并不直观，也不太容易进行初始化。所以通常来说，稀疏矩阵采用COO或者LIL进行初始化（我们可以在稀疏矩阵中有效添加元素），然后再转换为CSC或者CSR并进行计算。

更多关于稀疏矩阵的信息，详见： http://en.wikipedia.org/wiki/Sparse_matrix.

当我们创建了一个稀疏矩阵，我们要选择其存储形式，如：

In [51]:

```
from scipy.sparse import *
```

In [52]:

```
# 稠密矩阵
M = array([[1,0,0,0], [0,3,0,0], [0,1,1,0], [1,0,0,1]]); M
```

Out[52]:

```
array([[1, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 1, 1, 0],
       [1, 0, 0, 1]])
```

In [53]:

```
# 把稠密矩阵变为稀疏矩阵
A = csr_matrix(M); A
```

Out[53]:

```
<4x4 sparse matrix of type '<type 'numpy.int64'>'
        with 6 stored elements in Compressed Sparse Row format>
```

In [54]:

```
# 把稀疏矩阵变为稠密矩阵
A.todense()
```

Out[54]:

```
matrix([[1, 0, 0, 0],
        [0, 3, 0, 0],
```

```
        [0, 1, 1, 0],
        [1, 0, 0, 1]])
```

创建稀疏矩阵更有效的方法是，建立一个空矩阵，并用所在矩阵的位置填充（避免创建大的稠密矩阵）：

In [55]:

```
A = lil_matrix((4,4)) # 建立一个4×4的空矩阵
A[0,0] = 1
A[1,1] = 3
A[2,2] = A[2,1] = 1
A[3,3] = A[3,0] = 1
A
```

Out[55]:

```
<4x4 sparse matrix of type '<type 'numpy.float64'>'
        with 6 stored elements in LInked List format>
```

In [56]:

```
A.todense()
```

Out[56]:

```
matrix([[ 1.,  0.,  0.,  0.],
        [ 0.,  3.,  0.,  0.],
        [ 0.,  1.,  1.,  0.],
        [ 1.,  0.,  0.,  1.]])
```

In [57]:

```
A
```

Out[57]:

```
<4x4 sparse matrix of type '<type 'numpy.float64'>'
        with 6 stored elements in LInked List format>
```

In [58]:

```
A = csr_matrix(A); A
```

Out[58]:

```
<4x4 sparse matrix of type '<type 'numpy.float64'>'
        with 6 stored elements in Compressed Sparse Row format>
```

In [59]:

```
A = csc_matrix(A); A
```

Out[59]:

```
<4x4 sparse matrix of type '<type 'numpy.float64'>'
        with 6 stored elements in Compressed Sparse Column format>
```

我们可以像计算稠密矩阵一样，计算稀疏矩阵：

In [60]:

```
A.todense()
```

Out[60]:

```
matrix([[ 1.,  0.,  0.,  0.],
        [ 0.,  3.,  0.,  0.],
        [ 0.,  1.,  1.,  0.],
        [ 1.,  0.,  0.,  1.]])
```

In [61]:

```
(A * A).todense()
```

Out[61]:

```
matrix([[ 1.,  0.,  0.,  0.],
        [ 0.,  9.,  0.,  0.],
        [ 0.,  4.,  1.,  0.],
        [ 2.,  0.,  0.,  1.]])
```

In [62]:

```
A.todense()
```

Out[62]:

```
matrix([[ 1.,  0.,  0.,  0.],
        [ 0.,  3.,  0.,  0.],
        [ 0.,  1.,  1.,  0.],
        [ 1.,  0.,  0.,  1.]])
```

In [63]:

```
A.dot(A).todense()
```

Out[63]:

```
matrix([[ 1.,  0.,  0.,  0.],
        [ 0.,  9.,  0.,  0.],
        [ 0.,  4.,  1.,  0.],
        [ 2.,  0.,  0.,  1.]])
```

In [64]:

```
v = array([1,2,3,4])[:,newaxis]; v
```

Out[64]:

```
array([[1],
       [2],
       [3],
       [4]])
```

In [65]:

```
# 稀疏矩阵 - 稠密向量的乘积
A * v
```

Out[65]:

```
array([[ 1.],
       [ 6.],
       [ 5.],
       [ 5.]])
```

In [66]:

```
# 稠密矩阵得到同样的结果 - 稠密向量的乘积
A.todense() * v
```

Out[66]:

```
matrix([[ 1.],
        [ 6.],
        [ 5.],
        [ 5.]])
```

最优化¶

最优化问题（寻找函数的最大值或者最小值）是数学的一大领域，复杂函数的最优化问题，或者多变量的最优化问题可能会非常复杂。下面我们看一些很简单的例子，更多详细的对于使用SciPy处理最优化问题的介绍，请见：http://scipy-lectures.github.com/advanced/mathematical_optimization/index.html.

首先调用optimize:

In [67]:

```
from scipy import optimize
```

寻找最小值¶

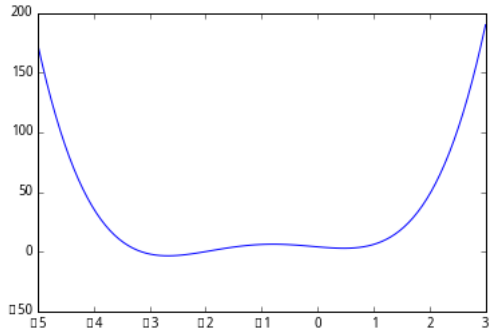
首先我们看如何寻找单变量的简单函数的最小值：

In [68]:

```
def f(x):
    return 4*x**3 + (x-2)**2 + x**4
```

In [69]:

```
fig, ax  = plt.subplots()
x = linspace(-5, 3, 100)
ax.plot(x, f(x));
```



我们可以用fmin_bfgs寻找函数的最小值：

In [70]:

```
x_min = optimize.fmin_bfgs(f, -2)
x_min
```

```
Optimization terminated successfully.
         Current function value: -3.506641
         Iterations: 6
         Function evaluations: 30
         Gradient evaluations: 10
```

Out[70]:

```
array([-2.67298167])
```

In [71]:

```
optimize.fmin_bfgs(f, 0.5)
```

```
Optimization terminated successfully.
         Current function value: 2.804988
         Iterations: 3
         Function evaluations: 15
         Gradient evaluations: 5
```

Out[71]:

```
array([ 0.46961745])
```

我们还可以用brent或者fminbound函数，它们采用了不太一样的语法和算法。

In [72]:

```
optimize.brent(f)
```

Out[72]:

```
0.46961743402759754
```

In [73]:

```
optimize.fminbound(f, -4, 2)
```

Out[73]:

```
-2.6729822917513886
```

寻找函数的解¶

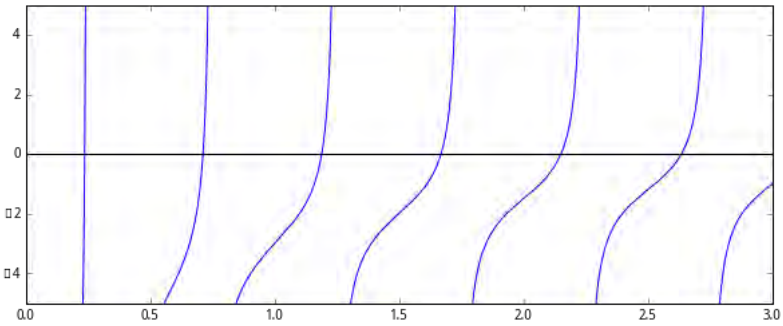
为了寻找函数f(x) = 0 的解，我们可以用fsolve函数，它需要一个初始的猜测值：

In [74]:

```
omega_c = 3.0
def f(omega):
    # 超越方程：低Q值的SQUID微波谐振器的共振频率
    return tan(2*pi*omega) - omega_c/omega
```

In [75]:

```
#import matplotlib
#matplotlib.rcParams['axes.unicode_minus']=False
fig, ax  = plt.subplots(figsize=(10,4))
x = linspace(0, 3, 1000)
y = f(x)
mask = where(abs(y) > 50)
x[mask] = y[mask] = NaN # 当函数变换符号时避免产生垂直线
ax.plot(x, y)
ax.plot([0, 3], [0, 0], 'k')
ax.set_ylim(-5,5);
```



In [76]:

```
optimize.fsolve(f, 0.1)
```

Out[76]:

```
array([ 0.23743014])
```

In [77]:

```
optimize.fsolve(f, 0.6)
```

Out[77]:

```
array([ 0.71286972])
```

In [78]:

```
optimize.fsolve(f, 1.1)
```

Out[78]:

```
array([ 1.18990285])
```

插值¶

插值在SciPy中能够很容易和方便的实现：interpid函数，当描述X和Y的数据时，返回值是被称之为x的任意值（X的范围）的函数，同时返回相应的插值y：

In [79]:

```
from scipy.interpolate import *
```

In [80]:

```
def f(x):
    return sin(x)
```

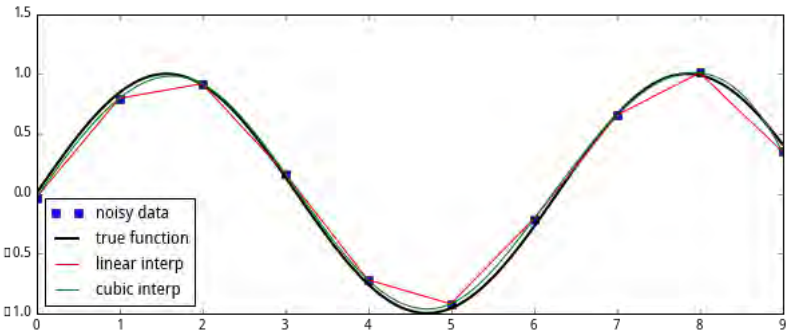
In [81]:

```
n = arange(0, 10)
x = linspace(0, 9, 100)
```

```
y_meas = f(n) + 0.1 * randn(len(n)) # 噪声模拟测量
y_real = f(x)

linear_interpolation = interp1d(n, y_meas)
y_interp1 = linear_interpolation(x)

cubic_interpolation = interp1d(n, y_meas, kind='cubic')
y_interp2 = cubic_interpolation(x)
```

In [82]:

```
fig, ax = plt.subplots(figsize=(10,4))
ax.plot(n, y_meas, 'bs', label='noisy data')
ax.plot(x, y_real, 'k', lw=2, label='true function')
ax.plot(x, y_interp1, 'r', label='linear interp')
ax.plot(x, y_interp2, 'g', label='cubic interp')
ax.legend(loc=3);
```



统计¶

scipy.stats包含了许多统计分布，统计函数和检验。完整的功能请见：http://docs.scipy.org/doc/scipy/reference/stats.html.

还有一个非常强大的统计模型的包叫做statsmodels，详见： http://statsmodels.sourceforge.net.

In [83]:

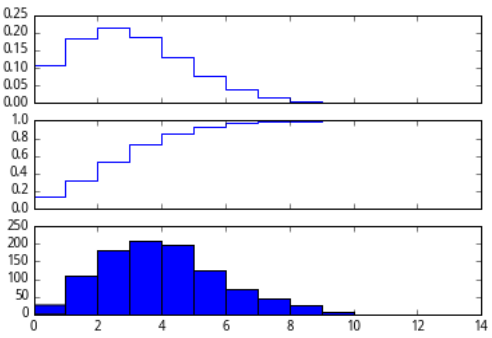
```
from scipy import stats
```

In [84]:

```
# 创建一个（离散）符合泊松分布的随机变量

X = stats.poisson(3.5) # 对于n = 3.5光子相干态的光子数分布
```

In [85]:

```
n = arange(0,15)

fig, axes = plt.subplots(3,1, sharex=True)

# 绘制概率质量函数 (PMF)
axes[0].step(n, X.pmf(n))

# 绘制累计分布函数 (CDF)
axes[1].step(n, X.cdf(n))

# 绘制随机变量 X 的1000次随机实现的直方图
axes[2].hist(X.rvs(size=1000));
```

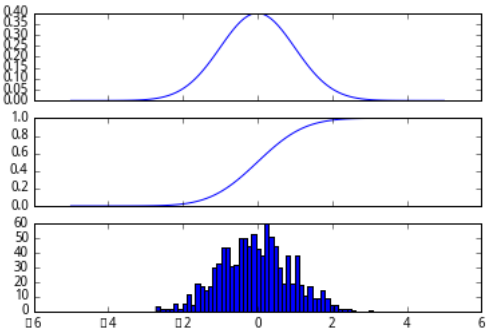


In [86]:

```
# 创建一个符合（连续）正态分布的随机变量
Y = stats.norm()
```

In [87]:

```
x = linspace(-5,5,100)

fig, axes = plt.subplots(3,1, sharex=True)

# 绘制概率分布函数 (PDF)
axes[0].plot(x, Y.pdf(x))

# 绘制累计分布函数 (CDF)
axes[1].plot(x, Y.cdf(x));

# 绘制随机变量 Y 的1000次随机实现的直方图
axes[2].hist(Y.rvs(size=1000), bins=50);
```



统计结果：

In [88]:

```
X.mean(), X.std(), X.var() # 泊松分布
```

Out[88]:

```
(3.5, 1.8708286933869707, 3.5)
```

In [89]:

```
Y.mean(), Y.std(), Y.var() # 正态分布
```

Out[89]:

```
(0.0, 1.0, 1.0)
```

统计检验¶

检验两组（独立）随机数据是否来自同一个分布：

In [90]:

```
t_statistic, p_value = stats.ttest_ind(X.rvs(size=1000), X.rvs(size=1000))

print "t-statistic =", t_statistic
print "p-value =", p_value
```

```
t-statistic = 0.215937352239
p-value = 0.829058623039
```

因为p值很大，我们不能拒绝原假设（两组随机数据有相同的均值）。

为了检验单个样本的数据是否均值为0.1（实际均值为0.0）：

In [91]:

```
stats.ttest_1samp(Y.rvs(size=1000), 0.1)
```

Out[91]:

```
(-2.1668968085719977, 0.030478493763385503)
```

p值很小，意味着我们可以拒绝原假设（Y的均值为0.1）。

In [92]:

```
Y.mean()
```

Out[92]:

```
0.0
```

In [93]:

```
stats.ttest_1samp(Y.rvs(size=1000), Y.mean())
```

Out[93]:

```
(0.10118531616888854, 0.91942365671744386)
```

延伸阅读¶

http://www.scipy.org - SciPy的官方网页

http://docs.scipy.org/doc/scipy/reference/tutorial/index.html - SciPy的一个教程

https://github.com/scipy/scipy/ - SciPy源代码.

版本¶

In [ ]:

```
%reload_ext version_information

%version_information numpy, matplotlib, scipy
```

【翻译搬运】Matplotlib - 用Python 绘制2D和3D图像

Matplotlib是Python中最常用的可视化工具之一，可以非常方便地创建海量类型的2D图表和一些基本的3D图表。本文翻译自Jupyter nbviewer中的第四讲，主要介绍了绘制2D图像的相关信息，图像的位置、大小，曲线的样式、宽度，坐标轴的刻度、数值、标签，以及图例、标题参数的设置，还包括各种类型的图像的绘制，如柱状图、色图、等高线图等等。作为延伸，又介绍了3D曲线图、框线图和投影图，以及动图的制作。最后作为了解，介绍了后端以及图片视频格式的相关内容。

作者：J.R. Johansson (邮箱：jrjohansson@gmail.com)

最新版本的用法介绍见网站http://github.com/jrjohansson/scientific-python-lectures. 其他相关介绍见http://jrjohansson.github.io.

In [1]:

```
# 利用matplotlib生成的图像嵌入notebook中，而不用每次生成图像时打开一个新的窗口，
# 后面会再次提到它的用法。如果你在使用旧版本的Python，请运行 '%pylab inline',如新版本，则输入
%matplotlib inline
```

简介¶

Matplotlib是一个绘制2D和3D科学图像的库，它包含了以下的优点：

1. 容易学习和掌握
2. 兼容LaTeX格式的标题和文档
3. 可以控制图像中的每个元素，包括图像大小和扫描精度。
4. 对于很多格式都可以高质量的输出图像，包括PNG，PDF，SVG，EPS和PGF.
5. 可以生成图形用户界面（GUI），做到交互式的获取图像以及无脑生成图像文件（通常用于批量作业）

Matplotlib最重要的一个特点，也是它作出的图像非常适合作为科学出版物的原因，是因为图像可以完全被程序所控制。这一点对于图像重现非常重要，同时为更新数据后重新作图以及改变图像形状提供了方便。更多关于Matplotlib网页请见http://matplotlib.org/

在Python中调用Matplotlib函数包有两种方法，一种是在pylab模式中包含一个星号（简单的方法）

In [2]:

```
from pylab import *
```

另一种是在matplotlib.pyplot模式下使用plt（整洁的方法）：

In [3]:

```
import matplotlib
import matplotlib.pyplot as plt
```

In [4]:

```
import numpy as np
```

MATLAB 样式的API¶

学习用matplotlib绘制图像最简单的方法使用matplotlib自身提供的类似MATLAB的API。它和MATLAB绘制图像的函数非常相近，所以熟悉MATLAB的用户可以非常容易的上手。采用在pylab模式中包含星号的方式可以使用matplotlib中的API：

In [5]:

```
from pylab import *
```

例：¶

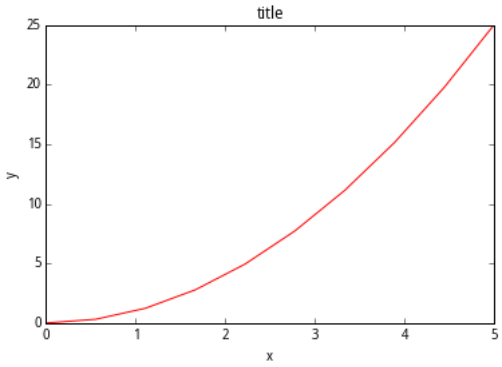
采用一个类似MATLAB作图的API，能够做出以下简单的图像：

In [6]:

```
x = np.linspace(0, 5, 10)
y = x ** 2
```

In [7]:

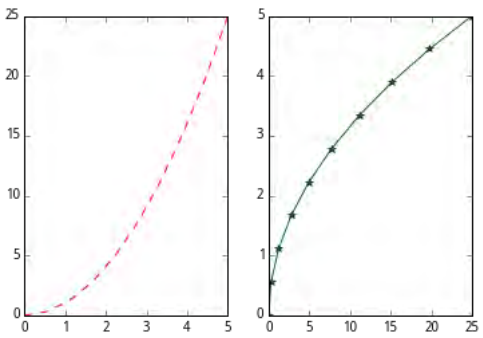
```
figure()
plot(x, y, 'r')
xlabel('x')
ylabel('y')
title('title')
show()
```



MATLAB中大多数绘图相关的函数都能在pylab模式下实现。例如将多个图像绘制在一个窗口中，以及选择颜色和线条类型：

In [8]:

```
subplot(1,2,1)
plot(x, y, 'r--')
subplot(1,2,2)
plot(y, x, 'g*-');
```



pylab这种MATLAB格式的API有一个优点，对于MATLAB熟悉的用户能够非常容易上手，而且对于绘制简单图像而言不需要花费很多精力去学习。

然而，对于并不是特别简单的图像，并不推荐使用MATLAB类似的API，学习使用matplotlib面向对象的绘图API是一种更好更强大的方法。对于多个复杂图像绘制在一个窗口中，插入图像和加入其它成分这样的复杂操作，matplotlib的API能够很好的解决。

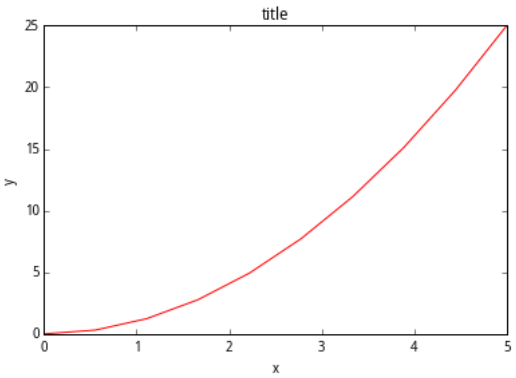
matplotlib 面向对象的API¶

面向对象的程序的主要思路是让用户能够面向对象来使用函数和进行操作，而不是像MATLAB类似的API一样采用全局的程序状态。Matplotlib的优势在绘制多个图像或者一个图像窗口中包含多个子图像的时候能够彰显出来。

我们这次采用面向对象的API来绘制和前一个例子相似的图像，但是这次我们存储一个引用在新创建的fig变量的图像中，而并不直接创建一个全局的图像，然后我们创建一个新的坐标轴图像axes（采用Figure函数族中的add_axes方法）：

In [9]:

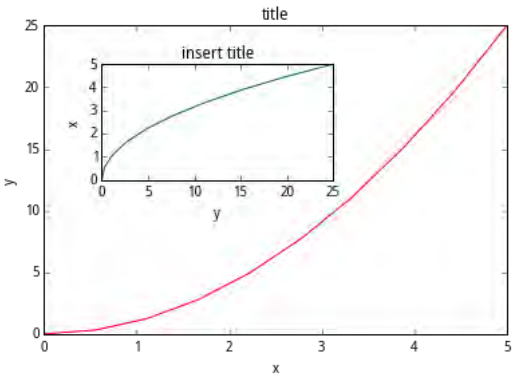
```
fig = plt.figure()

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # 左侧间距，底部间距，宽度，高度 (从0到1)

axes.plot(x, y, 'r')

axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



尽管代码看起来多了一点，但是我们现在能够完全掌控图像的坐标轴位置，并且能够在图像上轻易增加多个坐标轴：

In [10]:

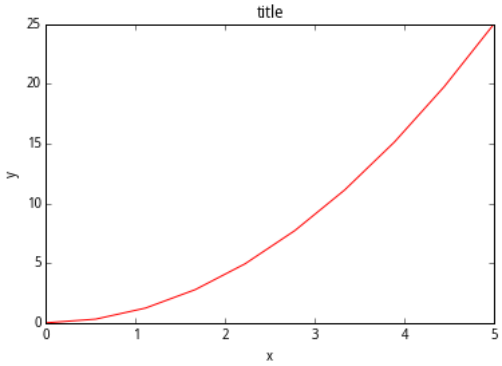
```
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # 主坐标轴
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # 插入的坐标轴

# 主要图像
axes1.plot(x, y, 'r')
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('title')

# 插入的图像
axes2.plot(y, x, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('insert title');
```



如果我们并不关心坐标轴的位置是否要明确处于画图窗口的哪个位置，我们可以采用matplotlib布局工具中的一个，例如subplots，用法如下：

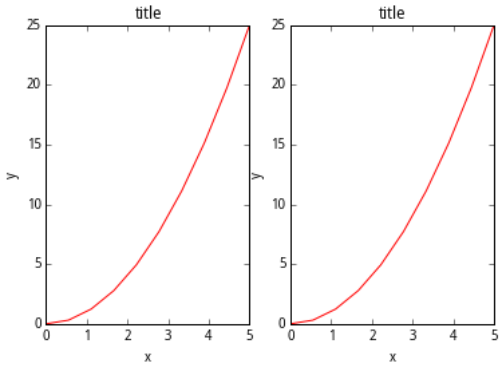
In [11]:

```
fig, axes = plt.subplots()

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



In [12]:

```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')
```

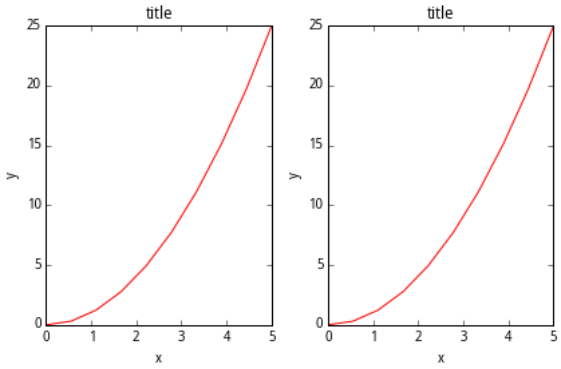


这样的代码很简单，但是如果坐标轴或者标签重合在一起，就显得不太美观了。

我们可以采用fig.tight_layout方法来解决这个问题，它能够更自动调整坐标轴在图像窗口的位置，从而避免重合的发生：

In [13]:

```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

fig.tight_layout()
```



图像大小，纵横比和图像精度¶

Matplotlib在绘制Figure对象时，允许用户确定图像纵横比、图像精度和大小，采用figsize和dpi关键字参数。figsize是关于图像宽度和高度（单位：英寸）的元组型变量，dpi是每英寸点数（像素）。为创建一个800×400像素，每英寸点数为100的图像，代码如下：

In [14]:

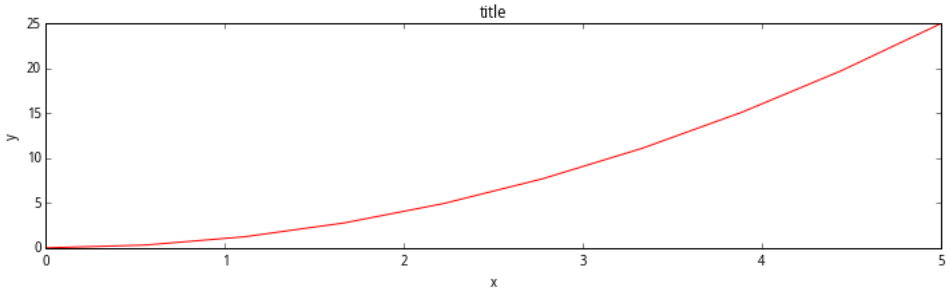
```
fig = plt.figure(figsize=(8,4), dpi=100)
```

```
<matplotlib.figure.Figure at 0x7f7f385be950>
```

同样的操作可以在布局工具中运行，例如subplots函数：

In [15]:

```
fig, axes = plt.subplots(figsize=(12,3))

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



保存图像¶

我们可以采用Figure函数族中的savefig方法来存储图像：

In [16]:

```
fig.savefig("filename.png")
```

这里我们还可以确定图像精度，以及选择不同的输出格式：

In [17]:

```
fig.savefig("filename.png", dpi=200)
```

可以存储什么样的格式？为获取最高质量图像，我们应该选择哪种格式？¶

Matplotlib可以产生各种高质量的输出格式，包括PNG，JPG，EPS，SVG，PGF和PDF。在学术论文中，如果可以的话推荐使用PDF格式（LaTeX文件编译pdflatex可以采用includegraphics来编译PDF）。在一些情况下，PGF格式也是一种好的选择。

图例、标签和标题¶

既然我们已经介绍了绘图和添加坐标轴的基本方法，我们现在来介绍如何添加图例、标签和标题。

标题¶

标题可以加在每个图像上，可以采用set_title方法来设置标题：

In [18]:

```
ax.set_title("title");
```

坐标轴标签¶

同样的，用set_xlabel和set_ylabel可以设置X和Y轴的标签：

In [19]:

```
ax.set_xlabel("x")
ax.set_ylabel("y");
```

图例¶

图像中曲线的图例可以用两种方式添加，一种是用坐标轴对象的legend指令，对于之前定义的曲线添加列表或元组形式的文本：

In [20]:

```
ax.legend(["curve1", "curve2", "curve3"]);
```

上面这种方法其实是MATLAB的API，如果图像上的曲线被添加或者删除时可能会报错（导致错误的添加图例）。

一种更好的方法是在绘图或添加其他元素的时候利用label="label text" 关键字参数，然后用无参数的legend指令把图例添加到图像上：

In [21]:

```
ax.plot(x, x**2, label="curve1")
ax.plot(x, x**3, label="curve2")
ax.legend();
```

这种方法的优点是，如果在图像上添加或者删除曲线，图例会随之自动更新。

legend函数有一个可供选择的关键字参数loc，用来确定图例添加的位置，loc参数的允许值是数值型代码，详见 http://matplotlib.org/users/legend_guide.html#legend-location. 下面列举了一些常见的loc值：

In [22]:

```
ax.legend(loc=0) # 由matplotlib确定最优位置
ax.legend(loc=1) # 右上角
ax.legend(loc=2) # 左上角
ax.legend(loc=3) # 左下角
ax.legend(loc=4) # 右下角
# .. 还有一些其他的选择，不一一列举
```

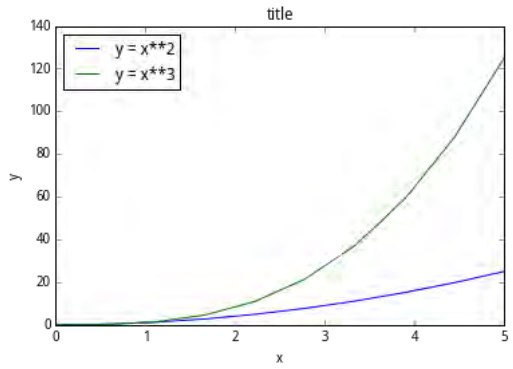
Out[22]:

```
<matplotlib.legend.Legend at 0x7f7f441f33d0>
```

运用上面介绍的关于标题，坐标轴标签和图例的内容，我们可以作出如下图像：

In [23]:

```
fig, ax = plt.subplots()

ax.plot(x, x**2, label="y = x**2")
ax.plot(x, x**3, label="y = x**3")
ax.legend(loc=2); # 左上角
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('title');
```



文本格式: LaTeX，字体大小，字体样式¶

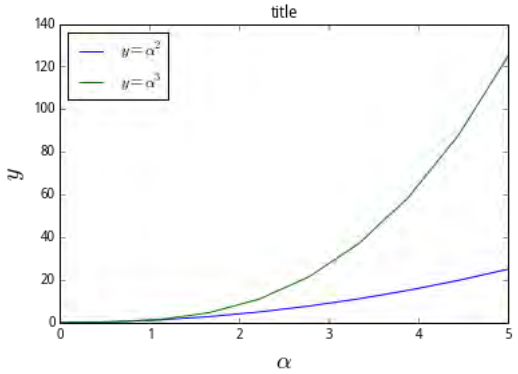
上面的绘制图像的方法都非常实用，但是还不能满足发表论文所需的标准。第一也是最重要的，我们需要采用LaTeX文本格式；第二，我们需要能够调整字体大小以适应出版社所需的要求。

Matplotlib与LaTeX非常兼容，我们只需采用美元符号来封装LaTeX的文本（图例，题目，标签等等），例如："$y = x^3$"。

但是这里我们可能在转换LaTeX代码和Python字符串的过程中出现一点问题。在LaTeX中，我们经常用反斜杠符号，例如用\alpha来产生符号α. 但是反斜杠在Python中已经有别的含义（转义码字符）。为了避免Python和LaTeX代码混淆，我们采用“原始”字符串。原始字符串带有前缀“r”，例如r"\alpha" 或者 r'\alpha' 而不是 "\alpha" or '\alpha':

In [24]:

```
fig, ax = plt.subplots()

ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # 左上角
ax.set_xlabel(r'$\alpha$', fontsize=18)
ax.set_ylabel(r'$y$', fontsize=18)
ax.set_title('title');
```



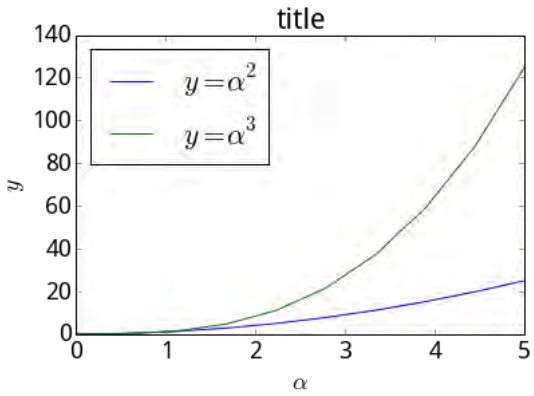
我们也可以改变全局的字体大小和字体样式，使得图像中的所有文本元素都适用（刻度标记、坐标轴标签，标题和图例等等）：

In [25]:

```
# 更新matplotlib的布局参数:
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'serif'})
```

In [26]:

```
fig, ax = plt.subplots()

ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # 左上角
ax.set_xlabel(r'$\alpha$')
ax.set_ylabel(r'$y$')
ax.set_title('title');
```



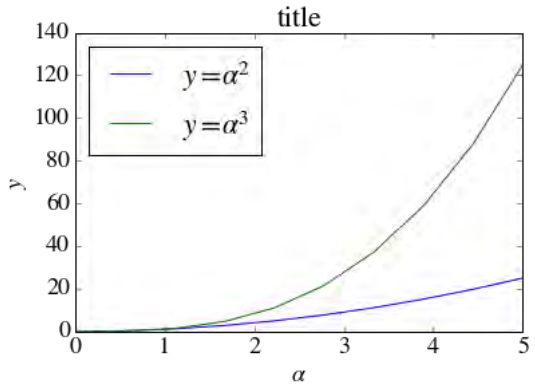
全局字体选择STIX字体样式是一个好的选择：

In [27]:

```
# 更新matplotlib的布局参数:
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral', 'mathtext.fontset': 'stix'})
```

In [28]:

```
fig, ax = plt.subplots()

ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # 左上角
ax.set_xlabel(r'$\alpha$')
ax.set_ylabel(r'$y$')
ax.set_title('title');
```



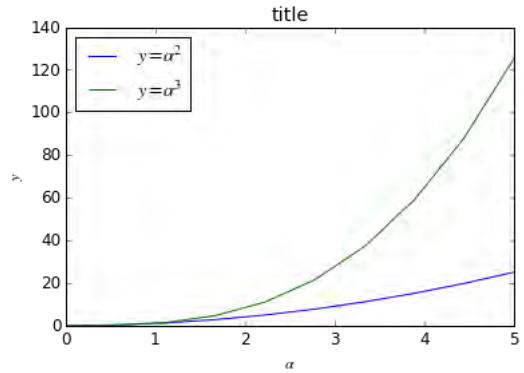
或者，我们可以要求matplotlib在图像中采用LaTeX文本元素：

In [29]:

```
matplotlib.rcParams.update({'font.size': 18, 'text.usetex': True})
```

In [32]:

```
fig, ax = plt.subplots()

ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # 左上角
ax.set_xlabel(r'$\alpha$')
ax.set_ylabel(r'$y$')
ax.set_title('title');
```



In [31]:

```
# 存储
matplotlib.rcParams.update({'font.size': 12, 'font.family': 'sans', 'text.usetex': False})
```

设置颜色，线条宽度和线条类型¶

颜色¶

用matplotlib，我们可以运用各种方法定义线条颜色和其他图像元素。首先，我们可以运用MATLAB的语法，定义'b'代表蓝色，'g'代表绿色，等等。同样，matplotlib也支持用MATLAB的API设置线条类型，例如：'b.-'代表蓝色虚点线：

In [33]:

```
# MATLAB样式的线条颜色和类型
ax.plot(x, x**2, 'b.-') # 蓝色虚点线
ax.plot(x, x**3, 'g--') # 绿色短划线
```

Out[33]:

```
[<matplotlib.lines.Line2D at 0x7f7f2892a790>]
```

我们也可以定义用颜色的英文名称定义，或者RGB十六进制码，或者用color和alpha关键字参数来选择性提供α值：

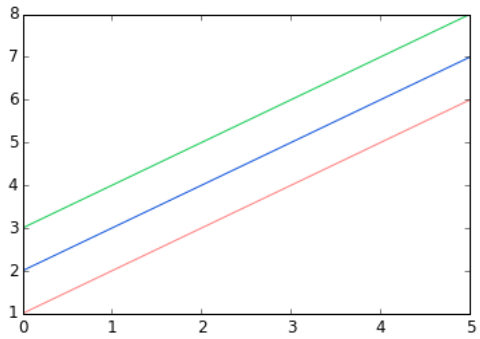
In [34]:

```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="red", alpha=0.5) # 半透明红色
```

```
ax.plot(x, x+2, color="#1155dd")        # 浅蓝色的RGB十六进制码
ax.plot(x, x+3, color="#15cc55")        # 浅绿色的RGB十六进制码
```

Out[34]:

```
[<matplotlib.lines.Line2D at 0x7f7f382a0050>]
```

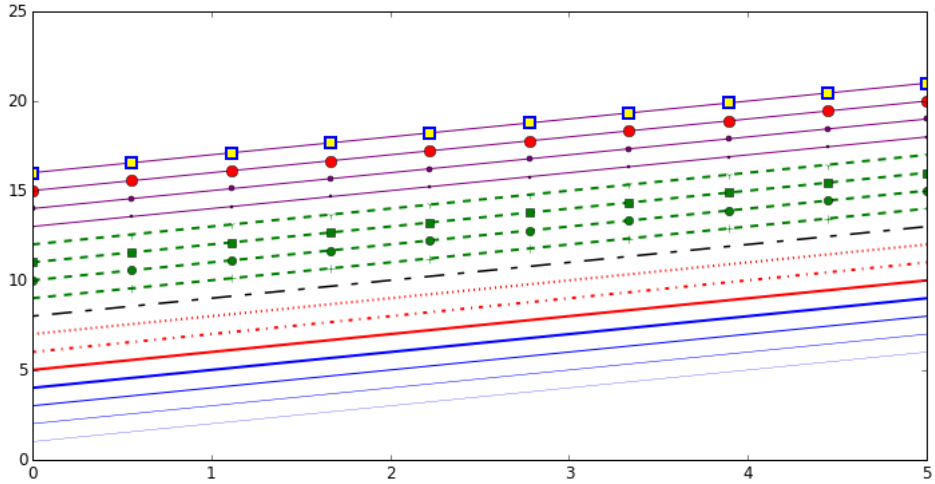


线条和标记样式¶

我们可以用linewidth或者lw关键字参数来调整线宽度，线条样式则可以在linestyle或者ls关键字参数中选择：

In [35]:

```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(x, x+1, color="blue", linewidth=0.25)
ax.plot(x, x+2, color="blue", linewidth=0.50)
ax.plot(x, x+3, color="blue", linewidth=1.00)
ax.plot(x, x+4, color="blue", linewidth=2.00)

# 线条样式选择
ax.plot(x, x+5, color="red", lw=2, linestyle='-')
ax.plot(x, x+6, color="red", lw=2, ls='-.')
ax.plot(x, x+7, color="red", lw=2, ls=':')

# 自定义设置
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # 格式：线长，间距，...

# 标记符号
ax.plot(x, x+ 9, color="green", lw=2, ls='--', marker='+')
ax.plot(x, x+10, color="green", lw=2, ls='--', marker='o')
ax.plot(x, x+11, color="green", lw=2, ls='--', marker='s')
ax.plot(x, x+12, color="green", lw=2, ls='--', marker='1')

# 标记大小和颜色
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="blue");
```



控制坐标轴外观¶

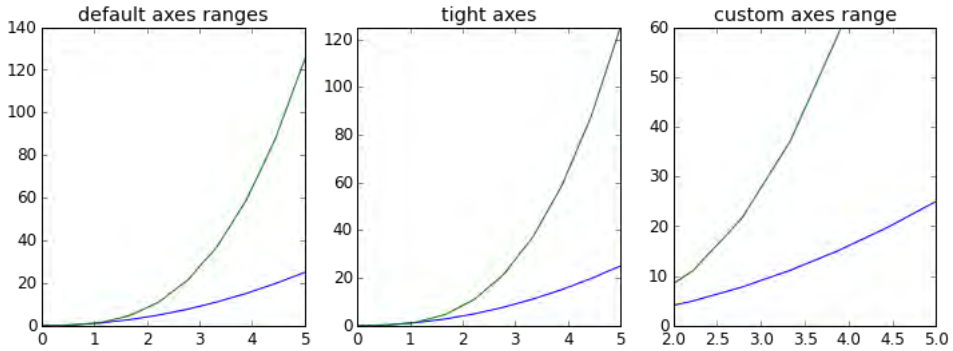
坐标轴外观是一个图像重要的方面，特别是我们经常需要更改它以满足出版刊物对于图像的要求。我们需要控制刻度和标签的位置，更改字体大小和坐标轴标签。这一节中，我们将会学习如何控制matplotlib图像的这些参数。

绘图范围¶

首先我们想要设置坐标轴的范围，可以运用坐标轴对象中的set_ylim和set_xlim，或者axis('tight')来自动设置“紧密结合”的坐标范围：

In [36]:

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title("custom axes range");
```

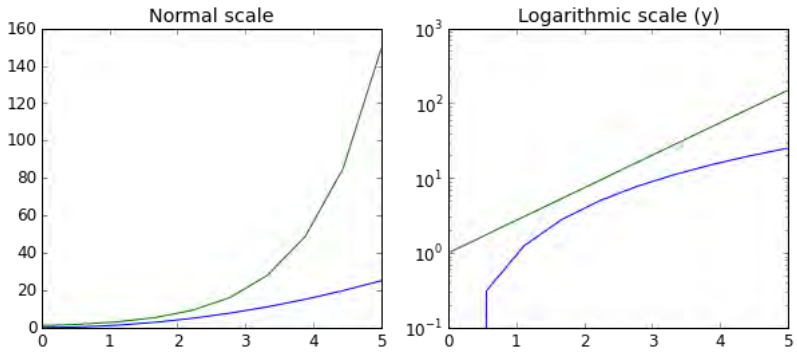


对数标度¶

对于单个或者两个坐标轴都可以设置对数标度。这个功能其实仅仅是Matplotlib全部变换系统的一个应用。每个坐标标度可以分别用set_xscale和set_yscale来设置（值填入“log”即可）：

In [37]:

```
fig, axes = plt.subplots(1, 2, figsize=(10,4))

axes[0].plot(x, x**2, x, np.exp(x))
axes[0].set_title("Normal scale")

axes[1].plot(x, x**2, x, np.exp(x))
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)");
```



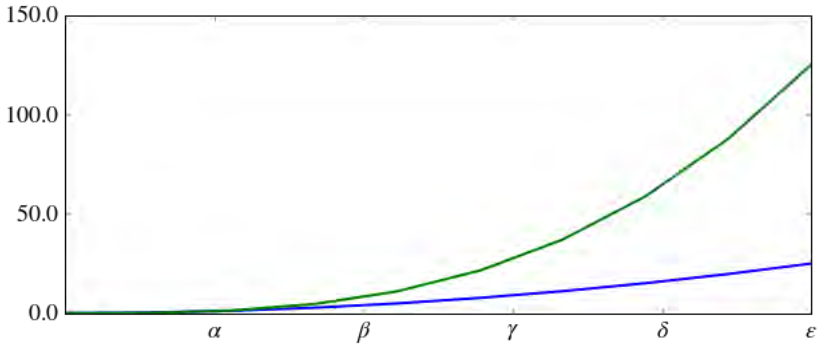
刻度的放置以及用户定义的刻度标签¶

我们可以用set_xticks和set_yticks来明确确定坐标轴的刻度位置，二者都需要提供一个列表型数值。对于每个刻度位置，我们可以用set_xticklabels和set_yticklabels来提供一个用户定义的文本标签：

In [38]:

```
fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(x, x**2, x, x**3, lw=2)

ax.set_xticks([1, 2, 3, 4, 5])
ax.set_xticklabels([r'$\alpha$', r'$\beta$', r'$\gamma$', r'$\delta$', r'$\epsilon$'], fontsize=18)

yticks = [0, 50, 100, 150]
```

```
ax.set_yticks(yticks)
ax.set_yticklabels(["$%.1f$" % y for y in yticks], fontsize=18); # 采用LaTeX格式标签
```



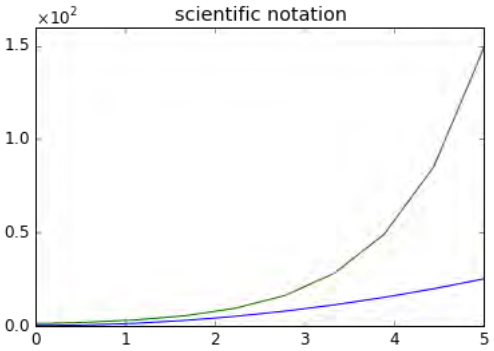
Matplotlib图像还有很多更为高级的方法来控制主刻度和副刻度的位置，比如在不同环境下自动确定其位置，详见http://matplotlib.org/api/ticker_api.html.

科学计数法¶

对于坐标轴上出现的较大的数字，通常运用科学计数法：

In [39]:

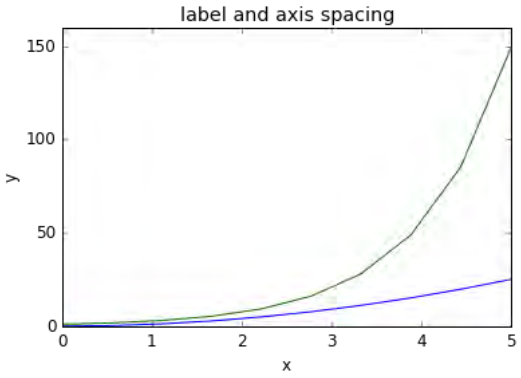
```
fig, ax = plt.subplots(1, 1)

ax.plot(x, x**2, x, np.exp(x))
ax.set_title("scientific notation")

ax.set_yticks([0, 50, 100, 150])

from matplotlib import ticker
formatter = ticker.ScalarFormatter(useMathText=True)
formatter.set_scientific(True)
formatter.set_powerlimits((-1,1))
ax.yaxis.set_major_formatter(formatter)
```



坐标数字以及坐标标签的位置¶

In [40]:

```
# x和y轴的距离和坐标轴上的数字
matplotlib.rcParams['xtick.major.pad'] = 5
matplotlib.rcParams['ytick.major.pad'] = 5

fig, ax = plt.subplots(1, 1)

ax.plot(x, x**2, x, np.exp(x))
ax.set_yticks([0, 50, 100, 150])

ax.set_title("label and axis spacing")

# 坐标轴标签和坐标轴数字的距离
ax.xaxis.labelpad = 5
ax.yaxis.labelpad = 5

ax.set_xlabel("x")
ax.set_ylabel("y");
```



In [41]:

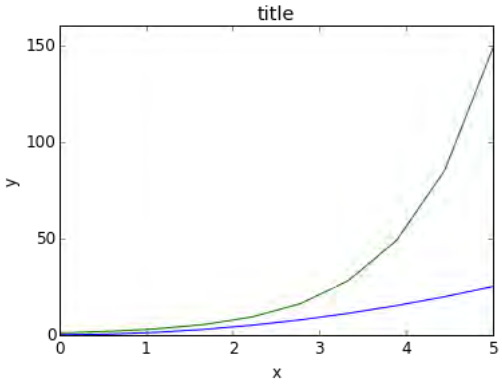
```
# 存储默认值
matplotlib.rcParams['xtick.major.pad'] = 3
matplotlib.rcParams['ytick.major.pad'] = 3
```

坐标轴位置调整¶

不幸的是，当保存图像时候，标签有时会被缩短，因此需要微调坐标轴的位置，这可以由subplots_adjust来实现：

In [42]:

```
fig, ax = plt.subplots(1, 1)

ax.plot(x, x**2, x, np.exp(x))
ax.set_yticks([0, 50, 100, 150])

ax.set_title("title")
ax.set_xlabel("x")
ax.set_ylabel("y")

fig.subplots_adjust(left=0.15, right=.9, bottom=0.1, top=0.9);
```

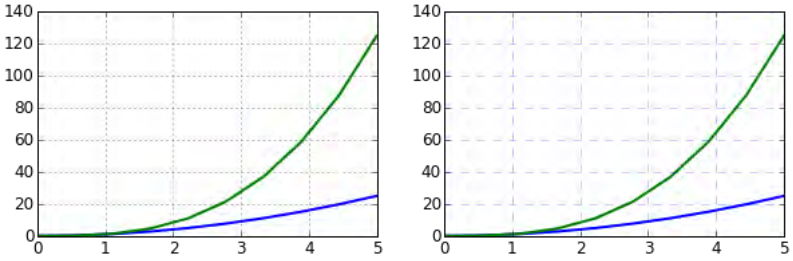


坐标轴网格¶

用坐标轴对象中的grid可以使用和取消网格线。我们也可以用plot函数中同样的关键字参数来定制网格样式：

In [44]:

```
fig, axes = plt.subplots(1, 2, figsize=(10,3))

# 默认网格外观
axes[0].plot(x, x**2, x, x**3, lw=2)
axes[0].grid(True)

# 用户定义的网格外观
axes[1].plot(x, x**2, x, x**3, lw=2)
axes[1].grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)
```

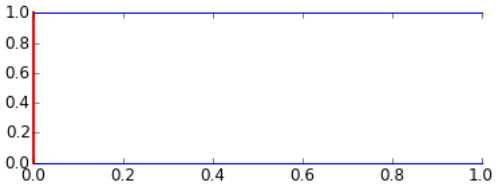


轴刻度标记线¶

我们也可以改变轴刻度标记线的参数：

In [45]:

```
fig, ax = plt.subplots(figsize=(6,2))

ax.spines['bottom'].set_color('blue')
ax.spines['top'].set_color('blue')

ax.spines['left'].set_color('red')
ax.spines['left'].set_linewidth(2)

# 取消右侧的坐标轴刻度
ax.spines['right'].set_color("none")
ax.yaxis.tick_left() # 只在左侧有刻度
```

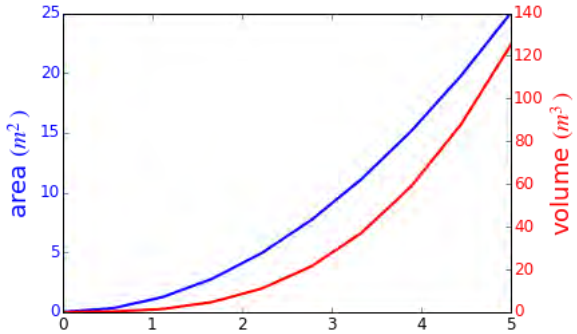


双刻度¶

有时在图像中采用两个x或y轴是十分有用的，例如单位不同的多条曲线画在一个图中。Matplotlib提供了twinx和twiny函数：

In [46]:

```
fig, ax1 = plt.subplots()

ax1.plot(x, x**2, lw=2, color="blue")
ax1.set_ylabel(r"area $(m^2)$", fontsize=18, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")

ax2 = ax1.twinx()
ax2.plot(x, x**3, lw=2, color="red")
ax2.set_ylabel(r"volume $(m^3)$", fontsize=18, color="red")
for label in ax2.get_yticklabels():
    label.set_color("red")
```

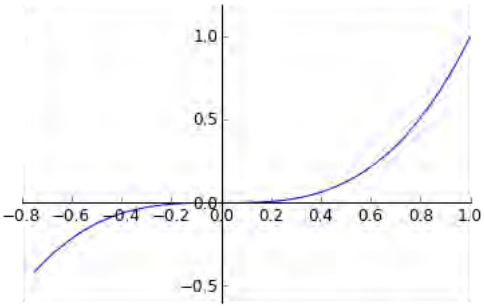


x和y值为0的坐标轴¶

In [47]:

```
fig, ax = plt.subplots()

ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
```

```
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0)) # 设置x坐标轴刻度位置于x=0

ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))   # 设置y坐标轴刻度位置于y=0

xx = np.linspace(-0.75, 1., 100)
ax.plot(xx, xx**3);
```



其他二维绘图样式¶

除了常规的plot方法，还有一些其他的函数能够实现不同样式的绘图，所有可以绘制的图像种类请见http://matplotlib.org/gallery.html. 下面展示一些有用的样式：

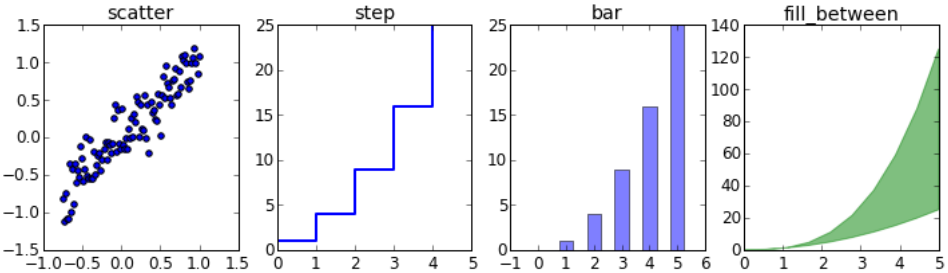
In [48]:

```
n = np.array([0,1,2,3,4,5])
```

In [49]:

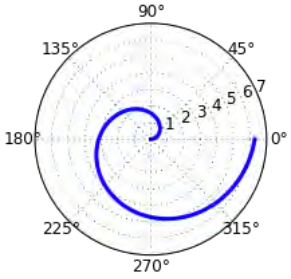
```
fig, axes = plt.subplots(1, 4, figsize=(12,3))

axes[0].scatter(xx, xx + 0.25*np.random.randn(len(xx)))
axes[0].set_title("scatter")

axes[1].step(n, n**2, lw=2)
axes[1].set_title("step")

axes[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
axes[2].set_title("bar")

axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5);
axes[3].set_title("fill_between");
```

```
/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/collections.py:590: FutureWarning: elementwise comparison failed; retu
  if self._edgecolors == str('face'):
```



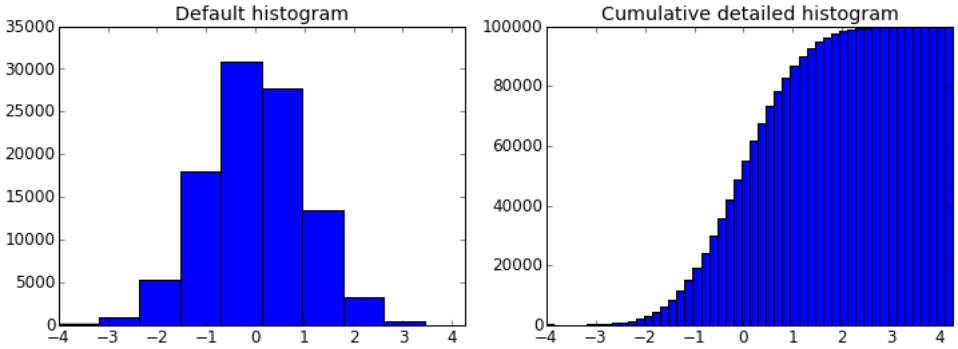
In [50]:

```
# 用add_axes的极坐标图和球极投影
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, t, color='blue', lw=3);
```



In [51]:

```
# 柱状图
n = np.random.randn(100000)
fig, axes = plt.subplots(1, 2, figsize=(12,4))

axes[0].hist(n)
axes[0].set_title("Default histogram")
axes[0].set_xlim((min(n), max(n)))

axes[1].hist(n, cumulative=True, bins=50)
axes[1].set_title("Cumulative detailed histogram")
axes[1].set_xlim((min(n), max(n)));
```

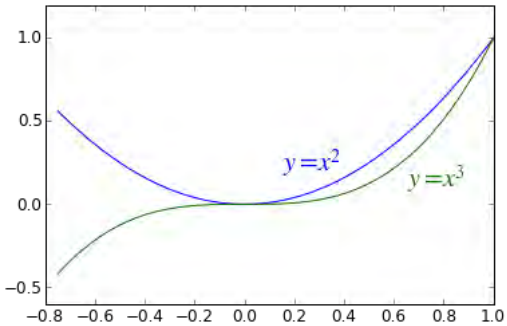


文字注释¶

采用text函数可以完成matplotlib图像的文字注释功能。和文字以及标题一样，它也支持LaTeX格式：

In [52]:

```
fig, ax = plt.subplots()

ax.plot(xx, xx**2, xx, xx**3)

ax.text(0.15, 0.2, r"$y=x^2$", fontsize=20, color="blue")
ax.text(0.65, 0.1, r"$y=x^3$", fontsize=20, color="green");
```



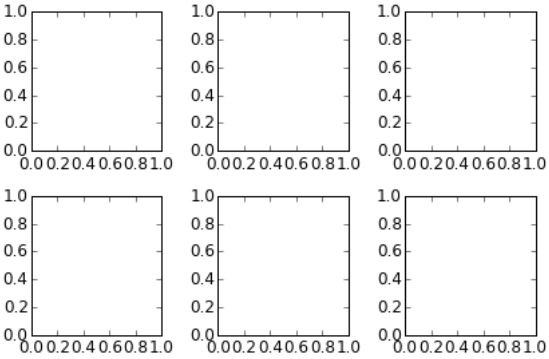
多个子图像的绘制和插入¶

采用fig.add_axes可以手动将坐标轴加入matplotlib图像中，或者用子图绘制的布局管理器，如subplots，subplot2grid或者gridspec：

subplots ¶

In [53]:

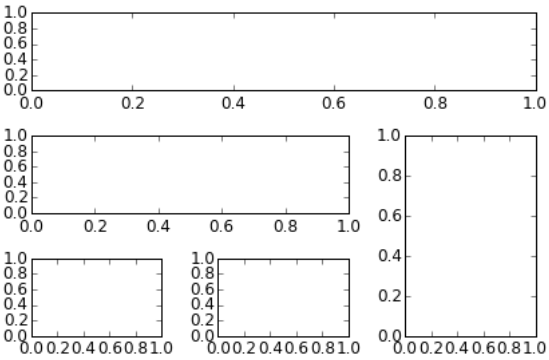
```
fig, ax = plt.subplots(2, 3)
fig.tight_layout()
```



subplot2grid ¶

In [54]:

```
fig = plt.figure()
ax1 = plt.subplot2grid((3,3), (0,0), colspan=3)
ax2 = plt.subplot2grid((3,3), (1,0), colspan=2)
ax3 = plt.subplot2grid((3,3), (1,2), rowspan=2)
ax4 = plt.subplot2grid((3,3), (2,0))
ax5 = plt.subplot2grid((3,3), (2,1))
fig.tight_layout()
```



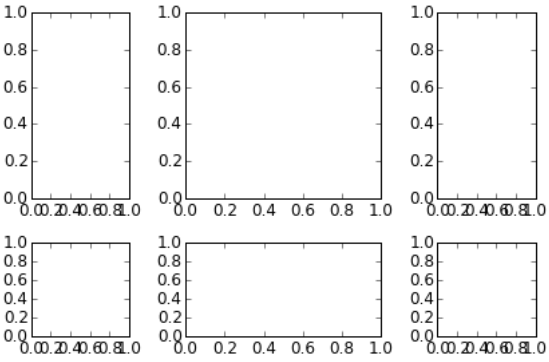
gridspec ¶

In [55]:

```
import matplotlib.gridspec as gridspec
```

In [56]:

```
fig = plt.figure()

gs = gridspec.GridSpec(2, 3, height_ratios=[2,1], width_ratios=[1,2,1])
for g in gs:
    ax = fig.add_subplot(g)

fig.tight_layout()
```

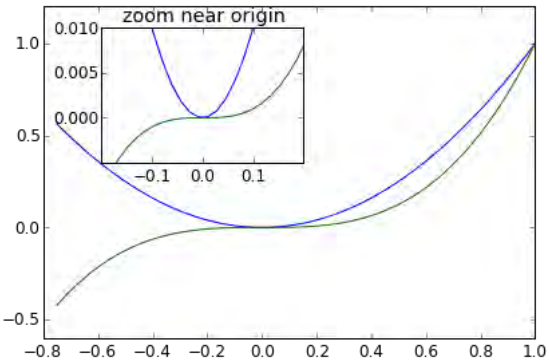


add_axes ¶

用add_axes手动添加坐标轴对于添加元素于图像中非常有用：

In [57]:

```
fig, ax = plt.subplots()

ax.plot(xx, xx**2, xx, xx**3)
fig.tight_layout()

# 插入
inset_ax = fig.add_axes([0.2, 0.55, 0.35, 0.35]) # X, Y, 宽度，高度

inset_ax.plot(xx, xx**2, xx, xx**3)
inset_ax.set_title('zoom near origin')

# 设置坐标轴范围
inset_ax.set_xlim(-.2, .2)
inset_ax.set_ylim(-.005, .01)

# 设置坐标轴刻度位置
inset_ax.set_yticks([0, 0.005, 0.01])
inset_ax.set_xticks([-0.1,0,.1]);
```



色图和等高线图¶

色图和等高线图对于两个变量的绘图函数非常有用。在大多数函数中，我们采用色图编码一个维度的数据。下面列出了一些之前定义好的色图，他们对于确定定制版的色图是一种直接的方法，详见：http://www.scipy.org/Cookbook/Matplotlib/Show_colormaps.

In [58]:

```
alpha = 0.7
phi_ext = 2 * np.pi * 0.5

def flux_qubit_potential(phi_m, phi_p):
    return 2 + alpha - 2 * np.cos(phi_p) * np.cos(phi_m) - alpha * np.cos(phi_ext - 2*phi_p)
```

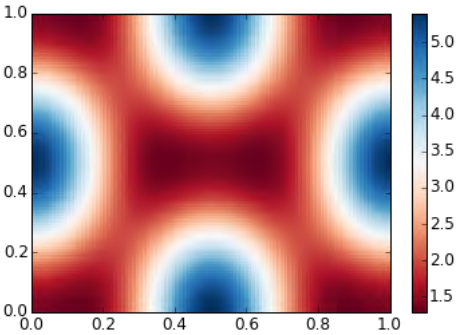
In [59]:

```
phi_m = np.linspace(0, 2*np.pi, 100)
phi_p = np.linspace(0, 2*np.pi, 100)
X,Y = np.meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T
```

pcolor 函数¶

In [60]:

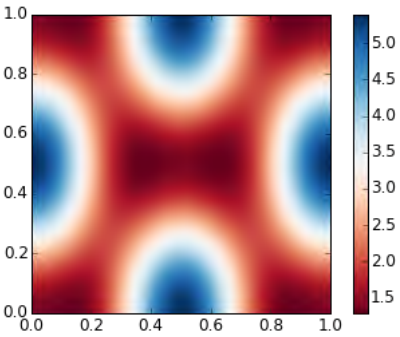
```
fig, ax = plt.subplots()

p = ax.pcolor(X/(2*np.pi), Y/(2*np.pi), Z, cmap=matplotlib.cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max())
cb = fig.colorbar(p, ax=ax)
```



imshow 函数¶

In [61]:

```
fig, ax = plt.subplots()

im = ax.imshow(Z, cmap=matplotlib.cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max(), extent=[0, 1, 0, 1])
im.set_interpolation('bilinear')

cb = fig.colorbar(im, ax=ax)
```

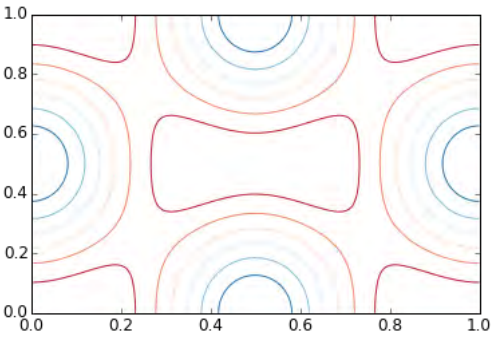


contour 函数¶

In [62]:

```
fig, ax = plt.subplots()

cnt = ax.contour(Z, cmap=matplotlib.cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max(), extent=[0, 1, 0, 1])
```

```
/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/collections.py:650: FutureWarning: elementwise comparison failed; retu
  if self._edgecolors_original != str('face'):
```



3D 图像¶

在使用matplotlib中的3D图像之前，我们首先需要创建Axes3D类。3D坐标轴和2D坐标轴创建的方法一样；或者更方便的方法是，在add_axes或者add_subplot中采用projection='3d'关键字参数。

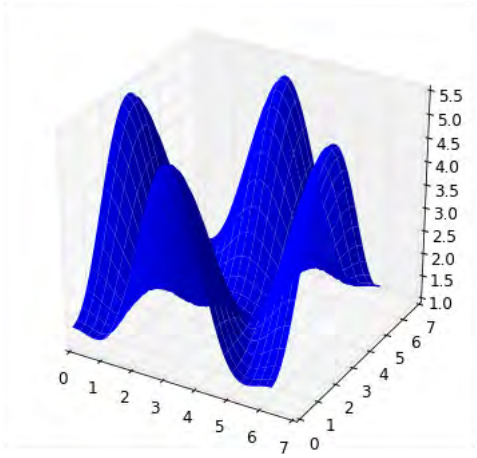
In [63]:

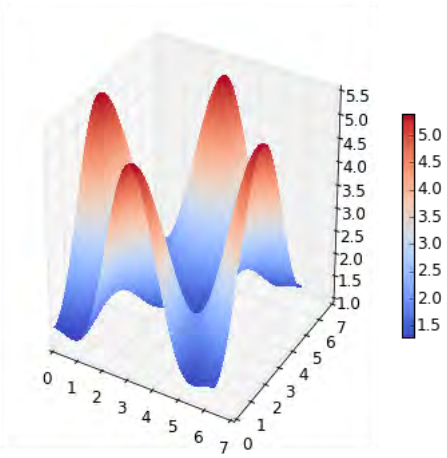
```
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

曲面图¶

In [64]:

```
fig = plt.figure(figsize=(14,6))

# `ax` 是一个3D坐标轴，由于添加了projection='3d'关键字参数于add_subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=0)

# 带有颜色梯度和颜色条的曲面图
ax = fig.add_subplot(1, 2, 2, projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=matplotlib.cm.coolwarm, linewidth=0, antialiased=False)
cb = fig.colorbar(p, shrink=0.5)
```

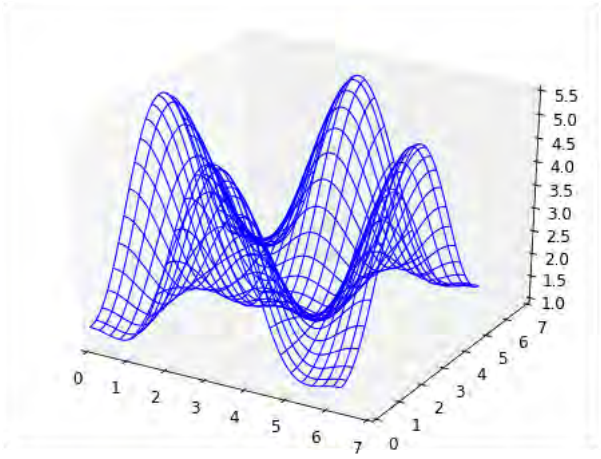




三维线框图¶

In [65]:

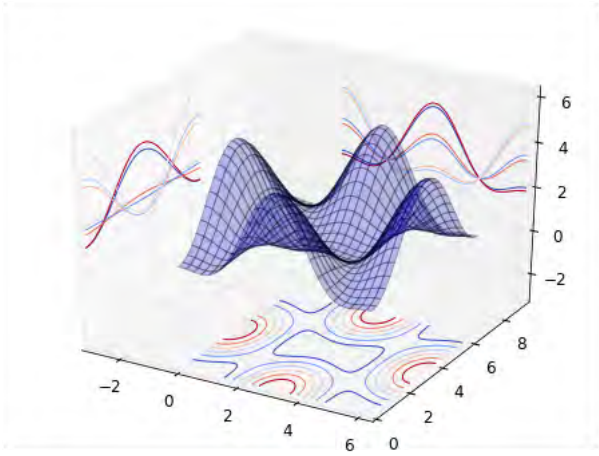
```
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1, 1, 1, projection='3d')

p = ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
```



带投影的等高线图¶

In [66]:

```
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1,1,1, projection='3d')

ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset = ax.contour(X, Y, Z, zdir='z', offset=-np.pi, cmap=matplotlib.cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-np.pi, cmap=matplotlib.cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=3*np.pi, cmap=matplotlib.cm.coolwarm)

ax.set_xlim3d(-np.pi, 2*np.pi);
ax.set_ylim3d(0, 3*np.pi);
ax.set_zlim3d(-np.pi, 2*np.pi);
```

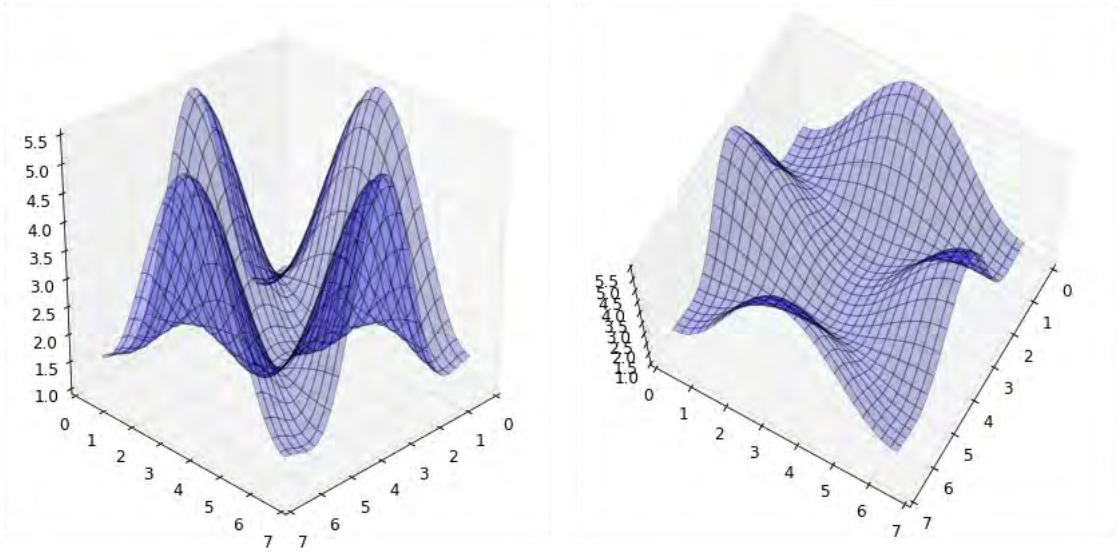


改变视角¶

采用view_init可以改变3D图像的视角，该命令有两个参数，elevation和azimuth角度（度数）：

In [67]:

```
fig = plt.figure(figsize=(12,6))

ax = fig.add_subplot(1,2,1, projection='3d')
ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
ax.view_init(30, 45)

ax = fig.add_subplot(1,2,2, projection='3d')
ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
ax.view_init(70, 30)

fig.tight_layout()
```



动图¶

Matplotlib也包含了一个简单的API用来产生一系列图像的动图。采用FuncAnimation函数，我们可以产生由一系列图像组成的视频文件。该函数采用了如下命令：fig图像窗口，func更新图像所用的函数，init_func组织图像的函数，frame产生的帧数，和blit指导动图函数只在帧数变动的时候更新（对于光滑动图而言）：

```
def init():
    # setup figure

def update(frame_counter):
    # update figure for new frame

anim = animation.FuncAnimation(fig, update, init_func=init, frames=200, blit=True)

anim.save('animation.mp4', fps=30) # fps = frames per second
```

为了使用matplotlib中的动图函数，我们首先调用matplotlib.animation：

In [68]:

```
from matplotlib import animation
```

In [69]:

```
# 解决复摆的ODE（常微分方程）问题

from scipy.integrate import odeint
from numpy import cos, sin

g = 9.82; L = 0.5; m = 0.1

def dx(x, t):
    x1, x2, x3, x4 = x[0], x[1], x[2], x[3]

    dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * cos(x1-x2) * x4)/(16 - 9 * cos(x1-x2)**2)
    dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * cos(x1-x2) * x3)/(16 - 9 * cos(x1-x2)**2)
    dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * sin(x1-x2) + 3 * (g/L) * sin(x1))
    dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * sin(x1-x2) + (g/L) * sin(x2))
    return [dx1, dx2, dx3, dx4]

x0 = [np.pi/2, np.pi/2, 0, 0]  # 初始状态
t = np.linspace(0, 10, 250) # 时间坐标
x = odeint(dx, x0, t)    # 求常微分方程的解
```

生成的动画显示了钟摆的位置作为时间的函数：

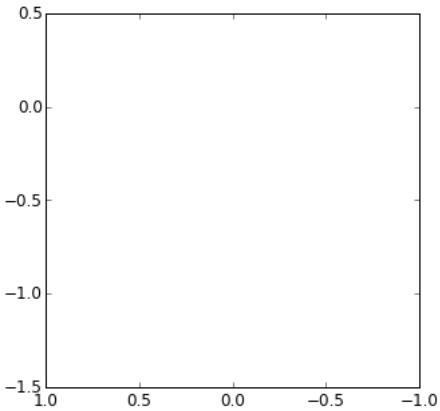
In [70]:

```
fig, ax = plt.subplots(figsize=(5,5))

ax.set_ylim([-1.5, 0.5])
ax.set_xlim([1, -1])

pendulum1, = ax.plot([], [], color="red", lw=2)
pendulum2, = ax.plot([], [], color="blue", lw=2)

def init():
    pendulum1.set_data([], [])
    pendulum2.set_data([], [])

def update(n):
    # n = 帧计数器
    # 计算复摆的位置
    x1 = + L * sin(x[n, 0])
    y1 = - L * cos(x[n, 0])
    x2 = x1 + L * sin(x[n, 1])
    y2 = y1 - L * cos(x[n, 1])

    # 更新数据
    pendulum1.set_data([0 ,x1], [0 ,y1])
    pendulum2.set_data([x1,x2], [y1,y2])

anim = animation.FuncAnimation(fig, update, init_func=init, frames=len(t), blit=True)

# anim.save可以采用不同的方法运行，在不同平台、
# 不同版本的matplotlib和视频编码器上可能有的方法不能使用
#anim.save('animation.mp4', fps=20, extra_args=['-vcodec', 'libx264'], writer=animation.FFMpegWriter())
#anim.save('animation.mp4', fps=20, extra_args=['-vcodec', 'libx264'])
#anim.save('animation.mp4', fps=20, writer="ffmpeg", codec="libx264")
anim.save('animation.mp4', fps=20, writer="avconv", codec="libx264")

plt.close(fig)
```

```
/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/animation.py:742: UserWarning: MovieWriter avconv unavailable
  warnings.warn("MovieWriter %s unavailable" % writer)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-70-41850fe2257c> in <module>()
     30 #anim.save('animation.mp4', fps=20, extra_args=['-vcodec', 'libx264'])
     31 #anim.save('animation.mp4', fps=20, writer="ffmpeg", codec="libx264")
---> 32 anim.save('animation.mp4', fps=20, writer="avconv", codec="libx264")
     33
     34 plt.close(fig)

/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/animation.pyc in save(self, filename, writer, fps, dpi, codec, bitrate
    747                                                     metadata=metadata)
    748                 except IndexError:
```

```
--> 749                     raise ValueError("Cannot save animation: no writers are "
    750                                      "available. Please install mencoder or "
    751                                      "ffmpeg to save animations.")

ValueError: Cannot save animation: no writers are available. Please install mencoder or ffmpeg to save animations.
```



Note: 为了产生视频文件，我们需要安装ffmpeg或者avconv. 在Ubuntu上安装的指令:

```
$ sudo apt-get install ffmpeg
```

或者（更新的版本）

```
$ sudo apt-get install libav-tools
```

在MacOSX中, 尝试:

```
$ sudo port install ffmpeg
```

有兴趣的用户可以自行安装，这里不再演示视频文件。

In [71]:

```
from IPython.display import HTML
video = open("animation.mp4", "rb").read()
video_encoded = video.encode("base64")
video_tag = '<video controls alt="test" src="data:video/x-m4v;base64,{0}">'.format(video_encoded)
HTML(video_tag)
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-71-dee50753e05a> in <module>()
      1 from IPython.display import HTML
----> 2 video = open("animation.mp4", "rb").read()
      3 video_encoded = video.encode("base64")
      4 video_tag = '<video controls alt="test" src="data:video/x-m4v;base64,{0}">'.format(video_encoded)
      5 HTML(video_tag)

IOError: [Errno 2] No such file or directory: 'animation.mp4'
```

后端¶

Matplotlib有许多“后端”对产生的图像负责，不同的后端能够更产生不同样式的图和视频。非交互式的后端（如 'agg'，'svg'，'pdf'等）是用来产生图像文件（如savefig函数），与此不同，交互式的后端（如Qt4Agg，GTK，MaxOSX）能够运行GUI窗口，供用户进行交互式的使用图像。

可供选择的后端有:

In [72]:

```
print(matplotlib.rcsetup.all_backends)
```

```
[u'GTK', u'GTKAgg', u'GTKCairo', u'MacOSX', u'Qt4Agg', u'Qt5Agg', u'TkAgg', u'WX', u'WXAgg', u'CocoaAgg', u'GTK3Cairo', u'GTK3Agg', u
```

默认后端是agg，它基于栅格图形库，非常适合生成像PNG这样的光栅格式。

通常来说，我们并不需要改变默认后端，但是有时转换到例如PDF或者GTKcariro（如果是Linux系统）时会非常有用，能够更产生高质量矢量图形而不是栅格图。

使用svg 后端产生SVG¶

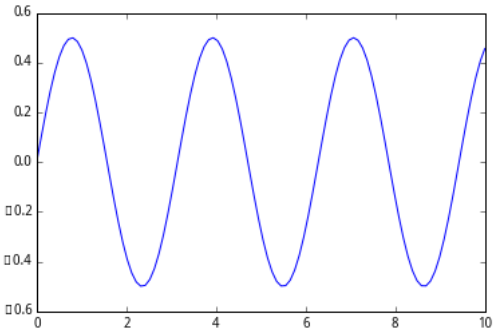
In [1]:

```
#
# 重启Notebook: matplotlib后端只能在pylab中选择
# (e.g. Kernel > Restart)
#
import matplotlib
matplotlib.use('svg')
import matplotlib.pylab as plt
import numpy
from IPython.display import Image, SVG
```

```
/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/__init__.py:1318: UserWarning:  This call to matplotlib.use() has no e
because the backend has already been chosen;
matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

  warnings.warn(_use_error_msg)
```

In [2]:

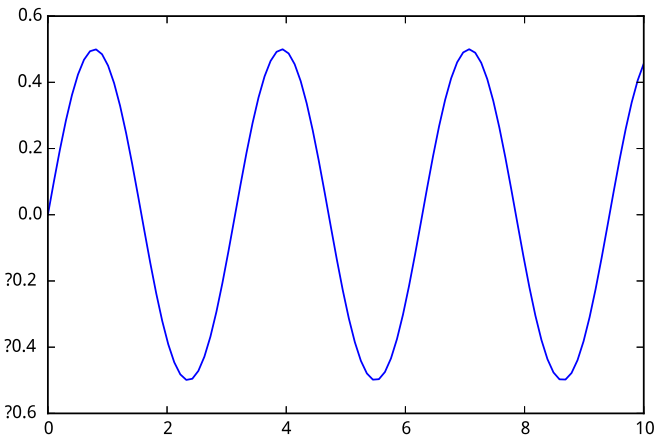
```
#
# 现在我们用svg后端产生SVG矢量图
#
fig, ax = plt.subplots()
t = numpy.linspace(0, 10, 100)
ax.plot(t, numpy.cos(t)*numpy.sin(t))
plt.savefig("test.svg")
```



In [3]:

```
#
# 显示产生的SVG文件.
#
SVG(filename="test.svg")
```

Out[3]:



IPython notebook　内联后端¶

当我们使用IPython notebook时，可以很方便的用matplotlib后端输出嵌入在notebook的图形文件。要激活这个后端，需要在开始的某处添加：

```
%matplotlib inline
```

采用如下格式也能够激活内联后端：

```
%pylab inline
```

不同之处在于%pylab inline调用了一系列函数包到全局地址空间（scipy，numpy），然而%matplotlib inline只在内联绘图时才调用。在IPython 1.0+的新的notebook中，建议使用%matplotlib inline，因为它更整洁，对于函数包的调用控制更多。通常，scipy和numpy分别通过如下形式调用：

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

内联后端有一系列的设置选择，可以通过IPython的命令%config来更新InlandBackend中的设置。例如，我们可以转换SVG图像或者更高分别率的图像通过：

```
%config InlineBackend.figure_format='svg'
```

或者

```
%config InlineBackend.figure_format='retina'
```

如需了解更多内容，请输入：

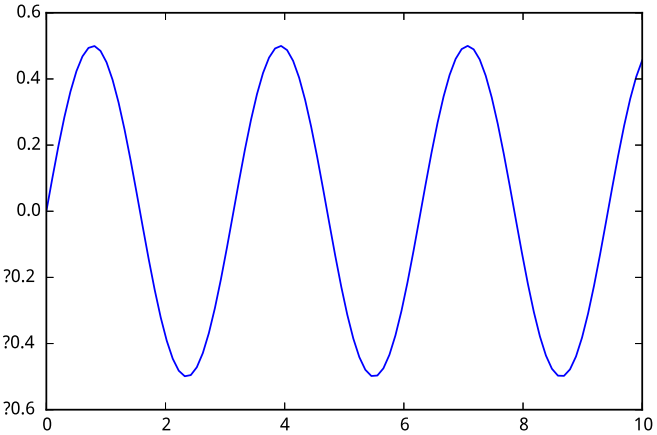
```
%config InlineBackend
```

In [1]:

```
%matplotlib inline
%config InlineBackend.figure_format='svg'

import matplotlib.pylab as plt
import numpy
```

In [2]:

```
#
# 现在我们替换notebook中的内联后端，使用SVG矢量图。
#
fig, ax = plt.subplots()
t = numpy.linspace(0, 10, 100)
ax.plot(t, numpy.cos(t)*numpy.sin(t))
plt.savefig("test.svg")
```



交互式后端（这使得Python 脚本文件更有意义）¶

In [1]:

```
#
# 重启Notebook: matplotlib后端只能在pylab中选择
# (e.g. Kernel > Restart)
#
```

```
import matplotlib
matplotlib.use('Qt4Agg') # or for example MacOSX
import matplotlib.pylab as plt
import numpy as np
```

```
/opt/conda/envs/python2/lib/python2.7/site-packages/matplotlib/__init__.py:1318: UserWarning:  This call to matplotlib.use() has no e
because the backend has already been chosen;
matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

  warnings.warn(_use_error_msg)
```

In [ ]:

```
# 现在采用Qt4Agg打开一个交互式的绘图窗口
fig, ax = plt.subplots()
t = np.linspace(0, 10, 100)
ax.plot(t, np.cos(t) * np.sin(t))
plt.show()
```

注意，当我们采用交互式后端是，需要调用plt.show()命令将图像显示在屏幕上。

推荐阅读¶

http://www.matplotlib.org - 官方网页 https://github.com/matplotlib/matplotlib - matplotlib源代码 http://matplotlib.org/gallery.html - 展示各种各样matplotlib函数包能够绘制的图像，强烈推荐！ http://www.loria.fr/~rougier/teaching/matplotlib - matplotlib课程 http://scipy-lectures.github.io/matplotlib/matplotlib.html - 其他参考文献

版本¶

In [ ]:

```
%reload_ext version_information
%version_information numpy, scipy, matplotlib
```

策略与应用

【量化课堂】股债轮动、股票指数间轮动和行业间轮动的一种新思路：GEYR策略

前言

GEYR一般定义为长期国债收益率与证券市场股利收益率的比值，可用于股债配置。当该值增大时，债券配置价值较高；反之则股票配置价值较高。GEYR策略的本质思想在于将标的资产视为零息债券，并对票息进行贴现以确定资产价值，进而得到何时应配置哪种资产。与股债间GEYR指数的计算类似，还可以用同样的思路对股票指数、股票行业的股利收益率做比，得到类似GEYR的比值，根据该比值上升或下降，在标的之间进行选择。

在选择的过程中，本文用到了两种方法，分别是隐马尔可夫模型和阈值模型。综合来看，隐马尔可夫模型在股债轮动之间表现尚可，但在指数轮动和行业轮动中表现一般；而阈值模型在指数轮动和行业轮动中表现不俗。根据分析，我们认为GEYR指标本身就包含了关于两标的价格、价值等较多信息，使用隐马模型得到的隐状态含义不明确，就有可能在计算隐状态、选择标的的过程中丢失信息。

```
本文由 JoinQuant 量化课堂推出。难度标签为进阶，理解深度标签：level-1
作者： YR
编辑： Mathematical23
```

一、GEYR指标简介

GEYR一般定义为长期国债收益率与证券市场股利收益率的比值，在本研究中采取10年期国债收益率与全市场PE比的倒数的比值，用公式表示为：

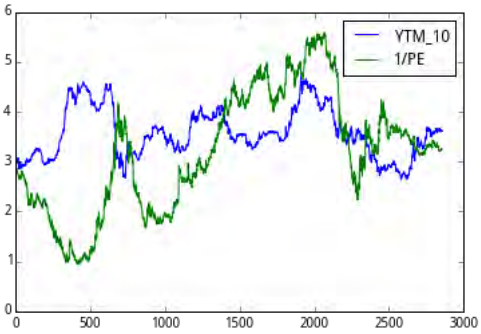
$$GEYR = \frac{YTM < em > 10}{\frac{\sum i=1^{n} MarketCap < em > i}{\sum i=1^{n} \sum < em > j=-1^{-4} np ij}}$$

分子是比较容易理解的。从经济学含义的角度来说，分母的实际意义为考察整个A股的价格与其盈利水平间的关系，计算方法类似PE值的计算。存在这样等量关系的原因是：

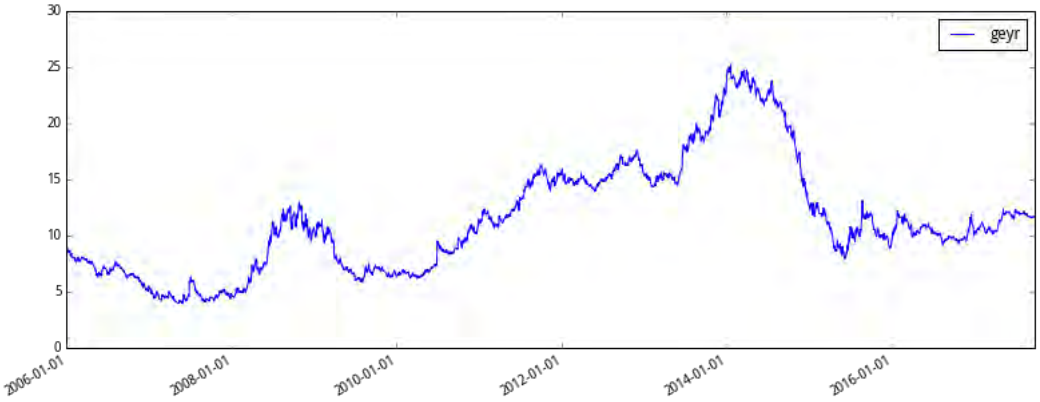
$$\frac{\sum < em > i = 1^{n} MarketCap < /em > i}{\frac{1}{4} \sum < em > i = 1^{n} \sum < /em > j = -1^{-4} np < em > ij} = \frac{MarketCap < /em > AShare}{\frac{1}{4} \sum < em > i = -1^{-4} np < /em > AShare\, i} = \frac{\overline{Price < em > AShare} * Sto}{\overline{EPS < /em > AShare} * Sto}$$

其中np为net profit，即净利润。

计算GEYR所需的10年期国债到期收益率和A股PE值图像如下：



计算所得GEYR图像如下：



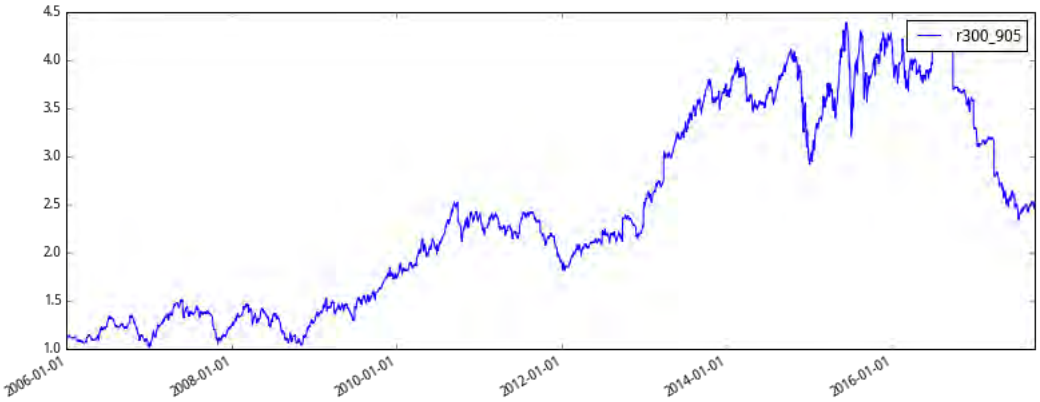
以GEYR作为标的的基本思想为：当GEYR值增大时，说明国债到期收益率升高，债券配置价值较大；反之，则说明A股PE值减小，即价格相对其盈利水平被低估，股市配置价值较大。以此为思想，我们可以通过更量化的指标来具体确定买入和卖出的时间。

有一个细节值得一提：这里在计算某一天GEYR值的时候，取的不是当天的全市场净利润，而是包含当期，向前推4个季度的净利润的平均值。之所以要向前滚动，是因为不同公司公布年报时点不同，且向前滚动一年能带来更高的稳定性。

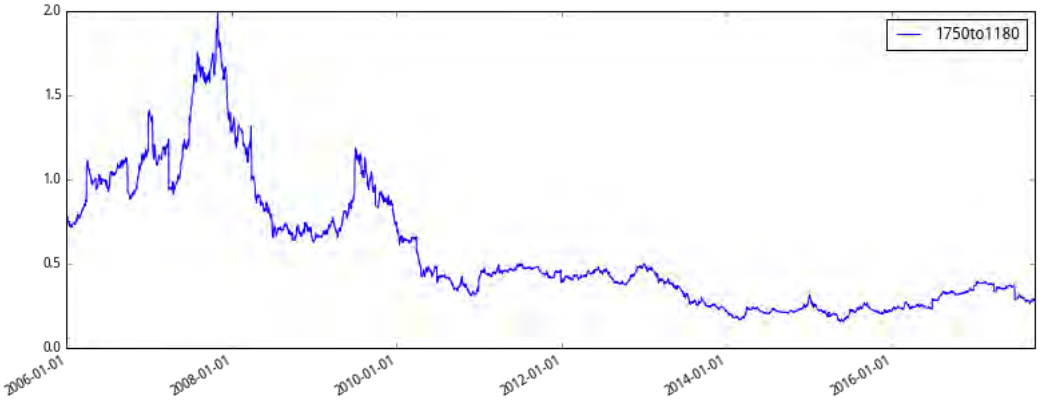
GEYR一般指股债之间的比值指标，但我们也可以借用类似的想法构建股票之间的类GEYR指标。举个例子，构建两指数000300和000905之间的GEYR指标，方法为：

$$GEYR300to905 = \frac{\frac{np300}{MarketCap300}}{\frac{np905}{MarketCap905}} = \frac{PE905}{PE_{300}}$$

该指标上扬时，说明000905的PE过高，应配置000300；反之，则配置000905。该指标的图像如图所示：



用同样方法可以对行业间进行轮动配置，这里以申万行业801750（计算机）和801180（房地产）为例，构建的GEYR图像如下：

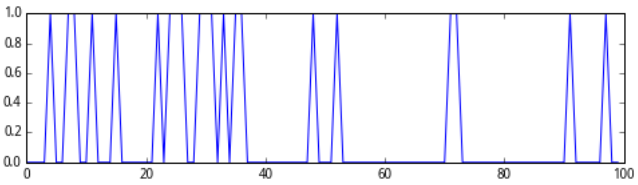


二、通过GEYR构建转换策略的方法

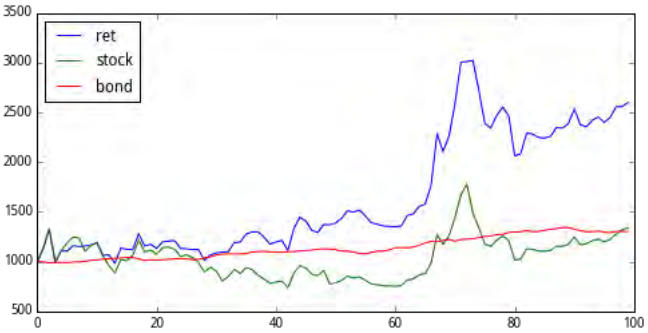
在计算得到GEYR序列后，我们需要通过该指标在两类资产之间进行轮动配置，那么何时进行资产类别的转换就成了关键问题。本次研究，我们构建了两种模型进行分析，分别是隐马尔可夫模型和阈值模型。

1、隐马尔可夫模型

隐马尔科夫模型（Hidden Markov Model, HMM）是被广泛使用在自然语言处理领域的一种统计学习模型，目前也被用于金融市场资产定价中。关于隐马尔可夫模型，可以参考量化课堂以前的文章【量化课堂】单特征因子的隐马尔可夫模型在商品期货中的应用 以及CSDN上刘建平的博客。
以股债轮动为例，我们假定有0和1两个隐状态，分别代表股票配置价值更高和债券配置价值更高。由于债券价格变动幅度较小，故我们计算每个月的GEYR，且以月为单位进行计算。我们选择前40个月的数据作为初始观察值，估计各个参数，然后计算第41个月即样本外的预测值，并之后滚动进行估计。
以此方法得到的隐状态转移图像为：

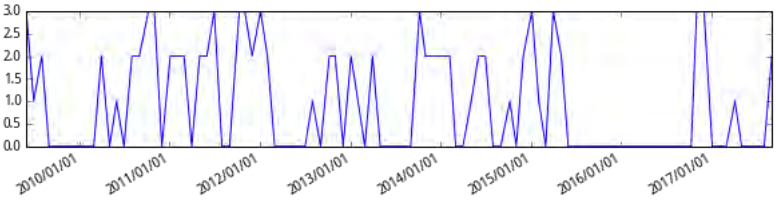


根据以下方法构建策略：当预测出下一时间点为隐状态0时，配置股票；为隐状态1时，配置债券。以1000为起始点，该策略得到的收益率图像为：

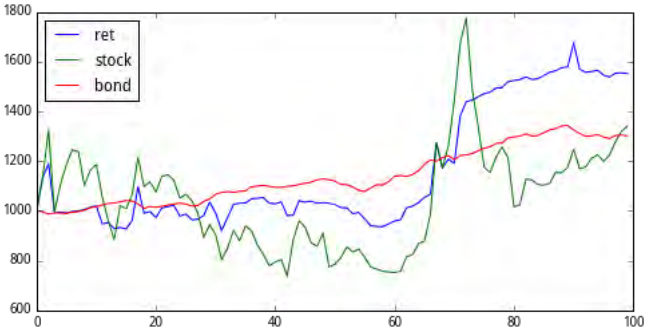


由隐状态转移图像和收益率图像我们可以发现，隐马模型对股票更“偏好”一些，导致策略在中后期获得了股票大幅上涨收益的同时也经受了大幅的回撤。故我们考虑将隐状态数量由两个增加至4个，记为0，1，2，3。当隐状态取0时，配置100%股票；取3时，配置100%债券。而取1和2时，分别配置2/3股票+1/3债券和1/3股票+2/3债券。我们希望通过这种操作来更多地增加对债券的配置，并平滑收益率曲线。

以此方法得到的隐状态转移图像为：

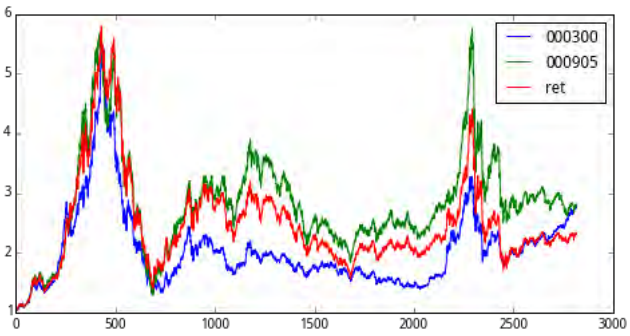


该策略得到的收益率图像为：

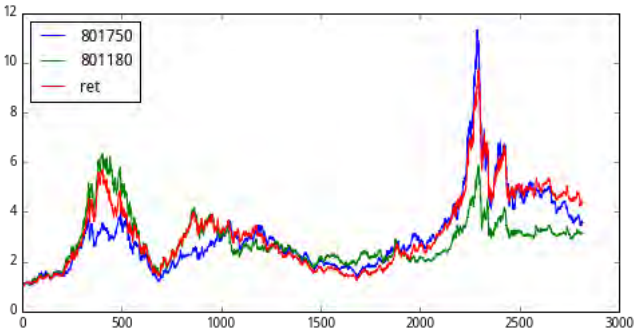


可以看到，2状态和4状态在隐状态转移图像中是有一些类似的，比如15年下半年起配置股票较多，这也符合我们的直觉，即两者差异不应太大。虽然该策略的最终收益不如2状态的收益，但可以看出其回撤明显更小，可以说是在减少收益的同时降低了风险。

我们尝试用类似的方法对指数间（000300和000905）、行业间（801750和801180）进行轮动配置，但由于股票市场变动速度比债券市场快，故我们采用的是日度数据。我们采用的是2隐状态+向前滚动40天的方法构建策略，得到的收益率图像如下：
指数间轮动：



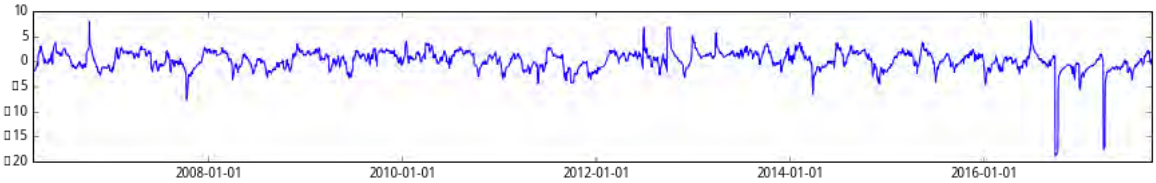
行业间轮动：



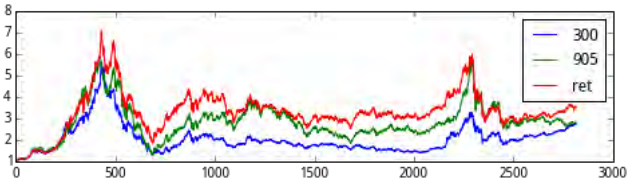
可以看出，这两个方法得到的策略的收益率均基本处在两标的收益率之间，效果不尽如人意。具体原因我们将在文章结尾进行分析。

2、阈值模型

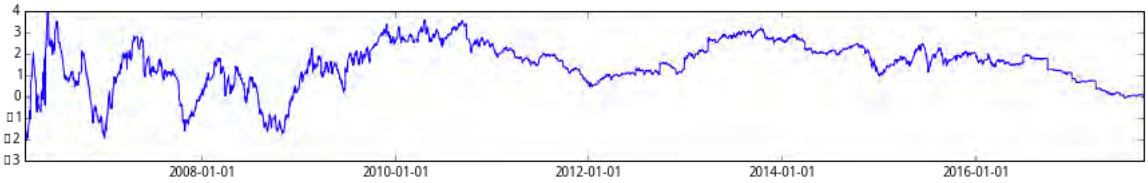
事实上，GEYR及类GEYR指标本身的含义是比较明确的，故我们考虑直接通过该指标进行计算并考察何时进行轮动。以000300和000905为例，与之前想法类似，我们采取滚动的方法进行估计。取前40天的数据作为观察值，进行均值-方差的标准化，当最后一天的值大于前39天的平均值时，认为GEYR有上扬趋势，配置000300；反之则配置000905。通过该方法计算的每天经标准化后的GEYR值为：



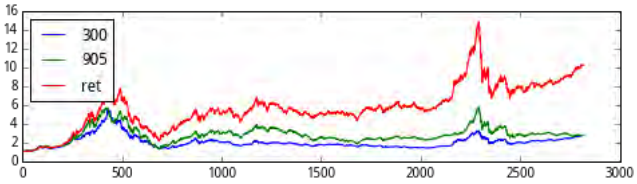
该策略的收益率曲线为：



在此方法基础上，我们又尝试将标准化使用的数据由前40天改为从起始日直到当天，其他方法不变。通过该方法计算的每天经标准化后的GEYR值为：

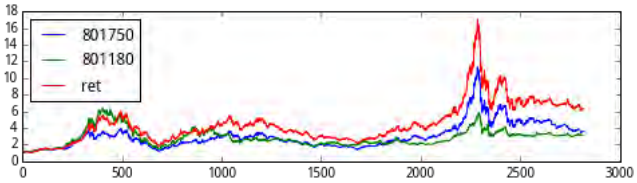


该策略的收益率曲线为：



可以看到，两个策略的表现都不错，都明显强于马尔科夫策略，而延长了标准化数据窗口时间得到策略的收益率更高。根据分析，我们认为后者收益率高于前者的主要原因在于，GEYR指标是将资产视为永续债券，考虑其未来现金流贴现得到的一个指标。由于股票价格波动较大，在计算GEYR阈值时如果取的窗口期较长，则该阈值会较为稳定，能除去不必要的杂音，带来更好的效果。

我们尝试对两行业间的轮动也采取同样策略，收益率图像如下：



可以看出，该方法的效果也好于马尔科夫方法。

三、反思和改进

在本研究中，一个比较明显的问题是，隐马尔可夫模型只在股债轮动中比较有效，在股票行业间、板块间的轮动效果都不尽人意，究其原因，我们认为是GEYR指标本身包含了丰富的股票市值、价格、盈利能力的信息，故直接用其进行判定就可以，而如果采取马尔科夫模型，在GEYR-隐状态-配置的过程中，在搞不清楚隐状态含义的情况下，可能会丢失部分信息，而利用GEYR阈值进行配置的方法可能更好地保留了信息，所以会取得更好的效果。

另外，本研究有频繁的调仓行为，故可能产生较高的手续费，并且债券流动性较差，可能会有较大滑点，但本研究中并未进行考虑，这些问题应该在未来更深入的研究中加以计算。

参考文献

《GEYR 策略在股票债券配置中的运用》，胡倩，海通证券
《Tutorial—hmmlearn 0.2.1 documentation》

【量化课堂】RSRS(阻力支撑相对强度)择时策略（下）

概述

本篇基于光大证券研报《基于阻力支撑相对强度(RSRS)的市场择时》，在上一期的基础上，进一步给出了RSRS右偏标准分交易策略，以及RSRS指标配合其他量化指标（价格、交易量相关性）的交易策略。

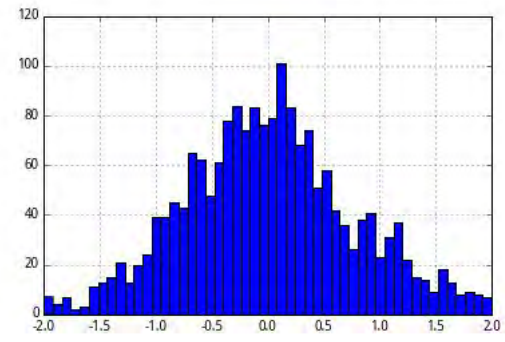
作者：一路向上
编辑：肖睿

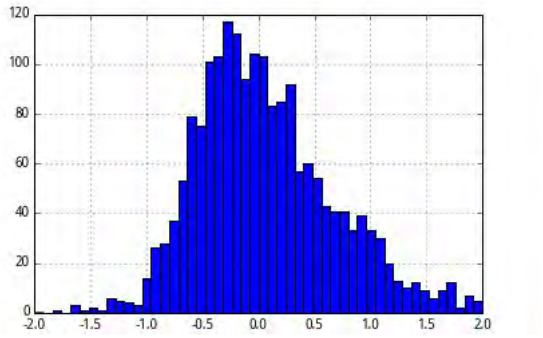
本文由JoinQuant量化课堂推出。难度为入门，深度为level-0。

四、RSRS右偏标准分交易策略

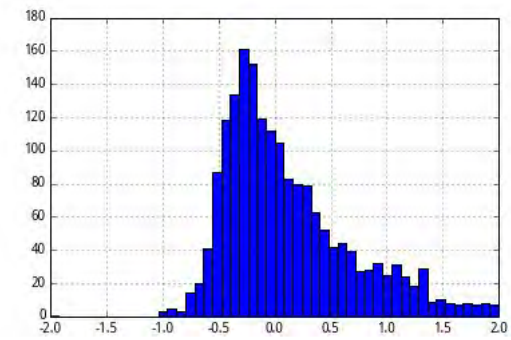
在使用斜率量化阻力支撑相对强度时，其量化效果很大程度上受拟合本身效果的影响。我们将RSRS标准分与决定系数相乘得到RSRS修正标准分，以此降低绝对值很大，但拟合效果很差的RSRS标准分对策略的影响。通过这种变换，修正RSRS标准分有明显的向正态修正的效果。

如下图所示，左图为RSRS标准分分布情况，其尾部较厚；右图为RSRS修正标准分，可以看出修正后更加接近于正态分布。





然而，修正标准分在预测性上的改善效果主要体现于标准分左侧，在做多策略中，左侧预测性改善对择时策略帮助并不大。我们将RSRS修正标准分与RSRS斜率值相乘得到RSRS右偏标准分。其分布如下图，可以看出右偏标准分左侧较薄，而右侧较厚。



同时给出如下RSRS右偏标准分交易策略（取值来自研报）：
1、计算RSRS右偏标准分$RSRS < em > rightdev$（N = 16, M = 300）
2、若$RSRS < /em > rightdev$大于$S < em > buy$时，全仓买入；若$RSRS < /em > rightdev$小于$S < em > sell$时，卖出平仓。（$S < /em > buy = 0.7, S_{sell} = -0.7$）

采用研报的参数值，得到策略情况如下：

基准和标的股票仍然选定沪深300指数，时间选取2010-01-01至2017-12-17。总收益率为209.03%，年化收益15.70%，胜率0.6，最大回撤24.989%，开平仓29次。

通过不断尝试调整参数值，我们希望得到一个更好的策略表现。N取值在10-30之间时，N=18依然表现最佳；但是M取值较大时，收益率有提升的趋势，特别当M从750变为800时，总收益率从190%跃升至250%左右。由于平台的数据从2005年开始，M值设为1200时将会超出数据范围。在可以取到的M值中，M = 1100时策略表现最佳：

总收益率为290.12%，年化收益19.24%，胜率0.636，最大回撤17.775%，开平仓28次。

无论从收益率、胜率上，还是从最大回撤上，上述参数值都比研报中给出的参数值表现要好。这可能由于计算标准分的方法不同。其研报中计算的标准分并没有用到时间区间之外的数据：M=600时，回测前600天并没有600个斜率数据来计算标准分，比如回测第50天就只使用了50个斜率数据进行标准化，这使得回测前期信号相对不稳定。而我们计算标准分时，固定了移动窗口的值，所有日期的标准分采用了相同个数的数据来计算。

五、RSRS指标配合量价数据优化策略

为了规避掉在大熊市买入的情况，我们尝试在开仓时，加入一个对目前市场状态的判断，过滤掉下跌行情中的开仓。下面给出基于价格趋势和基于交易量趋势两种优化。

1. RSRS指标 + 价格优化交易策略

一个直接的想法是从近期历史价格趋势进行判断，在回测中，使用前1日的20日均线值和3日前的20日均线值的相对大小牌判断近期市场状态。策略如下（取值来自研报）：
1、计算RSRS右偏标准分指标$RSRS < em > rightdev$。（N = 18, M = 600）
2、若$RSRS < /em > rightdev$大于$S < em > buy$，同时满足前1日的MA20的值大于前3日的MA20的值，则全仓买入。（$S < /em > buy = 0.7$）
3、若$RSRS < em > rightdev$小于$S < /em > sell$，同时满足前1日的MA20的值小于前3日的MA20的值，则卖出平仓。（$S_{sell} = -0.7$）

采用研报的参数值，得到策略情况如下：

总收益率为200.22%，年化收益15.27%，胜率0.818，最大回撤17.616%，开平仓12次。

调整参数发现，N = 18, M = 200时，策略表现略好于之前：

总收益率为213.85%，年化收益15.93%，胜率0.889，最大回撤15.156%，开平仓10次。

2. RSRS指标 + 交易量相关性优化交易策略

很多研究表明市场涨跌与交易量有明显的正相关性，因此我们尝试采用交易量与修正标准分的相关性来过滤误判信号。在相关性为正时，给出买入信号。策略如下（取值来自研报）：
1、计算RSRS右偏标准分指标$RSRS < em > rightdev$。（N = 18, M = 600）
2、若$RSRS < /em > rightdev$大于$S < em > buy$，同时满足前10日交易量与RSRS修正标准分之间的相关性为正，则全仓买入。（$S < /em > buy = 0.7$）
3、若$RSRS < em > rightdev$小于$S < /em > sell$，则卖出平仓。（$S_{sell} = -0.7$）

采用研报的参数值，得到策略情况如下：

总收益率为169.43%，年化收益13.67%，胜率0.636，最大回撤12.210%，开平仓22次。

调整参数发现，N = 18, M = 200时，从收益率上看远好于研报参数值：

总收益率为238.05%，年化收益17.05%，胜率0.714，最大回撤16.459%，开平仓20次。

后记

对于上述介绍的策略，还可以尝试配合其他量化指标过滤交易信号，也可以进一步调整买入卖出阈值（$S < em > buy, S < /em > sell$），还可以将第五部分中的RSRS右偏标准分换为标准分或修正标准分，不断尝试是否能够得到更好的策略表现。

v1.0，2017-12-18，文章上线

注：回测部分是RSRS右偏标准分交易策略(N = 18, M = 1100)

【量化课堂】RSRS(阻力支撑相对强度)择时策略（上）

概述

本篇基于光大证券研报《基于阻力支撑相对强度(RSRS)的市场择时》，给出了RSRS斜率指标择时，以及在斜率基础上的标准化指标择时策略。

作者：一路向上
编辑：肖睿

本文由JoinQuant量化课堂推出。难度为入门，深度为level-0。

一、 阻力支撑相关概念

阻力位是指指标价格上涨时可能遇到的压力，即交易者认为卖方力量开始反超买方，从而价格难以继续上涨或从此回调下跌的价位；支撑位则是交易者认为买方力量开始反超卖方，从而止跌或反弹上涨的价位。

常见的确定阻力支撑位的方法有，布林带上下轨突破策略（突破上轨建仓买入，突破下轨卖出平仓）和均线策略（如超过20日均线建仓买入，低于20日均线卖出平仓）。然而，布林带突破策略在震荡期间出现了持续亏损，均线策略交易交易成本巨大，且在震荡期间的回撤很大。

二、阻力支撑相对强度（RSRS）

阻力支撑相对强度(Resistance Support Relative Strength, RSRS)是另一种阻力位与支撑位的运用方式，它不再把阻力位与支撑位当做一个定值，而是看做一个变量，反应了交易者对目前市场状态顶底的一种预期判断。

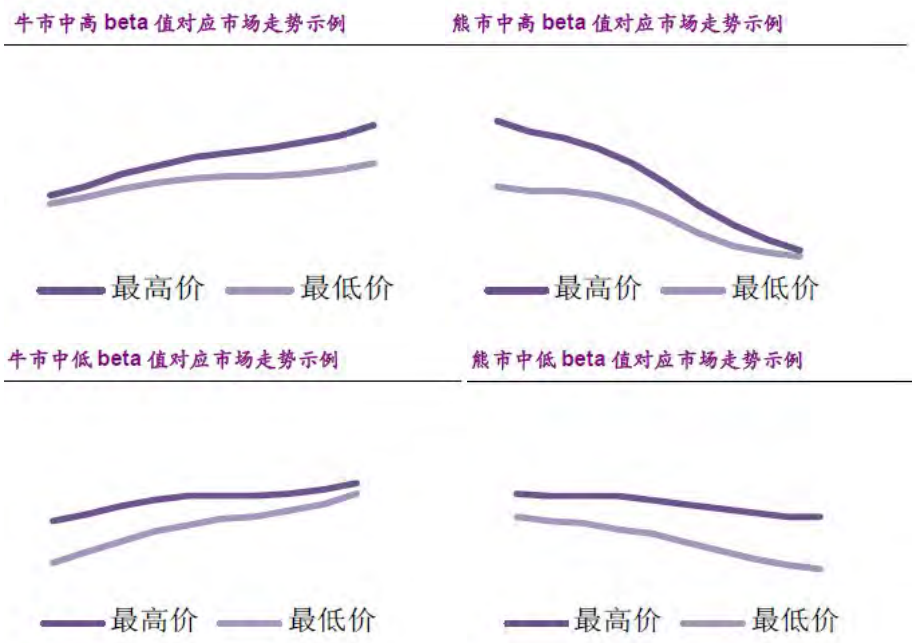
我们按照不同市场状态分类来说明支撑阻力相对强度的应用逻辑：

1.市场在上涨牛市中：
如果支撑明显强于阻力，牛市持续，价格加速上涨
如果阻力明显强于支撑，牛市可能即将结束，价格见顶
2. 市场在震荡中：
如果支撑明显强于阻力，牛市可能即将启动
如果阻力明显强于支撑，熊市可能即将启动
3.市场在下跌熊市中：
如果支撑明显强于阻力，熊市可能即将结束，价格见底
如果阻力明显强于支撑，熊市持续，价格加速下跌

每日最高价和最低价是一种阻力位与支撑位，它是当日全体市场参与者的交易行为所认可的阻力与支撑。一个很自然的想法是建立最高价和最低价的线性回归，并计算出斜率。即：

$$high = \alpha + \beta \cdot low + \epsilon, \epsilon \sim N(0, \sigma^2)$$

当斜率值很大时，支撑强度大于阻力强度。在牛市中阻力渐小，上方上涨空间大；在熊市中支撑渐强，下跌势头欲止。
当斜率值很小时，阻力强度大于支撑强度。在牛市中阻力渐强，上涨势头渐止；在熊市中支撑渐送，下方下跌空间渐大。
如图所示：



图片取自光大证券研报

三、阻力支撑相对强度（RSRS）指标择时策略

第一种方法是直接将斜率作为指标值。当日RSRS斜率指标择时策略如下：
1、取前N日最高价与最低价序列。（N = 18）
2、将两个序列进行OLS线性回归。
3、将拟合后的β值作为当日RSRS斜率指标值。
4、当RSRS斜率大于$S < em > buy$时，全仓买入，小于$S < /em > sell$时，卖出平仓。（$S < em > buy = 1, S < /em > sell = 0.8$）

由于市场处于不同时期时，斜率的均值有比较大的波动。因此，直接采用斜率均值作为择时指标并不太合适。我们尝试下面的方法。

第二种方法是在斜率基础上进行标准化，取标准分作为指标值。RSRS斜率标准分指标择时策略如下：
1、取前M日的RSRS斜率时间序列。（M = 600）
2、计算当日RSRS斜率的标准分$RSRS < em > std$:

$$RSRS < /em > std = \frac{RSRS - \mu < em > M}{\sigma_M}$$

其中μ_M为前M日的斜率均值，σ_M为前M日的标准差。
3、若$RSRS < /em > std$大于$S < em > buy$，则全仓买入；若$RSRS < /em > std$小于$S < em > sell$，则卖出平仓。（$S < /em > buy = 0.7, S_{sell} = -0.7$）

注：benchmark 和标的股票均为沪深300指数，尝试N取自10-30，M取自400-800，发现N=18，M=600时收益率最高。

该策略从2010-01-01至2017-12-15，开平仓35次，总收益186.67%，年化收益14.58%，胜率0.556，盈亏比4.568，最大回撤20.031%。

后记

下期将介绍基于RSRS标准分基础上的进一步优化。

v1.0，2017-12-16，文章上线

【量化课堂】高送转预测 -- 逻辑回归与支持向量机

前言

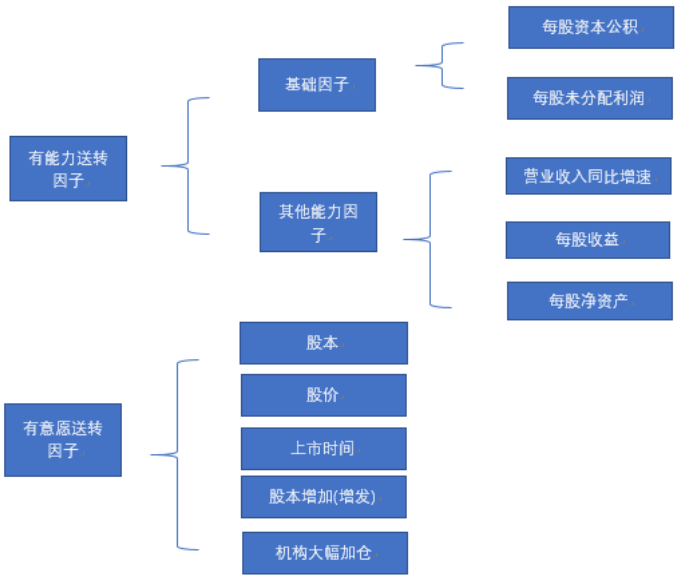
亲爱的朋友们大家好，又到了年末时节，每年这个时候到下一年3、4月份都是A股各上市公司的业绩预报、分配预报、年报正式报告的发布期，有意进行股票高送转的上市公司也会在这个时间段内发布高送转预案。众所周知，年报高送转题材的炒作是每年A股市场的开年大菜、必点精品，不少主题炒作投资者对此尤为热衷，并且随着A股市场的扩容，每年发布高送转预案的公司数量逐年增加。因此，为了更好地把握年报送转行情，本文将运用逻辑回归与SVM搭建高送转股票的预测模型，并对2017年年报高送转股票进行预测。

本文由 JoinQuant 量化课堂推出 。难度标签为入门，理解深度标签：level-0

作者： 李伟豪
编辑： 肖睿

导语

影响上市公司实施高送转的因素有很多，包括市场环境、财务状态、股票价格和监管层政策等。不少学者发现，高积累、高股价和股本小是上市公司实施"高送转"的先决条件，次新股、股本扩张与业绩一致增长的股票高送转的意愿比较强。这里，我们将影响高送转的因子分为以下几类：



虽然每年中报发布时候，也有一些股票发布高送转方案，但是总的来说，中报高送转的数量要远远小于年报高送转的股票。因此，这里我们只研究年报高送

转的数据。

数据集

1.因变量：

高送转（0-1变量），在这里，我们将送股比例+转股比例大于1的股票定义为高送转股票，即每十股送红股+转增股的加总大于10股，实施高送转的股票该变量为1，否则为0.

2.自变量：

初步选取每股资本公积+留存收益，每股总资产，总股本，每股盈余，营业收入同比增速，前20个交易日平均价，上市天数作为自变量，其中：每股资本公积+留存收益，每股总资产，总股本，每股盈余，营业收入同比增速的数据来源为每年的三季报，前20个交易日平均价，上市天数以每年11月1日为基准进行计算。
这里简单解释一下为什么不用定增指标，因为定增的股票基本不会是次新股，而在次新指标将结果往1拉的时候，这里的定增为0又会把结果往反方向拉回，特别是在本次使用数据集中送转与不送转、定增与不定增、次新与否的样本量高度不平衡的情况下，影响较大。当然我们也做了相应的检测，发现在加入了定增指标后预测效果确实不如之前。不过，定增与否的指标可以在选出可能送转股票后进一步筛选时使用。

3.特征选择：

(1).基于树的特征选择

	importance
per_zbgj_wflr	0.152848
per_TotalOwnerEquity	0.147104
capitalization	0.132174
eps	0.132444
revenue_growth	0.133320
mean_price	0.133576
days_listed	0.168533

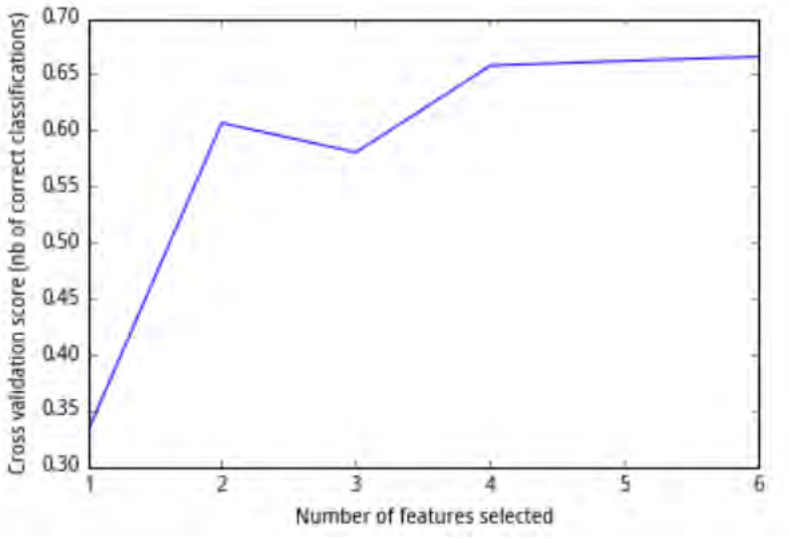
各变量对因变量重要性

	per_zbgj_wflr	per_TotalOwnerEquity	capitalization	eps	revenue_growth	mean_price	days_listed
per_zbgj_wflr	1.000000	0.988297	-0.032620	0.462543	0.011974	0.319627	-0.184551
per_TotalOwnerEquity	0.988297	1.000000	-0.009400	0.478124	-0.000433	0.323599	-0.132749
capitalization	-0.032620	-0.009400	1.000000	0.028189	-0.018424	-0.056660	0.005137
eps	0.462543	0.478124	0.028189	1.000000	0.075522	0.282604	-0.018372
revenue_growth	0.011974	-0.000433	-0.018424	0.075522	1.000000	0.038924	-0.051725
mean_price	0.319627	0.323599	-0.056660	0.282604	0.038924	1.000000	-0.055522
days_listed	-0.184551	-0.132749	0.005137	-0.018372	-0.051725	-0.055522	1.000000

各变量相关系数

可以看到，各个变量对因变量的重要性都相差不大，但是从相关性上来看，每股资本公积+未分配利润与每股净资产相关性非常高，这也很容易理解，因为净资产=股东权益=股本+资本公积+盈余公积+未分配利润。因此，这里我们将每股净资产这个变量删除，再做基于递归特征消除的特征判断。

(2).基于RFE的特征选择



RFE可以衡量随着特征数目的增加，模型整体分类准确率的变化，以此来确定最优的特征数目。直到自变量数增加到6，模型的交叉检验准确率一直在上升，因此，我们保留剩下的6个因变量，作为我们预测高送转的使用变量。

基于逻辑回归的预测

训练集：
假设当前时间为t年，则以t-1与t-2两年的数据作为训练集
测试集：
以t年数据作为测试集
例如：以2013年、2014年的股票指标以及是否高送转训练模型，将得到的参数用于2015年的测试集，得出15年测试集送转概率。

评估方法：
以训练集数据拟合模型得到参数后带入测试集数据，将测试集数据按股票送转的概率排序，分别取概率最大的前10、25、50
100、200只股票，计算这些股票中真实发生高送转的概率。

	10	25	50	100	200
accuracy_scale	0.3	0.20	0.24	0.22	0.260
accuracy_minmax	0.3	0.20	0.24	0.22	0.260
accuracy_no	0.1	0.16	0.32	0.32	0.275

2013年模型准确性

	10	25	50	100	200
accuracy_scale	0.6	0.60	0.58	0.55	0.465
accuracy_minmax	0.6	0.60	0.58	0.55	0.465
accuracy_no	0.7	0.48	0.46	0.47	0.375

2014年模型准确性

	10	25	50	100	200
accuracy_scale	0.9	0.68	0.64	0.65	0.595
accuracy_minmax	0.9	0.68	0.64	0.65	0.595
accuracy_no	0.9	0.64	0.66	0.53	0.445

2015年模型准确性

	10	25	50	100	200
accuracy_scale	0.8	0.84	0.84	0.62	0.460
accuracy_minmax	0.8	0.84	0.84	0.62	0.460
accuracy_no	0.4	0.20	0.20	0.28	0.245

2016年模型准确性

表中纵坐标下，scale ， minmax 表示两种不同的标准化方法，scale为均值方差标准化，minmax为0-1标准化（即将数据的规模压缩到0-1之间），no表示不做标准化。
accuracy_scale表示在均值方差标准化方法下，模型的预测准确率，accuracy_minmax表示在0-1标准化方法下模型的预测准确率。横坐标表示所选的送转概率最大的前10只，前25只，前50只等等。以第四个图为例，第一排第一列表示在预测送转概率最大的前十只股票中，有8(=0.8*10)只确实进行了高送转。
在逻辑回归中，模型的准确性对于两个不同的标准化方法表现稳健，但标准化下的效果要好于不标准化。从准确度上来看，2013年预测准确性较差，之后预测准确度逐渐变高，特别是16年，模型的前25只、前50只预测准确度都高达84%

基于SVM的预测

	10	25	50	100	200
accuracy_scale	0.6	0.36	0.30	0.30	0.265
accuracy_minmax	0.3	0.20	0.14	0.22	0.230
accuracy_no	0.0	0.00	0.12	0.07	0.055

2013年模型准确性

	10	25	50	100	200
accuracy_scale	0.7	0.64	0.66	0.51	0.475
accuracy_minmax	0.8	0.56	0.56	0.52	0.470
accuracy_no	0.2	0.12	0.08	0.11	0.065

2014年模型准确性

	10	25	50	100	200
accuracy_scale	0.5	0.68	0.62	0.56	0.545
accuracy_minmax	0.8	0.80	0.66	0.59	0.540
accuracy_no	0.2	0.12	0.14	0.12	0.130

2015年模型准确性

	10	25	50	100	200
accuracy_scale	0.7	0.60	0.66	0.54	0.410
accuracy_minmax	0.7	0.76	0.74	0.55	0.395
accuracy_no	0.1	0.08	0.08	0.05	0.040

2016年模型准确性

基于SVM模型的预测结果波动性较大，而且易受标准化方法的影响，13、14年scale标准化效果更优，而在15、16年minmax标准化效果却好于前者。与逻辑回归模型相比，SVM模型在13、14年效果优于逻辑回归，但在15年，scale标准化下弱于逻辑回归，minmax标准化下优于逻辑回归，16年各种标准化方法下都相比较差。但是明显看出，模型在13年的预测准确度要大幅低于后面几年，此结果同逻辑回归模型一致。
SVM模型时而更优时而更差，要在这两个模型中选择其一的话不好抉择，因此不如将两个模型联合起来，整合预测。

基于逻辑回归&SVM的预测

方法（排序打分）：
1.分别以逻辑回归与SVM进行高送转概率预测，按概率从大到小进行排序，基于排序给于相应分数，第一为1分，第二位2分。
2.将两个模型得分相加，按得分从小到大排序

	10	25	50	100	200
score_scale	0.4	0.36	0.32	0.28	0.275
score_minmax	0.4	0.24	0.18	0.23	0.250
score_no	0.1	0.08	0.24	0.32	0.270

2013年模型准确性

	10	25	50	100	200
score_scale	0.8	0.72	0.58	0.53	0.485
score_minmax	0.7	0.72	0.58	0.55	0.500
score_no	0.5	0.52	0.46	0.46	0.380

2014年模型准确性

	10	25	50	100	200
score_scale	0.8	0.84	0.72	0.64	0.560
score_minmax	0.7	0.80	0.68	0.62	0.585
score_no	0.9	0.60	0.64	0.53	0.450

2015年模型准确性

	10	25	50	100	200
score_scale	0.9	0.88	0.8	0.63	0.45
score_minmax	0.9	0.96	0.8	0.63	0.40
score_no	0.4	0.20	0.2	0.27	0.24

2016年模型准确性

将两个模型整合之后，预测准确性更加稳定，相对于SVM模型，受标准化方法的影响也更小。在对于2017年的预测中，我们使用联合的模型，使用scale标准化方法。

2017年预测结果

stock	score_logit	score_SVM	score	rank	stock	score_logit	score_SVM	score	rank
002627.XSHE	3	13	16	1	002788.XSHE	66	66	132	26
002872.XSHE	16	9	25	2	603979.XSHG	122	11	133	27
300184.XSHE	21	12	33	3	603033.XSHG	102	32	134	28
002682.XSHE	20	14	34	4	300485.XSHE	81	58	139	29
603385.XSHG	26	18	44	5	603556.XSHG	76	63	139	30
002772.XSHE	58	5	63	6	603558.XSHG	92	47	139	31
603878.XSHG	38	27	65	7	002541.XSHE	11	134	145	32
603599.XSHG	50	24	74	8	300391.XSHE	56	90	146	33
300537.XSHE	61	15	76	9	300407.XSHE	45	102	147	34
002164.XSHE	67	16	83	10	603535.XSHG	128	22	150	35
300510.XSHE	51	41	92	11	603367.XSHG	130	21	151	36
002564.XSHE	60	37	97	12	300338.XSHE	77	78	155	37
603839.XSHG	54	44	98	13	002377.XSHE	5	151	156	38
603928.XSHG	80	19	99	14	002574.XSHE	131	29	160	39
603926.XSHG	75	25	100	15	300178.XSHE	113	51	164	40
603116.XSHG	41	62	103	16	002791.XSHE	71	95	166	41
603167.XSHG	64	45	109	17	002746.XSHE	52	115	167	42
603086.XSHG	31	82	113	18	603226.XSHG	135	36	171	43
601900.XSHG	110	7	117	19	002740.XSHE	101	71	172	44
603768.XSHG	86	35	121	20	603117.XSHG	120	55.5	175.5	45
603035.XSHG	89	33	122	21	603630.XSHG	117	61	178	46
300349.XSHE	24	99	123	22	603017.XSHG	104	80	184	47
300569.XSHE	79	46	125	23	300587.XSHE	136	49	185	48
603980.XSHG	112	17	129	24	002734.XSHE	47	141	188	49
603586.XSHG	84	48	132	25	603225.XSHG	55	136	191	50

总结

通过上述模型，我们:
1.比较了两种模型预测高送转的优劣
2.整合两个模型为更稳定模型
3.较为有效的预测第二年年报高送转股票。
本篇提供了一种预测高送转股票的思路，对于进行高送转题材投资有一定参考作用，我们也将继续研究对于预测所得高送转大概率股票的策略研究，后续将继续更新。

注：以上历年送转数据均来自外部数据库，如需要可以通过链接下载

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime

import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/externals/joblib/_multiprocessing_helpers.py:28: UserWarning: [Errno 30] Read-only fil
  warnings.warn('%s.  joblib will operate in serial mode' % (e,))
```

In [2]:

```
#####获取年报高送转data
div_data = pd.read_csv(r'Div_data.csv',index_col=0)
##只留年报数据
div_data['type'] = div_data['endDate'].map(lambda x:x[-5:])
div_data['year'] = div_data['endDate'].map(lambda x:x[0:4])
div_data_year = div_data[div_data['type'] == '12-31']
div_data_year = div_data_year[['secID','year','publishDate', 'recordDate','perShareDivRatio',
       'perShareTransRatio']]
div_data_year.columns = ['stock','year','pub_date','execu_date','sg_ratio','zg_ratio']
div_data_year.fillna(0,inplace = True)
div_data_year['sz_ratio'] = div_data_year['sg_ratio']+div_data_year['zg_ratio']
div_data_year['gsz'] = 0
div_data_year.ix[div_data_year['sz_ratio'] >=1,'gsz'] = 1
```

In [3]:

```
####获取q1q2q3已经高送转data
q123_already_szdata = pd.read_csv(r'q1q2q3_already_sz_stock.csv',index_col=0)
```

In [4]:

```
###获取因变量因子

"""

每股资本公积、每股未分配利润、每股净资产、每股收益
营业收入同比增速、股本数量、股票价格、是否为次新股、上市日天数

"""
###将一些指标转变为每股数值
def get_perstock_indicator(need_indicator,old_name,new_name,sdate):
    target = get_fundamentals(query(valuation.code,valuation.capitalization,need_indicator),statDate = sdate)
    target[new_name] = target[old_name]/target['capitalization']/10000


    return target[['code',new_name]]

###获取每股收益、股本数量
def get_other_indicator(sdate):
    target = get_fundamentals(query(valuation.code,valuation.capitalization,\
                                    indicator.inc_revenue_year_on_year,\
                                    indicator.eps
                                    ),statDate = sdate)
    target.rename(columns={'inc_revenue_year_on_year':'revenue_growth'},inplace = True)

    target['capitalization'] = target['capitalization']*10000

    return target[['code','capitalization','eps','revenue_growth']]


###获取一个月收盘价平均值
def get_bmonth_aprice(code_list,startdate,enddate):
    mid_data = get_price(code_list, start_date=startdate, end_date=enddate,\
              frequency='daily', fields='close', skip_paused=False, fq='pre')
    mean_price = pd.DataFrame(mid_data['close'].mean(aixs = 0),columns=['mean_price'])
    mean_price['code'] =mean_price.index
    mean_price.reset_index(drop = True,inplace =True)

    return mean_price[['code','mean_price']]



###判断是否为次新股（判断标准为位于上市一年之内）
def judge_cxstock(date):
    mid_data = get_all_securities(types=['stock'])
    mid_data['start_date'] = mid_data['start_date'].map(lambda x:x.strftime("%Y-%m-%d"))
    shift_date = str(int(date[0:4])-1)+date[4:]
    mid_data['1year_shift_date'] = shift_date
    mid_data['cx_stock'] = 0
    mid_data.ix[mid_data['1year_shift_date']<=mid_data['start_date'],'cx_stock'] = 1
    mid_data['code'] = mid_data.index
    mid_data.reset_index(drop = True,inplace=True)

    return mid_data[['code','cx_stock']]

###判断是否增发了股票（相比于一年前）
def judge_dz(sdate1,sdate2):
    target1 = get_fundamentals(query(valuation.code,valuation.capitalization,balance.capital_reserve_fund),statDate = sdate1)
    target1['CRF_1'] = target1[ 'capital_reserve_fund']/target1['capitalization']/10000
    target2 = get_fundamentals(query(valuation.code,valuation.capitalization,balance.capital_reserve_fund),statDate = sdate2)
    target2['CRF_2'] = target2[ 'capital_reserve_fund']/target2['capitalization']/10000

    target = target1[['code','CRF_1']].merge(target2[['code','CRF_2']],on=['code'],how='outer')
    target['CRF_change'] = target['CRF_1'] - target['CRF_2']
    target['dz'] = 0
    target.ix[target['CRF_change']>1,'dz']=1
    target.fillna(0,inplace = True)

    return target[['code','dz']]

###判断上市了多少个自然日
def get_dayslisted(year,month,day):

    mid_data = get_all_securities(types=['stock'])
    date = datetime.date(year,month,day)
    mid_data['days_listed'] = mid_data['start_date'].map(lambda x:(date -x).days)
    mid_data['code'] = mid_data.index
```

```
    mid_data.reset_index(drop = True,inplace=True)

    return mid_data[['code','days_listed']]



def get_yearly_totaldata(statDate,statDate_before,mp_startdate,mp_enddate,year,month,day):

    """
    输入：所需财务报表期、20日平均股价开始日期、20日平均股价结束日期
    输出：合并好的高送转数据 以及 财务指标数据
    """
    per_zbgj = get_perstock_indicator(balance.capital_reserve_fund,'capital_reserve_fund','per_CapitalReserveFund',statDate)
    per_wflr = get_perstock_indicator(balance.retained_profit,'retained_profit','per_RetainProfit',statDate)
    per_jzc = get_perstock_indicator(balance.equities_parent_company_owners,'equities_parent_company_owners','per_TotalOwnerEquity',s

    other_indicator = get_other_indicator(statDate)
    code_list = other_indicator['code'].tolist()
    mean_price = get_bmonth_aprice(code_list,mp_startdate,mp_enddate)
    cx_signal = judge_cxstock(mp_enddate)
    dz_signal = judge_dz(statDate,statDate_before)
    days_listed = get_dayslisted(year,month,day)

    chart_list = [per_zbgj,per_wflr,per_jzc,other_indicator,mean_price,cx_signal,dz_signal,days_listed]
    for chart in chart_list:
        chart.set_index('code',inplace = True)

    independ_vari = pd.concat([per_zbgj,per_wflr,per_jzc,other_indicator,mean_price,cx_signal,dz_signal,days_listed],axis = 1)
    independ_vari['year'] = str(int(statDate[0:4]))
    independ_vari['stock'] = independ_vari.index
    independ_vari.reset_index(drop=True,inplace =True)

    total_data = pd.merge(div_data_year,independ_vari,on = ['stock','year'],how = 'inner')
    total_data['per_zbgj_wflr'] = total_data['per_CapitalReserveFund']+total_data['per_RetainProfit']


    return total_data
```

In [5]:

```
gsz_2016 = get_yearly_totaldata('2016q3','2015q3','2016-10-01','2016-11-01',2016,11,1)
gsz_2015 = get_yearly_totaldata('2015q3','2014q3','2015-10-01','2015-11-01',2015,11,1)
gsz_2014 = get_yearly_totaldata('2014q3','2013q3','2014-10-01','2014-11-01',2014,11,1)
gsz_2013 = get_yearly_totaldata('2013q3','2012q3','2013-10-01','2013-11-01',2013,11,1)
gsz_2012 = get_yearly_totaldata('2012q3','2011q3','2012-10-01','2012-11-01',2012,11,1)
gsz_2011 = get_yearly_totaldata('2011q3','2010q3','2011-10-01','2011-11-01',2011,11,1)


###不希望过大的营业收入增长影响结果，实际上营收增长2000%和增长300%对是否送转结果影响差别不大
for data in [gsz_2011,gsz_2012,gsz_2013,gsz_2014,gsz_2015,gsz_2016]:
    data.ix[data['revenue_growth']>300,'revenue_growth'] =300
```

特征选择¶

在每股资本公积+留存收益，每股总资产，总股本，每股盈余，营业利润同比增速，前20个交易日平均价，上市天数 这些变量中进行选择

In [32]:

```
###基于树的判断
traindata = pd.concat([gsz_2011,gsz_2012,gsz_2013,gsz_2014,gsz_2015,gsz_2016],axis = 0)
traindata.dropna(inplace = True)

x_traindata = traindata[['per_zbgj_wflr',\
       'per_TotalOwnerEquity', 'capitalization', 'eps', 'revenue_growth',\
       'mean_price', 'days_listed']]

y_traindata = traindata[['gsz']]
X_trainScale = preprocessing.scale(x_traindata)

from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()
model.fit(X_trainScale,y_traindata)
print(pd.DataFrame(model.feature_importances_.tolist(),index =['per_zbgj_wflr',\
       'per_TotalOwnerEquity', 'capitalization', 'eps', 'revenue_growth',\
       'mean_price', 'days_listed'],columns = ['importance'] ))
```

```
                     importance
per_zbgj_wflr          0.152848
```

```
per_TotalOwnerEquity    0.147104
capitalization          0.132174
eps                     0.132444
revenue_growth          0.133320
mean_price              0.133576
days_listed             0.168533
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:13: DataConversionWarning: A column-vector y was passed when a 1d array
```

各变量相关性

In [33]:

```
x_traindata.corr()
```

Out[33]:

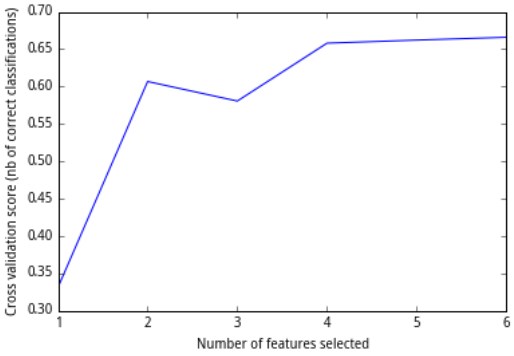
	per_zbgj_wflr	per_TotalOwnerEquity	capitalization	eps	revenue_growth	mean_pric
per_zbgj_wflr	1.000000	0.988297	-0.032620	0.462543	0.011974	0.319627
per_TotalOwnerEquity	0.988297	1.000000	-0.009400	0.478124	-0.000433	0.323599
capitalization	-0.032620	-0.009400	1.000000	0.028189	-0.018424	-0.056660
eps	0.462543	0.478124	0.028189	1.000000	0.075522	0.282604
revenue_growth	0.011974	-0.000433	-0.018424	0.075522	1.000000	0.038924
mean_price	0.319627	0.323599	-0.056660	0.282604	0.038924	1.000000
days_listed	-0.184551	-0.132749	0.005137	-0.018372	-0.051725	-0.055522

可以看到每股净资产与每股资本公积+未分配利润 相关度非常高，因此舍去每股净资产

In [34]:

```
###基于RFE（递归特征消除） 判断
traindata = pd.concat([gsz_2011,gsz_2012,gsz_2013,gsz_2014,gsz_2015,gsz_2016],axis = 0)
traindata.dropna(inplace = True)
x_traindata = traindata[['per_zbgj_wflr',\
        'capitalization', 'eps', 'revenue_growth',\
       'mean_price', 'days_listed']]
y_traindata = traindata[['gsz']]
X_trainScale = preprocessing.scale(x_traindata)

svc = SVC(C=1.0,class_weight='balanced',kernel='linear',probability=True)
rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(2),
              scoring='accuracy')
X_trainScale = preprocessing.scale(x_traindata)
rfecv.fit(X_trainScale,y_traindata)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
/opt/conda/lib/python3.4/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
```



逻辑回归预测¶

In [6]:

```
def get_prediction(x_traindata,y_traindata,x_testdata,standard='scale'):
    if standard == 'scale':
        #均值方差标准化
        X_trainScale = preprocessing.scale(x_traindata)
        scaler = preprocessing.StandardScaler().fit(x_traindata)
        X_testScale = scaler.transform(x_testdata)
    elif standard =='minmax':
        #min_max标准化
        min_max_scaler = preprocessing.MinMaxScaler()
        X_trainScale = min_max_scaler.fit_transform(x_traindata)
        X_testScale = min_max_scaler.transform(x_testdata)
    elif standard =='no':
        #不标准化
        X_trainScale = x_traindata
        X_testScale = x_testdata

    ###考虑到样本中高送转股票与非高送转股票样本的不平衡问题，这里选用调整的class_weight
    model = LogisticRegression(class_weight='balanced',C=1e9)
    model.fit(X_trainScale, y_traindata)
    predict_y = model.predict_proba(X_testScale)

    return predict_y

def assess_classification_result(traindata,testdata,variable_list,q123_sz_data,date1,date2,function = get_prediction):

    traindata.dropna(inplace = True)
    testdata.dropna(inplace = True)

    x_traindata = traindata.loc[:,variable_list]
    y_traindata = traindata.loc[:,'gsz']
    x_testdata = testdata.loc[:,variable_list]
    y_testdata = testdata.loc[:,'gsz']

    total = testdata.loc[:,['stock','gsz']]
    for method in ['scale','minmax','no']:
        predict_y = function(x_traindata,y_traindata,x_testdata,standard=method)
        total['predict_prob_'+method] = predict_y[:,1]

    ###过滤今年前期已经送转过的股票
    q123_stock = q123_sz_data['stock'].tolist()
    total_filter = total.loc[total['stock'].isin(q123_stock)==False]

    ###过滤ST股票
    stock_list = total_filter['stock'].tolist()
    st_data = pd.DataFrame(get_extras('is_st',stock_list , start_date=date1, end_date=date2, df=True).iloc[-1,:])
    st_data.columns =['st_signal']
    st_list = st_data[st_data['st_signal']==True]
    total_filter = total_filter[total_filter['stock'].isin(st_list)==False]

    ###衡量不同选股个数、不同标准化方法下的 预测准度
    result_dict ={}
    for stock_num in [10,25,50,100,200]:
        accuracy_list = []
        for column in total_filter.columns[2:]:
            total_filter.sort(column,inplace = True,ascending = False)
            dd = total_filter[:stock_num]
            accuracy = len(dd[dd['gsz']==1])/len(dd)
            accuracy_list.append(accuracy)
        result_dict[stock_num] = accuracy_list

    result = pd.DataFrame(result_dict,index =['accuracy_scale','accuracy_minmax','accuracy_no'])
```

```
    return result,total_filter
```

In [7]:

```
### 2013年预测结果
traindata = pd.concat([gsz_2011,gsz_2012],axis=0)
testdata = gsz_2013.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2013)&(q123_already_szdata['gs']>0)]
result_2013,total_2013 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,'2013-10-25','2013-11-01')
print (result_2013)
```

```
                10    25    50    100    200
accuracy_scale   0.3  0.20  0.24  0.22  0.260
accuracy_minmax  0.3  0.20  0.24  0.22  0.260
accuracy_no      0.1  0.16  0.32  0.32  0.275
```

In [8]:

```
###2014年预测结果
traindata = pd.concat([gsz_2012,gsz_2013],axis=0)
testdata = gsz_2014.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2014)&(q123_already_szdata['gs']>0)]
result_2014,total_2014 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,'2014-10-25','2014-11-01')
print (result_2014)
```

```
                10    25    50    100    200
accuracy_scale   0.6  0.60  0.58  0.55  0.465
accuracy_minmax  0.6  0.60  0.58  0.55  0.465
accuracy_no      0.7  0.48  0.46  0.47  0.375
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
```

In [9]:

```
####2015年预测结果
traindata = pd.concat([gsz_2013,gsz_2014],axis=0)
testdata = gsz_2015.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2015)&(q123_already_szdata['gs']>0)]
result_2015,total_2015=  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,'2015-10-25','2015-11-01')
print (result_2015)
```

```
                10    25    50    100    200
accuracy_scale   0.9  0.68  0.64  0.65  0.595
accuracy_minmax  0.9  0.68  0.64  0.65  0.595
accuracy_no      0.9  0.64  0.66  0.53  0.445
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
```

In [10]:

```
####2016年预测结果
traindata = pd.concat([gsz_2014,gsz_2015],axis=0)
testdata = gsz_2016.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2016)&(q123_already_szdata['gs']>0)]
result_2016,total_2016 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,'2016-10-25','2017-11-01')
print (result_2016)
```

```
                10    25    50    100    200
accuracy_scale   0.8  0.84  0.84  0.62  0.460
accuracy_minmax  0.8  0.84  0.84  0.62  0.460
accuracy_no      0.4  0.20  0.20  0.28  0.245
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
```

SVM分类¶

In [11]:

```
from sklearn.svm import SVC

def get_prediction_SVM(x_traindata,y_traindata,x_testdata,standard='scale'):
    if standard == 'scale':
        #均值方差标准化
        standard_scaler = preprocessing.StandardScaler()
        X_trainScale = standard_scaler.fit_transform(x_traindata)
        X_testScale = standard_scaler.transform(x_testdata)
    elif standard =='minmax':
        #min_max标准化
        min_max_scaler = preprocessing.MinMaxScaler()
        X_trainScale = min_max_scaler.fit_transform(x_traindata)
        X_testScale = min_max_scaler.transform(x_testdata)
    elif standard =='no':
        #不标准化
        X_trainScale = x_traindata
        X_testScale = x_testdata

    ###考虑到样本中高送转股票与非高送转股票样本的不平衡问题，这里选用调整的class_weight

    clf = SVC(C=1.0,class_weight='balanced',gamma='auto',kernel='rbf',probability=True)
    clf.fit(X_trainScale, y_traindata)
    predict_y=clf.predict_proba(X_testScale)
    return predict_y
```

In [12]:

```
### 2013年预测结果
traindata = pd.concat([gsz_2011,gsz_2012],axis=0)
testdata = gsz_2013.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2013)&(q123_already_szdata['gs']>0)]
result_2013,total_2013 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,'2013-10-25',\
                                        '2013-11-01',function = get_prediction_SVM)
print (result_2013)
```

```
                10    25    50    100    200
accuracy_scale   0.6  0.36  0.30  0.30  0.265
accuracy_minmax  0.3  0.20  0.14  0.22  0.230
accuracy_no      0.0  0.00  0.12  0.07  0.055
```

In [13]:

```
###2014年预测结果
traindata = pd.concat([gsz_2012,gsz_2013],axis=0)
testdata = gsz_2014.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2014)&(q123_already_szdata['gs']>0)]
result_2014,total_2014 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,\
                                        '2014-10-25','2014-11-01',function = get_prediction_SVM)
print (result_2014)
```

```
                10    25    50    100    200
accuracy_scale   0.7  0.64  0.66  0.51  0.475
accuracy_minmax  0.8  0.56  0.56  0.52  0.470
accuracy_no      0.2  0.12  0.08  0.11  0.065
```

In [14]:

```
####2015年预测结果
traindata = pd.concat([gsz_2013,gsz_2014],axis=0)
testdata = gsz_2015.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2015)&(q123_already_szdata['gs']>0)]
result_2015,total_2015 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,\
                                        '2015-10-25','2015-11-01',function = get_prediction_SVM)
print (result_2015)
```

```
                10    25    50    100    200
accuracy_scale   0.5  0.68  0.62  0.56  0.545
accuracy_minmax  0.8  0.80  0.66  0.59  0.540
accuracy_no      0.2  0.12  0.14  0.12  0.130
```

In [15]:

```
####2016年预测结果
traindata = pd.concat([gsz_2014,gsz_2015],axis=0)
testdata = gsz_2016.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2016)&(q123_already_szdata['gs']>0)]
result_2016,total_2016 =  assess_classification_result(traindata,testdata,variable_list,q123_sz_data,\
                                        '2016-10-25','2017-11-01',function = get_prediction_SVM)
print (result_2016)
```

```
                10    25    50    100    200
accuracy_scale   0.7  0.60  0.66  0.54  0.410
accuracy_minmax  0.7  0.76  0.74  0.55  0.395
accuracy_no      0.1  0.08  0.08  0.05  0.040
```

逻辑回归与SVM联合选择¶

In [16]:

```
def assess_unite_logit_SVM(traindata,testdata,variable_list,q123_sz_data,method_use,date1,date2):
    ###Logit 部分
    traindata.dropna(inplace = True)
    testdata.dropna(inplace = True)
    x_traindata = traindata[variable_list]
    y_traindata = traindata[['gsz']]
    x_testdata = testdata[variable_list]
    y_testdata = testdata[['gsz']]

    total_logit = testdata[['stock','gsz']].copy()
    for method in ['scale','minmax','no']:
        predict_y = get_prediction(x_traindata,y_traindata,x_testdata,standard=method)
        total_logit['predict_prob_'+method] = predict_y[:,1]


    ###########SVM部分
    traindata.ix[traindata['gsz']==0,'gsz']=-1
    testdata.ix[testdata['gsz']==0,'gsz']=-1
    x_traindata = traindata[variable_list]
    y_traindata = traindata[['gsz']]
    x_testdata = testdata[variable_list]
    y_testdata = testdata[['gsz']]

    total_SVM = testdata[['stock','gsz']].copy()
    for method in ['scale','minmax','no']:
        predict_y = get_prediction_SVM(x_traindata,y_traindata,x_testdata,standard=method)
        total_SVM['predict_prob_'+method] = predict_y[:,1]

    ###合并
    columns = ['stock','gsz','predict_prob_scale','predict_prob_minmax','predict_prob_no']
    total = total_logit[columns].merge(total_SVM[['stock','predict_prob_scale','predict_prob_minmax',\
                                                  'predict_prob_no']],on=['stock'])
    for method in ['scale','minmax','no']:
        total['score_logit'] = total['predict_prob_'+method+'_x'].rank(ascending = False)
        total['score_SVM'] = total['predict_prob_'+method+'_y'].rank(ascending = False)
        total['score_' + method] = total['score_logit']+total['score_SVM']

    ###过滤今年前期已经送转过的股票
    q123_stock = q123_sz_data['stock'].tolist()
    total_filter = total.loc[total['stock'].isin(q123_stock)==False]

    ###过滤ST股票
    stock_list = total_filter['stock'].tolist()
    st_data = pd.DataFrame(get_extras('is_st',stock_list , start_date=date1, end_date=date2, df=True).iloc[-1,:])
    st_data.columns =['st_signal']
    st_list = st_data[st_data['st_signal']==True]
    total_filter = total_filter[total_filter['stock'].isin(st_list)==False]

    result_dict ={}
    for stock_num in [10,25,50,100,200]:
        accuracy_list = []
        for column in ['score_scale','score_minmax','score_no']:
            total_filter.sort(column,inplace = True,ascending = True)
            dd = total_filter[:stock_num]
```

```
            accuracy = len(dd[dd['gsz']==1])/len(dd)
            accuracy_list.append(accuracy)
        result_dict[stock_num] = accuracy_list

    result = pd.DataFrame(result_dict,index =['score_scale','score_minmax','score_no'])

    return result
```

In [66]:

```
###2013年
traindata = pd.concat([gsz_2011,gsz_2012],axis=0)
testdata = gsz_2013.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2013)&(q123_already_szdata['gs']>0)]
result_2013_unite =  assess_unite_logit_SVM(traindata,testdata,variable_list,q123_sz_data,'minmax',\
                                        '2013-10-25','2013-11-01')
print (result_2013_unite)
```

```
             10    25    50    100    200
score_scale   0.4  0.36  0.32  0.28  0.275
score_minmax  0.4  0.24  0.18  0.23  0.250
score_no      0.1  0.08  0.24  0.32  0.270
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
```

In [67]:

```
###2014年预测结果
traindata = pd.concat([gsz_2012,gsz_2013],axis=0)
testdata = gsz_2014.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2014)&(q123_already_szdata['gs']>0)]
result_2014_unite =  assess_unite_logit_SVM(traindata,testdata,variable_list,q123_sz_data,'minmax',\
                                        '2014-10-25','2014-11-01')
print (result_2014_unite)
```

```
             10    25    50    100    200
score_scale   0.8  0.72  0.58  0.53  0.485
score_minmax  0.7  0.72  0.58  0.55  0.500
score_no      0.5  0.52  0.46  0.46  0.380
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
/opt/conda/lib/python3.4/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
```

In [68]:

```
####2015年预测结果
traindata = pd.concat([gsz_2013,gsz_2014],axis=0)
testdata = gsz_2015.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price', 'days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2015)&(q123_already_szdata['gs']>0)]
result_2015_unite =   assess_unite_logit_SVM(traindata,testdata,variable_list,q123_sz_data,'minmax',\
                                        '2015-10-25','2015-11-01')
print (result_2015_unite)
```

```
             10    25    50    100    200
score_scale   0.8  0.84  0.72  0.64  0.560
score_minmax  0.7  0.80  0.68  0.62  0.585
score_no      0.9  0.60  0.64  0.53  0.450
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
/opt/conda/lib/python3.4/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
```

In [69]:

```
####2016年预测结果
traindata = pd.concat([gsz_2014,gsz_2015],axis=0)
testdata = gsz_2016.copy()
variable_list = ['per_zbgj_wflr','capitalization', 'eps', 'revenue_growth',\
                 'mean_price','days_listed']
q123_sz_data =  q123_already_szdata[(q123_already_szdata['year']==2016)&(q123_already_szdata['gs']>0)]
result_2016_unite =  assess_unite_logit_SVM(traindata,testdata,variable_list,q123_sz_data,'minmax',\
                                        '2016-10-25','2016-11-01')
print (result_2016_unite)
```

```
             10    25   50    100   200
score_scale   0.9  0.88  0.8  0.63  0.45
score_minmax  0.9  0.96  0.8  0.63  0.40
score_no      0.4  0.20  0.2  0.27  0.24
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
/opt/conda/lib/python3.4/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A column-vector y was passed when a 1c
  y = column_or_1d(y, warn=True)
```

2017年预测¶

In [72]:

```
###取出2017年数据
statDate = '2017q3'
mp_startdate = '2017-10-01'
mp_enddate = '2017-11-01'
year = 2017
month = 11
day = 1

per_zbgj = get_perstock_indicator(balance.capital_reserve_fund,'capital_reserve_fund','per_CapitalReserveFund',statDate)
per_wflr = get_perstock_indicator(balance.retained_profit,'retained_profit','per_RetainedProfit',statDate)
per_jzc = get_perstock_indicator(balance.total_owner_equities,'total_owner_equities','per_TotalOwnerEquity',statDate)
other_indicator = get_other_indicator(statDate)
code_list = other_indicator['code'].tolist()
mean_price = get_bmonth_aprice(code_list,mp_startdate,mp_enddate)
cx_signal = judge_cxstock(mp_enddate)
days_listed = get_dayslisted(year,month,day)

chart_list = [per_zbgj,per_wflr,per_jzc,other_indicator,mean_price,cx_signal,days_listed]
for chart in chart_list:
    chart.set_index('code',inplace = True)

independ_vari = pd.concat([per_zbgj,per_wflr,per_jzc,other_indicator,mean_price,cx_signal,days_listed],axis = 1)
independ_vari['year'] = str(int(statDate[0:4]))
independ_vari['stock'] = independ_vari.index
independ_vari.reset_index(drop=True,inplace =True)

independ_vari['per_zbgj_wflr'] = independ_vari['per_CapitalReserveFund']+independ_vari['per_RetainedProfit']

gsz_2017 = independ_vari
gsz_2017.ix[gsz_2017['revenue_growth']>300,'revenue_growth'] = 300


traindata = pd.concat([gsz_2015,gsz_2016],axis=0)
testdata = gsz_2017
q123_sz_data = q123_already_szdata[(q123_already_szdata['year']==2017)&(q123_already_szdata['gs']>0)]
###Logit 部分
traindata.dropna(inplace = True)
testdata.dropna(inplace = True)
x_traindata = traindata[variable_list]
y_traindata = traindata[['gsz']]
x_testdata = testdata[variable_list]


total_logit = testdata[['stock']].copy()
method='scale'
predict_y = get_prediction(x_traindata,y_traindata,x_testdata,standard=method)
total_logit['predict_prob_'+method] = predict_y[:,1]


###########SVM部分
traindata.ix[traindata['gsz']==0,'gsz']=-1
x_traindata = traindata[variable_list]
y_traindata = traindata[['gsz']]
x_testdata = testdata[variable_list]
```

```
total_SVM = testdata[['stock']].copy()
method = 'scale'
predict_y = get_prediction_SVM(x_traindata,y_traindata,x_testdata,standard=method)
total_SVM['predict_prob_'+method] = predict_y[:,1]

###合并
columns = ['stock','predict_prob_'+method]
total = total_logit[columns].merge(total_SVM[['stock','predict_prob_'+method]],on=['stock'])

total['score_logit'] = total['predict_prob_'+method+'_x'].rank(ascending = False)
total['score_SVM'] = total['predict_prob_'+method+'_y'].rank(ascending = False)
total['score'] = total['score_logit']+total['score_SVM']

###过滤今年前期已经送转过的股票
q123_stock = q123_sz_data['stock'].tolist()
total_filter = total.loc[total['stock'].isin(q123_stock)==False]
###过滤ST股票
stock_list = total_filter['stock'].tolist()
st_data = pd.DataFrame(get_extras('is_st',stock_list ,start_date='2017-10-25', end_date='2017-11-01', df=True).iloc[-1,:])
st_data.columns =['st_signal']
st_list = st_data[st_data['st_signal']==True]
total_filter = total_filter[total_filter['stock'].isin(st_list)==False]
```

```
/opt/conda/lib/python3.4/site-packages/sklearn/preprocessing/data.py:160: UserWarning: Numerical issues were encountered when centeri
  warnings.warn("Numerical issues were encountered "
/opt/conda/lib/python3.4/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
```

预测前50只¶

In [75]:

```
total_filter.sort('score',inplace = True,ascending = True)
total_filter.reset_index(drop=True,inplace = True)
total_filter[:50]
```

Out[75]:

	stock	predict_prob_scale_x	predict_prob_scale_y	score_logit	score_SVM	score
0	002627.XSHE	0.985344	0.587367	3	13.0	16.0
1	002872.XSHE	0.931494	0.598285	16	9.0	25.0
2	300184.XSHE	0.916963	0.590618	21	12.0	33.0
3	002682.XSHE	0.918300	0.573175	20	14.0	34.0
4	603385.XSHG	0.909826	0.557251	26	18.0	44.0
5	002772.XSHE	0.851236	0.624213	58	5.0	63.0
6	603878.XSHG	0.888196	0.533011	38	27.0	65.0
7	603599.XSHG	0.866226	0.542158	50	24.0	74.0
8	300537.XSHE	0.846131	0.572538	61	15.0	76.0
9	002164.XSHE	0.842408	0.562850	67	16.0	83.0
10	300510.XSHE	0.866181	0.522639	51	41.0	92.0
11	002564.XSHE	0.847225	0.523973	60	37.0	97.0
12	603839.XSHG	0.859098	0.516115	54	44.0	98.0
13	603928.XSHG	0.829549	0.554954	80	19.0	99.0
14	603926.XSHG	0.835150	0.534565	75	25.0	100.0
15	603116.XSHG	0.881629	0.482789	41	62.0	103.0
16	603167.XSHG	0.844965	0.512905	64	45.0	109.0
17	603086.XSHG	0.897919	0.458643	31	82.0	113.0
18	601900.XSHG	0.802173	0.618932	110	7.0	117.0
19	603768.XSHG	0.821876	0.526580	86	35.0	121.0
20	603035.XSHG	0.817521	0.529033	89	33.0	122.0

	stock	predict_prob_scale_x	predict_prob_scale_y	score_logit	score_SVM	score
21	300349.XSHE	0.912351	0.440052	24	99.0	123.0
22	300569.XSHE	0.830428	0.512234	79	46.0	125.0
23	603980.XSHG	0.799044	0.559231	112	17.0	129.0
24	002788.XSHE	0.843138	0.478629	66	66.0	132.0
25	603586.XSHG	0.823848	0.510037	84	48.0	132.0
26	603979.XSHG	0.790512	0.591045	122	11.0	133.0
27	603033.XSHG	0.808966	0.529255	102	32.0	134.0
28	300485.XSHE	0.829367	0.492128	81	58.0	139.0
29	603556.XSHG	0.834741	0.481310	76	63.0	139.0
30	603558.XSHG	0.815226	0.511705	92	47.0	139.0
31	002541.XSHE	0.958416	0.405282	11	134.0	145.0
32	300391.XSHE	0.856482	0.450642	56	90.0	146.0
33	300407.XSHE	0.876355	0.438175	45	102.0	147.0
34	603535.XSHG	0.786474	0.547880	128	22.0	150.0
35	603367.XSHG	0.784653	0.548487	130	21.0	151.0
36	300338.XSHE	0.833377	0.460864	77	78.0	155.0
37	002377.XSHE	0.980316	0.390150	5	151.0	156.0
38	002574.XSHE	0.784113	0.531081	131	29.0	160.0
39	300178.XSHE	0.797867	0.506400	113	51.0	164.0
40	002791.XSHE	0.839194	0.442252	71	95.0	166.0
41	002746.XSHE	0.864586	0.421093	52	115.0	167.0
42	603226.XSHG	0.778350	0.526446	135	36.0	171.0
43	002740.XSHE	0.810678	0.473238	101	71.0	172.0
44	603117.XSHG	0.792407	0.500000	120	55.5	175.5
45	603630.XSHG	0.795253	0.484904	117	61.0	178.0
46	603017.XSHG	0.806854	0.460476	104	80.0	184.0
47	300587.XSHE	0.778195	0.508689	136	49.0	185.0
48	002734.XSHE	0.870310	0.398310	47	141.0	188.0
49	603225.XSHG	0.857836	0.402985	55	136.0	191.0

预测前100只¶

In [76]:

```
total_filter[:100]
```

Out[76]:

	stock	predict_prob_scale_x	predict_prob_scale_y	score_logit	score_SVM	score
0	002627.XSHE	0.985344	0.587367	3	13	16
1	002872.XSHE	0.931494	0.598285	16	9	25
2	300184.XSHE	0.916963	0.590618	21	12	33
3	002682.XSHE	0.918300	0.573175	20	14	34
4	603385.XSHG	0.909826	0.557251	26	18	44
5	002772.XSHE	0.851236	0.624213	58	5	63
6	603878.XSHG	0.888196	0.533011	38	27	65
7	603599.XSHG	0.866226	0.542158	50	24	74
8	300537.XSHE	0.846131	0.572538	61	15	76

	stock	predict_prob_scale_x	predict_prob_scale_y	score_logit	score_SVM	score
9	002164.XSHE	0.842408	0.562850	67	16	83
10	300510.XSHE	0.866181	0.522639	51	41	92
11	002564.XSHE	0.847225	0.523973	60	37	97
12	603839.XSHG	0.859098	0.516115	54	44	98
13	603928.XSHG	0.829549	0.554954	80	19	99
14	603926.XSHG	0.835150	0.534565	75	25	100
15	603116.XSHG	0.881629	0.482789	41	62	103
16	603167.XSHG	0.844965	0.512905	64	45	109
17	603086.XSHG	0.897919	0.458643	31	82	113
18	601900.XSHG	0.802173	0.618932	110	7	117
19	603768.XSHG	0.821876	0.526580	86	35	121
20	603035.XSHG	0.817521	0.529033	89	33	122
21	300349.XSHE	0.912351	0.440052	24	99	123
22	300569.XSHE	0.830428	0.512234	79	46	125
23	603980.XSHG	0.799044	0.559231	112	17	129
24	002788.XSHE	0.843138	0.478629	66	66	132
25	603586.XSHG	0.823848	0.510037	84	48	132
26	603979.XSHG	0.790512	0.591045	122	11	133
27	603033.XSHG	0.808966	0.529255	102	32	134
28	300485.XSHE	0.829367	0.492128	81	58	139
29	603556.XSHG	0.834741	0.481310	76	63	139
...	...	...	...	...	...	...
70	300442.XSHE	0.811955	0.405477	99	132	231
71	300432.XSHE	0.778761	0.440250	134	98	232
72	002513.XSHE	0.844120	0.371885	65	168	233
73	603003.XSHG	0.884385	0.354598	39	196	235
74	002775.XSHE	0.770174	0.443118	144	94	238
75	603615.XSHG	0.748619	0.480700	175	65	240
76	002521.XSHE	0.794655	0.412059	118	127	245
77	002743.XSHE	0.759337	0.457272	162	83	245
78	300421.XSHE	0.826244	0.375454	83	163	246
79	300200.XSHE	0.800044	0.403777	111	135	246
80	603808.XSHG	0.771102	0.436278	143	103	246
81	603139.XSHG	0.759555	0.451829	161	86	247
82	300157.XSHE	0.810745	0.391008	100	149	249
83	603038.XSHG	0.845250	0.355801	63	192	255
84	603689.XSHG	0.736977	0.481289	197	64	261
85	601233.XSHG	0.877884	0.341827	44	218	262
86	603801.XSHG	0.719112	0.530216	231	31	262
87	603018.XSHG	0.807858	0.376245	103	162	265
88	300233.XSHE	0.889060	0.331036	37	230	267
89	002623.XSHE	0.914365	0.321120	22	245	267
90	002611.XSHE	0.761795	0.429999	158	110	268
91	601155.XSHG	0.703473	0.550118	250	20	270
92	601717.XSHG	0.696049	0.608578	262	8	270

	stock	predict_prob_scale_x	predict_prob_scale_y	score_logit	score_SVM	score
93	000581.XSHE	0.838579	0.347750	72	202	274
94	603889.XSHG	0.762561	0.416682	156	121	277
95	300004.XSHE	0.775522	0.398924	138	140	278
96	603758.XSHG	0.720498	0.508334	228	50	278
97	603887.XSHG	0.736400	0.459910	198	81	279
98	603181.XSHG	0.745125	0.440732	187	97	284
99	603906.XSHG	0.731366	0.455659	207	84	291

100 rows × 6 columns

【量化课堂】双均线策略

导语： 双均线策略，通过建立m天移动平均线，n天移动平均线，则两条均线必有交点。若m>n，n天平均线“上穿越”m天均线则为买入点，反之为卖出点。该策略基于不同天数均线的交叉点，抓住股票的强势和弱势时刻，进行交易。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： 导数君
编辑： 宏观经济算命师

均线嘛，都是均线

对于每一个交易日，都可以计算出前N天的移动平均值，然后把这些移动平均值连起来，成为一条线，就叫做N日移动平均线。

比如前5个交易日的收盘价分别为10,9,9,10,11元，那么，5日的移动平均股价为9.8元。同理，如果下一个交易日的收盘价为12，那么在下一次计算移动平均值的时候，需要计算9,9,10,11,12元的平均值，也就是10.2元。

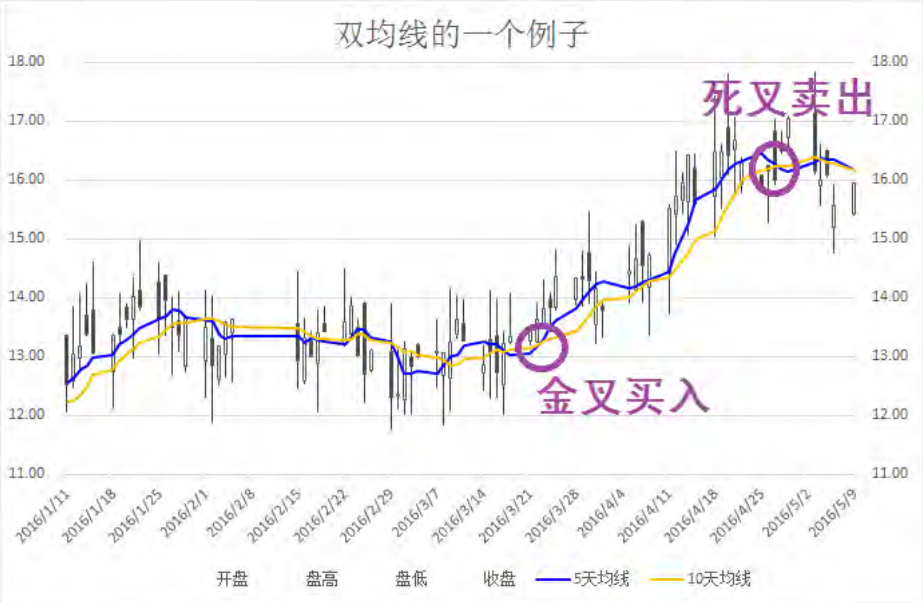
将这平均值连起来，就是均线。

如下图所示，收盘价是蓝线，橙色的线表示5日的移动平均线。

可以看到股票价格的波动比5天均线的波动要大，这是因为5天均线取的是前5个交易日的均值，相当于做了一个平滑。

双均线

顾名思义就是两条天数不同的移动平均线，比如说，一条是5天的移动平均线，另一条是10天的移动平均线。如图，蓝色的是5天均线，黄色的是10天均线。



金叉和死叉

由时间短的均线（如上图蓝色的线）在下方向上穿越时间长一点的均线（如上图黄色的线），为“金叉”，反之为“死叉”。
好了，现在可以构建一个简单的策略：我们认为，双均线金叉的时候，表明股票很强势，反之很弱势，我们就在强势的时候买一个好了，弱势的时候卖掉好了。

说了这么多，下面我们开始实战！

首先，我们可以看一下API中的策略示例里面的双均线策略（详见https://www.joinquant.com/api#双均线策略 ）直接复制粘贴就能运行，是不是很简单呢？

老师，可以了嘛？

当然不行，有时候我们也许会根据自己的需要对一些现有的策略进行改造，比如说，我想对均线进行加权呢？我想改造一个指数均线呢？

那我们得自己实现一下均线函数。方法不难，获得前N天的收盘价，然后计算一个算术平均数就可以了，各位读者可以先自己进行尝试，也可以参考回测代码块5（里面有代码和注释）

接下来，如果你想挑战更高难度，可以试一下计算指数移动平均的函数。

指数移动平均和算术平均或者加权平均的主要区别在于指数移动平均需要进行一个迭代，因此这可能是个有点挑战的地方：

$$EMA_i(N) = \begin{cases} MA_i(N), i = 1 \\ \alpha EMA_{i-1}(N) + (1-\alpha) p_{i-1}, \quad i > 1 \end{cases}$$

其中pi表示前一天的收盘价，且

$$\alpha = \frac{N-1}{N+1}$$

写出来没？如果没写出来的话可以参考回测代码块3和4（里面有代码和注释）

怎么样？是不是有点挑战性？如果写出这样的子函数，那么在主程序里面只要改一下函数输入参数，就可以轻松的在不同的参数之间来回切换，比较收益。

多股票

除了改为指数移动平均线以外，小编还加入多股票实现方法。

在交易日前，同时对多个股票进行判断，哪只金叉了就买入，死叉了就卖出，按日进行判断和交易。最大可以同时持有N只股票，用于实现这个策略的主函数可以参考回测代码块1和2。

好啦，看看小编选了5个股票的回测结果吧。从2005开始回测至今，你会发现在大部分时间里面，策略的收益比基准收益高，是不是很棒？各位读者可以自己尝试修改参数，看看参数应该如何选取。在这里，小编提供一个同时持有股票书N的选择的小tips，如果风险承受能力强的话可以少选一点，如果想分散风险，可以多选一些股票，但是，分到每个股票的资金最好不要少于两万，因为手续费是有最低限制的。

小结

我们这里是量化课堂的第三部分，主要给大家提供几个大范围的量化思路。本篇为第一讲，举了一个技术指标的例子。比较简单。有关技术指标的文章，百度上一搜一大把，我们平台里也有很多相关的帖子，python公开库ta-lib里也收录了很多指标，大家想进一步研究可以实现一下。也可以结合其他选股策略，或者根据需要制订资金等分的份数或者控制仓位等方法提高策略的收益。

好了，今天对双均线多股票选股策略的介绍就到这里了。

文章更迭记录：
v2.0，2016-07-16，更新为规范源码，添加“函数说明书”
v1.1，2016-07-04，添加“导语”
v1.0，2016-05-21，文章上线

【量化课堂】多因子策略入门

导语：每一位宽客都相信，影响股票涨跌的因素不胜枚举，而这些“因素”就是因子！本文作为一篇合格的入门教程，提供代码当做框架，各路宽客可以自己测试，查看收益率，亦可利用聚宽python平台自行构建代码。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： 导数君
编辑： 宏观经济算命师

因子

因子是什么？通俗来讲。选股择时，我们得有一个标准对不对？这些标准就叫做因子。比如，我认为营收增长率高的公司就是好公司！那我就把营收增长率大于30%的股票拉出来纳入石榴裙下好了。这个营收增长率大于30%就是因子，完毕。

因子有选股的因子（股票好不好），有择时的因子（好股票什么时候买）。由于择时往往跟技术指标关系紧密，本篇中就介绍基本面类的因子吧，偏财务向。

选取因子

最简单的方法，先物色一些自己喜欢的因子，比如增长率啦，市值啦，ROE啦，等等。然后一个个往里面加，看看效果如何，效果好了留下，效果差了删除，反复重复这个过程就能找到心仪的因子啦。

举个例子，小编先选两个因子，ROA和净利润/营业总收入(%)。ROA和利润率比较高的一般都是表现良好的公司，所以小编决定选取ROA和净利润/营业总收入前20名的股票买入，回测结果如下图所示：

表现比大盘略好一点，11年的收益有3.98倍。那换个因子试试？小编把净利润/营业总收入换成净利润环比增长率(%)，回测结果如下图所示：

然后发现收益率好了一点点，过去11年的收益有7.54倍。嗯，比刚才高了一些。如果我们把这三个因子都加进去会怎样呢？

收益率为5.34倍，没有刚才那么好了，还是刚才的ROA和净利润环比增长率(%)这两个因子比较好，那就保留两个吧。小编听说小市值股票收益好，如果把市值这个因子也加进去会怎么样呢？回测了一下，发现：

收益居然有28倍！真的是太不可思议了。

所以话说回来，虽然这种试错法选因子是一种比较基础的选因子方式，但其实还挺有用。经过反复试错，小编发现小市值和ROE高的股票收益比较高，回测结果显示，收益可以达到42倍之多！

是不是很惊艳！你可以直接编程构建代码，也可以用我们的代码当做框架。总之，可以自己测试一下，看看收益会不会爆表。
作为一篇合格的教程，我们接下来看看代码是如何实现的：

编写代码的一些问题

首先，财务面的数据有个问题，就是有些数据是不可获得的，这样的话对排序的影响比较大，因此涉及到一个清洗数据的步骤。一个很简单的办法就是用均值来填充，这个在Python的pandas库里面有个现成的函数，大家可以尝试使用里面的均值填充法。小编自己也写了一个填充均值的函数，大家也可以参考一下源码。

不过随着深入的研究，可能会发现用均值填充并不是一个完美的方法。这里小编再提供一个思路，大家感兴趣可以自己实现：如果某只股票这一期的某个财务数据空缺，但是上一期没有，我们可以根据该股票这个数据与上一期的平均值比例来确定。用公式说可能更清晰点：
空缺数值=本期该字段平均值*上期该字段数值/上期该字段平均值。

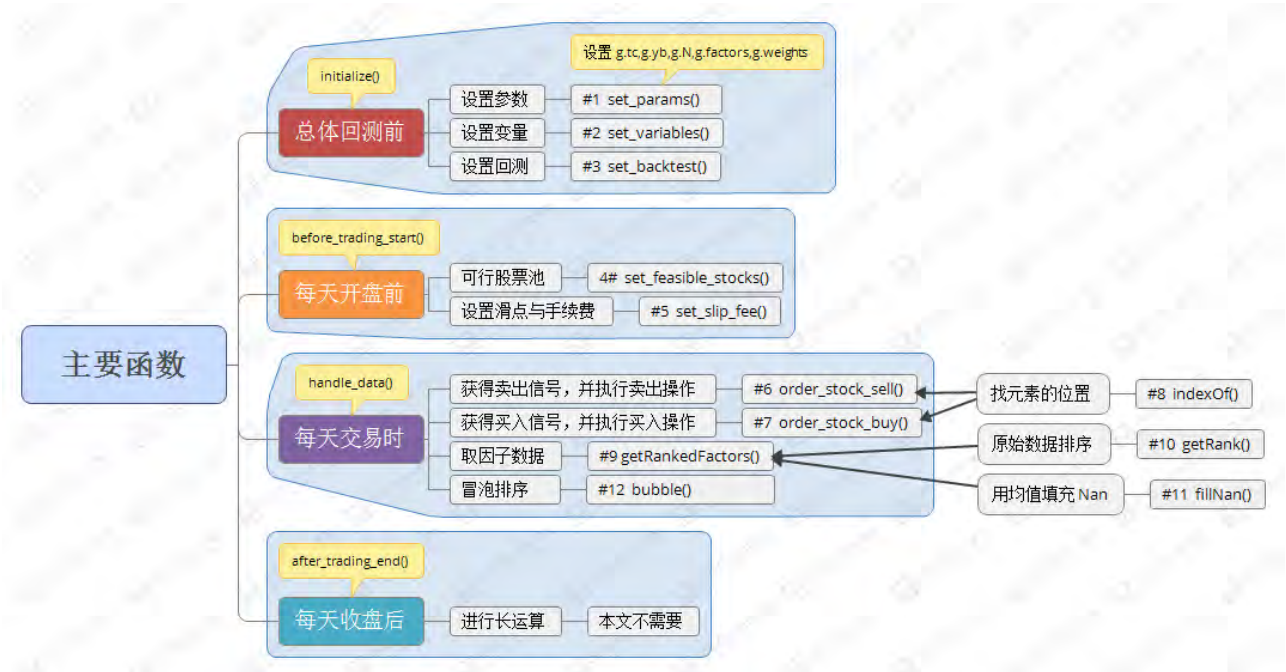
其次，我们对因子的单位要做一个统一。因为有的因子绝对值好几十亿（比如市值），有的可能只有十几（收益率）甚至是负的，因此因子和因子之间很难直接赋予权重进行计算。因此，我们可以考虑使用排名的方法，对这些因子进行排名。
Python自带有一个sort函数，不过为了练手起见，小编自己写了一个。用的是最简单的冒泡排序算法，高手也可以试一下堆排序或者归并排序以加快回测的速度。

最后，我们将上述功能汇总成一个函数，集中取数据-清洗数据。

有了以上的几个子函数，写主函数就很方便了，各位宽粉们赶紧尝试吧！

小结

我们这篇文章主要介绍了如何通过财务数据来构建一个多因子的策略。由于是入门向，我们构建多因子的方法比较简单，选取的因子依据是主观分析+排名。
如果想定量的分析，主流的方法是做回归分析，或者对各个因子进行打分，这些将会在进阶的量化课堂中有所介绍。如果还有其他的方法，当然也欢迎尝试。



文章更迭记录：
v2.1，2016-07-25，修正文字，感谢 liuzehong 指出

v2.0，2016-07-16，更新为规范源码，添加“函数说明书”
v1.1，2016-07-04，添加“导语”
v1.0，2016-05-21，文章上线

【量化课堂】海龟策略

导语：本篇介绍如何借鉴成熟的策略体系并在聚宽平台上实现。成熟的策略体系有很多种，例如海龟，羊驼，鳄鱼等等。今天的先举个海龟交易系统。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： Haozun
编辑： 宏观经济算命师

海龟

天然的海龟是一个比较成熟而完整的交易系统。构建交易系统的目的就是避免交易员自己做出主观的决策。这样才能真正的让概率发挥作用。海龟的主要捕捉的是趋势。其采用突破法来确定趋势，当价格突破时认为有买入的信号，而随着价格离当初突破的价格越来越远，我们认为趋势成立的概率就越来越高，加仓！那么，这个突破怎么确定呢？我们需要用到唐奇安通道的方法进行处理。

唐奇安通道

在海龟的系统的止盈止损中，实际上就是借助了唐奇安通道。那么，这唐奇安通道究竟是个什么东西呢？

首先引入上线中线下线的概念。上线=Max（前N个交易日的最高价），下线=Min（前N个交易日的最低价），中线=（上线+下线）/2，每个交易日结束之后更新当天的数据。这里N一般默认取20。那么唐奇安通道就是这个上线和下线所形成的走势区间。所谓的突破，也就是指今日盘中股价高过了上线。参见下图：

这里有个小插曲，为什么默认的N是20呢？这有个典故——神奇数字。Donchian在开发唐奇安通道的期间，碰巧阅读到整形外科医生Maxwel Maltz博士在1960年所作的“心理控制论”（这本书在1989年被重新发现）。Maltz博士称在整形外科手术过程中，患者最少需要21日来看到自己的新的容颜。这一事实震惊了Donchian，所以他也采用了这个说法。（小编也听说过，养成一个新的习惯需要21天嘛）

在聚宽平台实现这个策略，最好采用按分钟回测，这样可以准确的捕捉日内的买卖点，否则等日线的收盘价出来，说不定已经离突破很远了。同时请注意，在以往国外的实战当中，主要是在纽约和芝加哥的场内期货交易的，期货可以做多也可以做空，但是国内的股票只能买入和卖出，因此我们只能做向上突破。

止损

有了唐奇安通道，我们有了买入和卖出的依据了， 那么止损是干什么的呢？其实止损提出的初衷是，如果某笔交易是亏损的交易，造成的损失不要超过总仓位的k%。在完整的海龟系统里面，海龟用了一系列的公式进行仓位比例的计算，具体的内容可以参考下一节。

完整的海龟交易系统

完整的海龟交易系统包含以下内容：
1、市场：原版海龟选择交易纽约和芝加哥的场内期货。筛选标准则是高流动性，我大A股市场当然也符合这个标准啦。

2、仓位：这可以说是海龟交易系统最核心的部分。Richard Dennis期望通过市场的波动性水平来管理仓位。其构建了指数N来衡量波动性水平。指数的构建为以下四步
（注：如果暂时不能理解下面的公式，完全不用担心，这些都在代码中体现出来，大家可以在代码的实际使用中搞明白这些麻烦的公式）。

（1）True Range

$$True\ Range = Maximum(H-L, H-PDC, PDC-L)$$

公式中，True Range表示一天内的波动量，H为当日日内最高价，L为当日日内最低价，PDC为前一日收盘价。

（2）N

$$N = \frac{19 * PDN + TR}{20}$$

公式中，TR为True Range，即一天内波动量，PDN为前一日的N值。此公式的真是含义为计算之前20天（包括今天在内）的N的平均值

（3）Dollar Volatility

$$Dollar\ Volatility = N * Dollars\ Per\ Point$$

公式中，Dollar Volatility指的是波动的价格，Dollars per Point指的是标的股票每波动一个最小单位，1手股票的总价格变化量。在国内最小变化量是0.01元，1手是100股。所以Dollars per Point就是0.01×100=1

（4）Unit

$$Unit = \frac{1\%\ of\ Account}{Market\ Dollar\ Volatility}$$

公式中，Unit即为我们买卖的单位，1% of Account是总资产的1%，Market Dollar Volatility就是我们之前算出的Dollar Volatility，通过此公式计算出的Unit就是我们要买入的单位数量。此公式的意义是在一般情况下（市场波动率不大的时候），如果买入1Unit单位的资产，当天震幅使得总资产的变化不超过1%

3、入市：海龟将所有资金分为两部分，一部分资金按系统一执行，一部分资金按系统二执行
系统一
（1）若当前价格高于过去20日的最高价，则买入一个Unit（注意是分钟回测）
（2）加仓：若股价在上一次买入（或加仓）的基础上上涨了0.5N，则加仓一个Unit
系统二
与系统一相一致，但当如破55日最高价时才购买
（1）若当前价格高于过去55日的最高价，则买入一个Unit（注意是分钟回测）
（2）加仓：若股价在上一次买入（或加仓）的基础上上涨了0.5N，则加仓一个Unit
Example：若某只股票A的N为2，20日最高价为100
则当股价突破100时买入一个Unit，当股价突破100+0.5×2=101时加仓一个Unit，当股价突破101+0.5×2=102时加仓一个Unit。

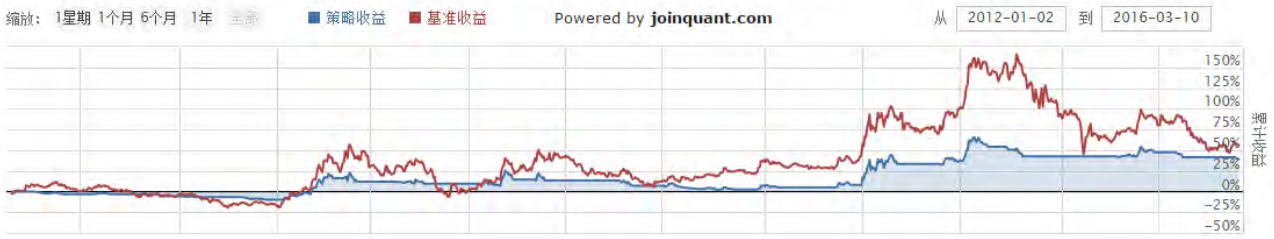
4、止损：即损失达到多少时就一定要卖出现有仓位。海龟交易系统规定，当价格比最后一次买入价格下跌2N时，则卖出全部头寸止损（也就是，在一般情况下，损失不会超过2%）。

5、止盈：
系统一
当股价跌破10日内最低价时（10日唐奇安通道下沿），清空头寸结束本次交易
系统二
当股价跌破20日内最低价时（20日唐奇安通道下沿），清空头寸结束本次交易

6、技巧：资金的调整。开始时设定两个比例：Loss和Adjust。若交易结束后损失的资金占总资金比例大于Loss，则今后只用现有投资资金的Adjust比例。
Example：若初始资金为100万，设定Loss=80%，Adjust=90%。则当总资产低于100×80%=80万时，进行一次资金调整，以后只使用80×90%=72万的资金用于投资行为

结果展示
在这里，小编用简单的海龟系统对几只股票进行了模拟，投入到系统一的资金比例设置为70%或80%，其他参数均不变，结果如下
平安银行 000001.sz

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
39.16%	8.49%	55.91%	0.024	0.281	0.257	0.403	-0.281	0.174	0.386	18.505%

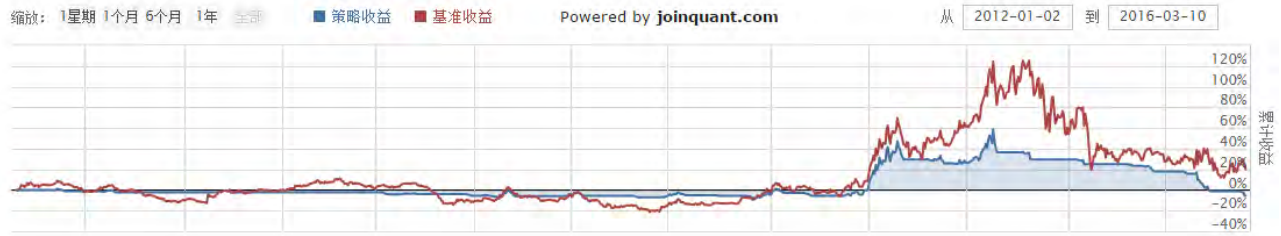


贵州茅台600519.sh

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
28.26%	6.33%	57.13%	0.001	0.283	0.151	0.196	-0.348	0.154	0.332	20.232%

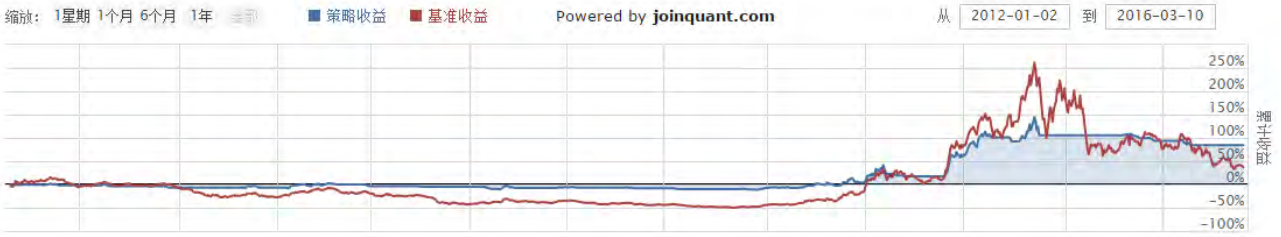
包钢股份600010.sh

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
-5.51%	-1.39%	20.19%	-0.055	0.231	-0.338	-0.348	-0.371	0.159	0.382	40.478%



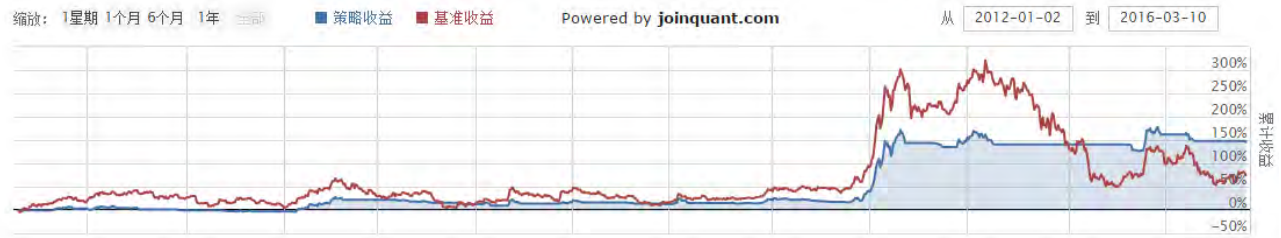
南方航空600029.sh

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
84.85%	16.35%	36.82%	0.114	0.243	0.606	0.772	-0.035	0.204	0.466	24.446%



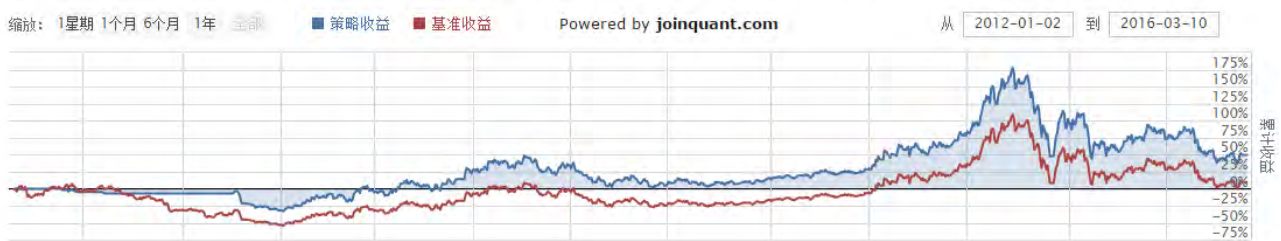
中信证券600030.sh

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
146.59%	24.92%	74.89%	0.180	0.268	0.981	1.716	0.000	0.213	0.463	16.694%



中兴通讯000063.sh

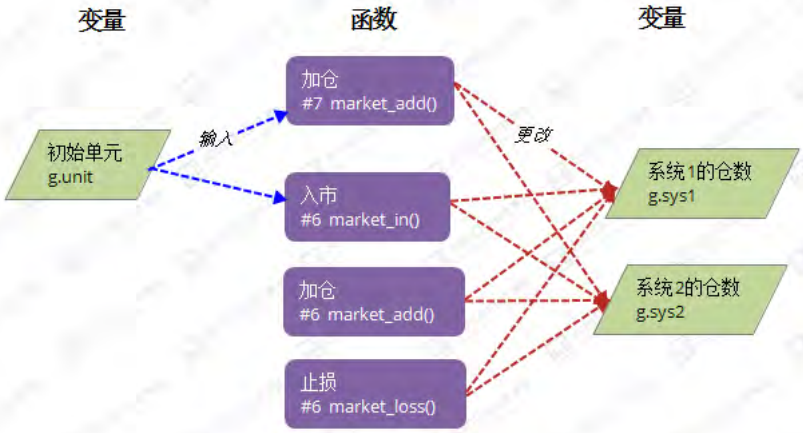
策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
49.73%	10.46%	10.01%	0.078	0.842	0.158	0.206	0.355	0.409	0.460	50.679%



你以为到这里就完了吗？嘿嘿，too young too simple

还有很多复杂的情况，你考虑到了嘛？比如说，通道上沿是昨天的，如果当天分钟价格反复在通道上下震荡，一个不小心就会造成反复的买入。再比如，如果当天刚刚买入，马上又要止损，但是日内不能卖出，我们明天要怎么处理呢？再比如说，如果今天要买入5000手，但实际情况下我们只能买入4000手，剩下的1000手我们要怎么处理呢？是明天补充，还是就此忽略？我们现在是设置的统一的止损线，我们是否可以对买入的每一次单都设置一个单独的止损线呢？

这些问题，没有标准的解法，你可以有自己的处理措施，但是小编只是想说：编写一个完整的策略，一定要思考到位。
小编在这里也只是列出了一部分问题，怎么解决这些问题就暂时留给各位读者作为家庭作业了！



文章更迭记录：
v2.0，2016-07-16，更新为规范源码，添加“函数说明书”
v1.1，2016-07-04，添加“导语”
v1.0，2016-05-21，文章上线

【量化课堂】雪球云蒙银行股搬砖

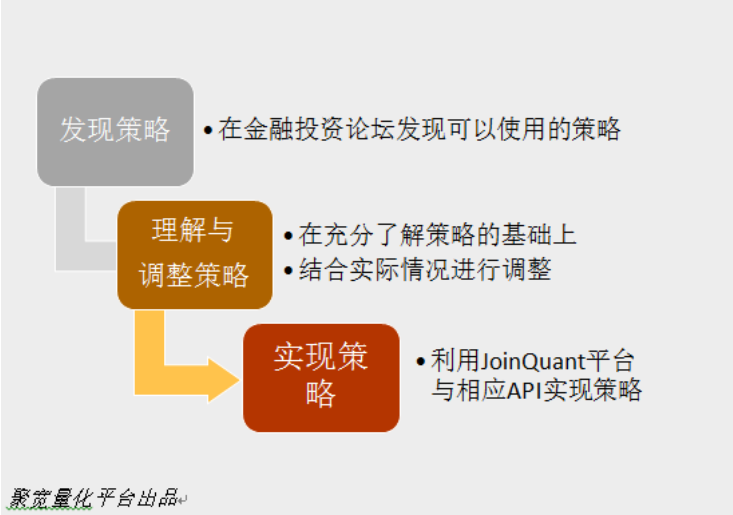
导语：银行股量化选股，即通过对盈利性指标、资本约束及稳健指标、以及资本回拨系数来对一个公司进行评判，得出一个综合打分，之后根据分数进行证券买卖和仓位管理。本文借鉴了雪球大V云蒙的策略，已经作者授权。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： Haozun
编辑： 宏观经济算命师

相信大家已经对怎么构建量化策略有了一个初步的了解。我们已经学习了基本的技术面择时策略以及基本面多因子选股，还学习了怎么样利用JoinQuant平台去构建海龟这样的成熟交易系统。但是，大家肯定会问，我们平时想不到什么好策略怎么办啊？！解决办法之一就是：看大V！

什么意思呢？就是说，有很多金融论坛的大V都在专栏中发表了自己的投资策略，那我们就可以在看懂这些公开出来的成熟策略后，自己在JoinQuant的平台实现该策略，进行回测检验。简单来说！就是下面的三步！



这篇文章中，我们借鉴了雪球大V云蒙的策略:银行股量化指标选股。雪球大V，神采飞扬，以善于投资银行股著称，搬砖运动的倡导者和发扬者。我们先来看一下云蒙在她的文章里提出的思路：

1、盈利性指标：2011-2015年加权平均净资产收益率分别为21.75%、24.17%、24.78%、23.12%、19.28%和17.09%；按照1:2:3:4:5:6的权重加起来除于21，得到一个6年加权净资产收益率，这个值为20.75%。考虑银行同质化相对严重，以80分为标杆，40分给予线性对比，40分给予同质化，这个盈利性指标就是ROE*200+40，这个值为81.51。

2、资本约束及稳健指标：考虑杠杆风险用核心资本充足率来衡量，由于目前银行整体系统性风险还是较小，每相对8.5%多一个点给予2%的溢价。招商银行年报核心资本充足率为10.83%，（10.83%-8.5%）*2+1=1.0466。

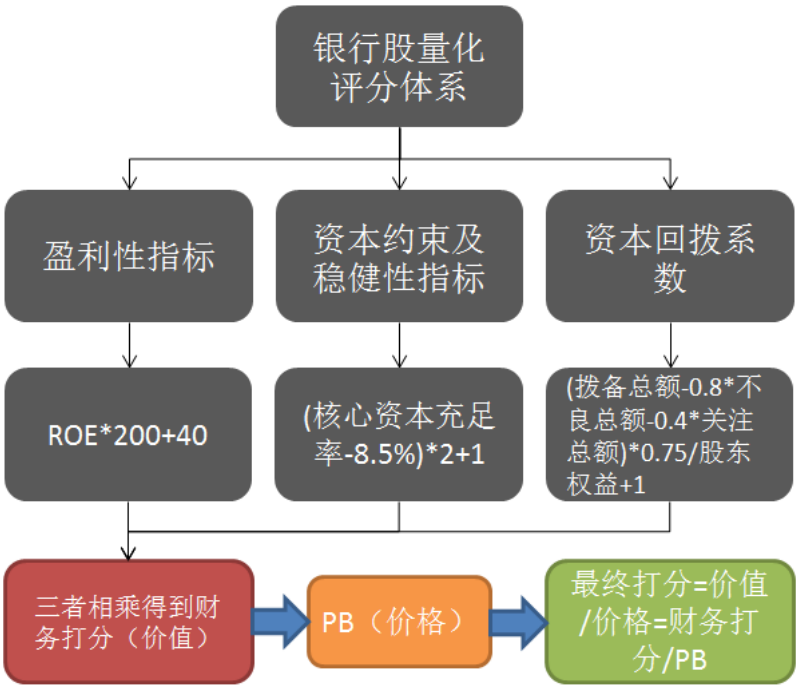
3、资本回拨系数：严格保守一点，80%的不良贷款损失掉，40%的关注类贷款损失掉，80%的逾期超90天损失掉，40%的逾期90天内损失掉，考虑逾期和这个关注不良是重复的除于2就能得到拨备还剩余多少或亏欠多少。然后再扣掉25%的税收，比上股东权益，就可以统一得到一个资本回拨系数，也就是拨备盈亏情况，也就是（拨备总额-0.5（80%不良贷款+40%关注贷款+80%超90天逾期+40%90天内逾期））*0.75/普通股股东权益+1。招商银行拨备总额为848亿，关注贷款738亿，不良贷款474亿，90天内逾期为354亿，超90天逾期和重组贷款为495亿，普通股股东权益为3608亿，计算下来就是1.0504。

4、年报财报评分为：81.51×1.0466×1.0504=89.61分。

5、年报数据比较全面，模型能最佳展示，对于季报没有公布的关注贷款按照不良贷款的同比增加，对于逾期重组贷款指标继续用年报数据，对于当年净资产收益率用当期同比乘去年全年数据，这样一季报也能给予评分，盈利性指标评分为78.20，资本约束指标为1.0645。资本回拨系数为6.45%，一季报评分89.29分。

6、4月30日，A股市净率1.181倍，H股市净率0.946倍。用财报的评分去除于市净率，A股的评分为89.29/1.181=75.6分，H股的评分为89.29/0.946=94.4分。到这一步，模型都是盲评的，是根据静态数据算出来的，也就是按照指标财报评分是都是，单位估值评分为多少，这个就是性价比，也是能够横向对比的值。

评分说明请看下图



评分导入

由于我们需要自己进行分数的计算，我们的结果是储存在一个CSV文件中，我们使用的格式如下。

code	2009-03-31	2009-06-30	2009-09-30	2009-12-31
000001.XS	71.57333191	106.1090291	113.3708452	76.44830163
002142.XS	80.68227165	93.41029163	92.25940732	75.85249637
600000.XS	89.32401288	124.4147076	124.5656868	94.84438969
600015.XS	62.23103047	84.0123636	84.09315087	67.51060623
600016.XS	71.49669208	104.6391255	106.0156006	79.58751086
600036.XS	85.03492873	107.1850327	107.1886106	87.10948315
601009.XS	87.78836905	92.70132946	91.34920394	77.84739518
601166.XS	88.90539595	112.3554268	112.0194401	91.19036572
601169.XS	106.5679726	111.3589456	112.07897	91.46693148
601288.XS	44.20308151	56.00005753	56.1121674	62.1211661
601328.XS	69.10395288	83.34113928	82.08069925	72.24986346
601398.XS	77.59135548	96.74887492	97.45513907	75.92230504
601818.XS	79.5691435	97.32826879	97.45045423	84.88002603
601939.XS	80.99286669	99.0779037	100.3436898	78.66074065
601988.XS	67.83317647	90.57139227	90.52856043	68.05021043
601998.XS	72.6130722	88.01543976	88.44880354	68.34289802

那么，要怎么在回测中使用这个CSV文件中的数据呢？大家可以戳链接：读取文件（密码：g9y9）

具体步骤：我们首先要在研究模块上传我们的CSV文件，点击upload，选择相应文件。

之后，在我们的回测中使用下列命令就可以读取数据。

```
from pandas import Series, DataFrame
from six import StringIO
g.data_body = read_file("bankshares.csv")
g.rating_data = pd.read_csv(StringIO(g.data_body))
```

策略构建

我们已经构建出了一个评级的指数，接下来要做的全部事情就是根据这个评级指数进行买卖调仓。

1. 一共有16只银行股，我们从中选出num=4只，加入我们的投资组合。
2. 每当有财务报表更新时，我们的评级指数也会发生变化，这时就要判断是否要对仓位进行调整。始终保持选择评级最高的num=4只股票加入我们的投资组合。但是这里要注意，我们只有当能赚到k=3%的利润时才会进行调仓，也就是说，假设原先的前4名是A,B,C,D。现在的前四名是A,B,C,E。只有当E的评级指数比D高3%以上时，我们才会进行调仓。具体操作见下图。
3. 每当排名发生变化时，从新考虑是否满足超过3%的条件，满足则再次调仓。
4. 对于这num=4只股票，我们会为他们分配一个权重，初始权重按照评分从高到低为4:3:2:1，之后的权重调整见下图。我们将按照这个权重分配全部的资产，即假设一共有100万的资产，则如果当前的四只股票比例为4:3:2:1，则第一只股票买100*4/(1+2+3+4)万元。

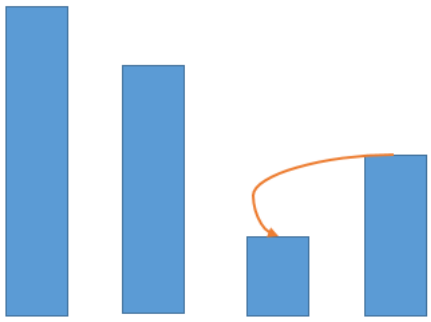
评分与调仓

这算是云蒙的策略在实际实践时的一个小麻烦，我们下面详细的讲解一下具体做法。

假设原始的评分最高4只股票为A100分，B95分，C80分，D60分，所以权重为A4：B3：C2：D1。经过新一轮的重新评分后，假设最高的4项为A110分，B100分，D85分，C75分。则AB的排名没有变化，不用去管，D上升到了第三名，所以理论上D应该增加一份，C应该减少一份。但我们要先判断一下D=85分是否比C=80分上升了3%以上，计算后发现是大于3%的，所以D增加一份，C减少一份。最后为A4：B3：D2：C1。当股票数目更多时，我们就要对多组股票的分数进行比较，这里遵循一个原则，始终从右面开始比较！具体什么叫从右面开始比较呢？我们看下图。

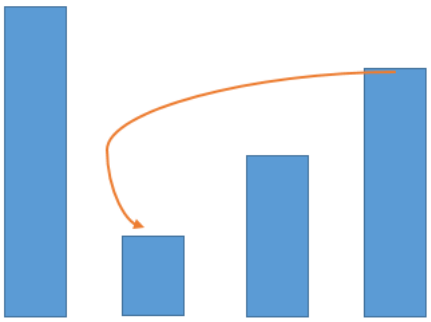
示例1

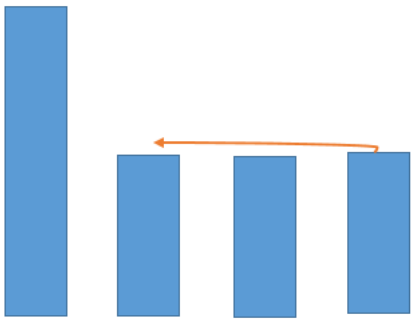
当从新排序后，我们发现顺序发生变化。我们遵循从右边开始的原则，发现现在排在第四位的股票有两份（因为上一期排在第三），而这个位置理论上只应该有一份。所以我们又遵循从右边开始的原则，看第四只股票往左数的第一只（即第三只）股票，判断一下是否超过了3%，若超过，则按红线所示移动一份的份额。



示例2

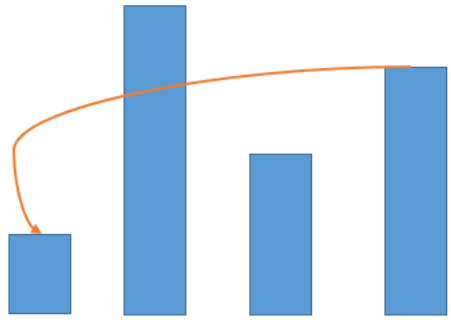
从右边开始看，发现第四只股票份额不对，应该有一份，但实际有三份（因为上一期排在第二），所以看他左边第一只（第三只）是否少份数，发现不少。则再向左一只（第二只），发现他少两份，则比较是否超过3%，若超过则从第四只移动两份的份额到第二只。



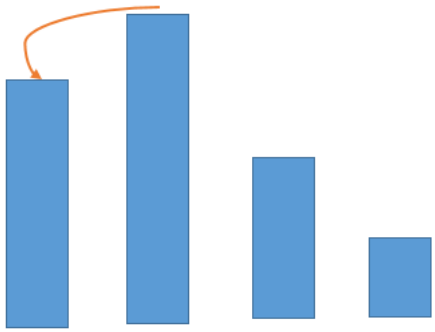


示例3

从右边开始看起，发现第四只的理论份额和实际份额不符，多两份，看他的左数第一只（第三只），发现不少份数，再往左看一只（第二只）同样不少份数。再往左看一只（第一只），发现少。则比较是否满足3%原则，满足则第四只移动两份到第一只。



结束后，看第三只，发现份数正确。结束后，看第二只，发现多一份，往左数一只（第一只），发现少一份。如果满足3%原则，则第二只往第一只移动一份的份数。



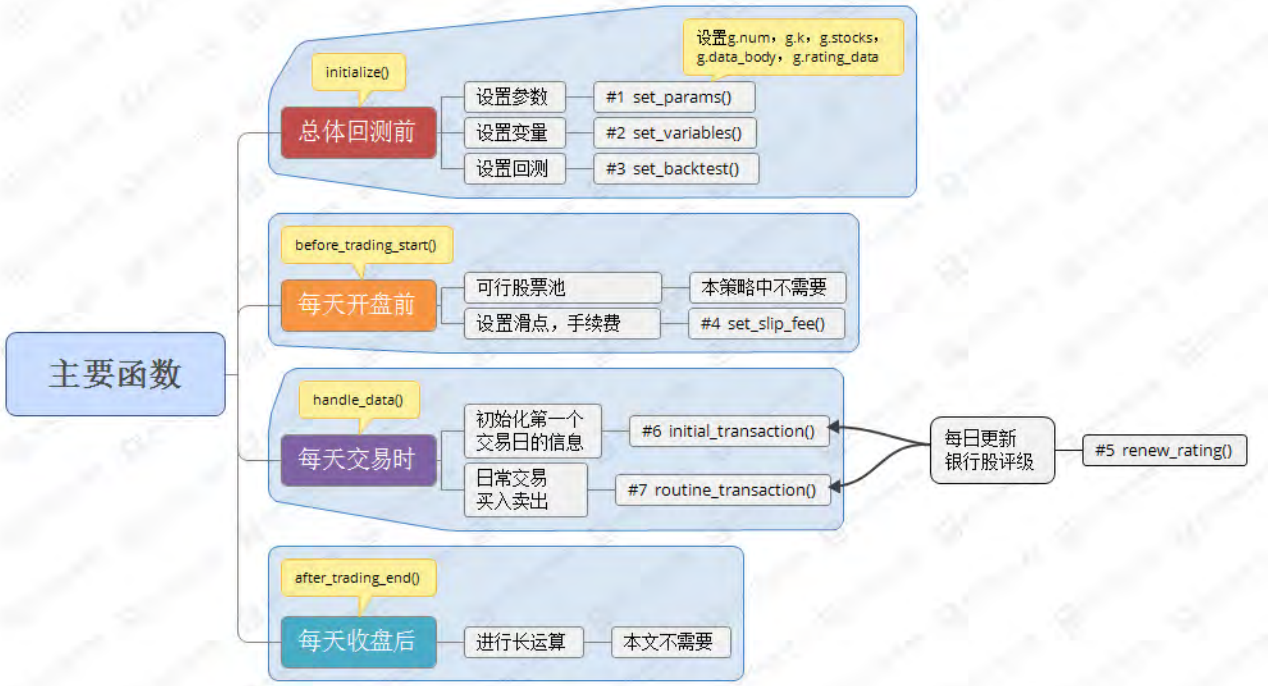
从上面的例子我们可以看出，如果有的时候，个别股票之间排名虽然发生了变化，但如果未满足3%的条件，则四只持仓股票的份额会发生变化，即不再是4:3:2:1。这是策略中的正常现象~

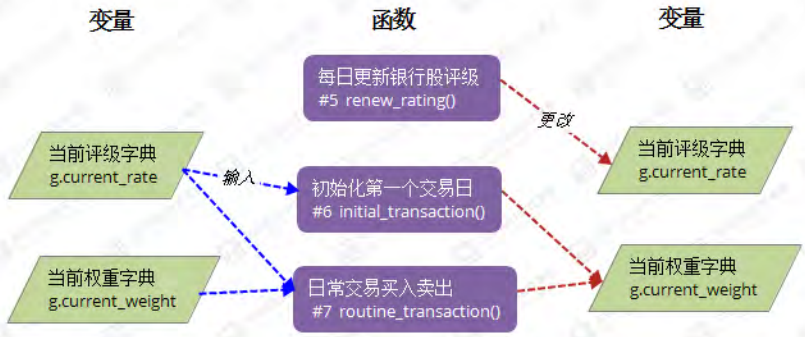
结果展示

我们在JoinQuant回测了一下该策略，基准收益设置为银行股指数的收益，结果如下所示

总结

以上就是我们从雪球大V云蒙处的策略进行适度修改所得到的量化交易策略！大家觉得怎么样呢？不放克隆下面的回测，自己看一看代码，跑一跑，从而更全面的了解整个策略！





文章更迭记录：
v2.0，2016-07-19，改为规范代码，添加函数说明书
v1.2，2016-07-18，添加CSV文件
v1.1，2016-07-04，添加“导语”
v1.0，2016-05-21，文章上线

【量化课堂】多因子策略-APT模型

导语：APT（Arbitrage pricing theory）定价理论是CAPM的一个推广，它们都是均衡状态下的模型，不同的是：CAPM把收益单纯的归为市场变化这一个因子引起的。APT把收益归因在不同的因子上面。本文旨在运用多元线性回归选股的方法，构建多因子策略。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： 导数君
编辑： 宏观经济算命师

阅读前需要了解：
线性回归模型，理解深度：level-0
APT模型，理解深度：level-0

备注：以上达到了解程度即可，即知道内容的思想，应用场景，输入输出。不需要证明。对于知识点本身，暂时安装一个外部链接，量化课堂将会自己产生内容覆盖，敬请期待。

多因子选股策略在之前的初阶教程中有所提及。对于因子的选取，我们当时做了一些简单的处理，例如人工选股以及等权重等。

今天小编带着大家研究一下如何运用多元线性回归选股的方法。

首先，我们来看一下多元线性回归模型：

多元线性回归简介

在线性回归分析中，如果有两个或两个以上的自变量，就称为多元线性回归。事实上，一种现象常常是与多个因素相联系的，由多个自变量的最优组合共同来预测或估计因变量，比只用一个自变量进行预测或估计更有效，更符合实际。因此多元线性回归比一元线性回归的实用意义更大，尤其在选股模型中，股价的涨跌的影响因素一般来说不止一个。

从数学的角度来说，我们假设因变量是自变量X1,X2,X3..Xk的线性函数，用方程来表示就是：

$$Y < em > i = \beta_0 + \beta_1 X < /em > 1i + \beta < em > 2X < /em > 2i + \ldots + \beta < em > kX < /em > ki + \varepsilon_i$$

其中Yi表示因变量（被解释变量）的第i个观测值，而Xki则是第k个自变量（解释变量）的第i个观测值，是自变量Xk的系数，Ei是第i组观测值的残差项。在金融领域，beta0有时候会写成α，用方程来表示就是：

$$Y < em > i = \alpha + \beta_1 X < /em > 1i + \beta < em > 2X < /em > 2i + \ldots + \beta < em > kX < /em > ki + \varepsilon_i$$

然而，字母的改变并不会影响模型的本质。

多元线性回归通常采用普通最小二乘法（OLS）进行估计，普通最小二乘估计法的思路是改变

$$\beta_0, \beta_1, \beta_2, \ldots, \beta_k$$

使得残差的平方和最小。其参数估计结果为：

Misplaced &

其中

Misplaced &

$$\begin{bmatrix} 1 & amp; X < /em > 11 & amp; X < em > 12 & amp; \ldots & amp; X < /em > 1k\,1 & amp; X < em > 21 & amp; X < /em > 22 & amp; \ldots & a \end{bmatrix}$$

Misplaced &

好，这个模型就介绍到这里，我们这里只要知道这个多元线性回归的一种用法就行了：现在你猜测，一件事情可能被好几个因素所影响，那么可以尝试一下线性模型。

关于这个模型的详细介绍，大家可以参阅聚宽量化课堂的数学部分（敬请期待）。

套利定价理论（APT）

套利定价理论认为，套利行为是现代有效率市场（即市场均衡价格）形成的一个决定因素。如果市场未达到均衡状态的话，市场上就会存在无风险套利机会。并且用多个因素来解释风险资产收益，并根据无套利原则，得到风险资产均衡收益与多个因素之间存在（近似的）线性关系，其定价方程是：

$$r < em > i = \alpha + \beta_1 F < /em > 1i + \beta < em > 2F < /em > 2i + \ldots + \beta < em > kF < /em > ki + \varepsilon_i$$

其实小编总觉的那些口口声声说套利的，其实根基都是建立在统计之上，给我感觉就像城市建立在流沙之上。

在我的理解里（或者本篇文章的体系下），Arbitrage pricing theory定价理论是CAPM的一个推广，由APT给出的定价模型与CAPM一样，都是均衡状态下的模型，不同的是CAPM嘛，其实你把市场变化看成一个因子也行，只不过CAPM把收益单纯的归为市场变化这一个因子引起的。APT把收益归因在不同的因子上面。

是股票i的收益率， 是第i个股票第k个因子的值，这个方程和多元线性回归模型的方程是一致的，因此可以用线性回归的方法来估计参数并根据当前的因子值预测未来的收益。

那么我们应该如何选择因子呢？

如何选择因子？

根据入门级内容，我们可以先主观的选定如下因子：

类型	聚宽的因子名称	正显著	负显著	总的显著
财务指标	eps	8	13	21
财务指标	roa	6	9	15
财务指标	adjusted_profit	5	8	13
财务指标	roe	7	6	13
财务指标	inc_return	6	7	13
财务指标	operation_profit_to_total_revenue	5	7	12
财务指标	net_profit_margin	4	7	11
财务指标	gross_profit_margin	5	5	10
财务指标	net_profit_to_total_revenue	3	7	10
财务指标	ga_expense_to_total_revenue	7	3	10
财务指标	goods_sale_and_service_to_revenue	4	6	10
财务指标	inc_total_revenue_year_on_year	7	3	10

那么，是不是直接使用这些因子进行回测就高枕无忧了呢？事实上并不是的，细心的读者可能会发现，这些因子中的一部分本来就存在一定的相关关系，比如roe和inc_return。这两者具有高度相关性。这个现象被称作“多重共线性”，指的是线性回归模型中的解释变量之间彼此之间度相关，从而使模型失真。那么多重共线性应该如何处理呢？

多重共线性怎么办？

主流的解决方法有如下几种：
1）排除引起共线性的变量：找出引起多重共线性的解释变量，将它排除出去，以逐步回归法得到最广泛的应用。
2）差分法：时间序列数据、线性模型：将原模型变换为差分模型。
3）减小参数估计量的方差：岭回归法；等等等等。

这几种方法大家都可以尝试一下，以后我们深入讨论。小编这里做一个简（偷）化（懒），以第一种方法为例，讲述如何排除引起共线性的变量。就实排除的方法也有很多种，比如相关系数法、修正的R方法、主成分分析法、人工判断法等。我们继续简化，用相关系数法进行处理。思路是采用因子的数据，直接计算普通的相关系数，然后从相关系数较大的几个因子中选择一个。参见下表：

	eps	roa	adjusted_	roe	inc_retur	operatior	net_profi	gross_prc	net_profi	ga_expens	goods_sal	inc_total
eps	1	0.704427	0.174677	0.322964	0.317088	0.305367	0.150508	0.275649	0.151978	-0.04453	-0.0249	0.045905
roa	0.704427	1	0.137924	0.397507	0.393515	0.461183	0.30339	0.387853	0.304454	-0.04101	-0.04856	0.065504
adjusted_	0.174677	0.137924	1	0.061289	0.064079	0.048721	0.020728	0.057139	0.021368	-0.02893	0.007614	0.001386
roe	0.322964	0.397507	0.061289	1	0.993475	0.174488	0.128135	0.125114	0.129268	-0.02034	-0.03323	0.027557
inc_retur	0.317088	0.393515	0.064079	0.993475	1	0.172835	0.126109	0.128698	0.127424	-0.02154	-0.03426	0.030076
operatior	0.305367	0.461183	0.048721	0.174488	0.172835	1	0.218268	0.318926	0.213927	-0.51571	-0.03518	0.013181
net_profi	0.150508	0.30339	0.020728	0.128135	0.126109	0.218268	1	0.370714	0.99402	0.272279	0.033729	-0.00042
gross_prc	0.275649	0.387853	0.057139	0.125114	0.128698	0.318926	0.370714	1	0.372156	0.254361	0.091187	-0.01059
net_profi	0.151978	0.304454	0.021368	0.129268	0.127424	0.213927	0.99402	0.372156	1	0.271173	0.034172	-0.00043
ga_expens	-0.04453	-0.04101	-0.02893	-0.02034	-0.02154	-0.51571	0.272279	0.254361	0.271173	1	0.073159	-0.0322
goods_sal	-0.0249	-0.04856	0.007614	-0.03323	-0.03426	-0.03518	0.033729	0.091187	0.034172	0.073159	1	-0.04997
inc_total	0.045905	0.065504	0.001386	0.027557	0.030076	0.013181	-0.00042	-0.01059	-0.00043	-0.0322	-0.04997	1

有没有发现roe和inc_return的相关系数0.99十分显眼？把这个inc_return拉出去砍了！去掉后结果如下图所示：

	eps	roa	adjusted_	inc_retur	operatior	net_profi	gross_prc	net_profi	ga_expens	goods_sal	inc_total
eps	1	0.704427	0.174677	0.317088	0.305367	0.150508	0.275649	0.151978	-0.04453	-0.0249	0.045905
roa	0.704427	1	0.137924	0.393515	0.461183	0.30339	0.387853	0.304454	-0.04101	-0.04856	0.065504
adjusted_	0.174677	0.137924	1	0.064079	0.048721	0.020728	0.057139	0.021368	-0.02893	0.007614	0.001386
inc_retur	0.317088	0.393515	0.064079	1	0.172835	0.126109	0.128698	0.127424	-0.02154	-0.03426	0.030076
operatior	0.305367	0.461183	0.048721	0.172835	1	0.218268	0.318926	0.213927	-0.51571	-0.03518	0.013181
net_profi	0.150508	0.30339	0.020728	0.126109	0.218268	1	0.370714	0.99402	0.272279	0.033729	-0.00042
gross_prc	0.275649	0.387853	0.057139	0.128698	0.318926	0.370714	1	0.372156	0.254361	0.091187	-0.01059
net_profi	0.151978	0.304454	0.021368	0.127424	0.213927	0.99402	0.372156	1	0.271173	0.034172	-0.00043
ga_expens	-0.04453	-0.04101	-0.02893	-0.02154	-0.51571	0.272279	0.254361	0.271173	1	0.073159	-0.0322
goods_sal	-0.0249	-0.04856	0.007614	-0.03426	-0.03518	0.033729	0.091187	0.034172	0.073159	1	-0.04997
inc_total	0.045905	0.065504	0.001386	0.030076	0.013181	-0.00042	-0.01059	-0.00043	-0.0322	-0.04997	1

还可以发现net_profit_to_total_revenue和net_profit_margin相关性也很大，而且eps和roa相关性也较大，我们继续手起刀落手起刀落。结果如下图所示：

	roa	adjusted_	inc_retur	operatior	net_profi	gross_prc	ga_expens	goods_sal	inc_total
roa	1	0.137924	0.393515	0.461183	0.30339	0.387853	-0.04101	-0.04856	0.065504
adjusted_	0.137924	1	0.064079	0.048721	0.020728	0.057139	-0.02893	0.007614	0.001386
inc_retur	0.393515	0.064079	1	0.172835	0.126109	0.128698	-0.02154	-0.03426	0.030076
operatior	0.461183	0.048721	0.172835	1	0.218268	0.318926	-0.51571	-0.03518	0.013181
net_profi	0.30339	0.020728	0.126109	0.218268	1	0.370714	0.272279	0.033729	-0.00042
gross_prc	0.387853	0.057139	0.128698	0.318926	0.370714	1	0.254361	0.091187	-0.01059
ga_expens	-0.04101	-0.02893	-0.02154	-0.51571	0.272279	0.254361	1	0.073159	-0.0322
goods_sal	-0.04856	0.007614	-0.03426	-0.03518	0.033729	0.091187	0.073159	1	-0.04997
inc_total	0.065504	0.001386	0.030076	0.013181	-0.00042	-0.01059	-0.0322	-0.04997	1

等等！上图有个误杀的，想了想，还有用eps吧。roa君，手动再见！

在删掉了相关性较大的因子之后，终于选出了9个因子：

eps（每股收益EPS(元)）
adjusted_profit（扣除非经常损益后的净利润(元)）
inc_return（净资产收益率(扣除非经常损益)(%)）
operation_profit_to_total_revenue（营业利润/营业总收入(%)）
net_profit_margin（销售净利率(%)）
gross_profit_margin（销售毛利率(%)）
ga_expense_to_total_revenue（管理费用/营业总收入(%)）
goods_sale_and_service_to_revenue（销售商品提供劳务收到的现金/营业收入(%)）
inc_total_revenue_year_on_year（营业总收入同比增长率(%)）

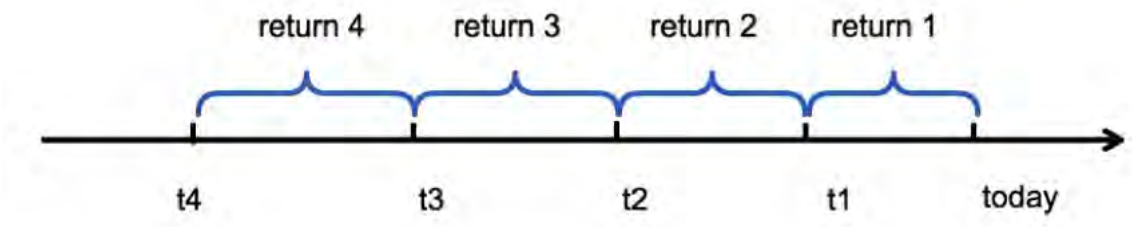
策略实现

我们选择了沪深300 作为股票池，我们暂且认为，这些股票池里面的股票性质相似，因此我们假设同一个因子对这些股票起到的贡献是相同的（未来会针对这一假设推翻）。

选定一个调仓频率。比如每个季度，我们设定一个调仓日期，然后每到调仓日进行回归分析和调仓。

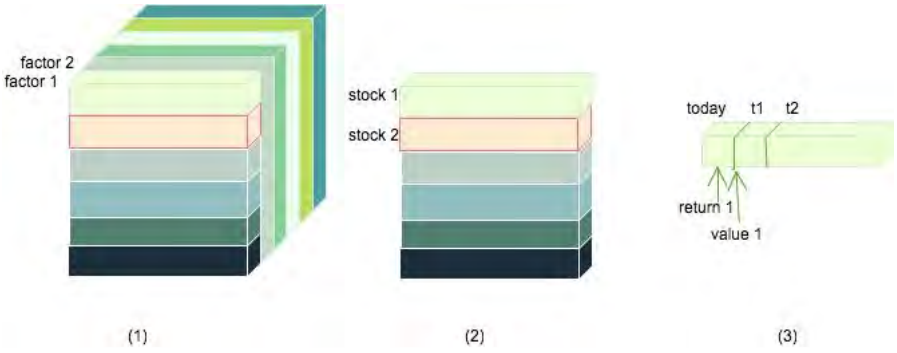
回归主要分析因子对股票（所有股票）的重要程度是多少。那么到底是谁和谁回归呢?

下图表示单个股票单只因子的取法。对于单只股票的单个因子，t1日的因子值和t1~today期间的平均对数收益率return 1组成了一组观测值，t2日的因子值和t2-t1时期的平均对数收益率return 2组成了一组观测值，一直到t4日的因子值和t4-t3时期的平均对数收益率return 4组成了一组观测值。



如果我们取N支股票，滚动向前的4期财报数据（讲道理的话，年报和下一年的一季报几乎一起公布，所以有效的数据只有三期？这里我们暂且认为有4期吧），对于单个因子，可以得到4N个观测值，用这4N的观测值可以回归出因子1在这段时期（t4-today）对这 N支股票的影响。然后将所有的待选因子采取这样的方法，放到同一个回归方程里，估出方程的回归系数，即β_1,β_2…β_k。

以上的话有点绕口，但是小编写了半天，其实也没啥好方法，就把它们画出来了：



先从(3)看起，有没有觉得跟上述时间轴图很像呢？没错，他代表一只股票关于一个因子的时间序列数据。

当我们将N只这样的股票数据排个队，便得到了图(2)。这些数据都是关于一个因子的，此时已经可以做单因子回归了。
我们把这样的很多因子叠在一起，即图(1)就是我们所需要的全部数据啦。

最后是预测。用最近发布的因子，比如图中的t1时刻公布的数据，预测各个股票未来的日均对数收益率，选择预期日均收益率较大的一些股票进行持仓。

这个是小编的回测结果，每次选取预测最好的前20只股票进行回归：

自2006年至今的绝对收益率为579.21%，同期的基准收益率为231.64%。我把收益率的数据导出，放到excel表格里面看看超额收益率

可以看出，这个策略收益总体能超越大盘，有正的alpha，可以尝试往对冲策略这个方向发展，但是超额收益其实跟择时也有很大的关系。在2008年那波熊市之前，超额收益率做了一个过山车被吐回来了。真正产生超额收益率的是09至15年的牛市顶峰。虽然看似15年以后超额收益率回撤挺大。但是由于比例尺放大了造成的。我们贴一个从15年开始的回测：

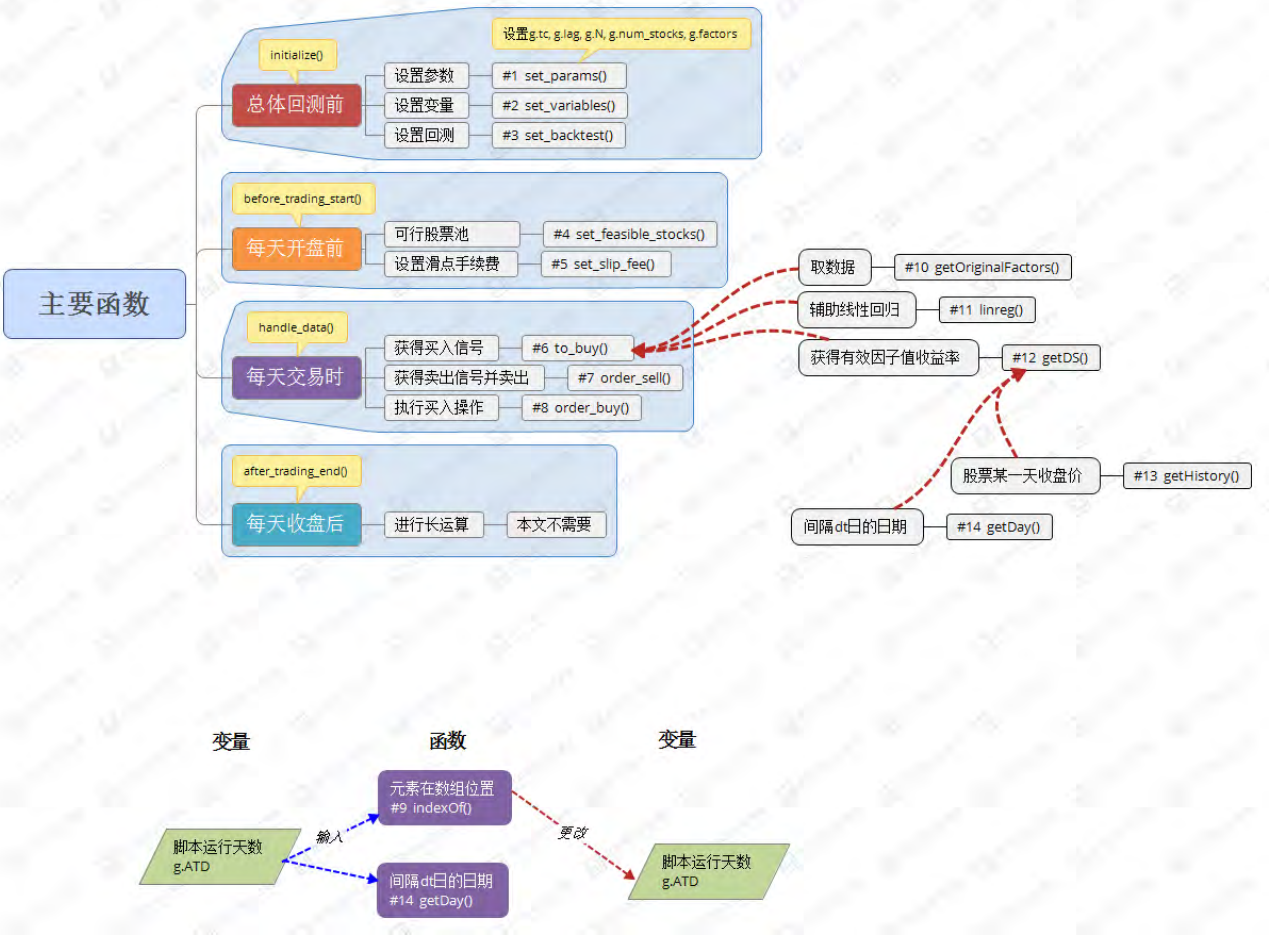
可以看到基本上策略与基准打平。但是别忘了15年这么牛熊一折腾，基差由正基差变成负基差了诶，所以真正的跑起对冲来，还是有基差可以吃的。

老师，收益率不高回撤大啊，能不能再给力点啊?
你们一个二个的想干嘛。真想找好策略，得自己去挖啊。
作为一个演示策略，啥参数都没调，其实已经像模像样了：）

未来你可以改进方向：

1. 扩充股票池，扩充因子，不要只选沪深300，沪深300可能起作用的因子都是偏价值的，但是成长系的白马黑马股统统没考虑啊，妖股小市值也没有入池啊。这些小伙伴可以都带上一起玩耍。
2. 因子的筛选再考究些，比如分两步筛选，第一步入选百只左右的因子，然后按照某种统计方法将因子筛出。
3. 目前函数回测进行调仓的日期是固定的63天，但是随着时间的推移，原来在1月份的调仓的日期慢慢就偏移到2月3月去了，我们可以固定一个日期进行回测。等等？！难上文所说的07年和15年效果不好是偏移造成的？待我修改修改再回测。
4. 我们可以把商榷一下假设。不太可能一个因子对所有股票的影响都是一样的，所以可以对单独的股票单独寻找因子回归嘛。
5. 对很多细节可以提高，比如对传入的因子进行数学处理，等等。
6. 本策略此如果能够结合择时策略，收益会更加好。这个择时策略就留给各路英雄尝试了~

好了，进阶篇之运用多元线性回归法选股策略就介绍到这里了，如果有兴趣，克隆代码块，自己修补一下。



文章更迭记录：
v2.0，2016-07-18，更新为规范源码，添加“函数说明书”
v1.1，2016-07-04，添加“导语”
v1.0，2016-05-31，文章上线

【量化课堂】均值回归入门

导语：假设“跌下去的迟早要涨上来“，则一支股票的价格低于其均线越多的时候，它回归的可能性就越大，也就越值得买。本文选取偏离度最高的几支股票并进行调仓，意指在价格震荡中博取反弹。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者：肖睿
编辑：宏观经济算命师

均值回归

均值回归（Mean Reversion），是在价格震荡中博取反弹的交易思路，它是基于Poterba和Summers（1987）首先提出的一种现象，如果要用一句话总结，那就是“跌下去的迟早要涨上来”。想理解均值回归，我们要先讲一下价格的波动性。

看过K线图的话，我们都知道股票的价格从来都不会平滑地上涨或下跌，而总是在移动的过程中上下波动，画出一些波浪。

如果市场满足Fama提出的市场有效性假说，那么这种波动现象应该是不存在的，然而现实中的市场并不是完全有效的。有许多研究尝试解释价格波动的现象，专家们普遍认为是诸如投资热点、对新闻的过度反应以及投资者的投机心态等非有效因素致使价格偏离了合理定价，从而造成了价格的波动。

均值回归的理论基于以下观测：价格的波动一般会以它的均线为中心。也就是说，当标的价格由于波动而偏离移动均线时，它将调整并重新归于均线。那么如果我们如果能捕捉偏离股价的回归，就可以从此获利。

举例来说，下图是一支股票的日线图，其中紫色曲线是20日均线，蓝色箭头为在股价大幅度偏离时买入股票可以获得的收益。

一个最简单的策略

根据均值回归的思路，我们认为一支股票的价格低于其均线越多的时候，它回归的可能性就越大。因此，我们可以以一支股票的价格与其均线的偏离程度作为评估标准，并选择买入该偏离度最高的股票。

那么偏离度该如何计算呢？如果用P代表现在的股价，用MA代表均线的价格，那么这个度量是 (MA-P)/MA。

好了，你也许有两个问题：

1. 为什么是均线减价格？
   因为我们认为价格低于均线越多越值得买，因此用均线减去价格算出的差率越高我们认为越值得买。
2. 为什么要除以均线？
   因为价格和均线的差的单位是元，需要除以均线才可以在股票之间相互对比。比如股票A的价格是1，均线是2，股票B的价格是99，均线是100；它们的均线和价格差都是1，但很明显按照我们的思路，股票A更值得买，这要除以均线才能体现出来。

那么策略如下：
· 先决定好参数：选定股票池，以N日移动均线作为比价基准，按每B天为周期调换仓位，以及仓内持有的股票数量num_stocks。
· 在每个调仓日进行以下操作：

1. 计算池内股票的N日移动均线；
2. 计算池内所有股票价格与均线的偏离度；
3. 选取偏离度最高的num_stocks支股票并进行调仓。

回测结果

以下回测都是以沪深300成分股作为股票池，每次调仓选择10支股票。

首先是均线长度20天，持仓5天的回测。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
437.99%	18.15%	233.34%	0.053	1.017	0.386	0.490	0.349	0.367	0.304	68.527%

效果还算可以，但是回撤巨大。可以看出在暴跌环境中，大家一起往死里跌，均值回归什么的都无所谓了。

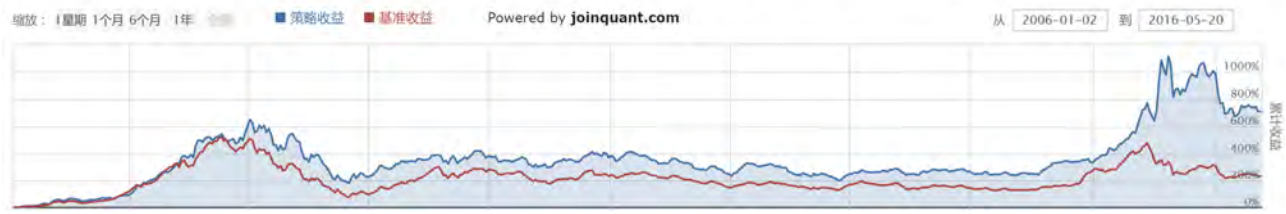
然后是均线长度5天，持仓1天。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
-82.93%	-16.07%	233.34%	-0.263	0.716	-0.700	-0.800	-1.461	0.287	0.304	96.799%

诶哟，我的天呐…

最后是均线长度30天，持仓30天。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
711.62%	23.07%	233.34%	0.110	0.926	0.563	0.750	0.523	0.339	0.304	64.085%



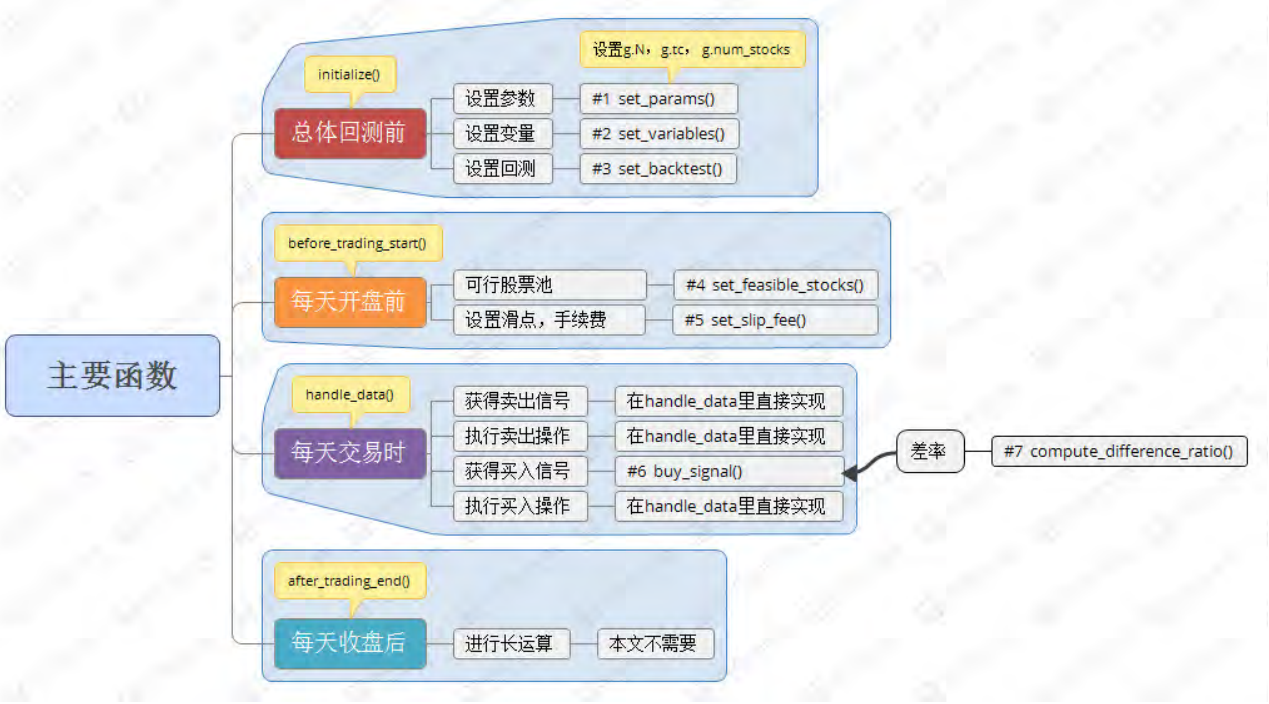
这个结果就不错了哟~~

策略的难点

从上面的回测可以看出，以均值回归的思路做出的简单策略虽然收益可观，但实则回撤猛烈，很不稳定。通过分析不难看出，主要的问题就是，买在半山腰了。

既然是博取反弹，那么难免会买在半山腰，比如在下图K线的情况中。假设我们在那根大阴棒的底部判断“哦，这个价格与均线偏差够大了”，然后买了进去，那就会接着吃跌。最后价格虽然又触碰到均线，但其实是价格在低位停留太久把均线拉了下来，实际上没有给我们什么收益。

所以，本策略的改进方向：一是通过定量统计分析的方法，判断更合理的入场时机，二是在买在半山腰后的止损判定。



文章更迭记录：
v2.0，2016-07-16，更新为规范源码，添加“函数说明书”
v1.1，2016-07-04，添加“导语”
v1.0，2016-06-21，文章上线

【量化课堂】Fama-French 三因子火锅

导语： CAPM模型认为，收益风险同源。市场风险是唯一能给股票带来超额收益的风险。但是事实上除了市场风险外，Fama-French认为市场上还存在还有市值风险，账面市值比风险等，据此建立的模型被称为“Fama-French三因子模型”。本文旨在深入浅出介绍三因子模型的思想并提供一个选股应用。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： 导数君
编辑： 宏观经济算命师

三因子火锅，既美味又营养

阅读前需要了解：
线性回归模型，理解深度：level 0
CAPM模型，理解深度：level 0

Fama-French 三因子模型概述

看到CAPM模型，大家有没有想过，为什么有的股票有正的超额收益，有的股票的超额收益却是负的？是不是市场风险不能够完全解释个股的超额收益？

是的。

Fama和French这两个人研究股票超额收益率的时候发现了一个神奇的现象：有两类股票的历史平均收益率一般会高于CAPM模型所预测的收益率。它们是小公司股票、以及具有较高股权账面-市值比的股票。Fama和French认为：1）市值比较小的公司通常规模比较小，公司相对而言没那么稳定，因此风险较大，需要获得更高的收益来补偿；2）账面市值比就是账面的所有者权益除以市值（下以简称B/M）。B/M较高则说明市场上对公司的估值比公司自己的估值更低。这些公司一般都是销售状况或者盈利能力不是十分好的公司，因此相对于低B/M的公司来说需要更高的收益来补偿。

这个三因子模型的本质就是把CAPM中的α（未被解释的超额收益）分解掉，将其分解成市值因素、B/M因素和其他未被解释的因素（可以看成是新的α），可以用如下公式表达：

$$R_i = a_i + b_i R_M + s_i E(SMB) + h_i E(HML) + \varepsilon_i$$

其中$R_i = E(r_i - r_f)$，指股票i比起无风险投资的期望超额收益率。$R_M = E(r_M - r_f)$，为市场相对无风险投资的期望超额收益率，E(SMB)是小市值公司相对大市值公司股票的期望超额收益率，E(HML)则是高B/M公司股票比起低B/M的公司股票的期望超额收益率，而 ε_i 是回归残差项。

对三因子模型的理解

上面这个三因子模型和CAPM模型在表达式上面的区别就是多了几个回归的自变量。因此，
打眼一看，不少的读者（包括小编第一次接触的时候）可能会觉得这个模型只是为传统多因子模型提供了两个因子（市值、B/M）而已，然后用传统的因子打分、回归等方法进行选股。

小编只想说，然而并不是这样。

上面这种理解也可以用来建模，但它并不是本来Fama和French想表达的！那么他们究竟想说什么呢？我们再仔细看一下三因子模型的表达式：

$$R_i = a_i + b_i R_M + s_i E(SMB) + h_i E(HML) + \varepsilon_i$$

R_i和R_M是什么，前文刚介绍过。但这个E(SMB)怎么理解呢？Fama把市场里面的所有股票按市值排序，然后等分成三份：第一份是大市值股票（市值在所有股票中最大的1/3），第二份是中市值股票，第三份是小市值股票（市值在所有股票中最小的1/3）。记大市值股票的平均期望收益率为$E(r_S)$，小市值股票的期望收益率为$E(r_B)$。那么$E(SMB) = E(r_S) - E(r_B)$。E(HML)的定义也类似。

小编认为，三因子模型的贡献，在于发现了股票的期望收益不仅仅与市场的系统风险有关，还和市值风险和账面市值比风险有关。市值和B/M这一类因子是对市场整体进行一个衡量的，而不是对个股的衡量。

对于市场的衡量，我们也可以用多元线性回归的方法来估计。三因子模型的表达式中a_i, b_i, s_i, h_i都是回归系数，b_i描述的是股票本身的市场方面风险的大小，s_i描述的是股票本身的市值方面风险的大小，h_i描述的是股票本身的账面市值比方面风险的大小。

FF三因子模型套利法选股

举一个具体应用的例子：
我们再看一眼三因子模型的表达式（是最后一眼了，各位别烦躁）：

$$R_i = a_i + b_i R_M + s_i E(SMB) + h_i E(HML) + \varepsilon_i$$

如果默认三因子模型是正确的，而且市场风险、市值风险、账面市值比这三类风险能很好地解释个股的超额收益，a_i的长期均值应该是0。那么，如果对于某

个时期的股票，回归得到$a_i < 0$，说明这段时间里面收益率偏低（因此股价也偏低），而根据有效市场假设，出来混总是要还的，今天的偏离在未来要涨回来的。

所以我们的选股思路非常简单：

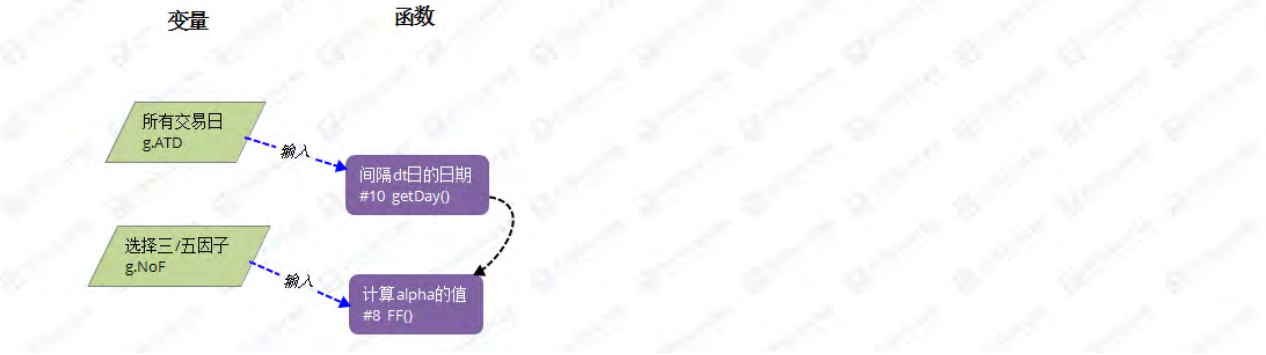
1. 先设定一个调仓频率，每T=10天调仓一次
2. 设定一个样本长度S=63天。
3. 然后在调仓日对于过去S天的数据进行回归分析，计算出每个股票在过去的S天里面α观测值，
4. 然后买入α最小的N（N=10）支股票即可。

以上参数皆可调整。2006年至今的收益率高达1484%，不仅跑赢了大盘，还跑赢了不少转化为多因子模型的方法的选股策略。下图表示这个策略的收益情况（股票池就是沪深300本身）：

这个策略的Alpha高达18.4%，而且beta非常接近1，因此可以使用沪深300指数来对冲市场风险从而获得超额收益，下图表示这个策略的净值和沪深300组合的净值的比值：

从这个图中我们可以看出，虽然在熊市的时候回撤较大，但是如果用适合仓位的沪深300股指期货进行对冲的话，收益还是比较稳定的，因此说明了这个策略还是非常有效的。

好啦，本篇到此为止，欢迎关注后续文章！



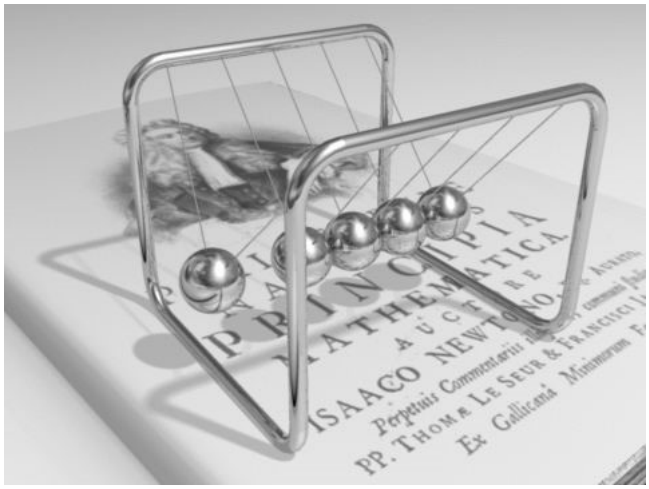
文章更迭记录：
v2.0，2016-07-16，添加“函数说明书”
v1.1，2016-07-09，更新为规范源码
v1.0，2016-06-29，文章上线

【量化课堂】动量策略入门

导语：假设"股票的收益率有延续原来的运动方向的趋势“。通俗来讲：今天涨了，明天大概率会接着涨，强者恒强。根据这种假设，我们可以用收益率、成交量、换手率等等作为动量因子，来构建动量策略。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： 何忆嘉年
编辑： 宏观经济算命师



Motivation

其实同一级别拆开来看，市场的状态无非就是两种，趋势和震荡。我们怎么在股票中赚钱呢？答：上涨趋势的时候死死拿着，震荡行情中高抛低吸，下跌趋势趁早收手。听上去好像不难？那为啥大家都不赚钱？
难点在于，你分不清当前是震荡还是趋势。没骗你，没说笑。真不好分，无数前人已经试过了。100%的分开是不可能的，能的话全市场的钱都是你的。比如今天突破了，你怎么知道明天是接着上涨呢？还是一棍子跌回去了？

所以，我们就换一种思路，从概率的视角下看待问题。只要发出趋势成立的信号，我就按照未来是趋势的假设来操作，跌破了，我就止损，到了合适的点位，我就止盈。那么只要上述信号带来的收益期望值大于0，我认为这些操作是会赚钱的。

动量效应（Momentum effect ）呢，是捕捉趋势的一种方法论，一般又称“惯性效应”，由Jegadeesh和Titman（1993）提出。他们认为，股票的收益率有延续原来的运动方向的趋势，即过去一段时间收益率较高的股票，在未来依旧会取得高于平均的收益率。通俗说，今天涨了，明天大概率会接着涨，强者恒强嘛。

根据这种假设，我们构建的策略就叫动量策略。

动量类因子的定义

上面说了，动量效应就是认为股票强者恒强。但是具体怎么操作，怎么算“强”呢？我们认为具有某些特征的股票就是“强者”，这些特征，可以被理解成信号，或者因子，换个名字而已。
一般而言，除了最广泛应用的收益率以外，成交量、换手率等等都可以作为动量因子，随手贴一个表格，光大证券研报里抠出来的：

PM_24M	过去 24 个月的动量=过去 24 个月的收益率
PM_12M	过去 12 个月的动量=过去 12 个月的收益率
PM_6M	过去 6 个月的动量=过去 6 个月的收益率
PM_3M	过去 3 个月的动量=过去 3 个月的收益率
PM_1M	过去 1 个月的动量=过去 1 个月的收益率
PM_6MC1M	过去 6 个月动量-过去一个月动量
PM_1DC1M	过去一个月收益率的最大值
Success	1-过去一个月收益率排名/股票总数
PM_1D	过去一日收益率
Price_3M	股价/过去 3 周股价平均值-1
PM_5D_Indu1	过去 5 日收益率，减去行业均值
Volume_1MC3M	过去 1 个月交易量/过去 3 个月交易量
Volume_1M	(月末交易量/过去日均交易量-1)*过去 1 个月收益率
PM_12MCH6M	(1+过去 12M 收益率)/(1+6 个月前 12M 收益率)-1

（参考《光大证券——多因子研究系列（五）：动量类因子测试》）

在不同的市场阶段、不同风格、不同行业和不同成分股的测试中，策略表现最好的动量因子均有所变化。下面，我们以最简单的收益率动量作为因子，简单实现一下纯动量策略。

具体策略

1.设置参数，包括股票池，调仓周期holdingPeriod，收益率计算周期returnPeriod，每次持仓股票占股票池的比例ratio，并计算每次持股数量N等。
2.计算股票池中所有股票的上一周期的收益率 （这里采取简单收益率 =
(昨天的收盘价- returnPeriod+1天前的收盘价)/ returnPeriod+1天前的收盘价）
3.将股票池内（除了已持仓的）所有股票按照收益率排序，始终持有上一收益率计算周期(returnPeriod)收益率前N的股票。
4.每holdingPeriod天进行Rebalance。

测试时间范围：2006-04-01 – 2016-04-01
(有关创业板指数的回测范围：2012-01-01 – 2016-04-01)

接下来我们来看看策略结果~
1. 调整收益率回测区间和调仓周期

股票池	持仓股数	returnPeriod	holdingPeriod	收益(%)	最大回撤(%)
沪深300成分股	10	126	20	54	70.88
	10	30	30	104	69.21
	10	20	20	43	74.29
	10	5	5	-77	90.36
	10	2	1	-91	93.93
沪深300基准收益	300			203.64%	

调仓周期和收益率回测区间较长时，动量策略表现一般，低于大盘。直觉上解释，股票一般兼具动能效应和均值回复的特性，而长周期更加偏向均值回复。同样，调仓周期和收益率回测区间较短时，动量因子表现很差，收益率为负。一方面，是因为调仓过于频繁，交易费用很高。另一方面，短期的收益率更偏向于随机波动，其动量很难被捕捉到。
从上述回测结果看来，最佳的收益率计算周期为30天，调仓周期为30天。但不论是收益率还是最大回撤，策略都没有跑赢沪深300指数。

让我们换种思路看看策略还有没有改进的地方。如果调整周期效果不明显，我们试试不同的股票池。比如，用创业板指成分股?
2. 调整股票池
N=10 returnPeriod=30 holdingPeriod=30股票池：创业板指数

果然好很多！不过这里使用系统默认基准——沪深300。由于股票池不同，策略与基准不具有可比性。把基准换成创业板试试。
N=10 returnPeriod=30 holdingPeriod=30股票池：创业板 基准：创业板

唔，我们发现，虽然策略跑赢了沪深300，但当将其与创业板指数对比时，还是不尽如人意。策略几乎与股票池完全贴合。会不会是我们持仓的股票数量太多了呢
3.调整持仓数量
N=15 returnPeriod=30 holdingPeriod=30 股票池：创业板 基准：创业板

N=5 returnPeriod=30 holdingPeriod=30 股票池：创业板 基准：创业板

N=3 returnPeriod=30 holdingPeriod=30股票池：创业板 基准：创业板

从收益率、最大回撤上看，N=5为最理想的参数。
从图形上看，在牛市期间，策略有明显的超额收益(即蓝线高于红线的部分)，其中N=3表现最佳。

小结:

就本文的测试结果来说，沪深300股票池的30日收益率动量因子比较有效、创业板作股票池比沪深300表现更好。动量策略在牛市能跑赢基准，在别的市场阶段表现不佳。

如何判断策略的适用性、如何调整参数，是很多策略所面临的问题。动量有效性与很多因素有关。如果深入研究下去，主要的方向还是尽可能的区分动量策略适用于怎么样的市场、适合什么类型的股票、以及收益率何时表现为均值回复、动能持续，亦或随机波动。本文提供了一个最简单的调参范例。除了人工手动代入，我们还可以通过编程迭代、统计机器学习等方法去调试参数。不过在这过程中不要一味地追求结果最好，这样很容易先入过分拟合的陷阱。量化课堂以后的内容也会深入迭代这一步分内容。

动量类策略也有一些明显缺陷，比如波动率、回撤比较大。作为研究的另外一个方向，就是看看怎么把上面这些问题解决了。比如可以设置一些止盈止损条件啦，多策略混合啦，或者用股指期货进行对冲啦。等等。

谁说跑不赢大盘的策略不是好策略，既然他稳定跑不赢大盘，我们反过来操作不就能稳定跑赢大盘了嘛！->请戳《均值反转策略》

好了，就到这里！
备注：该回测的结果并非最优，设置不同的参数，会产生不同的结果。如有需求，用户可以查看源码调整参数。

文章更迭记录：
v2.0，2016-07-18，更新为规范源码，添加“函数说明书”
v1.1，2016-07-04，添加“导语”
v1.0，2016-07-02，文章上线

【量化课堂】Fama-French 五因子模型

导语：上篇关于Fama-French三因子模型的文章很受欢迎（Fama-French三因子模型），我们再接再厉，推出了Fama-French五因子选股策略。
早在1993年，Fama和French两个人就已经发表了他们的三因子模型，认为股票的超额收益可以由市场风险、市值风险、账面市值比风险来共同解释。后来，这两个人发现了除了上述风险，还有盈利水平风险、投资水平风险也能带来个股的超额收益，并在2013年发表了五因子模型。本文旨在对五因子模型以及这几个因子进行简单的介绍，并给出一个简单而有效的五因子选股策略。

规范源码已更新！请大家克隆研究。
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者： 导数君
编辑： 宏观经济算命师

阅读前需要了解：
线性回归模型，理解深度：level 0
CAPM模型，理解深度：level 0
Fama-French三因子模型，理解深度：level 0
（详见Fama-French三因子模型）

Fama-French 五因子模型概述

我们先来回顾一下三因子模型，然后再引入五因子模型。

1993年Fama和French提出了著名的三因子模型，把个股的超额收益率分解成市值因素、账面市值比因素和其他未被解释的因素（可以看成是新的α），用数学式来表示就是：

$$< br > R_i = a_i + b_i R_M + s_i E(SMB) + h_i E(HML) + e_i < br >$$

其中$R_i = r_i - r_f$，就是股票i比起无风险投资的期望超额收益率，$R_M = r_M - r_f$，就是市场相对无风险投资的期望超额收益率，E(SMB)小市值公司相对大市值公司股票的期望超额收益率，E(HML)则是高B/M公司股票比起低B/M的公司股票的期望超额收益率，而 e_i是回归残差项。

如果三因子模型中的三个因子可以完全解释各种风险带来的超额收益，那么任何一个股票以及任何一个投资组合的a_i的真实值应该为0。在过去20年里面，很多学者对三因子模型进行实证分析，发现有些股票的a_i显著不为0，这说明三因子模型中的三个风险（因素）并不能解释所有超额收益。后来，Fama和French发现在上述风险之外，还有盈利水平风险、投资水平风险也能带来个股的超额收益，并于2013年提出了五因子模型。该模型更加充分地解释个股的超额收益，其表达式为：

$$< br > R_i = a_i + b_i R_M + s_i E(SMB) + h_i E(HML) + r_i E(RMW) + c_i E(CMA) + e_i < br >$$

从表达式看，五因子模型比三因子模型里面多出来了两项：E(RMW)是高/低盈利股票投资组合的回报之差，而E(CMA)则是低/高再投资比例公司股票投资组合的回报之差。这两项分别描述了盈利水平风险、投资水平风险（注意这里的投资水平不是二级市场的投资水平，而可以通俗的解释为企业扩大再生产的能力）。与三因子类似，参数估计的方法仍然是用多元线性回归的方法，这里的a_i则是五因子模型里面尚未解释的超额收益。

五个因子简单介绍

本小节主要简单介绍一下市场风险、市值风险、账面市值比风险、盈利水平风险、投资水平风险这五个风险。Fama和French在他们五因子模型的文章里面提到，这个模型的表现相对于因子的具体选取并不敏感。举个例子：不论你用市值还是用总资产来描述市值因素，对模型的影响并不会太大。因此如果在其他文献中看到的定义与本文有所不同，也无须大惊小怪。
2.1市场风险

市场风险是指大盘走势变化所引起的不确定性。简单来说，就是大盘波动导致个股也跟着波动的风险。比如表现比较好的公司，其股票价格却伴随着大盘下降了，或者表现不怎么好的公司，股价却跟着牛市上涨了。

市场风险是股票持有者所面临的所有风险中最难对付的一种，它给持股人带来的后果有时是灾难性的（比如2015年中让股民们心有余悸的“千股跌停”，表现再好的股票也难逃魔掌）。根据风险收益对等的原则，对于市场风险，应该有对个收益的超额补偿，这部分补偿计算方式为大盘指数相对于无风险投资的超额收益：

$$< br > R_M = r_M - r_f < br >$$

2.2市值风险

市值风险是指公司的规模对该公司股票的风险有着接影响：资产规模小，风险就会相对增加，反之，资产规模大，风险就会相对减少。企业的资产规模与风险的这种关系已经被广泛投资者所接受（例如聚宽平台上的小市值系列策略）。国际上亦有许多知名研究机构和研究人员发表过有关文章，阐述了资产规模与投资回报率之间的负相关关系。例如普华永道研究得出来的R=17.074-2.7lgA，其中R是股票收益率，A是公司总资产账面值（旨在描述公司规模）。

在五因子模型中，对市值风险的超额回报仍然用$E(SMB)$描述。$E(SMB)$的计算方式是：首先把市场里面的所有股票按市值排序，然后等分成三份：第一份是大市值股票（市值在所有股票中最大的1/3），第二份是中市值股票，第三份是小市值股票（市值在所有股票中最小的1/3）。记小市值股票的平均期望收益率为$E(r_S)$，大市值股票的期望收益率为$E(r_B)$。那么$E(SMB) = E(r_S) - E(r_B)$。

2.3账面市值比风险

账面市值比就是账面的所有者权益除以市值（下简称B/M）。账面市值比风险描述了公司的额外财务困境风险，说明市场上对公司的估值比公司自己的估值要低。这些公司一般都是销售状况或者盈利能力不是十分好的公司，因此相对于低B/M的公司来说需要更高的收益来补偿。

在五因子模型中，对市值风险的超额回报仍然用E(HMI)描述。E(HMI)的计算方式是：首先把市场里面的所有股票按B/M排序，然后等分成三份：第一份是高B/M股票（B/M在所有股票中最大的1/3），第二份是低B/M股票，第三份是低B/M股票（市值在所有股票中最小的1/3）。记高B/M股票的平均期望收益率为$E(r_H)$，低B/M股票的平均期望收益率为$E(r_L)$。那么

$$< br > E(HML) = E(r_H) - E(r_L) < br >$$

2.4盈利水平风险

盈利水平风险是指，盈利能力较高的行业一般会伴随着更高的风险。我们用ROE来衡量盈利水平。记做 $E(RMW)$，其计算方法和$E(SMB)$、$E(HML)$类似（也是将股票分成三份，然后计算高/低盈利水平的股票期望收益率之差）。

2.5投资水平风险

投资水平可以用再投资率来衡量，我们认为投资率偏低的公司风险较大，投资者对这些公司有更高的收益率要求，反之亦然。Fama和French在他们五因子模型的文章里面提供了一种计算再投资比例的方法：用总资产年增长率来计算再投资比率。投资水平风险带来的超额收益$E(CMA)$计算方法和$E(SMB)$、$E(HML)$、$E(RMW)$类似（也就是将股票分成三份，然后计算低/高再投资比例公司股票期望收益率之差）。

五因子模型选股应用

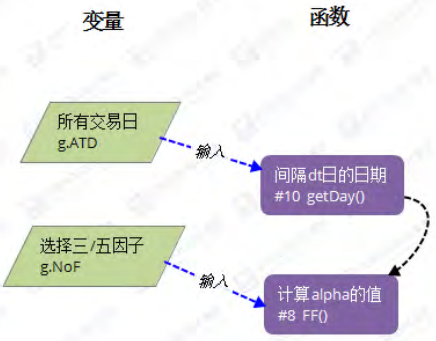
和《Fama-French三因子火锅》里面提到的选股思路差不多。假设五因子模型中的五类风险能够很好地解释个股的超额收益，a_i的长期均值应该是0。因此，如果对于某个时期的股票，$a_i < 0$，说明这段时间里面收益率偏低（因此股价也偏低），而由于均值需要回归，所以这个股票在未来要涨回来的。

这个选股思路非常简单，就是：
先设定一个调仓频率，比如每T天调仓一次。
设定一个样本长度，比如S天。
然后在调仓日对于过去S天的数据进行回归分析，计算出每个股票在过去的S天里面的a_i观测值。
然后买入a_i最小的N支股票即可。详细的回测代码及其注释见附录。

以下是我们将T设置成10天，S设置为63，N设置为10的回测结果：

虽然这个策略的收益没有三因子策略的收益高，但是仍然能跑赢沪深300指数。下图表示这个策略净值和沪深300组合净值之比:

确实收益率相对于三因子模型来说有所下降，其原因有待各位读者进一步的探索。不过小编要声明一下，其实策略的收益率只是诸多考量因素的一个罢了，对于对冲型策略来说，小编宁愿牺牲一些收益，换取更稳定的增长。



别着急哦，未来还有脑洞文，敬请期待。

文章更迭记录：

v2.0，2016-07-18，更新为规范源码，添加“函数说明书”
v1.0，2016-07-01，文章上线

【量化课堂】基于协整的搬砖策略

导语：配对交易（Pairs Trading）是通过一买一卖的手段来赚取两只股票走势的差价的投资策略。本文介绍如何通过协整关系实现配对交易，以及在缺乏卖空机制情况下的搬砖策略。

作者：Haozun，肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，难度为进阶下，理解深度为level-0。

阅读本文需要掌握协整（level-0）的知识。

配对交易

相信很多同学都了解过 Pairs Trading，即配对交易策略。其基本原理就是找出两只走势相关的股票。这两只股票的价格差距从长期来看在一个固定的水平内波动，如果价差暂时性的超过或低于这个水平，就买多价格偏低的股票，卖空价格偏高的股票。等到价差恢复正常水平时，进行平仓操作，赚取这一过程中价差变化所产生的利润。

使用这个策略的关键就是“必须找到一对价格走势高度相关的股票”，而高度相关在这里意味着在长期来看有一个稳定的价差，这就要用到协整关系的检验。

在量化课堂介绍协整关系的文章里，我们知道如果用 X_t 和 Y_t 代表两支股票价格的时间序列，并且发现它们存在协整关系，那么便存在实数 a 和 b，并且线性组合 $Z_t = aX_t - bY_t$ 是一个(弱)平稳的序列。如果 Z_t 的值较往常相比变得偏高，那么根据弱平稳性质，Z_t 将回归均值，这时，应该买入 b 份 Y 并卖出 a 份 X，并在 Z_t 回归时赚取差价。反之，如果 Z_t 走势偏低，那么应该买入 a 份 X 卖出 b 份 Y，等待 Z_t 上涨。所以，要使用配对交易，必须找到一对协整相关的股票。

这里要提醒读者，无论是原始的 Pairs Trading 策略，还是本篇的搬砖策略，在寻找股票对时，数据上的检验都只是辅助手段。我们首先要做的还是在基本面的角度进行分析，分析公司的主营业务，产品链，业内地位等。在此基础上，我们才会对有可能具有协整关系的股票进行数据上的检验。这是非常重要的。

协整关系的检验

我们想使用协整的特性进行配对交易，那么要怎么样发现协整关系呢?
在 Python 的 Statsmodels 包中，有直接用于协整关系检验的函数 coint，该函数包含于 statsmodels.tsa.stattools 中。
首先，我们构造一个读取股票价格，判断协整关系的函数。该函数返回的两个值分别为协整性检验的 p 值矩阵以及所有传入的参数中协整性较强的股票对。
我们不需要在意 p 值具体是什么，可以这么理解它： p 值越低，协整关系就越强；p 值低于 0.05 时，协整关系便非常强。

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
```

```
# 输入是一DataFrame，每一列是一支股票在每一日的价格
def find_cointegrated_pairs(dataframe):
    # 得到DataFrame长度
    n = dataframe.shape[1]
    # 初始化p值矩阵
    pvalue_matrix = np.ones((n, n))
    # 抽取列的名称
    keys = dataframe.keys()
    # 初始化强协整组
    pairs = []
    # 对于每一个i
    for i in range(n):
        # 对于大于i的j
        for j in range(i+1, n):
            # 获取相应的两只股票的价格Series
            stock1 = dataframe[keys[i]]
            stock2 = dataframe[keys[j]]
            # 分析它们的协整关系
            result = sm.tsa.stattools.coint(stock1, stock2)
            # 取出并记录p值
            pvalue = result[1]
            pvalue_matrix[i, j] = pvalue
            # 如果p值小于0.05
            if pvalue < 0.05:
                # 记录股票对和相应的p值
                pairs.append((keys[i], keys[j], pvalue))
    # 返回结果
    return pvalue_matrix, pairs
```

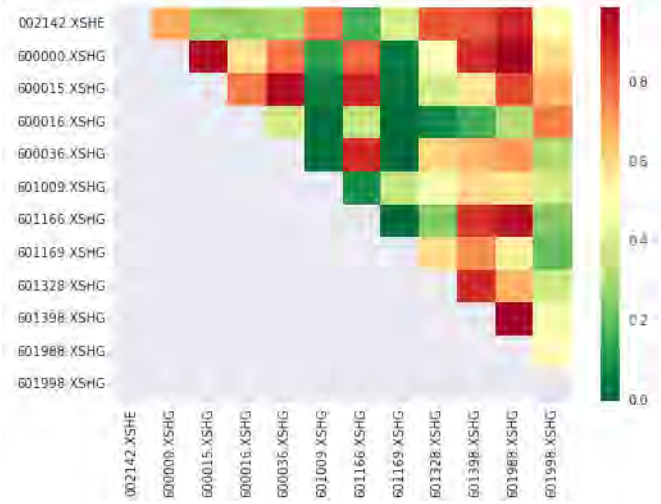
其次，我们挑选10只银行股，认为它们是业务较为相似，在基本面上具有较强联系的股票，使用上面构建的函数对它们进行协整关系的检验。在得到结果后，用热力图画出各个股票对之间的 p 值，较为直观地看出他们之间的关系。

我们的测试区间为2014年1月1日至2015年1月1日。热力图画出的是 1 减去 p 值，因此颜色越红的地方表示 p 值越低。

```
stock_list = ["002142.XSHE", "600000.XSHG", "600015.XSHG", "600016.XSHG", "600036.XSHG", "601009.XSHG",
```

```
            "601166.XSHG", "601169.XSHG", "601328.XSHG", "601398.XSHG", "601988.XSHG", "601998.XSHG"]
prices_df = get_price(stock_list, start_date="2014-01-01", end_date="2015-01-01", frequency="daily", fields=["close"])["close"]
pvalues, pairs = find_cointegrated_pairs(prices_df)
sns.heatmap(1-pvalues, xticklabels=stock_list, yticklabels=stock_list, cmap='RdYlGn_r', mask = (pvalues == 1))
print pairs
```

```
[('600000.XSHG', '600015.XSHG', 0.03384597872190559), ('600000.XSHG', '601988.XSHG', 0.012356888964434419), ('600015.XSHG', '600036.XSHG', 0.0
38816934807400762), ('601166.XSHG', '601988.XSHG', 0.04478308256413578 2), ('601398.XSHG', '601988.XSHG', 0.010624349301350767)]
```



可以看出，上述10只股票中有5对具有较为显著的协整性关系的股票对（红色表示协整关系显著）。我们选择使用其中 p 值最低（0.0106）的工商银行（601398.XSHG）和中国银行（601988.XSHG）这一对股票来进行研究。首先调取工商银行和中国银行的历史股价，画出两只股票的价格走势。

```
stock_df1 = prices_df["601398.XSHG"]
stock_df2 = prices_df["601988.XSHG"]
plot(stock_df1); plot(stock_df2)
plt.xlabel("Time"); plt.ylabel("Price")
plt.legend(["601988.XSHG", "601998.XSHG"],loc='best')
```

接下来，我们用这两支股票的价格来进行一次OLS线性回归，以此算出它们是以什么线性组合的系数构成平稳序列的。

```
x = stock_df1
y = stock_df2
X = sm.add_constant(x)
result = (sm.OLS(y,X)).fit()
print(result.summary())
```

OLS Regression Results

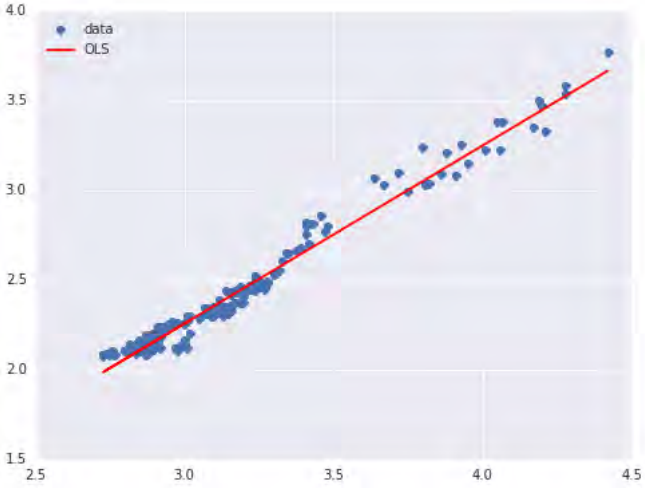
Dep. Variable:	601988.XSHG	R-squared:	0.972
Model:	OLS	Adj. R-squared:	0.972
Method:	Least Squares	F-statistic:	8437.
Date:	Thu, 14 Jul 2016	Prob (F-statistic):	1.09e-190
Time:	15:12:22	Log-Likelihood:	363.71
No. Observations:	245	AIC:	-723.4
Df Residuals:	243	BIC:	-716.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	-0.7248	0.034	-21.117	0.000	-0.792	-0.657
601398.XSHG	0.9938	0.011	91.851	0.000	0.972	1.015

Omnibus:	7.092	Durbin-Watson:	0.225
Prob(Omnibus):	0.029	Jarque-Bera (JB):	8.337
Skew:	0.256	Prob(JB):	0.0155
Kurtosis:	3.745	Cond. No.	34.0

系数是 0.9938，画出数据和拟合线。

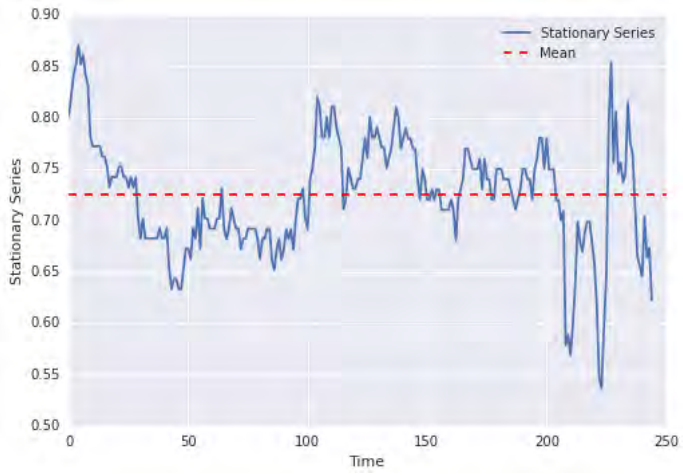
```
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="data")
ax.plot(x, result.fittedvalues, 'r', label="OLS")
ax.legend(loc='best')
```



设中国银行的股价为 Y，工商银行为 X，回归拟合的结果是
$Y = -0.7248 + 0.9938 \cdot X$
也就是说 $Y - 0.9938 \cdot X$ 是平稳序列。

依照这个比例，我们画出它们价差的平稳序列。可以看出，虽然价差上下波动，但都会回归中间的均值。

```
plot(0.9938*stock_df1-stock_df2);
plt.axhline((0.9938*stock_df1-stock_df2).mean(), color="red", linestyle="--")
plt.xlabel("Time"); plt.ylabel("Stationary Series")
plt.legend(["Stationary Series", "Mean"])
```



买卖时机的判断

这里，我们先介绍一下 z-score。z-score 是对时间序列偏离其均值程度的衡量，表示时间序列偏离了其均值多少倍的标准差。首先，我们定义一个函数来计算 z-score：
一个序列在时间 t 的 z-score，是它在时间 t 的值，减去序列的均值，再除以序列的标准差后得到的值。

```
def zscore(series):
    return (series - series.mean()) / np.std(series)
```

对于工商银行与中国银行的平稳线性组合，用上面的函数计算 z-score 并绘出图。

```
plot(zscore(0.9938*stock_df1-stock_df2))
plt.axhline(zscore(0.9938*stock_df1-stock_df2).mean(), color="black")
plt.axhline(1.0, color="red", linestyle="--")
plt.axhline(-1.0, color="green", linestyle="--")
plt.legend(["z-score", "mean", "+1", "-1"])
```

我们认为，当两这个序列的 z-score 突破 1 或者 -1 时，说明两支股票的价差脱离了统计概念中的合理区间，如果它们的协整关系能够保持，那么它们的价差应该收敛。所以，在发现上述序列突破 1 或 -1 时，应该按照比例买多一支股票并做空另外一支，从而赚取之后收敛的差价。

结合上图，当 z-score 突破上方红线时，说明工商银行的价格相对于中国银行高估，因此我们买入 1 份中国银行并卖空 0.9938 份工商银行（系数根据前面的线性回归得出），并当 z-score 回归于 0 时清仓获利。如果 z-score 突破下方绿线的话，反方向操作即可。

中国式 Pairs Trading 策略：搬砖

标准的配对交易策略是通过一买一卖的行为来对冲掉系统性风险，从而以较低的风险赚取到持续的利润。但目前A股市场是不允许进行直接卖空操作的。融券的渠道我等散户又搞不定。因此，我们对原始的配对交易进行修改，只进行做多操作，不进行做空操作。这种操作被形象的成为“搬砖”。其目标不是追求绝对收益，而是追求收益率比一直持有一个股票的高。

在选定一组协整关系为 $aX - bY$ 的股票后，我们有以下策略：
- 选定比例 p 和 q，初始仓位为 $p\%$ 的 X 和 $q\%$ 的 Y。选定测试 z-score 天数的参数 test_days。
- 每天执行：
    - 计算两支股票的线性组合序列 $aX - bY$ 在过去 test_days 的标准差和均值，以此计算 z-score。
    - 如果当天 z-score 高出 1，则将仓位调整为全仓 Y。如果当天 z-score 小于 -1，则将仓位调整为全仓 X。
    - 如果上一交易日处于全仓一支股票的状态，并且今日 z-score 回归 0 点，则调整回至 $p\%$ 和 $q\%$ 的比例。

在之前章节中我们通过分析发现，工商银行和中国银行在2014年全年有着非常强的协整关系，但我们不能在14年进行回测，因为这样等同于使用未来函数（比如文章最后的回测），因此我们在2015至2016的区间内进行回测。使用的参数是 $p\% = 50\%$，$q\% = 50\%$，$\text{test\_days} = 120$。

由于没有对冲机制，所以不能依靠绝对收益评估策略的效益，而要用它和配对的两支股票对比。如果分别跑赢了两支股票，那么说明该策略是有效的。

首先是和工商银行的对比，我们的策略完全跑赢了这支股票。

其次是和中国银行的对比，虽然前半年被工商银行拖了后腿，但后半年还是稳稳地跑赢了。

本策略全程满仓，持有股票只限于以上两支。通过判断两支股票的相对强度进行仓位调整，最后收益胜过了其中的任何一支，效果是非常显著的。

策略的应用

搬砖的策略能跑赢个股，但也只能跑赢个股，所以若想投入使用并赚取稳定收益，还要结合其他策略一起使用。比如，使用基本面或者技术面的思路来判断银行指数在未来一段时间中的走势。如果趋势乐观，则可以在指数成分股中选取两支具有协整性质的个股进行搬砖。如果趋势不乐观，则空仓对待。这样，通过搬砖，在其他策略之上产生更多的超额收益。

代码说明书与变量说明书

函数说明书

全局变量说明书

文章更迭记录：
v1.2，2016-07-28，更改难度标签
v1.1，2016-07-16，更新函数说明书和回测
v1.0，2016-07-15，文章上线

【量化课堂】多头趋势回踩策略

导语：雪球文章《多头趋势回撤点：一个好懂又好用的均线策略》介绍了一个择时选股的辅助判断方法。这篇文章将根据这个思路构建一个交易策略，并使用历史回测来检验它的效果。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，难度为进阶上，理解深度为level-0

多头趋势回撤点

多头趋势回撤的思路，是根据若干条均线呈现出的形态判断一支股票是否处于强势状态，并抓住回调的时机低位买入。顾名思义，这个策略的要点分为两部分：多头趋势和回撤点。

多头趋势

如果天数从短到长的移动均线呈从上到下排列的态势，我们判断股价处于多头趋势。

我们可以将较短的均线视为短线投资者的意向，较长的均线视为长线投资者的意向。那么当股价处于多头趋势时，表示短、中、长线投资者一致看多，即股票价格处于一个强势阶段。

相应的，同样是利用多根均线，如果天数从短到长的均线呈从下到上排列状态，那么股价则处于空头趋势，这时股价倾向于下跌。如果我们发现几根均线在一段时间里反复交叉，纠缠在一起，那说明股价正处于震荡的阶段。

举例来说。在下图的情况中，5、10、20、60、120 五根均线为从上至下依次排序，由此判定股价处于多头趋势。

下图里均线呈完全相反的排列顺序，是空头趋势。

再如下图中均线反复交叉的情况，则为震荡趋势。

我们要做的是追随趋势，买入呈现出多头态势股票。问题在于，这些股票都是处于上涨阶段，我们该如何选择合适的点位入场呢？

回撤点

假设股票处于多头趋势中。如果价格回撤到某一根均线，并且不破坏多头趋势的格局，我们将其称为一个回撤点。

股价在上涨过程中，短线投资者获利了结会造成股价向下调整，属于正常现象。如果在调整中依然破坏多头趋势的形态，那意味着股票依然处于强势阶段并且会继续上涨，那么这个回撤正是适合入场的时机。

如上图所示，股票的均线呈现出多头趋势。在红圈标明处，股价收盘于 10 日均线之下，并且没有打乱五根均线的多头排列，我们将其视为入场信号。

当然，任何方法都不是万能的，不是所有的回撤点都会保证收益。举例来说，

在上图中有两个 10 日回撤点，其中的第一个在买入后的确产生了收益，但第二个却持续下跌。为了避免这种情况，我们可以结合其他的方法和思路来强化分析判断的结论。另外，也应该设置判断错误时的止损方案，可以按照百分比止损，也可以根据均线的形态止损：比如当 5 日均线交叉 20 日均线，或者当价格跌破 20 日均线时，止损卖出。

交易策略

仅仅运用多头趋势回撤的思路，我们构建策略如下：

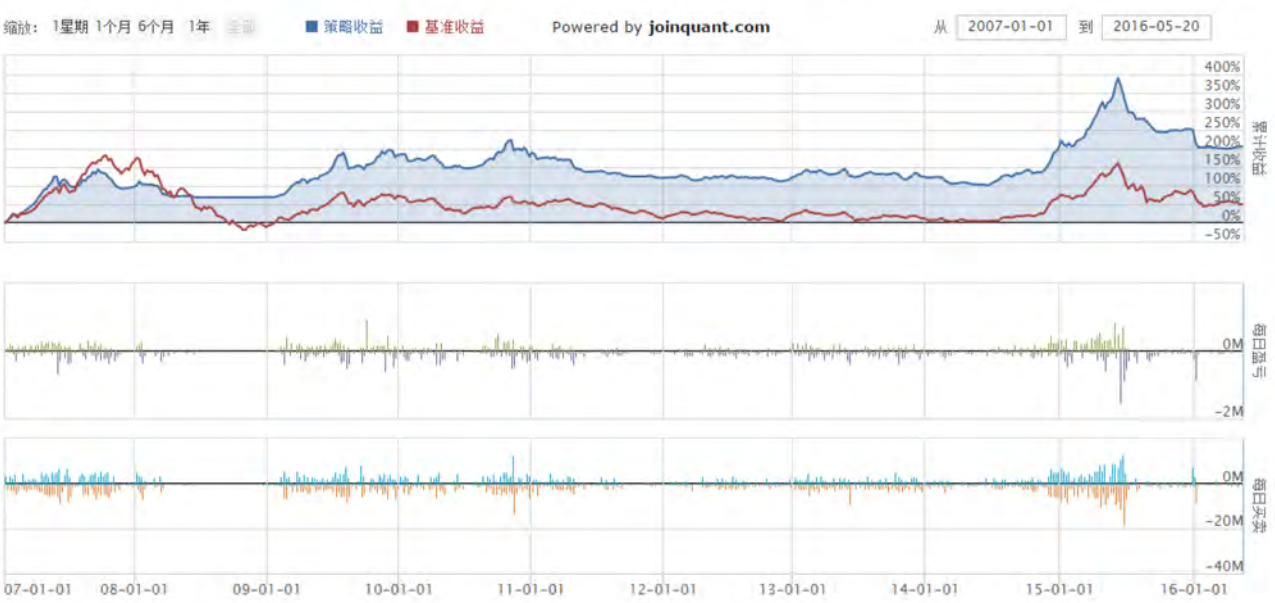
一、选定一股票池，并且选定一系列系数：
二、一组均线天数 $[N_1, N_2, N_3, \ldots, N_k]$：总数量 k 限制，按照从小到大 $N_1 \leq N_2 \leq \cdots \leq N_k$ 排列。当相应天数的移动均线是从大到小排列时，是多头排列的格局；
三、趋势天数 T： 当上面指定的移动均线在 T 天内都处于多头排列时，我们才判断价格处于多头趋势；
四、回撤均线 M：当前一天的最收盘价低于 M 日均线时判断为回撤；
五、持有股票上限 num_stocks：同时最多持仓 num_stocks 支股票。
六、止损比例 d 和止盈比例 u：当股票价格搞出买入价的 u 倍，或低于买入价的 d 倍时，卖出股票。

每日执行以下操作
产生信号：
一、选出股票池中所有在过去的 T 个交易日内，$[N_1, N_2, N_3, \ldots, N_k]$ 日均线组按照多头排列的股票，判定为多头趋势；
二、在处于多头趋势的股票中选出前一日收盘价低于 M 日均线的股票，判定为发生回撤点；
三、在所有出现多头趋势回撤点的股票中，去掉已持仓的股票，其余的发出买入信号。
调换仓位：
一、全仓卖出所有达到止盈或止损线的股票；
二、在有现金的情况下，买入所有发出信号的股票，每支股票的买入总值为总资产净值除以 num_stocks。

回测结果

以下的回测股票池为沪深300成分股。我们多头排列考虑 5、10、20、60、120 日移动均线，将 $T = 7$ 天以内都保持多头排列的股票筛选出来；前一日收盘价低于 $M = 10$ 日均线的情况判定为发生回撤。最大持有股票为 15 支。并且止损和止盈点为 $u = d = 5\%$，也就是说无论先盈利 5%，还是先亏损 5%，都会触发清仓（这样赚钱的来源就完全取决于信号发出后先触碰正 5% 还是负 5%了，未来可以跟凯利公式结合使用）。回测时间从 2007 到 2016，平均年化收益率 13%，较指数有稳定盈利。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
205.49%	13.02%	50.82%	0.088	0.385	0.458	0.554	0.195	0.197	0.311	40.250%(2015-06-15,2016-04-20)

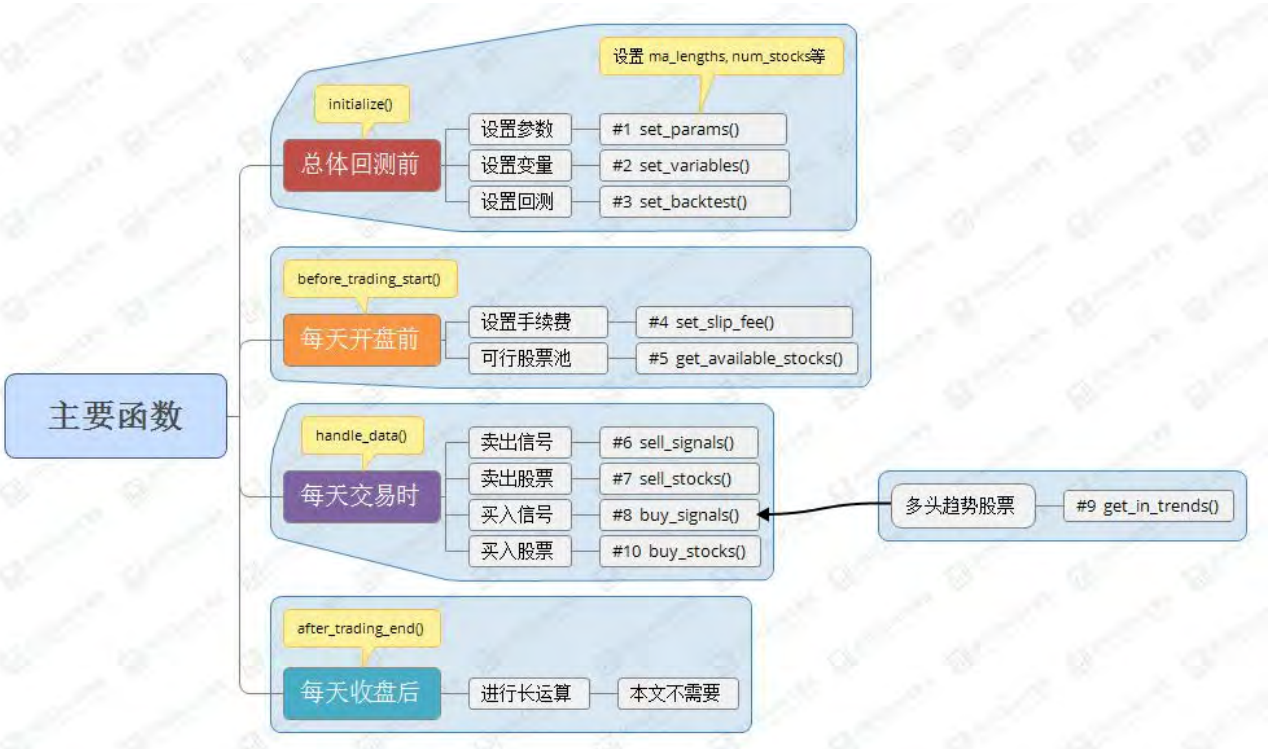


结语

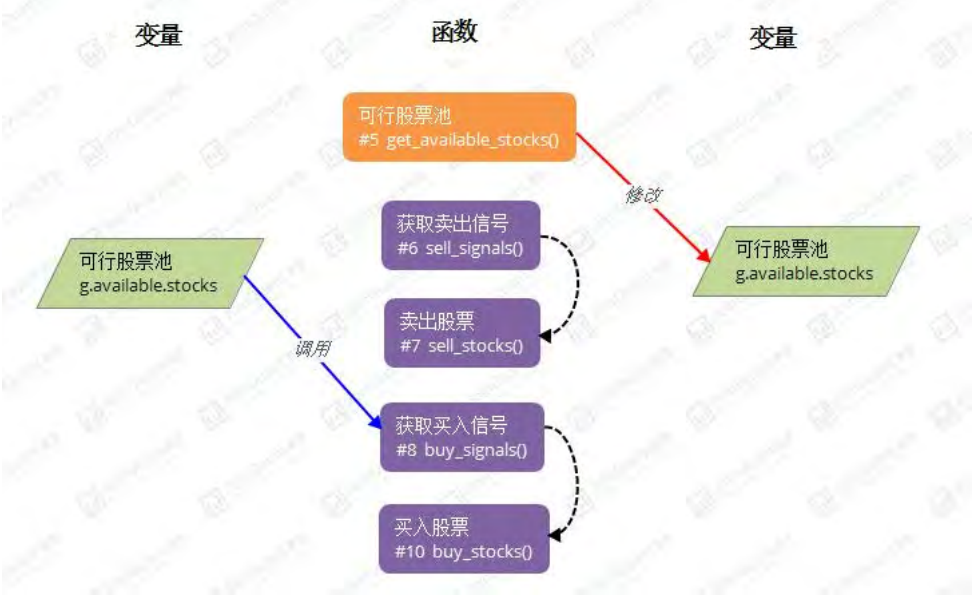
最后也重申一下雪球原文提到的一点，这里介绍的是一个辅助择时和选股的思路，并不能单独作为一个完整的交易体系。上面的策略和回测也是对这个思路的一个尝试和检验，验证了它的有效性。但若想投入实际应用，还必须结合其他的分析判断和交易策略。

函数和变量说明书

函数说明书（点击放大查看）



全局变量说明书（点击放大查看）



文章更迭记录：
v1.2，2016-11-01，修正策略逻辑，感谢 sunyanzi1986 指出
v1.1，2016-07-28，修改代码注释，感谢 James_3 指出
v1.0，2016-07-25，文章上线

【量化课堂】手把手教你如何应用机器学习

导语：

在我们的数学课堂中，我们给大家简单介绍了几种机器学习方法的算法原理（SVM，朴素贝叶斯，随机森林等等），在每篇文章的最后，我们都放了一个非常小的例子来帮助大家使用这些算法。这一篇就给大家展示一个更贴近实际的例子，来帮助大家更好的使用这些算法。

本文中，为了降低提取的特征中的噪声，我们将常见技术分析指标进行离散化，作为特征值，如RSI，MACD，等等。然后应用SVM，朴素贝叶斯和随机森林算法来预测下一个交易日的涨跌情况。

为了和实际应用场景更接近，我们每天都会向训练数据中添加今天的数据，因为训练数据集发生了改变，我们每天都会训练一个新的模型。下面我们正式开工。

*本文由*JoinQuant*量化课堂推出。难度标签为进阶上，理解深度标签：*level-0

作者：yongpeng.r
编辑：宏观经济算命师

阅读前需要了解：
文中涉及到SVM，随机森林，朴素贝叶斯算法，可以参照量化课堂中的 支持向量机入门，随机森林入门，朴素贝叶斯入门。

训练预测代码介绍:

本文是之前的支持向量机，随机森林，朴素贝叶斯等机器学习算法的应用篇，具体的代码可以参照文中后面的研究模块，下面先将一些重要的代码拿出来说说，以方便大家理解代码的逻辑。

特征计算
我们先通过get_price方法得到该支股票的前35个交易日的开高低收和交易量数据，然后利用talib包提供的特征计算API计算相应的特征。在这次实验中，我们选取了SMA，WMA，MOM，STCK，STCD，MACD，RSI，WILLR，CCI，MFI，OBV，ROC，CMO等技术指标作为训练模型的特征。计算SMA指标的代码如下：

```
trading_days = get_all_trade_days()
start_day = trading_days[index - 35]
end_day = trading_days[index]
stock_data = get_price(test_stock, start_date=start_day, end_date=end_day, \
                       frequency='daily', fields=['close','high','low','volume'])
close_prices = stock_data['close'].values
sma_data = talib.SMA(close_prices)[-1]
```

分类标签确定
如果下一个交易日的收盘价比当前交易日的收盘价高，label的值为True，否则为False。相关代码如下：

```
start_day = trading_days[index]
end_day = trading_days[index + 1]
stock_data = get_price(test_stock, start_date=start_day, end_date=end_day, \
```

```
                    frequency='daily', fields=['close','high','low','volume'])
close_prices = stock_data['close'].values
label = False
if close_prices[-1] > close_prices[-2]:
label = True
```

特征离散化
通过talib库计算出来的特征都是连续值，由于股市中的噪声非常多，如果直接将连续值特征放到机器学习算法中训练，得到的结果很可能过拟合的（回测结果好，实盘结果差）。为了避免过拟合，本文中结合各个特征的含义，将连续值特征离散化为二值特征。比如对于SMA特征，离散化的方法就是和当日收盘价相比较，若SMA小于当前收盘价，离散之后的特征值是1，若SMA大于等于收盘价，离散之后的特征值是-1。SMA离散化的代码如下，x_all是存放所有特征值的二维数组，x_all中的第1列存放的就是SMA特征数据。

```
# SMA
if x_all[index][0] < x_all[index][-1]:
x_all[index][0] = 1
else:
x_all[index][0] = -1
```

应用机器学习算法
在准备好数据之后，就需要利用机器学习算法来训练模型并且利用训练好的模型预测下一个交易日的涨跌情况了。下面是通过SVM算法预测的代码：

```
clf = svm.SVC()
clf.fit(x_train, y_train)
prediction = clf.predict(x_test)
```


版本更迭信息：V 1.0 文章上线

【量化课堂】均值回归进阶策略

导语：我们经常在抓反弹时感觉像在抓一只刺猬，不知道该怎么下手。左侧交易会不会买在半山腰？右侧交易会不会进场太晚？什么时候买入真是让人头疼。切莫苦恼，你完全可以通过历史数据统计判断反弹的时机。本文就将抛砖引玉地介绍一种简单的统计方法，它不能保证抓反弹次次成功，但可以让你对多错少，累积起来就是真实的收益！

```
作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，难度为进阶（上），深度为 level-0。
```

本文是量化课堂的《均值回归入门》的进阶版，并且用到《指标效果的统计分析》一文中的统计方法。

动机

我们先回顾一下均值回归的基本概要：

股票的价格会以它的均线为中心进行波动。也就是说，当标的价格由于波动而偏离移动均线时，它将调整并重新归于均线。那么如果我们如果能捕捉偏离股价的回归，就可以从此获利。

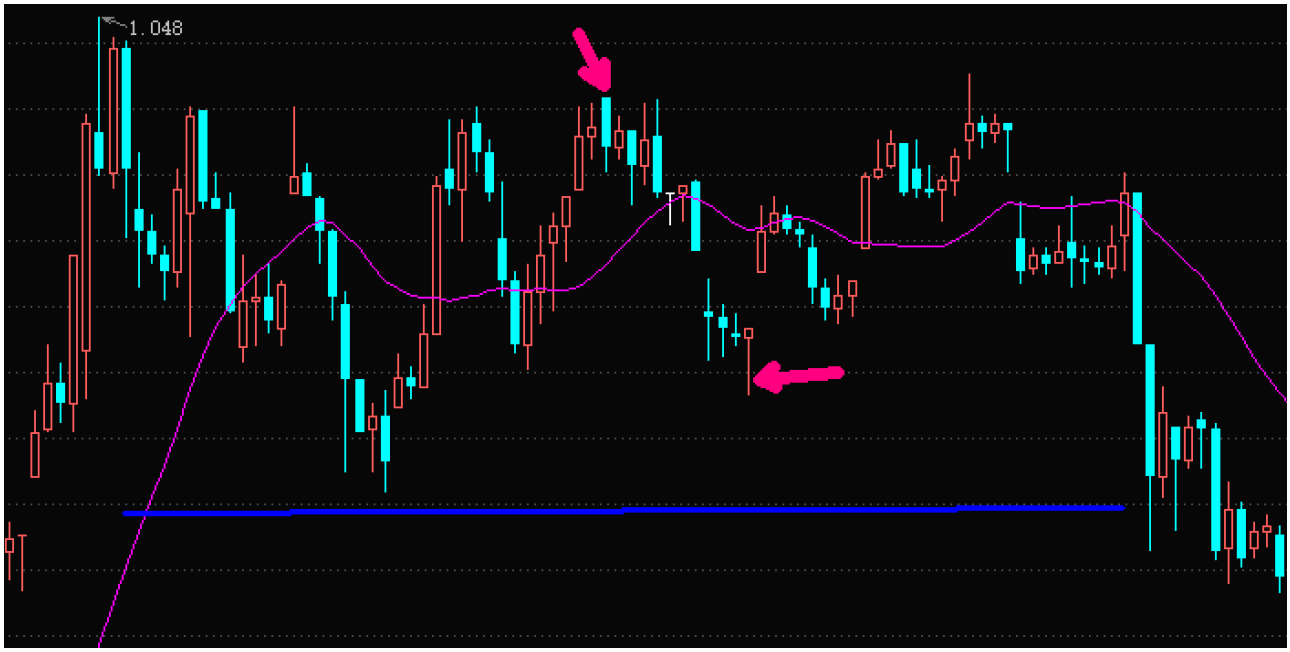
在入门篇文章里我们尝试了一个非常简单的均值回归策略，也发现了相应的问题：那就是无法准确判断何时买入，也缺乏买在半山腰之后的处置机制。本篇要讲的是，统计历史数据中价格偏离所对应的反弹幅度，选择胜率最高的偏离度作为入场信号。

波动大小的相对性

在入门篇文章中我们粗略地以价格和均线的偏差作为买入的评估标准，但它有一个弊端，那就是忽略了股价波动的相对性。假如股票 A 在一段时间里每天振幅 3%，股票 B 在同样一段时间里每日振幅 0.5%，今天我们发现 A 和 B 的股价都和均线偏离了 2%，我们认为那支的偏离更显著呢？是 B 对吧。

举实例来说。下图是某支股票的 60 分钟线，中间紫色曲线为股价的 5 日均线。可以看出在蓝色线段标明的时间里，股价波动平缓。在两个粉色箭头处，股价比均线分别高 4.1% 和低 5.9%。这些波动较前段时间更为剧烈，可以告诉我们股价发生了偏离。

再看下图，这是同一支股票在另一段时间的 60 分钟线。这段时间里价格波动幅度大，在粉色箭头的位置，价格和均线的差距分别为5.3% 和 5.1%，但我们不将其视为显著的价格偏离。



那么我们该如何测量近期波动幅度的大小呢？这就要涉及到统计学中的标准差概念。

标准差

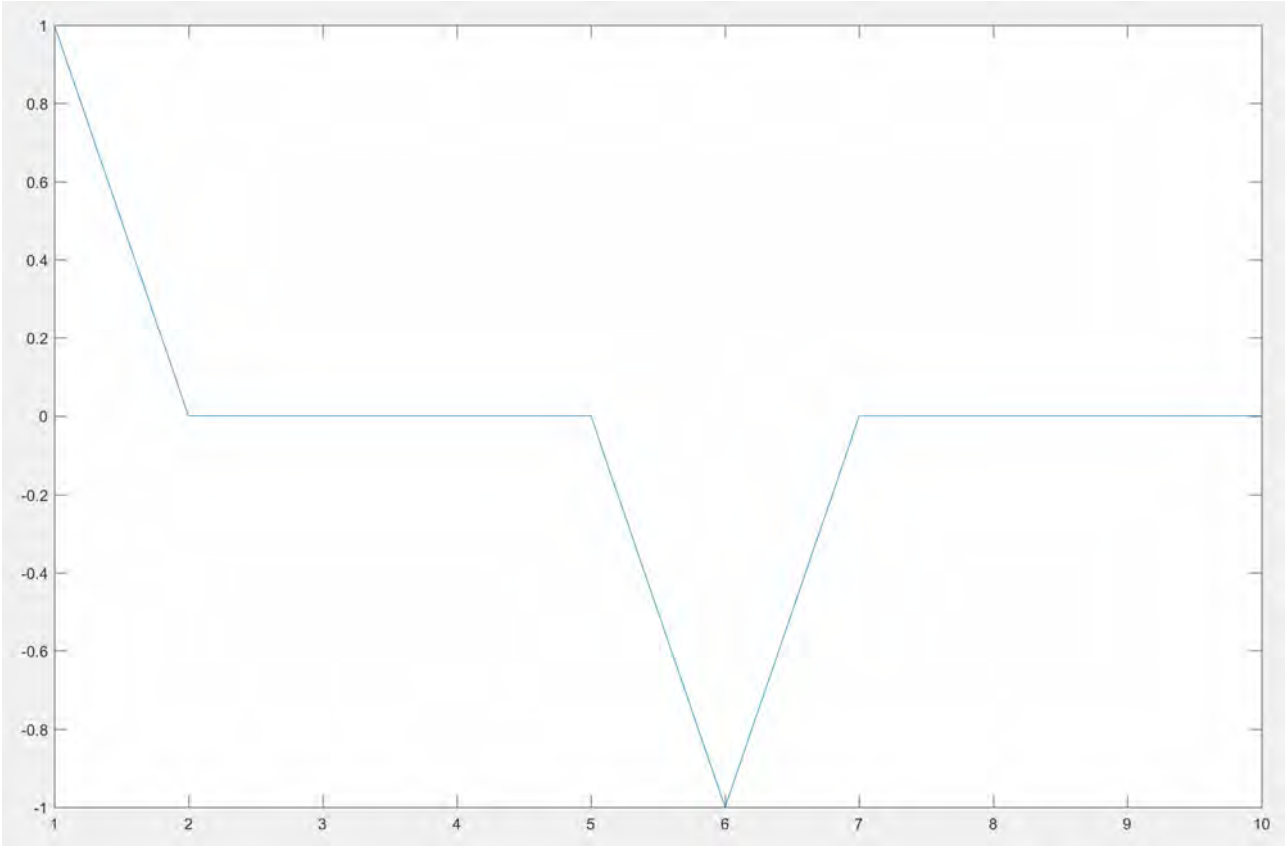
标准差（standard deviation），通常用小写希腊字母 σ（sigma，读“西格玛”）表示。通俗地讲，一组数据的标准差就是这组数据离均值的普遍差距。

假设 $x_1, x_2, \ldots, x_n$ 是我们观测到的一组数据，这组数据的均值是 $\mu = (x_1 + x_2 + \cdots + x_n)/n$
标准差的计算公式为

$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_n - \mu)^2)}{n}}$$

这里 $x_t - \mu$ 计算的是第 t 项数据和均值的差值。将这 n 项差值的平方取了平均，再算出平方根，我们得到的是这组数与均值偏差的一个标准。如果这组数据的波动较大，那么 σ 相应也会较大；相反的，如果这组数据波动小，那么 σ 会更接近零。

举例来说，假设我们有一组数据 $A = (1, 0, 0, 0, 0 - 1, 0, 0, 0, 0)$ 如下

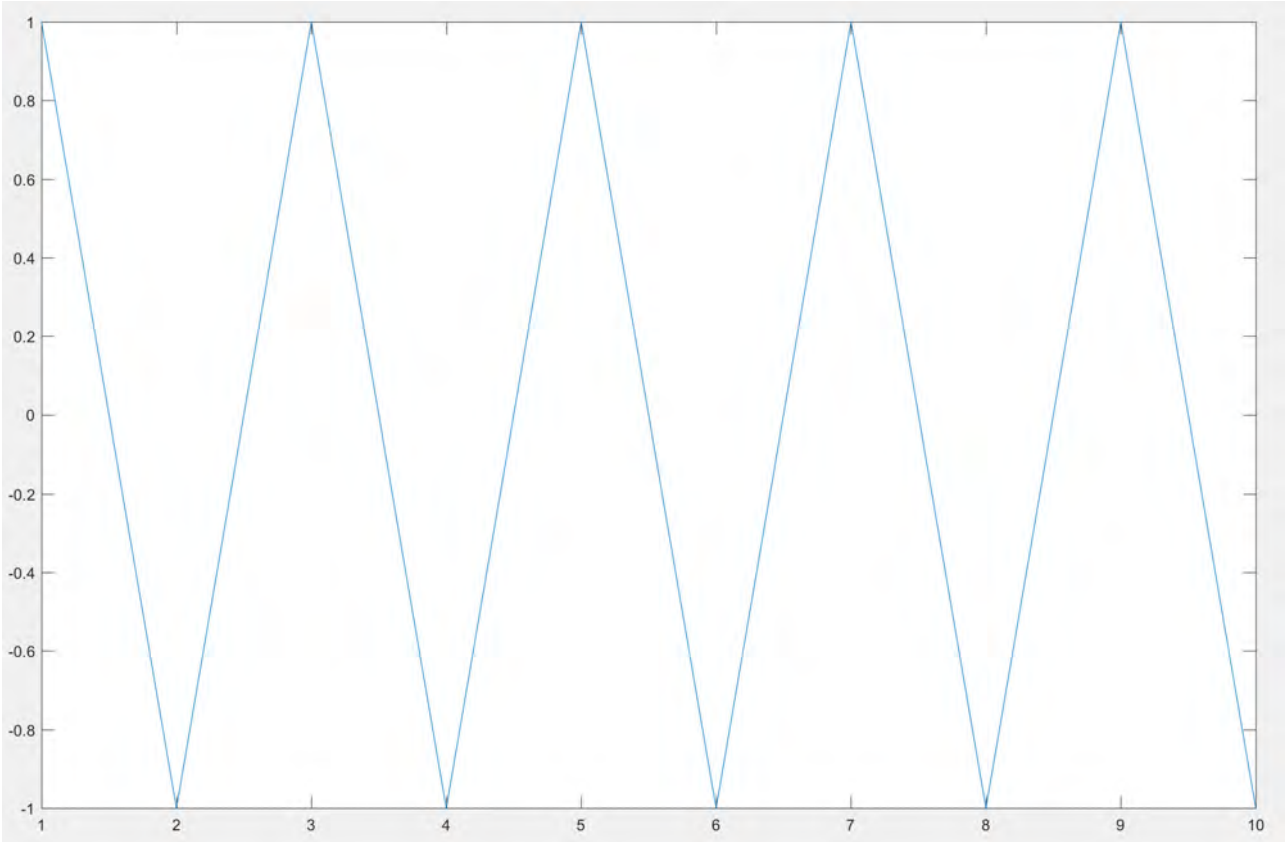


它的平均值是 0，标准差是

$$\sigma = \sqrt{\frac{(1-0)^2 + (-1-0)^2)}{10}} = \sqrt{1/5} \approx 0.45$$

这时如果下一项出现的数据是 $x_{11} = 1$，可以算出它和均值的差是 $(1-0)/\sigma \approx 2.23$ 倍标准差，可以被视为是一个较大的波动。

在另一种情况里，假设我们的数据是 $B = (1, -1, 1, -1, 1, -1, 1, -1, 1, -1)$ 如下图



可以直观地看出波动较大。这组数据同样是均值等于 0，它的标准差是

$$\sigma=\sqrt{\frac{(1-0)^2+(-1-0)^2+\cdots+(-1-0)^2)}{10}}=\sqrt{1}=1$$

明显比数据 A 的标准差更大。这时如果出现新的数据 $x_{11}=1$，由于它和均值的差距等同于标准，我们不认为它是一个很大的波动。

有了这个概念，我们可以构造一种利用标准差来测量当日股价波动幅度的方法。取过去 N 天收盘价，并取其标准差 σ，设今日股价为 P 并且 N 日均线值为 MA_N，计算

$$\rho=\frac{P-\mathrm{MA}_N}{\sigma}$$

这个值得含义是，相较过去 N 天的价格，今日股价与均值的偏差为 ρ 倍的标准差。如果绝对值 $|\rho|$ 比较大，我们判定今日价格波动比历史波动更为突出；反之，如果 $|\rho|$ 比较小，我们判定今日的波动很普通。嗯，这个 ρ 嘛，为了方便起见，作者就自行起一个名字吧，就叫它“N 日偏离倍数好了”。

好，可以合理地判定波动大小了，那么应该如何利用这些信息判断何时入场呢？

偏离倍数和赢输的统计

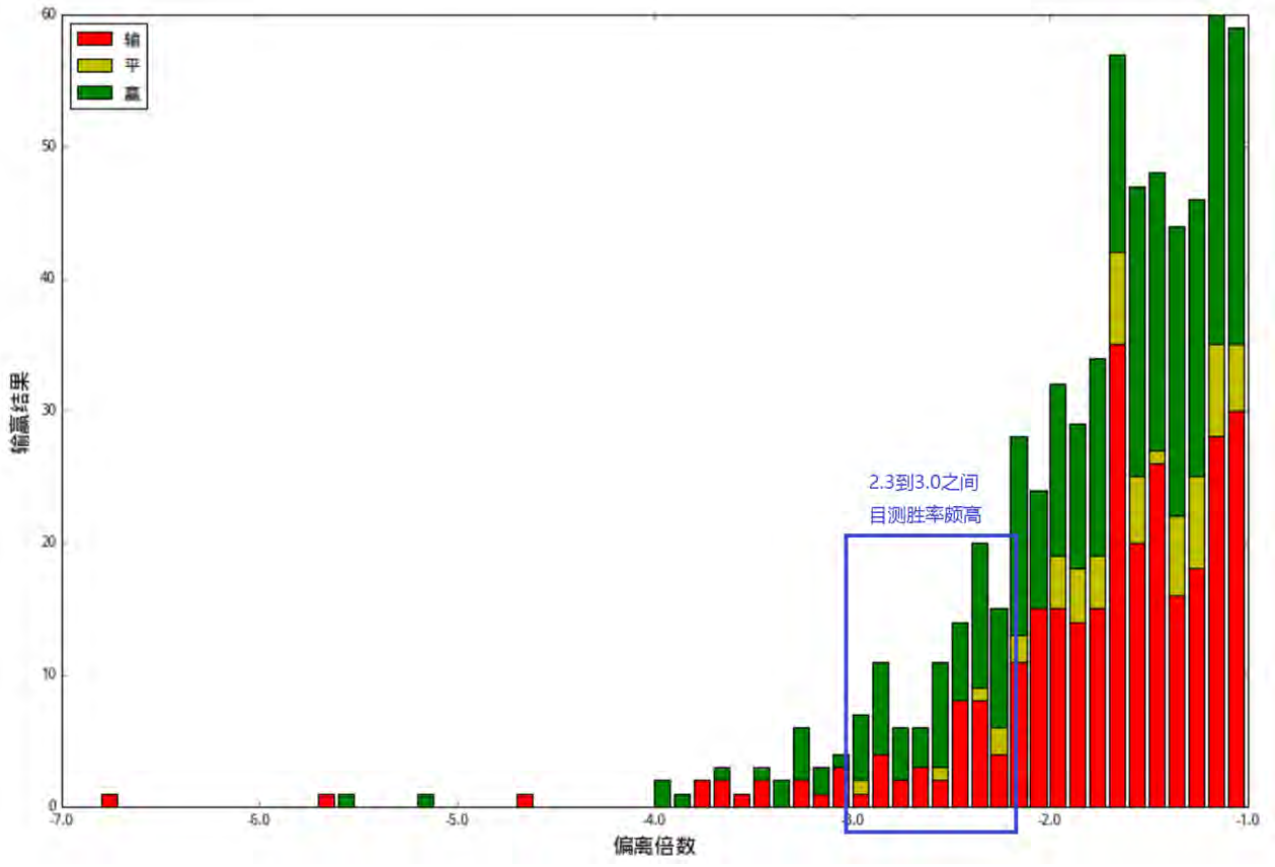
我们没法凭空预测偏离倍数在多大的情况下价格会反弹，也不能盲目地认为偏离倍数越高越好。我们干脆穷举，对历史上所有的偏离倍数（范围），统计其后后股价的走势，然后将胜率最高的偏离倍数视为入场信号，在其出现时买入。

接下来的统计方法请参阅量化课堂文章《指标效果的统计分析》。

按照文中的思路，我们要计量的指标是上一节中介绍的 N 日偏离倍数 ρ。另外，我们只记录 Misplaced & 的情况，因为如果 $\rho\geq-1$，那说明价格偏离不大，则不构成回归信号。

我们将要统计的结果进行如下判定：设定观测输赢的天数 T，止盈的 σ 倍数 u，止损的 σ 倍数 d。如果在未来的 T 天里，股价先上涨 $\sigma\cdot u$，则记作“赢”；先下跌 $\sigma\cdot d$，则记作“输”；如果 T 天股价都没离开这个区间，则记作“平”。

以某股票从 2006 年到 2016 年的数据为例，统计结果如下：



通过直接看图可估测出 ρ 在 2.3 到 3.0 的区间里胜率很高。我们再通过文章中取最佳指标区间的算法获取一个胜率最高的偏离倍数区间。这里，我们想选定一个区间宽度 w，并在以上统计中计算一个宽度为 w 的横轴区间（w 个连续竖条），需要它的胜率（绿色区域总和除以红色区域总和）是最高的。当然，还需要设置一个最小数据比 θ，区间内的数据量必须高于总数据量的 θ 倍，不然区间内的统计缺乏可信性。

我们设置参数：区间宽度 $w=5$，最小数据比 $\theta=0.05$。通过计算，得到区间 $[-2.6, 3.1]$，区间中的数据如下：

value	win	even	lose
-3.0	5	1	1
-2.9	7	0	4
-2.8	4	0	2
-2.7	3	0	3
-2.6	8	1	2

其中赢 $(5, 7, 4, 3, 8)$ 次，输 $(1, 4, 2, 3, 2)$。计算出输赢率为

$$\frac{5+7+4+3+8}{1+4+2+3+2}=\frac{27}{12}$$

也就是说，根据历史统计，如果当偏离倍数在 -3.1 和 -2.6 之间时买入该股票，输赢的预期是 27 赢比 12 输，当然，还有 2 次平局。

策略和回测

其实策略的主要部分已经完成，只要加上对资金和仓位的管理，就可以构成一个可运行的策略。这里我们就构建一个很简单的方案。

设定如上一节所述每的参数：均线天数 N，统计输赢天数 T，止盈倍数 u，止损倍数 d，区间宽度 w，和最小数据比 θ。每交易日执行以下操作：
一、全仓卖出应当止盈或止损的股票，或持有超过 T 天的股票；
二、执行上一节的统计方法来更新每一支股票的最佳偏离倍数区间；
三、如果空仓，选定任意一支偏离倍数在最佳区间内的股票并全仓买入，如果所有股票都不符合则空仓。买入时记录止盈线 u 倍 σ 和止损线 d 倍 σ，并设置如果 T 日内未触碰止盈或止损线，也全部卖出。

来看一看回测结果。以下回测使用的股票池只包含 1 或 2 支标的股，这样便于看出策略对于个股发出的信号。文章最后的策略代码只适用于小规模的股票池；对于更大的股票池，信号会更加密集，因此需要调整每支股票的持仓量并且分别止盈止损，读者可以自行进行修改，或等待量化课堂未来的文章。另外，我们的回测从 2011 年开始，这样保证回测开始时就有一定的历史数据储备。

首先是股票 002013 的回测，使用参数 $N=30$，$T=30$，$u=d=1$，$w=4$，$\theta=0.05$，回测结果如下。

策略在震荡市中产生比较稳定的收益，并保持比较小的回撤。在牛市中完全不产生信号，因此没有半点反应，倒是在股灾之后稳稳地抓好了反弹，在短期内获取大量收益。

还使用同样的参数，我们看 002230 的回测。

同样是在震荡市中表现良好，但在股灾中抓反弹一度失利，不过后期在弥补损失之余获得了更高的收益。

观看回测最下方的“每日买卖”图可看出这个策略产生的信号和交易实则不多，在股票池里放入多支股票的话可以轮回抓取反弹，产生叠加的收益。比如，下图是将上面两支股票的作为标的回测，对比的指数是沪深300。

结语

嗯，我们看看啊…总结就是三点。首先是均值回归核心思路：

价格偏离均线太多就会弹回来

然后是，我们怎么判断偏离多少呢？

价格减去均线除以历史波动的标准差可以丈量偏离的程度

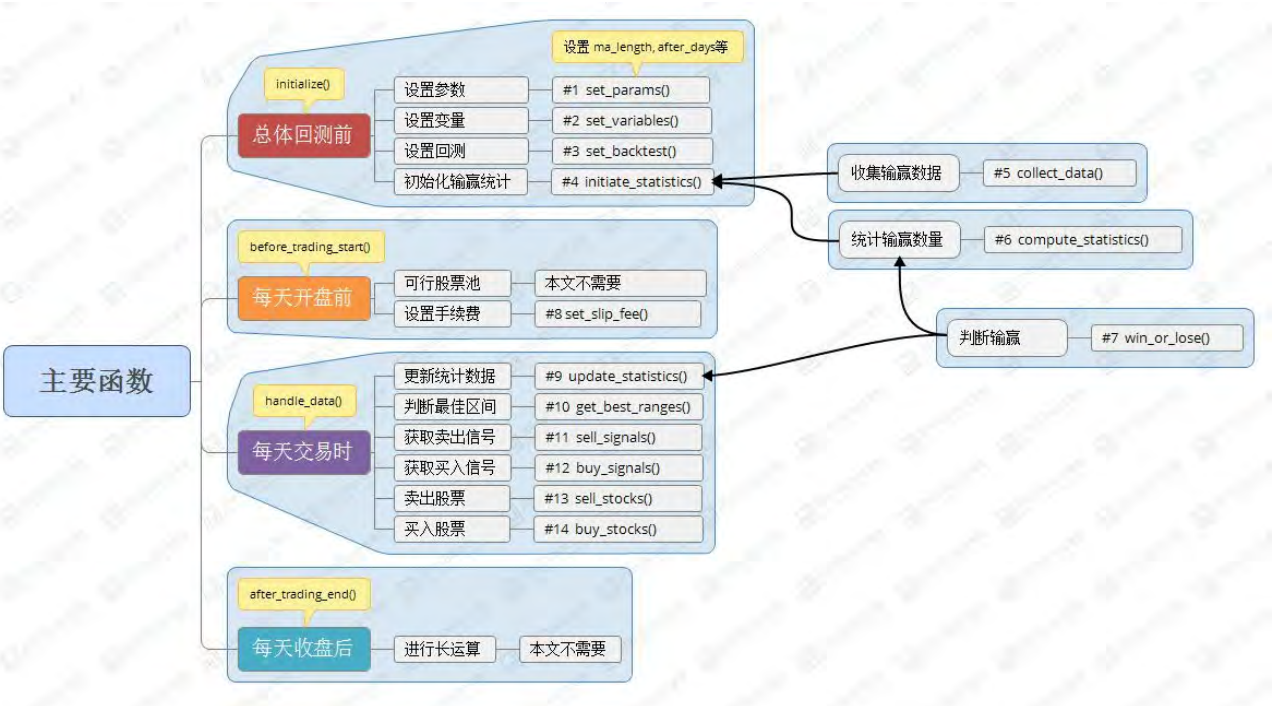
判断入场的话…

利用历史统计的偏离倍数对应的输赢率判断入场时机

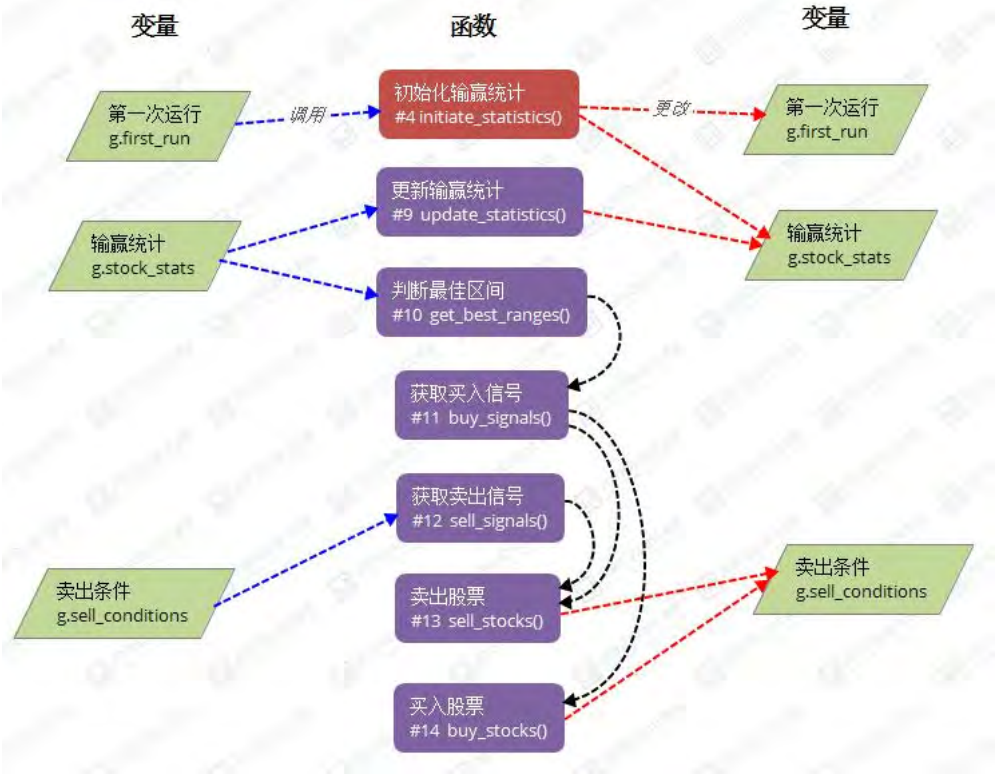
最后还要指出，本篇文章旨为抛砖引玉，展示一种思路。文中的统计方法比较粗糙，策略也比较简单。想要在实战中应用的话，还需要进一步拓展、打磨并结合其他的方法，才能构造出一套成熟的交易策略。量化课堂未来的文章会在此策略上升级改进，结合凯利公式进行动态的仓位调整，敬请期待。

函数和变量说明书

函数说明书



全局变量说明书



v1.1，2016-08-11，添加统计分析文章链接
v1.0，2016-07-25，文章上线

【量化课堂】CAPM模型应用策略

导语：CAPM模型公式解释了金融资产的系统性风险β，但事实上股票中还有这个公式不能解释的收益，叫作α。本文将展示如何通过α选股构建投资组合，并且获得超额收益。

作者：刘凡
编辑：肖睿

本文中的数学符号：σ_s为股票标准差，β为股票和市场组合的相关系数，σ_M为市场标准差，r_f为无风险利率，r_s为股票收益率，r_M为市场组合收益率

超额收益选股

CAPM模型(capital asset pricing model)相信大家应该都很熟悉。其中心思想是利用指数模型简化了马克维茨边界条件，推导出系统风险可以用股票与市场的相关系数和市场风险表示：$\sigma_s = \beta \cdot \sigma_M$，并由此计算在正确定价的情况下股票的收益率与市场组合收益率的关系为：

$$E[r_s] = r_f + \beta(E[r_M] - r_f)$$

那么我们知道，当一只股票的价格被正确定价的时候，它的风险溢价与市场组合的风险溢价是呈现完全的线性关系，将这样的股票收益率和市场组合的收益率做线性回归，得到的应该是一条斜率为β，截距为r_f的拟合直线。
现实中，每只股票的收益率与市场组合的收益率进行回归的时候截距都不会严格的等于r_f，此时我们定义

$$\alpha = (E[r_s] - r_f) - \beta(E[r_M] - r_f)$$

α被称作超额收益，意为股票除了承担系统风险所获得的收益之外的额外收益。按照CAPM模型，当正确定价时，股票的α值应该为零，所有α值不为零的股票我们都可以认为被错误的定价了。被错误定价的股票可以分为两类

$$\begin{cases} \alpha < 0，\text{此时股票的价格被低估了} \\ \alpha > 0，\text{此时股票的价格被高估了} \end{cases}$$

假设CAPM理论是成立，这些错误定价的股票都会最终回到正确的定价上去，因此，我们在选股时应该买入α更低的股票，这就是我们策略构造的基础。

策略和回测

我们构建策略的是，如下：

- 设定某一股票池，并选定一指数作为市场组合（一般情况下股票池应该是该市场组合的成分股）；
- 选定参数：调仓周期tc、计算α所使用的收盘价天数N、持仓的股票数量num；
- 每隔tc个交易日执行：

- 使用过去N天收益率数据进行线性回归，计算股票池中所有股票对应市场组合的α；
- 选择α最小的num支股票，调仓持有这些股票，持有比例为α越小权重越大。

下面的回测使用股票池为沪深300成分股，对应的市场组合是沪深300指数；参数有tc=15、N=60、num=20。调用2012年12月至今的数据进行回测的结果如下：

我们可以看到，策略收益始终在市场组合之上，是因为我们购买的20支股票基本上已经将风险足够分散化，并且持有15天之后，这些被暂时错误定价的股票都回到了正确的定价，所以策略的收益与指数的收益是同步的，但是始终比指数的收益要高。

函数和变量说明书

函数说明书

全局变量说明书

文章更迭记录：
v1.0，2016-07-27，文章上线

【量化课堂】斗牛蛋卷二八轮动原版策略实现

导语：这篇文章以二八轮动为例，尝试介绍如何从完全入门开始构建一个策略，适合小白入门级阅读。

本文由JoinQuant量化课堂推出 。难度标签为进阶（上），理解深度为 level-0

作者：邓佳佶
编辑：肖睿

前言

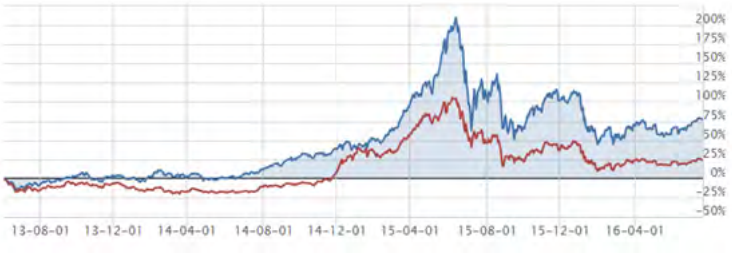
量化投资并不是一个神秘不可知的领域，比如说，一个最简单的策略——购买昨天涨停的股票持有到明日收盘时卖出，就可以通过量化的方式实现：第一步，筛选出昨日涨停的股票；第二步，计算每只股票买入比例；第三步，开盘买入。这个简单策略背后的投资逻辑是：涨停的股票还会有继续增长的趋势；同样，我们也可以购买昨天跌停的股票持有到明日收盘时卖出，它背后的投资逻辑是：跌停的股票在之后的交易中会回归均值。

虽然这是两种极不靠谱的交易策略，但其中蕴含了量化投资中如何发现趋势的基本思想。第一种交易是典型的动量策略，即根据涨跌趋势都会继续保持的特点，继续买入持续上涨股票，卖出下跌股票——用通俗的话来讲，就是“追涨杀跌”。第二种交易是典型的反转策略，即认为股票价值是被低估的，在未来的时间内会出现均值回归，就是所谓的“低吸高抛”。

动量策略和反转策略看上去是冲突的，但实际上并不矛盾，因为在实际的交易中，我们是根据市场的不同信息（通常我们称之为信号）解读市场究竟处于持续上升或下跌还是反转。信号是决定采用何种策略的关键。

“二八轮动”就是根据A股市场中大盘股和小盘股走势不同作为信号判断的。（所谓二，就是指数量占20%的大盘股、权重股；所谓的八,就是数量占80%左右的中小盘股，非权重股；其轮动就是指在两者之间不断切换，轮流持有。）
大盘股和小盘股的区分就是根据公司的流通股本的多少，大盘股通常指流通股本大于1亿的上市公司股票，而小盘股则与大盘股相对，通常指流通股本不足3000万的股票。沪深300通常可以近似表示大盘股的整体走势，中证500指数近似表示小盘股的整体走势。

在A股市场中，大盘股与小盘股往往存在着分化，走势并不同步。如下图所示，是13年8月到16年7月沪深300（红线）和中证500（蓝线）的走势：



简单来看，中证500整体涨幅和跌幅都高于沪深300，并且两者走势在部分时段并不同步。因此基本的策略可以设定为：当中证500上涨高于沪深300追涨中证500，在中证500下跌高于沪深300时切换到沪深300。

最简单的二八轮动策略

根据上述信号，我们可以制定这样的策略：每周五(或者本周的最后一个交易日)临近收盘时，将沪深300指数和中证500指数切换到周线状态，分别查看两者过去四周的累计涨幅。哪个过去四周涨幅大，那么就在收盘前买入对应的ETF持有一周，直至下一次的切换，其策略可以写为以下两步：

1、在每周五，对比当前交易日收盘数据与第前二十个交易日的收盘价，选择沪深300指数和中证500指数中涨幅较大的一个；

2、如果没有买入任何股指，则买入；如果涨幅较大的指数已买入，则保持仓位不变；否则执行调仓交易。

蛋卷二八轮动

这是雪球上「蛋卷基金」的实盘交易策略：斗牛蛋卷二八轮动，相关推荐阅读：蛋卷二八轮动，好策略还看未来好执行

上面的二八轮动策略无法规避大熊市中资产的大幅贬值，因为所交易的资产全部都在股市当中，尽管在熊市中从中证500切换到沪深300下跌幅度减小了不少，但是仍然是下跌的。同时设定在每周五交易频率过低，无法及时捕捉市场行情，因此可以改为每日判断。据此，我们可以对二八轮动策略进行一个改进：添加国债指数，当沪深300指数和中证500指数与前20日比都下跌时，切换为国债指数。

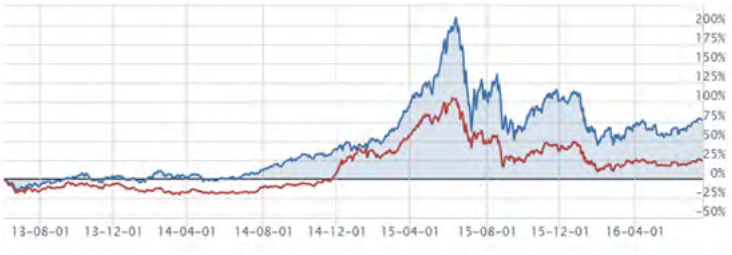
将上述的交易分步描述成可执行操作，就是蛋卷斗牛二八轮动的策略：

1、对比当前交易日收盘数据与二十个交易日前的收盘数据，选择沪深300指数和中证500指数中涨幅较大的一个；

2、于收盘前执行调仓交易；

3、如果两个指数与前20日比均为下跌，则切换为持有国债指数（也可理解为空仓）

根据上述策略得到的回测结果（2013.7-2016.7）如下：



如上图所示，是以中证500为基准收益（红线）对比的二八轮动策略（蓝线）的收益。总体上看，二八轮动在股票大涨大跌是的行情表现往往较好：

1、在2014-2015年的大牛市中，二八轮动策略（即蓝线）捕捉到了趋势，在牛市中基本是满仓持有中证500，和指数收益基本持平。但因为有20日延迟，因此在大牛市中稍弱于指数。

2、二八轮动完美避开了几次大跌（红色箭头的部分），特别是股灾级别的快速下跌段，下跌时间越久，二八轮动相比于其他指数的优势就更加明显。从回测结果看，该策略主要收益也就是来自于在大涨时跟大盘获得收益，在大跌开始时就退出止损。在一个包含大牛市和大熊市的完整牛熊大周期里，可以大幅跑赢所有指数。

3、但同时，正是因为二八轮动策略靠大涨大跌赚钱的特点，在震荡行情中往往表现不佳，频繁换手过程中往往被两边打脸。

（如上图所示，如果以沪深300作为基准收益，则二八轮动的超额收益更高，其主要原因是，小盘股在牛市中表现优于大盘股，而二八轮动在这种时候就会选择小盘股跟风获得收益，而在大跌时止损保证了之后收益率不会随大盘大跌）

蛋卷二八轮动的优化

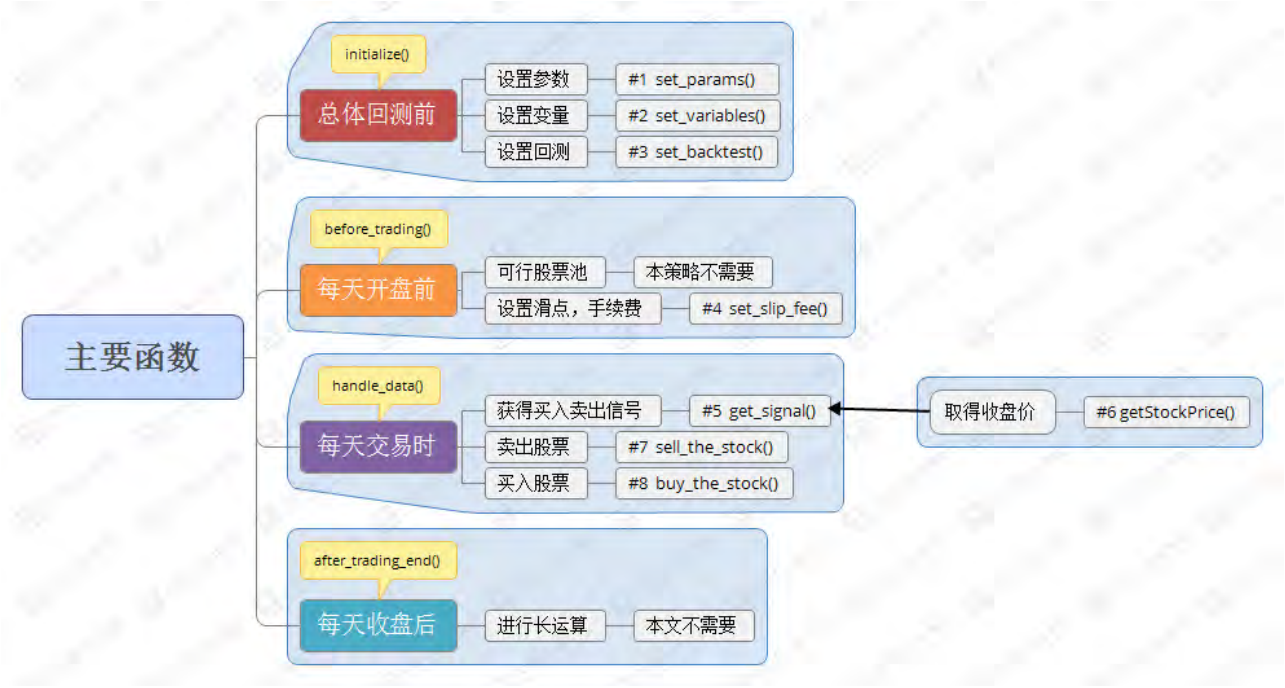
1、二八轮动无法捕捉短期的上涨信号（脉冲式波动），因为取20日前的股票价格往往会无法识别短期的上涨或者下跌。因此可以在二八轮动的基础上添加对短期上涨的识别，比如在持有国债的情况下，如果有一个指数当天上涨3%以上，此时把国债改为买入该种指数。

2、在估值ETF的选择中，我们只选择了中证500和沪深300。实际上，比中小盘更受整体股市牛熊影响的还有创业板。如果创业板指数加入，在沪深300指数、中证500指数和创业板指数三者之前选择最强，收益率可提升20%以上。（但需要判断这样的策略是否存在过度拟合：之所以收益提升，是因为前几年创业板的行情好，但是在接下来的几年内不一定。）

3、二八轮动本质上是一个择时策略，判断整体股市持续上涨和持续下跌的情况。如果在大牛市时直接买入小市值股票而不是中证500，那么回测的收益率更高。

函数和变量说明书

函数说明书



文章更迭记录：
v1.1，2016-07-28，修改排版
v1.0，2016-07-27，文章上线

【量化课堂】CAPM+APT多因子模型

导语: 本文结合CAPM和APT模型的思路，通过CAPM模型计算股票收益的α，再通过《多因子策略-APT模型》一文中的方法分析多因子对α的影响，并以此为标准进行选股。

作者：刘凡
编辑：肖睿

阅读本文前请掌握：

基本金融概念。
线性回归模型，理解深度level-0

APT模型，理解深度level-0
CAPM模型，理解深度level-0

CAPM模型概述

大部分投资者都是风险厌恶型的，也就是说，对于高风险的股票，投资者要求的期望收益率会更加高。分散投资组合可以分散风险。但有一些风险是不能被分散的，比如市场风险、国家风险等。

美国学者夏普、林特尔、特里诺和莫辛等人于1964年在资产组合理论的基础上发展出来了一个资本资产定价模型（简称CAPM模型）。这个模型根据股票本身的风险和市场风险的相关性估算出单个股票的期望收益率。

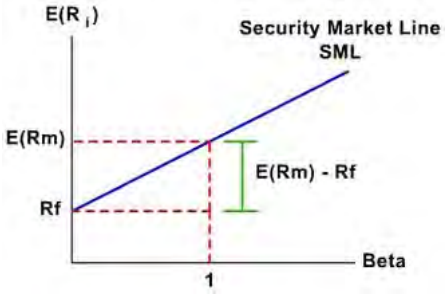
市场的风险通常用大盘指数的波动率来衡量（波动即风险），由于风险的存在，大盘期望的收益率r_M要比无风险利率r_f要高一些，其超额收益率（作为对风险的补偿）记为$R_M = r_M - r_f$，对于单个股票，采用类似的方法，记为$R_i = r_i - r_f$，那么，第i个股票的预期收益率记为

$$R_i = \beta_i R_M$$

或者更常见地，写成

$$r_i = r_f + \beta_i(r_M - r_f)$$

用图像来表示就是



由上图可知，在β=0的时候，为无风险收益，没有市场风险，也没有超额收益。β越大，超额收益也越大，资产风险和市场风险的相关程度也随之升高。作为补偿的期望超额收益也越大。
不完美的CAPM
但是，CAPM模型是不是完美的呢？显然不是，CAPM首先就假设了市场风险是唯一风险因子。但在现实中，仅仅由市场风险是不足以解释资产回报的，因此，人们对CAPM做了改进，认为资产的收益率中的一部分是资产内秉特性，记为α，那么有

$$r_i = r_f + \beta_i(r_M - r_f) + \alpha_i$$

或者等价于

$$\alpha_i = r_i - [r_f + \beta_i(r_M - r_f)]$$

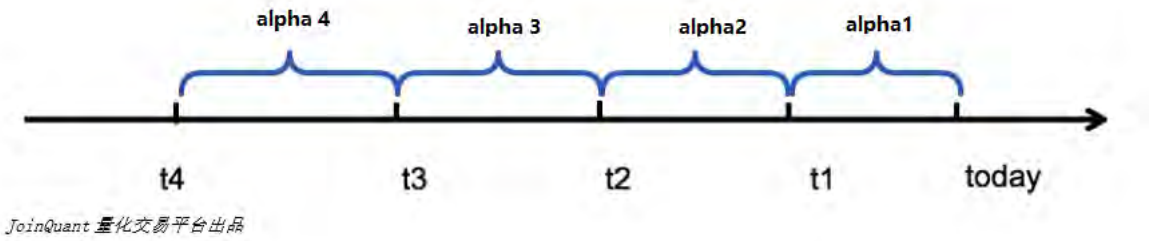
再次强调一下。CAPM模型本身是没有alpha的，我们引入这个α_i目的是要把股票的收益率归结为两个方面，一个是大盘的因素，用beta系数表示，一个是个股的因素，用α表示。比如广为人知的市值因素，动量因素，各种财务因素，等等。
CAPM+APT选股应用
根据上文，我们有：

$$r_i = r_f + \beta_i(r_M - r_f) + \alpha_i$$

这个公式把单一资产的收益率分解成三部分：无风险收益率r_f，与整体市场相关的部分$\beta_i(r_M - r_f)$，以及个股内秉因素α_i

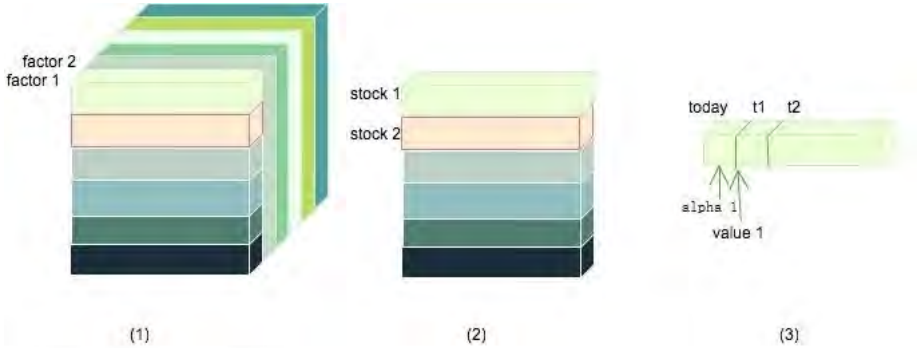
这个α_i怎么刻画呢？还请大家回顾一下量化课堂文章《多因子策略-APT模型》，同那篇文章的思想，我们有相信α与公司的财务指标密切相关，故我们可以通过历史的财务指标和历史α进行线性回归，并用计算出的回归系数结合当期的财务指标预测下一期的α。

下面的图来自于APT模型的原文，我们要把过去几个时期的财务数据和α值进行统计和回归分析。



和上篇文章的方法类似，对于单只股票 i 的所有财务因子，t1日的因子值和t1~today期间的超额收益α_i1组成了一组观测值；t2日的因子值和t2~t1时期的超额收益α_i2组成了一组观测值；一直到t4日的因子值和t4~t3时期的超额收益α_i4组成了一组观测值。

我们取N支股票，按照财报周期向前获取4期的财务因子和期间内的α，以此获得4N组观测值，数据结构如下。



JoinQuant 量化交易平台出品

我们实际选取的财务因子有：市盈率、扣除非经常损益后的净利润、净资产收益率等九个指标，具体的因子选取方法请参见APT模型的文章。

我们将获取的数据组放入回归模型进行线性回归计算。如果用$F_1, F_2, \ldots, F_n$表示我们考虑的因子的话，回归的模型就是

$$\alpha = b_1 F_1 + b_2 F_2 + \cdots + b_n F_n$$

我们估算出回归系数$b_1, b_2, \ldots, b_n$。在这之后，我们可以将现期的财务因子和计算出的回归系数，代入上面线性函数计算出未来一段时间的α的预估，并调仓持有α预估值最高的股票。

策略和回测

我们构建策略如下：
第一步：根据CAPM模型计算得到股票池中每只股票对应财报季度的α值。
第二步：将对应季度的财报因子数据和α值进行多元线性回归
第三步：利用第二步中得到的回归参数对下一期的股票α值进行预测
第四步：将所得的预期α值进行排序，选择N支α最大的股票（若不足N支α大于0，则取所有α大于0的股票），并且将资产平均配置到股票组合中

在实际操作中，可以通过买入预期alpha>0的股票组合卖空沪深300指数进行操作。实现的方法只需要在聚宽社区的回测方法的基础上把日均收益率换成同期的alpha观测值就行了。

如下图所示，使用股票池为沪深300，股票数量N=0，回测周期是2006-01-01到2016-06-06，策略收益率是545.56%。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	最大回撤
545.56%	20.21%	244.23%	0.074	0.985	0.499	0.636	0.547	0.325	0.304	71.121%

从这个图中我们可以看出，策略的收益率总体上比大盘略高，在最近一两年尤为明显，因此如果结合对冲的策略，收益应该会更好。

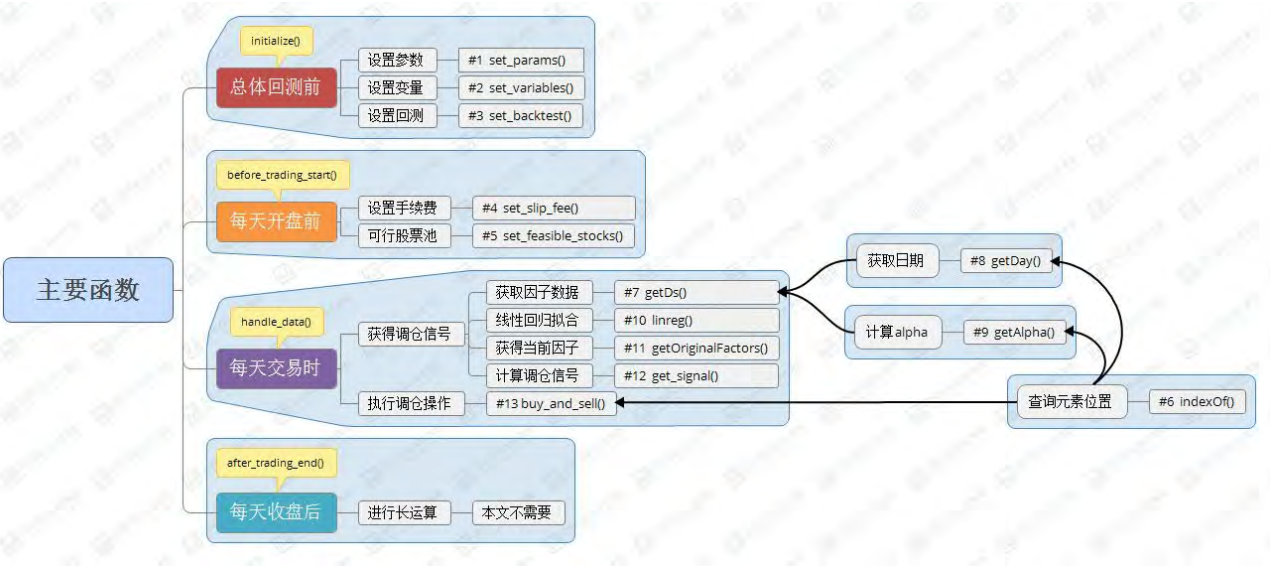
这里使用策略净值/大盘净值的方法来描述超额收益率：

JoinQuant 量化交易平台出品

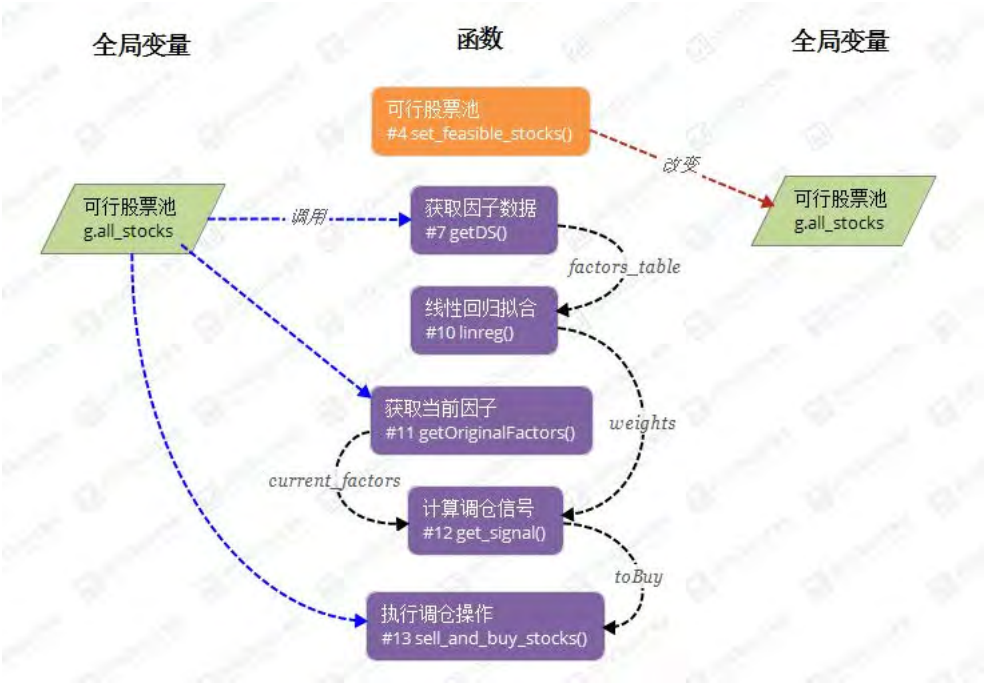
这进一步证实了之前的猜想，这个策略总体上08年以后就有效了，在牛市期间表现比较好，但是在熊市的时候还是会略有回撤。也许有的读者注意到了，虽然看似15年以后超额收益率回撤挺大，但这是由于比例尺放大了造成的。因为在你资产价值已经是开始的好几倍的情况下，而沪深300指数期货只使用原来资产价值对应的仓位进行对冲，当然对冲效果不好了。
解决这个问题的一个方法是动态调仓。就是根据当前的资产价值来更新沪深300指数对冲的仓位，还请读者期待股指期货的回测上线。

函数和变量说明书

函数说明书：



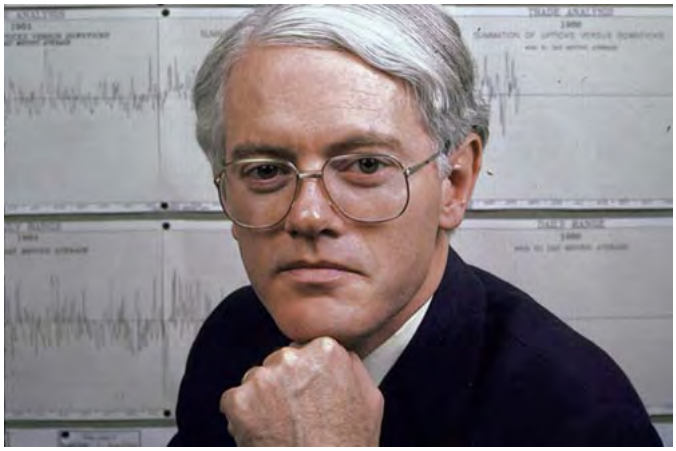
变量说明书：



文章迭代记录
v1.0，2016-07-30文章上线

【量化课堂】彼得·林奇的成功投资

导语：投资大师彼得·林奇（Peter Lynch）有过一个著名的论断：任何一家公司股票如果定价合理的话，市盈率就会与收益增长率相等。这就是PEG估值法，PEG在综合考虑了低风险以及未来成长性的因素，可用于股票价值评估。本文浅谈“PEG估值法”，并给出策略，进行回测。



本文由JoinQuant量化课堂推出 。难度标签为入门，理解深度标签：level-0

作者： 宏观经济占卜师
编辑： 宏观经济算命师

彼得·林奇与PEG估值法

财富的化身--彼得·林奇
彼得·林奇（Peter Lynch）生于1944年1月19日，是一位卓越的投资家，曾被《时代杂志》评为首席基金经理。1977年至1990年，在彼得·林奇管理麦哲伦基金的13年间，基金规模大幅扩增，由2000万美元成长至140亿美元！一跃成为全球之最。
彼得·林奇在投资上的功绩和影响，几乎无人能及。今天，我们介绍一下彼得·林奇推广的“PEG估值法”，来体会大师的投资思想。

PEG估值法简介
达·芬奇说过：”把最复杂的变成最简单的，才是最高明的。“PEG估值法就很好的体现了这个思想。下面，小编给出PEG的估值原理。
一言以蔽之：计算每只股票的PEG值，并排序，取PEG值最小的前n支股票，作为待买股票即可！

那么，这个PEG究竟是什么呢?
先介绍几个基本概念:

1.EPS(Earnings Per Share) 表示每股收益（一般按年计算）:

$$< br > EPS = \frac{\text{归属于普通股股东的当期净利润}}{\text{当期实际发行在外的普通股加权平均数}} < br >$$

2.PE（Price to Earning Ratio ）表示市盈率，是当前股价（P）相对每股收益（EPS）的比值:

$$< br > PE = \frac{P}{EPS} < br >$$

将EPS按照不同的计算法方法取出，我们可以得出适用范围不同的PE（市盈率）:

若每股收益(EPS)取最近4次的季报的每股收益的平均值，则计算出“滚动市盈率”（又称市盈率TTM）；

若每股收益(EPS)取去年的12个月的每股收益，则计算出“静态市盈率”（又称市盈率LYR）；

若每股收益(EPS)取预测下一年的每股收益，则计算出“动态市盈率”。

本文取“ 市盈率TTM”，更加接近现实，靠谱。

3.G（Growth Rate of Expected Profit ）表示企业的收益增长率。收益增长率的计算方法不一，简便起见，本文取EPS增长率：

$$< br > G = \frac{EPS\ this\ year - EPS\ last\ year}{EPS\ last\ year} < br >$$

4.基于以上几个指标，得出PEG的计算公式:

$$< br > PEG = \frac{PE}{G < em > 100} < br >$$

从以上公式可以看出，PE蕴含着股价的信息，PEG是一个股价相对于收益增长率的比值。直观来讲，PEG越高，代表该公司的股价有被高估的可能性，不建议买。反之，PEG越低，代表该公司的股价有被低估的可能性，考虑买入(*一般情况下，PEG越低越好。

PEG估值法的适用条件

PEG是一个综合指标，既考察价值，又兼顾成长性。不难看出，PEG估值法侧重于成长型的公司，非常适合对成长型公司的股票价值评估。马克思说过:“任何真理都有自己适用的条件和范围。”在投资的世界里，倘若不在合适的范围内进行操作，就不是一个好策略。

要想成为一个好策略，就必须预先过滤股票池，筛选出符合PEG估值法运用条件的股票。以下几种情况就不适合用PEG估值法进行估值:

类别	原因
周期性行业	利润基础稳定不住，使用PEG容易造成误差
非成长股	收益增长率或市盈率为负的股票，不符合高增长的条件
项目类公司	利润高低依赖于公司接的项目数，利润基础稳定不住
稳定的大型公司	这类公司往往稳定有余而成长不足
融资依赖型的企业	因融资带来的高增长，不能稳定持续

本文仅对股票池的“非成长股”进行排除，有兴趣的用户，可以进行多次过滤。

利用PEG寻找成倍牛股

PEG估值法是彼得·林奇用作评估成长型公司价值的方法。PEG数值通常可分为四档:

PEG值	股票价值评估
0至0.5	相对低估
0.5至1	相对合理
1至2	相对高估
大于2	高风险区

目前，我国处于高速发展的经济中，我们对A股高成长型公司的合理估值可以定为：PEG=1。举几个例子:

公司名称	市盈率	收益增长率	PEG	股票价值评估
A公司	66倍	30%	2.2	高风险区
B公司	30倍	30%	1	估值合理
C公司	30倍	100%	0.3	低估，买入建仓

大家理解了吗？C公司就是那支能为你带来N倍收益的牛股，依照PEG估值法的思想，PEG值越低越牛哦！！！

策略思路

step1：
设置沪深300为初始股票池，实际情况中，当天停牌的股票是无法进行买卖操作的，所以在整体回测前，将当日停牌的股票剔除，得到可行股票池。

step2:
前面已说明，本策略仅对成长股有效，所以仅仅过滤掉当日停牌的股票是不完善的。仍需过滤掉市盈率（PE）为负值，或收益增长率（G）为负值的股票。聚宽平台的取数据函数get_fundamentals可以直接取PE，G值（详见【数据】模块的股票财务数据）。get_fundamentals函数的默认日期是context.current_dt的前一天（详见API）,因为当天是无法知道今日的某些数据的。本策略使用默认值（缺省），避免未来函数，不建议修改。
step3：
整体思路是非常简洁的，下面对股票的PEG进行排序，取出PEG最小(且全都小于0.5)的前n只支股票，作为调仓时待买入的股票列表。
step4：
每次调仓时，先卖后买，腾出资金。对不在待买入列表的股票，执行卖出操作。对在待买入列表的股票，分配资金，执行买入操作。

应用PEG估值法的误区

在克隆（见文末，附源码）本策略之前，小编给大家列了一些预先需要注意的地方，防患于未然哈：

1.取市盈率和收益增长率数据时，应该取回测当天的前一天的数据，避免未来函数。在聚宽平台，取数据函数get_fundamentals时间默认值是回测前一天，保持默认即可，不建议修改。

2.计算出的PEG值并非越小越好，因为计算PEG时所用的收益增长率，是过去n年平均指标这样的历史数据。实际上，决定上市公司潜力的是其未来的增长率。目前小PEG的公司并不代表其今后这一数值一定就小。

3.怎样选择高增长的公司?
重要的是你选择的公司是要处在高增长的行业，比如互联网、环保、新材料、医药等，行业高增长是企业高增长的基石。选对了行业，再选择行业内的佼佼者即可。

4.市盈率或收益增长率为负的公司怎么计算PEG?
这种公司对于小编来说没有研究价值。Just kick them O~U~T~

5.市盈率用静态LYR?? 滚动TTM? 还是动态的?
市盈率LYR: 以上一年度每股收益计算的市盈率。靠谱，但信息陈旧。
市盈率TTM: 以最近四个季度每股收益计算的市盈率。靠谱，更接近现实。
动态市盈率：以未来一年每股收益计算的市盈率。取决于行业研究员的财务报表的数据分析，具有预测未来的效果。
其中，市盈率LYR和市盈率TTM可以用聚宽平台get_fundamentals函数直接取（见API和数据模块的股票财务数据）。
本策略使用“市盈率TTM”。

文章更迭记录：
v1.0，2016-07-30，文章上线

【量化课堂】Hurst 指数与应用（单股票）

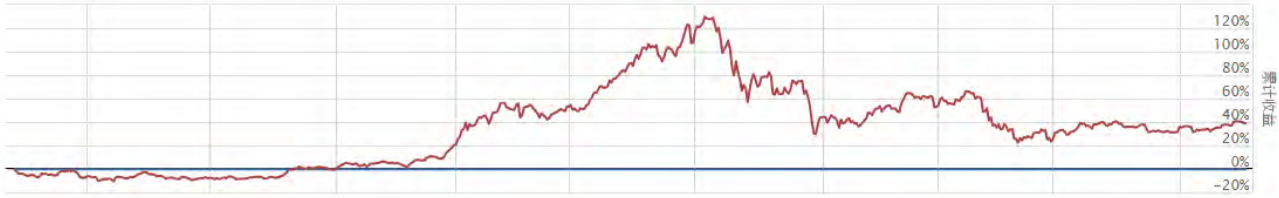
导语：真实中，股票收益率的分布并不像我们所期望的那样是独立正态分布，而是有偏的——收益率时间序列往往存在自相关性。如果能够根据收益率的记忆性分析出它的趋势，那么便可以将此作为一个择时的依据，制定量化交易策略。那么，如何刻画这一统计特性？使用Hurst指数便是其中一种办法。

本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0
作者：胖子邓

编辑：宏观经济算命师

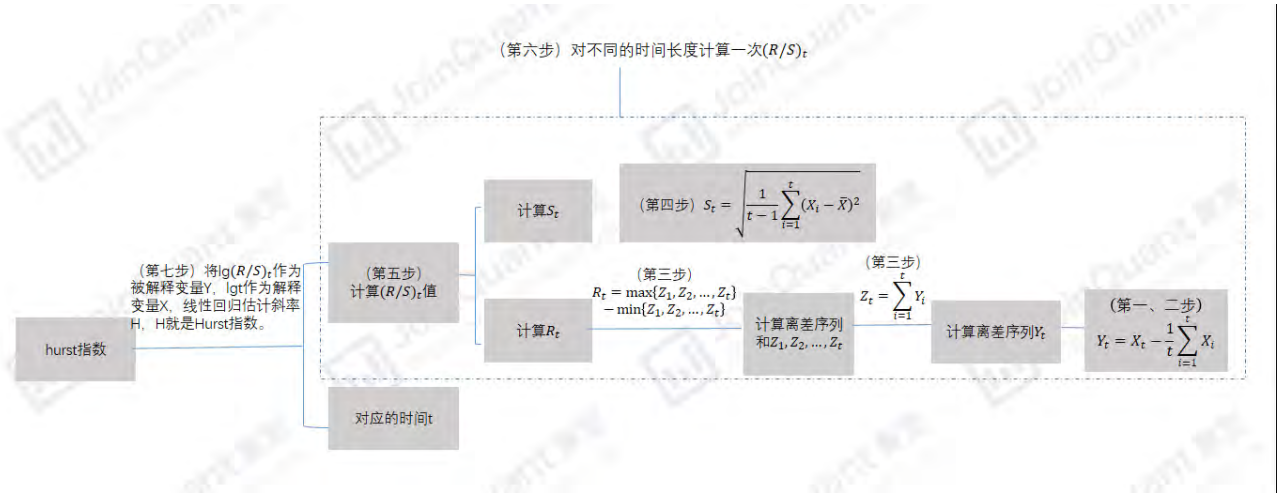
在标准金融学模型中，我们基本都假定股票的波动是随机独立的。一个随机独立的波动率对应的股票变动如下图所示：

但是实际上，真实的股票数据并非这样——股票的低迷往往会持续很长的时间，而突然的增长则是远超过平均收益的暴涨。

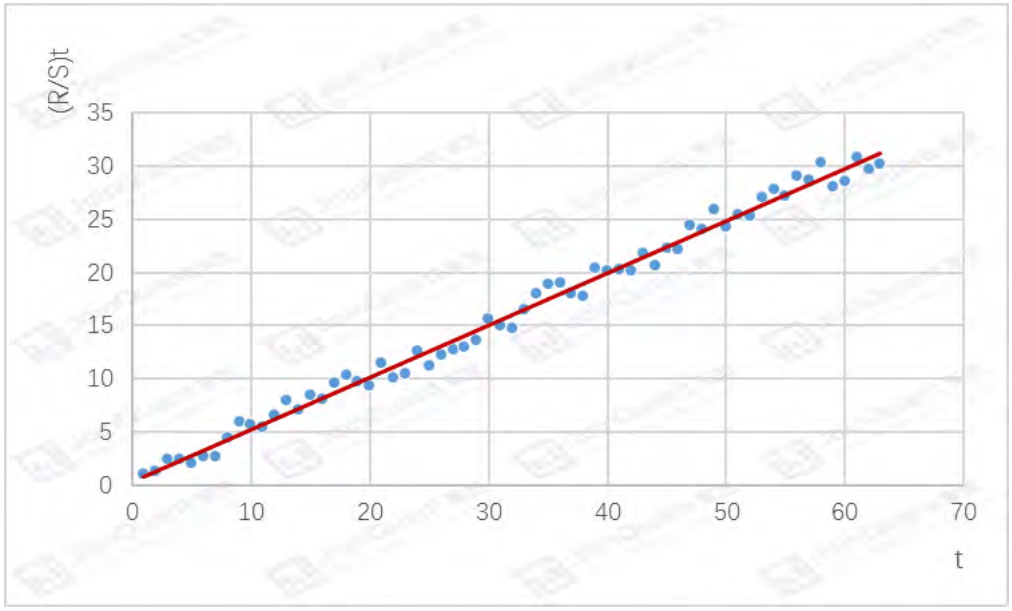


直观上的原因在于，看上去随机的收益率并不是那么的随机——它们具有一些自相关性。例如，在大部分股票市场，当前收益率持续较高的时候，之后的收益率也更可能高（牛市往往持续一段时间），当前收益率低时，之后的收益率也低（熊市往往也会持续很长时间）。
听起来有点像动量效应，其实吧，我们得再次声明下，任何CTA类策略，要么就是抓趋势（偏动量）的，要么就是抓震荡（偏均值回归）的。
自相关性的应用其实起源于其他场景。Hurst指数就可以刻画这一统计性质。Hurst指数的来源就是Hurst本人在研究河流流量之后发现，干旱的时间愈久，就更可能出现持续的干旱；大洪水之后，也更可能出现大洪水。
我们可以像hurst本人刻画河流流量一样，将hurst指数用于刻画股票的记忆性，通过识别股票价格的波动趋势，买入更有可能持续上涨的股票，实现收益。

1.1 Hurst 指数的计算



（上图是计算hurst指数的流程图，可配合以下文字一起咀嚼）
简单来讲，hurst指数的计算，是计算一个序列$(R/S)_t$（接下来会具体介绍$(R/S)_t$的计算方法）对t的变化斜率。



如上图所示，如果我们有一系列（t，$(R/S)_t$）的数据，那么回归得到的红线的斜率就是hurst指数的值。
而这里的t就是选择的样本区间的长度（例如选择了八十天的数据，那么t=80），接下来的关键就是计算$(R/S)_t$值。

取沪深300日回报率(daily return)n天内的数据作为时间序列，记回报率$X = X_1, X_2X_3, \ldots, X_n$为一串时间序列。

第一步：计算这个序列的均值:

$$m < em > t = \frac{1}{t} \sum < /em > i = 1^t X_i$$

第二步：计算离差序列(deviation):

$$Y_t = X_t - m_t$$

第三步：将离差序列求和，计算出离差序列和的极差(widest difference):

$Z < em > n = \sum < /em > i = 1^n Y < em > i$
$R_t = \max(Z_1, Z_2, \ldots, Z_n) - \min(Z_1, Z_2, \ldots, Z_n)$
$Y =$ 离差序列
$Z =$ 离差序列和
$R =$ 离差序列和的极差

第四步：计算序列X的标准差(standard deviation):

$$S_t = \sqrt{\frac{1}{t-1} \sum < /em > i = 1^t (X_i - \overline{X})^2}$$

第五步：计算R/S值:

$(R/S)_t = R_t/S_t$
$R =$ 第3步中计算的离差序列和的极差
$S =$ 第4步中计算的标准差

根据上面1-5步的计算，我们得到了一个序列$X = X_1, X_2, X_3, \ldots, X_n$对应的R/S，因为这个序列样本有n个，因此R/S可以写作$(R/S)_n$
如前所述，我们需要计算一系列的（t，$(R/S)_t$），因此我们需要重复上面的步骤——

第六步：重复1-5步计算R/S，但分别取不一样的t值（t=2,3,…,n），这相当于我们对序列：

X_1, X_2
X_1, X_2, X_3
X_1, X_2, X_3, X_4
$\ldots$
$X_1, X_2, X_3, X_4, \ldots, X_n$
分别计算按照上述分段方法得到的$(R/S)_t$值。

第七步：计算Hurst指数
a. 将每种分段方法的片段大小（即所取得时间区间t）和R/S对10取对数
b. 这样我们就有了对数序列。将$lg(R/S)_t$作为被解释变量Y，lgt作为解释变量X，线性回归估计斜率H，H就是Hurst指数。
对Hurst 指数的直观解释是：当所选取的样本越多，样本的变化也就越丰富，因此离差序列和的极差（以下简称极差）也会越大。如果序列有相关性时，序列长期向着一个方向变化，计算所得的极差就会比完全随机的情形更大，因此当极差随样本增加增长得越快时，序列的记忆性就越强。而hurst就是描述样本

规模和R/S的上述关系，当hurst指数大的时候，其记忆性自然也就越强。
（我们这个是level0的理解深度，未来的量化课堂文章会讲原理）

1.2 Hurst 指数的性质

Hurst指数体现了时间序列的自相关性，尤其反映了序列中隐藏的长期趋势，统计学上称为长期记忆(long-term memory)。Hurst指数居于0-1之间。以0.5为间隔，时间序列在不同的区间会表现不同的特性：

H=0.5：标准布朗运动。此时，序列为随机游走，表现马尔可夫链特性。
H≠0.5：分形布朗运动，这种运动不是完全随机的，具有长期相关性——

```
当H>0.5时，表现为持续性（即持续之前序列的性质）；
当H<0.5时，表现为反持续性。
```

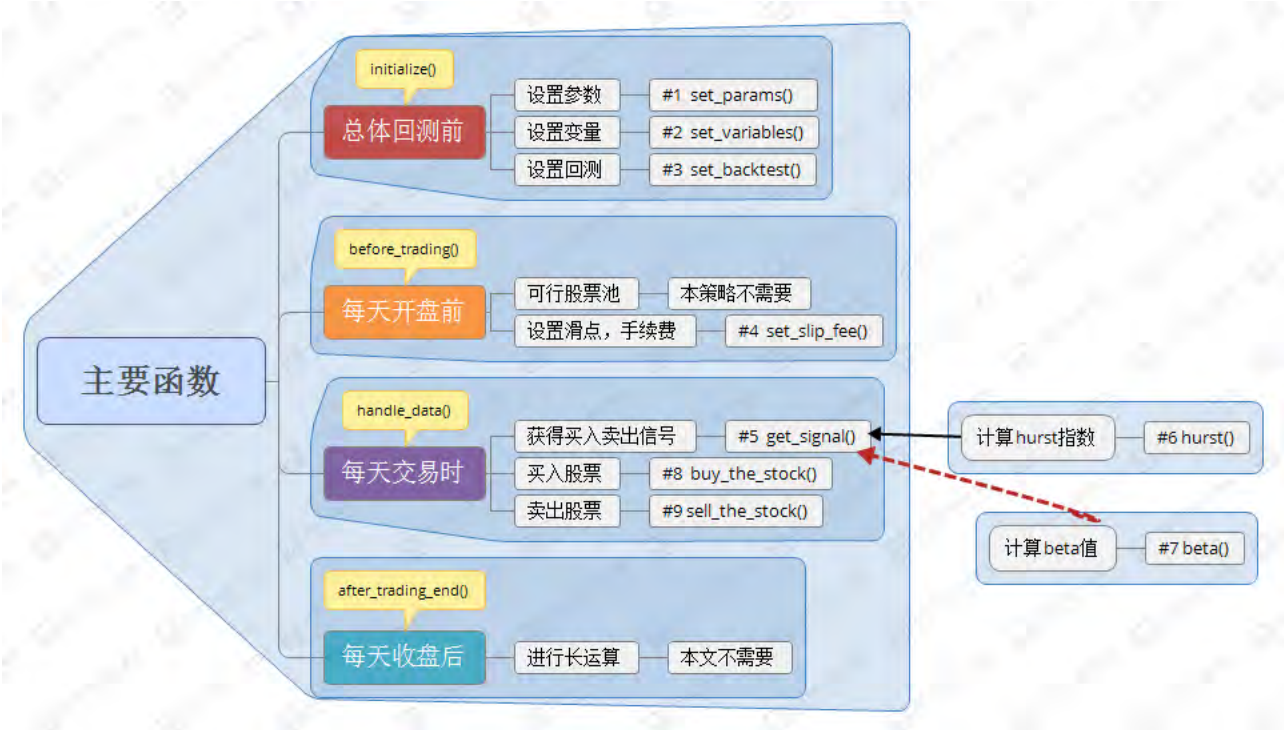

1.3 Hurst 指数的应用

思路：根据收益率变化趋势与hurst指数，判断应该买入还是卖出——
如果过去收益率增加并且序列有正相关性，买入股票；
如果过去收益率降低并且序列有正相关性，卖出股票；
如果过去收益率降低并且序列有负相关性，买入股票；
如果过去收益率增加并且序列有负相关性，卖出股票；

具体而言，在我们的策略中，采用hurst指数来衡量序列相关性。当hurst>0.6时，认为序列有趋势性，当hurst<0.4时，认为hurst有负相关性；

而对于判断过去收益增加还是减少，一种办法是直接用当前收盘价和前N日收盘价比较，如果当日收盘价高，则认为收益增加。但是在我们的设定中，这种办法的缺陷是这实际上只能判断收益增加，但是不能判断收益率的变化，因此在本文的策略中，采用的是N日收益率对时间回归。如果回归所得的斜率为正，则认为在过去N日中收益率是增长的趋势（注意：是收益率向上，而不是股票走势向上哦！）；如果回归所得的斜率为负，则认为在过去N日中收益率是减少的趋势。
注：在应用中，我们取N=59（收盘价数据取N=60天的数据）。

根据回测结果，可以看出hurst的择时在应对大跌时效果较好，但因为要保证收益率是持续增长，往往在大牛市中会提前空仓，但一整个牛熊市下来，收益率远高于大盘。



文章迭代记录
v1.0，2016-07-30文章上线

【量化课堂】高息股策略

导语：选股是量化投资中极重要的一个方面，可以通过基本面行情判断行业选股，也可以通过公司财务指标选股，也可以采用技术分析选股。这里，我们以选择高息股作为选股策略，顺便演示一下国泰安数据的使用方式。

本文由JoinQuant量化课堂推出 。难度标签为入门，理解深度标签：level-0

作者： 胖子邓
编辑： 宏观经济算命师

为了方便大家更容易获取股票和公司的相关信息，joinquant免费推出国泰安数据库，包含上市公司基本信息、分红、重大事项、宏观经济等多项数据，可在joinquant的数据中查看。

顾名思义，高息股就是指市场中分红较多的股票。我们可以根据如下的方式计算其股息率：

$$
 股息率 = \frac{分红金额 \times 每年分红次数}{股票价格} \times 100$$

其中分红金额和分红次数可以在
国泰安数据 > 融资与分配 > 红利分配信息表 > 每10股派现（税前）& 分红次数中查看（股息率的计算还可以参考joinquant上的另一篇文章：《PM 教你计算股息率》）

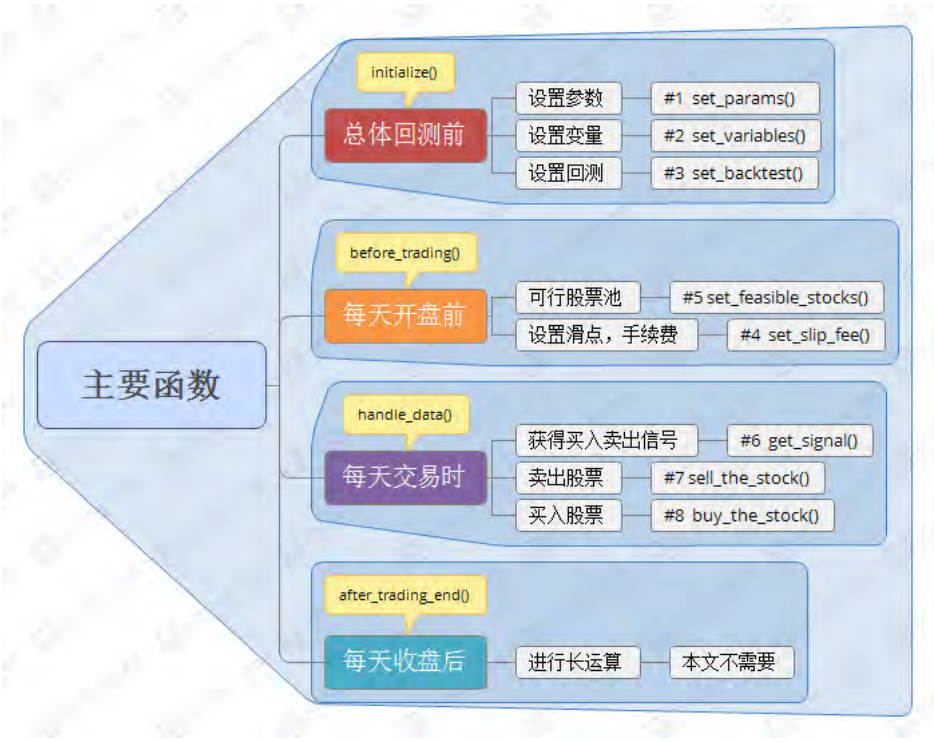
我们的策略很简单：调仓日期选择每年1,4,7,10月份交易日的最后一天（按季度调仓），将当天开盘的股票按照股息率（参考价格为昨收盘）从大到小排列，并且选取在特定排序区间的股票买入。

我们将回测时间设定为2006年1月1日至2016年7月31日，股票池设定为沪深300本身，并且每次调仓买入股息率排名5~20的股票。

从回测的结果上看，06年至今的收益率为632%，有一定的超额收益率存在。
但是，还存在改进的空间：

1. 高股息并不能完全反映公司的财务状况，作为单一指标的选股策略，存在一定缺陷；
2. 高股息策略完全没有考虑择时的问题，而这在股票投资中是相当重要的。
3. 在沪深300股票池中选择高股息，很大程度上就反映了大盘的平均水平，从回测结果来看，也基本上如此：虽然高息股策略优于大盘，但是超额收益不是稳步增加，同时波动基本是跟着大盘走的，受市场影响很大；

高息股策略只是一个引子，大家可以通过我们提供的股票、公司、基本面数据尽情地尝试和测试自己的交易策略:-D



文章更迭记录：
v1.0，2016-08-09，文章上线

【量化课堂】指标效果的统计分析：思路之一

我们经常会在交易中使用一些技术指标，但很多时候只是知道这个指标的核心思路，并不太确定它们的实际效果。可能它们的效果和我们的想法有偏差。可能它们的效果是因股票而异的。也可能我们连一个具体的思路都不清楚，只是天马行空地想到，“咦？这个也许能行？”

我们把这些指标写到策略里，有时候回测看着像这么回事，可有时候又不像那么回事，懵懵懂懂的我们也不知道是怎么回事了。

“那…如果，”你说，“嗯，不对，让我再想想…”

时过半晌，“如果，”你又打断了沉默，“有一个系统性的统计方法，能让我们直观地看到这个指标的效果，那就好了。”

诶哟你说得太对了，我就是来找你谈这个的。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，本文的难度属于进阶（上），深度为 level-1。

统计的对象

现代金融理论认为，证券价格的时间序列在大部分时间服从随机游走，但是在一些特定的时刻（比如基本面发生变化，或者供需关系发生变化），价格序列会偏离随机游走，并且选择向上或者向下的方向。当价格在脱离随机游走时，通常都会产生一些现象或征兆，就是我们常说的“信号”，如果我们捕捉到这些信号，就可以赚取收益。

我们使用利用股票的历史数据计算各种各样的指标（比如MACD，RSI），就是为了通过指标的数值来判断是否出现信号。但很多时候指标的使用是粗糙并且模糊的，比如“当RSI低于20时买入”是一个发出信号的标准，但为什么是20呢？18或者22会不会效果更好？我们想通过历史统计来分析这个问题，那么进行统计的对象就有两个：指标出现的数值，以及出现该数值之后的涨跌结果。
指标

一般而言，一支股票的指标是一个函数Ind(T)，它输入的是时间T（确切地说还有时间T之前的所有股票数据），返回的是在那个时间点的指标数值。比如说，我们要考量的指标是过去5天的收益率，那么指标的输出就是今日收盘价除以五天前收盘价的商再减1。
结果

结果指的就是，我们认为指标所预测的事件到底有没有发生，由此把结果分为“赢”、“平”和“输”。举例来说，假设我们认为过去5天的收益率越高，未来两天的收益率就越大；那我们要观测的结果就是未来两天交易量的情况，如果未来两天的平均交易量大于今天的110%，就记作“赢”，小于今天的100%就记作“输”，其余情况记作“平”。这里“赢平输”的计算标准有一定拍脑袋的成分，这么做的缺点就是统计的数据不全面，优点是结果更直观更方便应用，并且如果统计结果不理想，我们可以更改输赢的决定方法再重新来过。

指标和结果的统计

数据统计

一、提取历史所有交易日的股票数据，并去掉停牌日的信息；
二、记录每一天的指标和输赢结果。对于每一个不停牌的交易日T，做：
    a. 计算当日的指标Ind(T)，方便统计需要，四舍五入到合适的小数点位；
    b. 计算从T日起观测的输赢结果；
    c. 记录事件组(Ind(T),赢或平或输)。
三、统计指标的某个值出现后赢和平和输分别发生过多少次。

举例个简单的例子。假设我们认为过去5天涨得越多则未来五天越可能跌。那么把五日收益率设为指标；未来第五天收盘价小于今日收盘价则记为赢，大于记输，等于则记平。

首先收集数据，注意要把停牌时的数据去掉，这里是从06年2月2日到16年2月2日。

```
prices = get_price('002200.XSHE', start_date="2006-02-02", end_date="2016-02-02", frequency='daily', fields=['close'], skip_paused=True)['close'

然后进行统计，第二步第三步可以一气呵成

# 要用到pandas
import pandas as pd

def get_stats(prices):
    # 设置一个计数器
    i = 5
    # 设置一个空list
    data = []
    # 从第一个到倒数第二个价格
    while i < len(prices)-4:
        # 计算五日收益率，留小数点后2位
        ratio = 0.01 * ((prices[i]/prices[i-5]) //0.01)
        # 如果第五天收盘价更低
        if prices[i+4] < prices[i]:
            # 那结果记赢
            result = 'win'
        # 如果第五天收盘价更高
        if prices[i+4] > prices[i]:
            # 结果记输
            result = 'lose'
        # 收盘价不变的话
        if prices[i] == prices[i+4]:
            # 记平
            result = 'even'
        # 看看该比例有无记载过
        ratio_recorded = False
        # 翻看data
        for data_dict in data:
            # 如果比例被记载过
            if data_dict['value'] == ratio:
                # 那就好，更新输赢
                data_dict[result] += 1
                # 记载过为是
                ratio_recorded = True
        # 如果翻完了发现没记载过，
        if ratio_recorded == False:
            # 那么就记载下来
            data_dict = {'value':ratio, 'win':0, 'lose':0, 'even':0}
            data_dict[result] += 1
            data.append(data_dict)
        # 别忘更新i
        i += 1
    # 转换成DataFrame
    df = pd.DataFrame(data,  columns=['value', 'win','even', 'lose'])
    # 按照value列从小到大排序
    df = df.sort(['value'], ascending = True)
return(df)
```

然后就可以看到结果啦

```
stats = get_stats(prices)
stats.head(10)
```

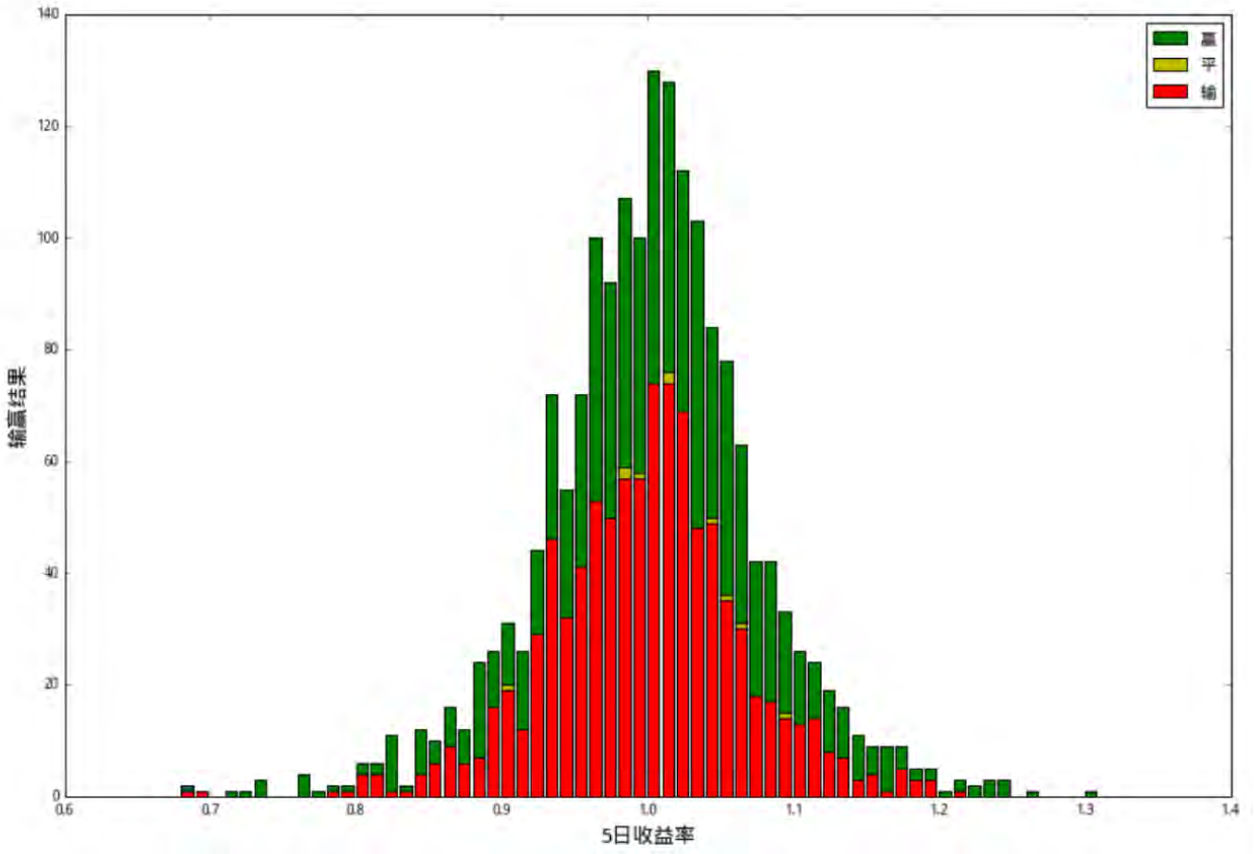
	value	win	even	lose
49	0.68	1	0	1
55	0.69	0	0	1
52	0.71	1	0	0
51	0.72	1	0	0
48	0.73	3	0	0
50	0.76	4	0	0
14	0.77	1	0	0
22	0.78	1	0	1
46	0.79	1	0	1
31	0.80	2	0	4

制图

把统计好的数据画出来就可以看出效果了。基于以上数据的特性，我们要画一个叠加条状图。该图的x坐标轴是指标的值，y轴则是该指标值出现的次数，并按颜色划分成“赢平输”叠加的条状图。代码如下：

```
# 要用到pyplot包
Import matplotlib.pyplot as plt

# 设置图的大小
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 10

# 把统计的DataFrame中竖列抽出来
values = stats['value']
wins = stats['win']
evens = stats['even']
loses = stats['lose']

# 画输的数目，颜色为红
p1 = plt.bar(values, loses, width = 0.008, color='r')
# 画平的数目，颜色为黄，底部在输之上
p2 = plt.bar(values, evens, width= 0.008,  bottom = loses, color='y')
# 画赢的数目，颜色为绿，底部在输+平之上
p3 = plt.bar(values, wins, width=0.008, bottom = evens+loses, color='g')

# 标注坐标轴和注释
plt.ylabel('输赢结果',size = 15)
plt.xlabel('收盘价除以22',size = 15)
plt.legend((p1[0], p2[0],p3[0]), ('输', '平','赢'))
```

得到下图

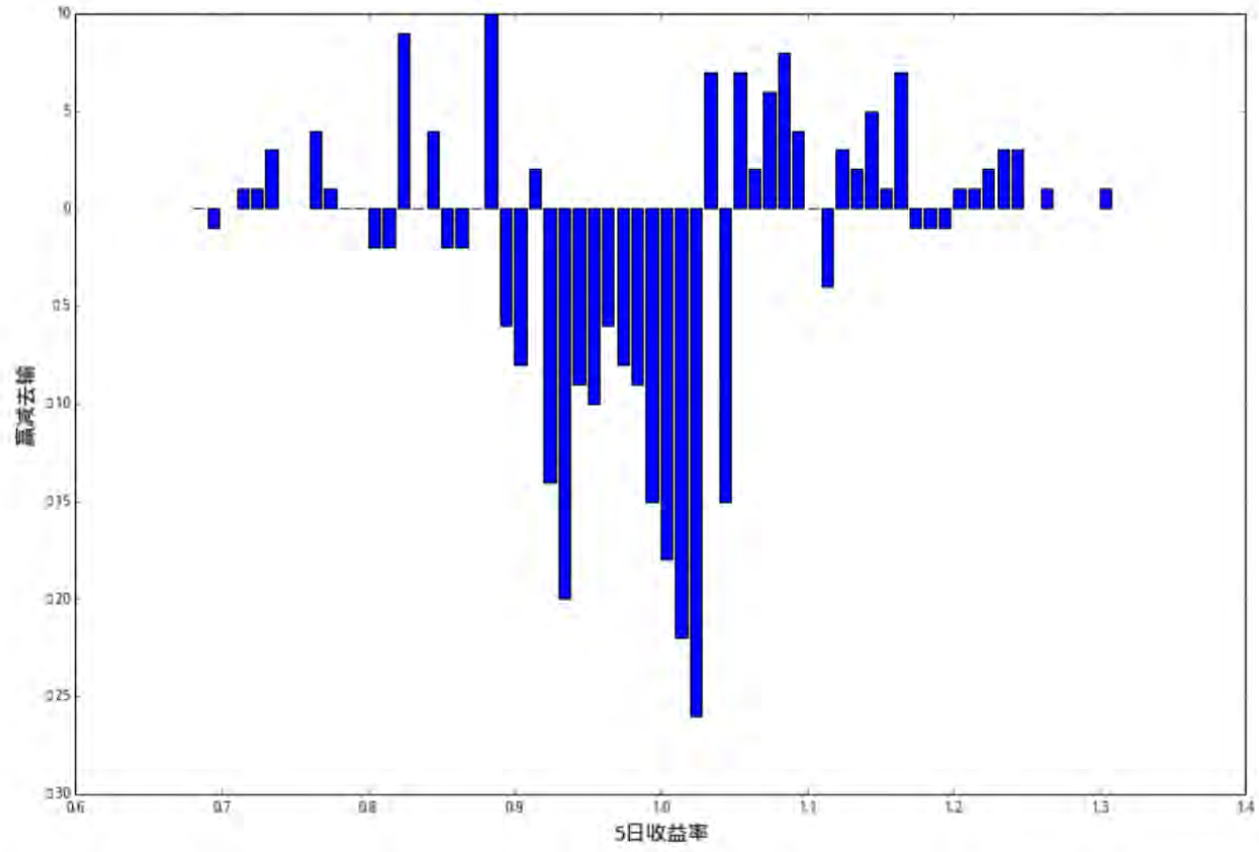


为了方便看出赢和输的差距，可以把赢减输画出来。

代码

```
# 画赢数减去输数
pdifference = plt.bar(values, wins - loses, width = 0.008, color='b')

# 标注坐标轴和注释
plt.ylabel('赢减去输',size = 15)
plt.xlabel('5日收益率',size = 15)
```

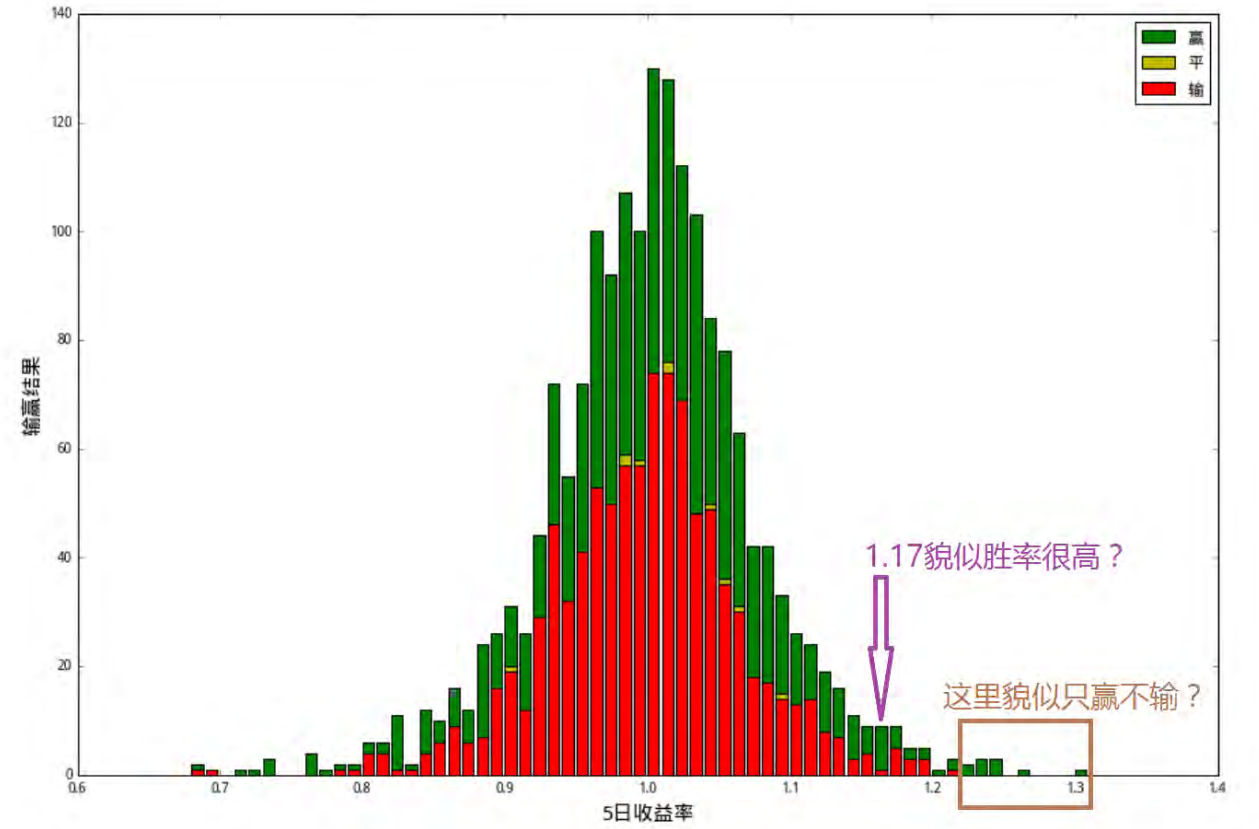
得到



这样就可以清晰地看到一些规律。

合理地使用统计结果

统计结果不可以乱用
我们单单把图画了出来，那是不是就可以在策略中用了？当然不是。拿之前的统计举例，可以看见两个比较明显的问题

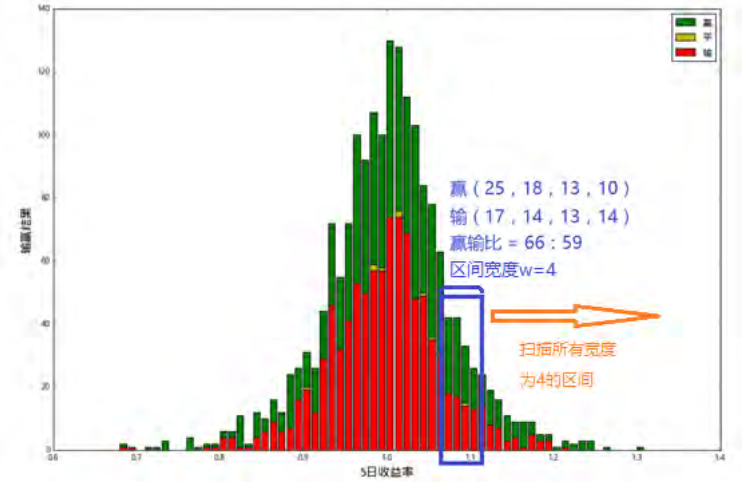


先看紫色标注的位置，那里统计的胜率很高。如果们选择在5日收益率1.17时买入，会怎么样？首先5日收益率正好为1.17的概率很小，可能若干年都不出现信号。其次，可以看见它右边的1.18的胜率并不高，那么实际上当指标计算出为1.175时就无法合理判断。

再看棕色框起的区域，这里只有赢没有输，但是这个区域中样本量太小了，完全无法保证统计结果的准确性。
选择胜率最高的区间

一个可行的解决办法，是设定一个指标区间的宽度w，并且设定最低数据比θ。如果一个宽度为w的x轴区间里的数据量占总数据量的θ以上，我们就计算并记录该区间里的输赢比；如果区间内数据比例少于θ，我们则抛弃该区间。最后选出输赢比最高的区间作为产生信号的指标值。

比如我们设w=4，那么在1.08到1.11的区间里，赢输比为66:58。



我们要选的是依照该计算方法得到的赢输比最高的区间。

下面函数的4个输入分别为：
statistics – 之前算出的DataFrame统计数据
tick_width – 指标值的最小间隔（跳动值，比如上面的示例里就是0.01）
least_percentage – 要求区间数据量至少有总数据量的多少
band_width – 区间的宽度，整数，可理解为进行多少次tick_width的跳动。

代码如下

```
# 返回一Series，内容 low=区间低点，high=区间高点，ratio=区间赢输比
def find_best_region(statistics, tick_width, least_percentage, band_width):
    # 取出统计df的列
    values = statistics['value']
    loses = statistics['lose']
    wins = statistics['win']
    evens = statistics['even']
    # 算总数据量
    num_data = sum(wins) + sum(loses) + sum(evens)
    # 起始一list
    mydata = []
    # 计算指标统计出的最低值除以间距，取整数，方便后面移动计算。
    low_bound = int(statistics['value'].iloc[0]/tick_width)
    # 计算指标统计出的最高值除以间距，减去区间宽度
    high_bound = int(statistics['value'].iloc[-1]/tick_width - band_width + 1)
    # 对于上限和下限之间的所有整数
    for n in range(low_bound, high_bound ):
        # 选取统计中所对应的区间
        statistics1 = statistics[values >= float(n)*tick_width]
        stat_in_range  = statistics1[values <= float(n + band_width - 1) * tick_width]
        # 计算区间中的赢输比。输数加一，避免除以零。
        ratio = float(sum(stat_in_range['win'])) / float(sum(stat_in_range['lose'])+1)
        # 计算区间中数据量
        range_data = float(sum(stat_in_range['win']) + sum(stat_in_range['lose']) + sum(stat_in_range['even']))
        # 如果区间数据量除以总数据量大于最低数据比
        if range_data / num_data >= least_percentage:
            # 记录区间的最低值，最高值，和区间内的赢输比
            mydata.append({'low': float(n) * tick_width, 'high': float(n+band_width) * tick_width, 'ratio': ratio})
    # 制作DataFrame
    data_table = pd.DataFrame(mydata)
    # 按照赢输比排序
    sorted_table = data_table.sort('ratio', ascending = False)
    # 返回第一行
return(sorted_table.iloc[0])
```

对于上面的例子，tick_width = 0.01，再设 least_percentage = 0.03，以及 band_width = 4，计算

```
find_best_region(stats,0.01, 0.03, 4)
```

得到区间[1.12,1.16]，区间内输赢比1.43。
还不过瘾?
不过瘾好啊，那么与其固定区间的宽度，我们不如把所有的宽度都计算一遍。做法就是，先固定一个最短的宽度w_0，对于所有宽度为w_0的区间进行上面的计算，然后将宽度换为w_0+1再重复一遍，然后是w_0+2，以此类推，直到达到了设定的最大宽度。最后取这个过程中算出的最大的输赢比，并取相应的区间。

具体的计算已经由上面的函数完成，现在只要再包装一个函数来迭代地以不同的宽度呼出上面的函数，得到每个长度的最佳区间，再从这里面选出胜率最好的。
这里输入的statistics，tick_width，least_percentage和之前相同，另外的两个输入是
least_width – 最短的区间宽度
most_width – 最大的区间宽度
代码

```
# 返回一Series，内容 low=区间低点，high=区间高点，ratio=区间赢输比
def find_absolute_best_region(statistics, tick_width, least_percentage, least_width, most_width):
    # 创建标注列的空DF
    columns = ['low', 'high', 'ratio']
    df = pd.DataFrame(columns = columns)
    # 对于所有在最短和最长标准之间的宽度
    for band_width in range(least_width, most_width + 1):
        # 运行上面函数，得到该宽度的最佳区间
        best_width_region = find_best_region(statistics, tick_width, least_percentage, band_width)
        # 将结果加入DF
        df = df.append(best_width_region, ignore_index = True)
    # 将赢输比从大到小排列
    sorted_table = df.sort('ratio', ascending = False)
    # 返回第一行
return(sorted_table.iloc[0])
```

还是同样的例子，这次least_percentage选0.05，最小宽度least_width=2，最大宽度most_width=30。然后

```
find_absolute_best_region(stats, 0.01, 0.05, 2, 30)
```

得到宽度为19的区间[1.12,1.31]，这之间的赢输比为1.72，这也是使用该指标时保证5%数据量的情况下的最大赢输比!

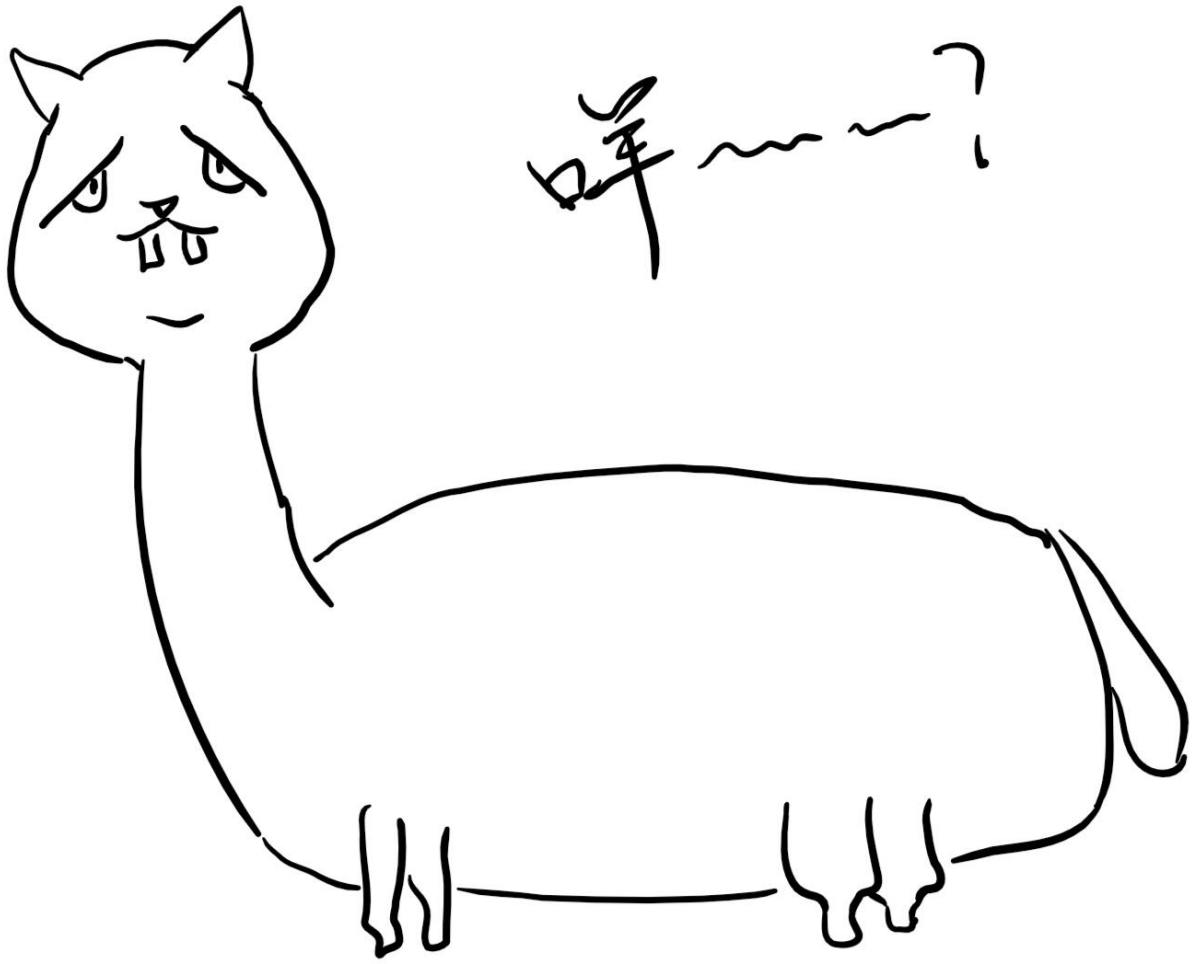
对于在策略中运用的补充

如果想把本篇统计算法的输出嵌入到交易策略中，应该注意以下几点：
不同股票要分开统计

因为每支股票的“个性”不同，所以对于不同指标的关联程度是不一样的，一支股票的最高胜率区间用在另一支股票上也许效果会截然不同，所以一定要把不同的股票分开统计。
每日更新统计数据

使用本方法的话最好把整个算法嵌入到策略中去，并且每日更新统计数据和最佳区间。如果不这么做的话，一则指标的最佳区间可能会过时，二则由于现在的最佳区间对于历史交易日来说是未来数据，所以以其进行回测会产生虚高的效果。
运用凯利公式
本篇的统计方法会计算出依照指标进行判断的成功率，利用该信息我们可以使用凯莉公式来优化投资收益。
凯莉公式点我，
具体的使用方法就留给大家当作业啦!

【量化课堂】羊驼交易系统



作者：何忆嘉年、yongpeng.r、Sunx
编辑：肖睿

本文由JoinQuant量化课堂推出。难度为入门，深度为level-0。

听说我们的神兽草泥马，哦不，羊驼，能选股？

美国“旧金山纪事报”曾做过大猩猩选股实验，让大猩猩随机挑选出5只股票。发现这个股票组合比《华尔街日报》8位知名分析师精心计算分析挑选的5只股票组合表现还好！2014年甘肃卫视的《马上知道》栏目中，节目组让羊驼随机选择一堆股票进行持仓，然后每天卖掉持有的股票中收益率最差的一只，再买入羊驼随机选择的一只。结果也得到了很可观的收益率！是不是很神奇！

稍加思考，我们可以发现其实这是一个将动量因子和遗传算法（这是啥？）相结合的策略。听上去很高深的样子，别急，我们先来看看什么是遗传算法。

物竞天择，适者生存

达尔文进化论认为，生物之间存在着生存争斗，适应者生存下来，不适者则被淘汰，这就是自然的选择。而股票市场中也存在这样一种选择，投资者就是大自然。投资者偏好高收益股票，厌恶低收益股票。于是，高收益股票生存下来，低收益股票被淘汰。羊驼算法，就是在一轮轮的竞争中，剔除表现最差的股票，保留表现优异的股票，让优良的基因得以传承。这个表现呢，就用过去一段时间的收益率（动量因子）来刻画。

羊驼嘛，就羊驼嘛

绕了这么大一个圈子，终于到了我们的主题。
羊驼策略，简单来说就是无脑选股，周期性调仓——每次剔除收益率最差的n支股票，买入随机选择的n支。为了让我们的策略是可复现的，我们选择用历史收益率最低的股票来代替随机选股，逻辑嘛：根据均值回归 的思路，我们买入前期跌幅最大的股票并期待它们的走势会反转。在一个调仓周期之后，当期反转力度最强的股票会被留下，让它继续升值；而反转力度最差的会被卖掉，换成股票池里跌幅最大的。当然，如果持有的股票不仅是持仓内跌幅最大的，并且还是全股票池跌幅最大的，那还是拿着不动吧。

所以这是一个结合动量和反转的策略，好像有点不伦不类的样子吧，但是它有收益呀有收益！为什么不用动量的方法选股呢你问。我们试了，哈哈哈哈… 在反复调参之后勉强可以跑平大盘！

策略实现

1. 设置参数，包括股票池，调仓周期 holdingPeriod (代码中为tc)，收益率计算周期 holdingPeriod (代码中为N)，每次持仓股票数目为 num_stocks ，每次换仓换股数量 change_No等。
2. 计算股票池和持仓中所有股票的上一周期的收益率:

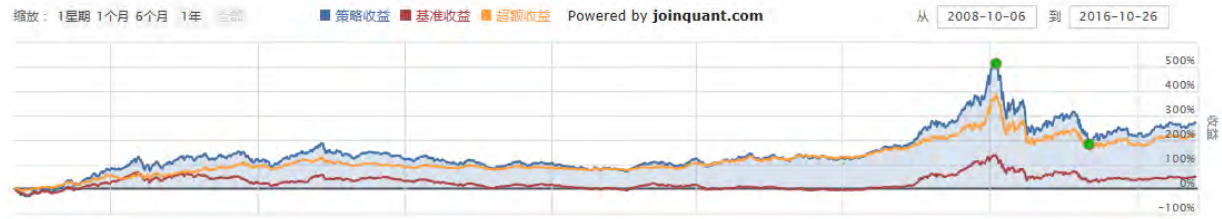
$$< br > 收益率 = \frac{昨天的收盘价 - (\text{returnPeriod} + 1)天前的收盘价}{(\text{returnPeriod} + 1)天前的收盘价} < br >$$

3. 将可行股票池内的股票按照上一周期(returnPeriod)收益率排序。将目前持仓股票按照上一周期(holdingPeriod)收益率排序。
4. 卖出当前持仓中收益率最低的 change_No 支股票，卖出股票得到的现金和原来库存现金等金额买入回测期收益排名最差的 change_No 支股票。如果持仓中的某支股票在持仓中排在收益率最差 change_No 中，同时又在所有可行股票池最差收益率排名 change_No 中，则仅调仓不换股。
5. 每 holdingPeriod 天进行 Rebalance ，调整持仓中股票到手中所有股票市值/ num_stocks 。

对应参数选择为持股20支，每次换取2只，回测期60天，调仓期60天，从2008年10月开始。股票池是沪深300。这个策略还是有比较稳定的超额收益的。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
271.03%	18.20%	49.52%	0.129	1.033	0.455	0.584	0.946	0.312	0.273	0.603	0.551	4.795	38

亏损次数	最大回撤
25	53.726%(2015-06-12,2016-01-28)



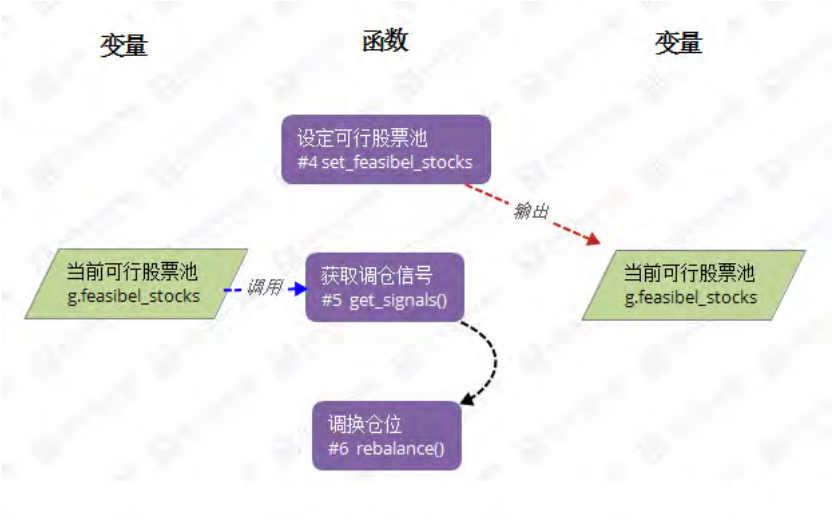
回测结果与改进空间：

1、 以回测期内收益最低购买是均值回归的思想，但是我们还可以结合各股其他指标判断来更精准地选择股票。
2、 本策略中收益走势与大盘较为一致，可以结合大盘走势分析进行择时，避免熊市带来的亏损；当然也可以卖空股指期货进行对冲。

函数和变量说明书：

函数说明书

变量说明书



v1.1，2016-11-10，修改代码，原调仓目标带入循环导致每次手中剩余部分现金
v1.0，2016-10-28，文章上线

【量化课堂】Foster Friess 积极成长策略

Foster Friess 是美国的一个成功投资家，他著名的选股策略是以股票基本面的数据来判断积极成长中的股票，是一个结合基本面和技术分析的投资思路。

作者：swlaw、石佛刘昊尊
编辑：宏观经济算命师、肖睿

</br>

Foster Friess 积极成长选股策略

福斯特•佛莱斯1940 年生于美国威斯康星州，他被CNBC 评为“20世纪最伟大的投资人之一”。1974年创立佛莱斯联合公司，1985 年发行白兰地基金(Brandywine Fund)，至2000 年止的15年间，白兰地基金平均年报酬率超出S&P500 指数达2.24%，累积报酬率达11.33 倍，且15年之中只有一年小幅亏损0.65%，基金净值成长至58亿美元，是美国家喻户晓的成长型基金之一。
福斯特•佛莱斯联合公司旗下除了白兰地基金之外，主要还有于1991 年成立的白兰地蓝筹股基金(Brandywine Blue Fund)，至2000 年底止，佛莱斯联合公司管理的资产达85 亿美元，总部设在威斯康星州的杰克森市，另在格林威尔、凤凰城、惠灵顿市等地皆有分公司。委托该公司管理资产的除了个人投资者外，还有公司户、退休基金计划、州郡政府基金、校产基金及民间基金会等，值得注意的是连著名的诺贝尔基金(Nobel Prize)也是福斯特•佛莱斯的重要客户之一
福斯特•佛莱斯从盈利增长、盈利能力、财务健康、盈利惊喜等几个方面考察公司。同时，对市盈率有一定要求，以避免选中估值过高的股票。福斯特•佛莱斯的投资策略，部分涉及主观判断，但同时也包括可数量化的部份。在投资决策方面则以市盈率水平决定是否买进一支股票。福斯特•佛莱斯认为16 倍是合理的水平，超过25 倍他就不予考虑。即使某只股票成长率达40%，他也不会用40 倍市盈率的价位去买
接下来我们介绍几个重要的财务指标。

财务指标概念：

市盈率（PE）：市盈率=每股股价（P）/每股净收益（EPS），其主要影响因素有公司所处的行业，经济周期，企业运行状况等因素。其中对于一些成长性比较好的朝阳产业来说（例如IT产业），其市盈率通常高于50，而对于一些传统行业（水泥，化工）市盈率可能就比较低了。

资产负债率（Debt Ratio ）资产负债率=总负债/总资产。根据广为流传的MM理论我们知道在考虑公司需要缴纳税款的情形下，公司负债经营能够提高公司的价值，具体分析如下：
我们先从最简单的入手，即单单只提高负债比例的前提下，会使得公司破产风险增大，因此为吸引股东投资于该股票必须提高股票收益率，在不考虑税收的前提下，公司的权益收益率 R_e 的计算公式如下:

$$R_e = R_u + (R_u - R_f)\frac{D}{E}$$

其中 R_u 代表公司没有负债时的公式权益收益率，R_f 代表公司债务收益率，我们是在理想化模型下进行推导，以市场无风险利率代表公司的债权收益率。
接下来我们计算一下公司的加权收益率 WACC:

$$\mathrm{WACC} = \frac{D}{D+E}R_f + \frac{E}{D+E}R_e$$

将上述 R_e 带入 WACC 表达式即可得到公司的加权收益率为:

$$\mathrm{WACC} = R_u$$

结果显示在不考虑税收的情况下，公司承担负债并不会增加公司的加权收益率。这既是MM理论定理一的简单说明。

下面我们考虑有税收的情形（MM理论定理二），用 T 表示税率，此时 R_e 的表达式变为下述样式：

$$R_e = R_u + (R_u - R_f)\frac{D}{E}(1-T)$$

$$\text{WACC} = \frac{D}{D+E}R_f + \frac{E}{D+E}R_e = \frac{E}{D+E}R_u + \frac{D}{D+E}R_u(1-T)$$

可以看出其与不考虑税收的情形还是有一定区别的，之所以会产生这一区别是因为企业所缴纳的利息具有一定的“避税效应”，即企业交完利息后才会计算其所得税基数，因此相同的蛋糕，由于分给政府的那一块减少了，相应属于股东和债权人的部分就增大了。这也就说明了企业的价值增大的原因。
但在考虑负债增加时，会使得公司的破产风险加大，因此我们需要在利息税盾效应与公司破产风险这两者的之间进行权衡，选择我们能够接受的一个资产负债率。

主营业务利润占比：主营业务利润占比=主营业务利润/总利润，它表现了企业的营业健康程度。主营业务利润占比越高，说明企业利润来源越稳定，受偶然某一年的利润突然增大影响较少。进而投资者承担的风险也就越低。

营业利润率：营业利润率=营业利润/总营业收入，这个很好理解，它反应了企业的盈利水平，利润率越高，这个企业赚钱能力越就越强，越值得我们投资。

Foster Friess 选股条件

市盈率<25

PE Ratio<25

资产负债率<30%

Total Liability/Total Assets < 0.3

主营业务利润占比 > 80%

Operating Profit/Total Profit > 0.8

营业利润率 > 10%

Operating Profit/Total Operating Revenue > 0.1

当然，这些财务指标的阈值并不是绝对的，我们在制定策略的时候可以使用其他的数值进行挑选，也许会获得更好的结果。

根据这一组选股条件，我们构建以下的交易策略：
设定参数tc、pe_ratio、lar、otr、prr；
每tc天调仓一次：挑选所有满足市盈率PE 小于pe_ratio，资产负债率小于lar，主营业务利润占比大于otr，营业利润率大于prr的股票，按照等权分配进行调仓。
</br>

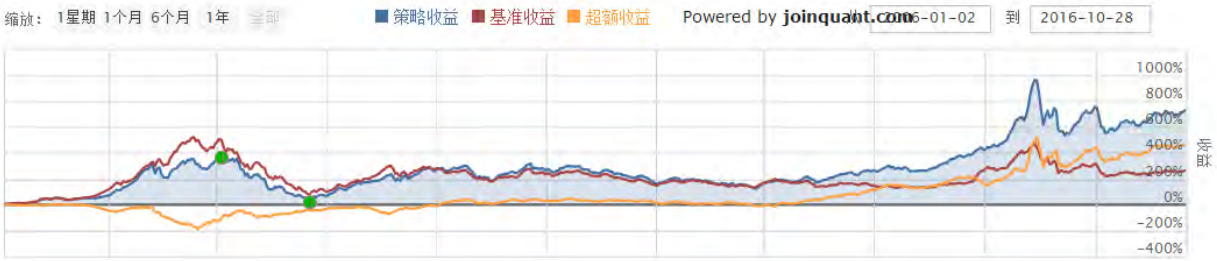
回测结果

选择调仓频率为30个交易日，PE<25，资产负债率<0.3，主营业务利润率>0.8，营业利润率>0.1。得到的回测结果如下所示：

可以看出整体收益从2006年至2016年10月28日表现还不错，但其最大回撤较大，需要承担较高的风险。下面我们将持仓频率调增为120日，PE<25，资产负债率<0.3，主营业务利润率>0.8，营业利润率>0.1。再一次回测得到以下数结果：

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率
723.57%	22.21%	261.70%	0.091	1.014	0.541	0.656	0.618	0.337	0.299	0.572

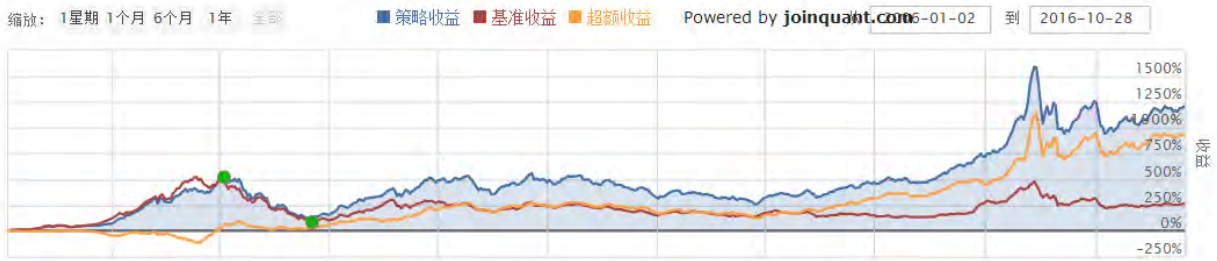
日胜率	盈亏比	盈利次数	亏损次数	最大回撤
0.553	2.283	542	406	73.212%(2008-01-15,2008-11-04)



可以看出，此策略的收益劣于刚才所述的交易频率为30天的情形，策略收益仅为30天调仓频率收益率的一半多。下面我们将交易频率提高到60天，PE<25，资产负债率<0.3，主营业务利润率>0.8，营业利润率>0.1。观察一下结果会有什么变化：

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率
1197.80%	27.61%	261.70%	0.148	0.983	0.720	0.884	0.903	0.328	0.299	0.507

日胜率	盈亏比	盈利次数	亏损次数	最大回撤
0.557	2.324	831	809	69.519%(2008-01-15,2008-11-04)

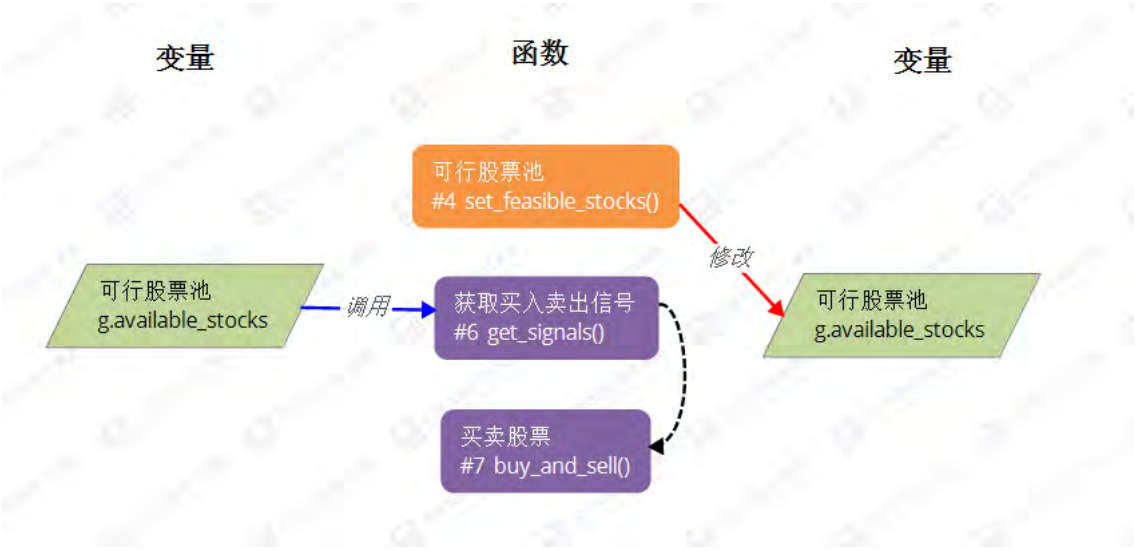


可以看出比天时稍微好一些，但仍然略逊于30天的表现。

小结

这一策略只是简单的描述了如何根据已有的指标从A股股票中筛选股票池，具体操作中一些可能对结果产生影响的因素是在调仓日需要将已停牌的股票从可行股票池中删除，以免产生不合理的股票组合。为了使得该策略取得更好的表现，可以改进的方向为：
(1)可以在调仓时对组合中每一股票的买入数量按照股票市值权重进行分配，按照多因子策略的思路来分配股票权重。
(2)构造组合时选择的股票组合筛选条件中可以加入一些更能体现股票成长性得因素，例如净利润增长率等，这可以使得策略在一定程度上能够兼顾价值性投资与成长性投资。
(在这个选股基础上，做了一个小小的进阶---底部放量选股策略，欢迎大家移步探讨：底部放量选股策略)

函数和变量说明书



文章更迭记录：
v1.0，2016-11-07，文章上线

【量化课堂】因子研究系列之一 -- 估值和资本结构因子

导语：不同因子的作用折射出每个宽客各自对于金融市场的理解，本文目的在于展现不同因子与市场表现帮助各位看官查看是否有所遗漏，同时提供因子提取和使用的简易实现代码。

作者：Sunx
编辑：肖睿

本文由JoinQuant量化课堂推出。难度为入门，深度为level-0。

本文是一系列因子研究中的第一篇文章。本系列的文章有：

1. 【量化课堂】因子研究系列之一 -- 估值和资本结构因子
2. 【量化课堂】因子研究系列之二 -- 成长因子
3. 【量化课堂】因子研究系列之三 -- 技术因子
4. 【量化课堂】因子研究系列之四 -- 市值与行业的中性化

因子简介

量化课堂推出了多份因子模型的介绍材料，包括多因子策略入门、多因子策略-APT模型、Fama-French三因子火锅、Fama-French五因子模型、CAPM+APT多因子模型等。也包括彼得·林奇的成功投资中的PEG指标，既可以看作是由因子构成的指标，也可以看作一个影响股票价格变动的因子。

每个宽客可能心中都有自己偏好的因子，这也是因子模型的一个好处，广泛可选取的因子指标能够涵盖并刻画股票的众多侧面，通过综合多方面信息可以得到一个选股结果。因子模型的另一个好处就是：“多因子模型的表现相对来说比较稳定，因为在不同的市场情况下，总有一些因子发挥作用（潘凡，2011年1月26日，安信证券《基于有效因子的多因子选股模型》）。”无论是否是高手，对于基础指标变动理解的加深肯定对于股票整个系统的认识是有帮助的。从数据而言，聚宽仅股票财务数据就包括市值数据（12项）、资产负载数据（84项）、现金流数据（57项）、利润数据（41项）和财务指标数据（34项）。

潘凡（2011）的《基于有效因子的多因子选股模型》是一份广泛验证因子有效性的材料，丁鹏《量化投资——策略与技术》中介绍多因子模型部分也援引了该材料。潘凡（2011）文中称“借鉴了包括Barra，Credit Suisse 等多因子模型并参考了国内外各种文献，选取了 30 个因子”。我们社区中@陈小米大牛基于这个材料写过一个实现策略量化选股——多因子模型，基于丁鹏的对应内容，其中结合聚宽数据构成估值、成长性和资本结构三方面的10个因子进行验证。本文同时补充Foster Friess 积极成长策略中给出没包含在前文的因子。本系列文章在充分挖掘聚宽提供的基础财务指标后，提取总计26个因子，给出基础指标说明和单因子策略分析，并在之后进行一系列更深固然的研究。本篇是次列文章中的第一篇，文中将会介绍估值因子（6项）和资本结构因子（3项）。

回测设定简介

使用上证股票池2005年至今数据，选取上一个月（21个交易日）没有停牌的股票为股票池，每个月初第一个交易日进行交易，选取对应指标最大（或最小）1%可行股票池股票总数股票进行仓位调整，股票权重为平均分配。

1账面市值比 $\mathrm{BP} = 1/\mathrm{pb\_ratio}$

账面市值比为股东权益除以股票市值，在计算中这里使用聚宽财务数据中的pb_ratio（市净率=每股市值/每股净资产）倒数计算得出，前一种方法（股东权益除以）的指标处理可以在Fama-French三因子火锅中获取。由于该指标是比值型，因此每股指标与整体指标得到的因子数值相同，较为常用的比较标准为与1进行比较。BP因子属于估值因子，即通过对于BP值的高低可以直观反映出股票估值是否符合公司价值。结合BP值可以直观表达为：BP值高于1，说明股份对应的公司权益的价值高于股价，也就是说股票的价值被低估了，如果能用市场上的股票价格把公司买下来，按斤卖了也能赚（可以对应理解相反情况）。因为公司包含了制度管理、企业文化、公司商誉、品牌价值等非物质不好度量部分，一般会导致公司的市值高于公司净资产价格。根据中证指数有限公司的数据显示，上海A股的平均BP为59%。个股的BP指标需要同时结合整个股市的BP指标进行解读，同时BP指标也受行业影响。往往在牛市时很多投资者乐于使用BP作为指标，因为这能代表股票的安全边界。

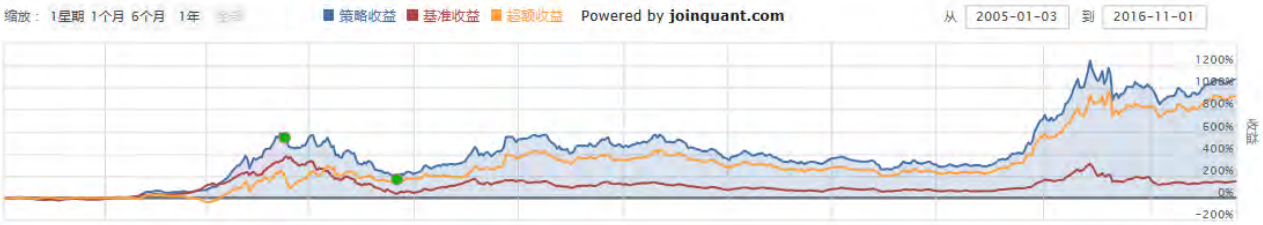
最大1%BP策略为每次选取可行股票池BP值最大1%的股票买入，每月调仓一次。而最小策略选取的BP指标刚好相反，最小的1%。最大1%因子策略中包含了均值回归的逻辑，即高BP可能表示其被低估，在后期有上涨反弹的可能。但是对比两种策略我们反而看到与预期不相符的情况，最低1%估值在同样年限

中获利比前者多将近1.9倍。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1072.99%	23.91%	151.25%	0.161	0.884	0.616	0.789	0.690	0.323	0.276	0.574	0.480	1.898	287

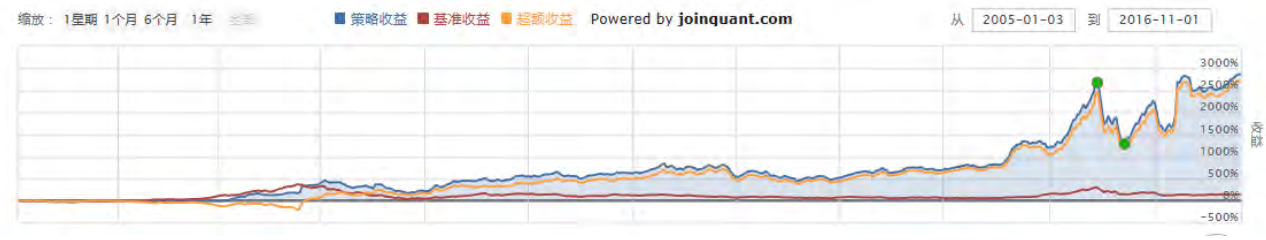
亏损次数	最大回撤
213	59.017%(2007-10-09,2008-11-03)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
2888.39%	34.42%	151.25%	0.282	0.506	0.823	1.597	0.642	0.370	0.276	0.634	0.525	2.130	255

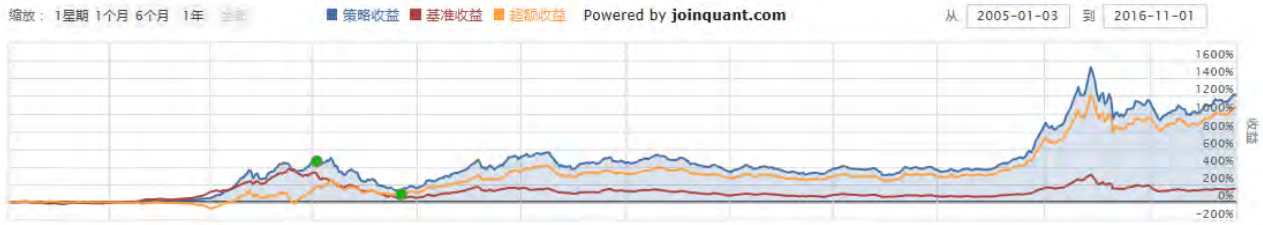
亏损次数	最大回撤
147	53.981%(2015-06-12,2015-09-15)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1207.61%	25.09%	151.25%	0.166	1.022	0.664	0.797	1.064	0.318	0.276	0.565	0.522	2.171	1441

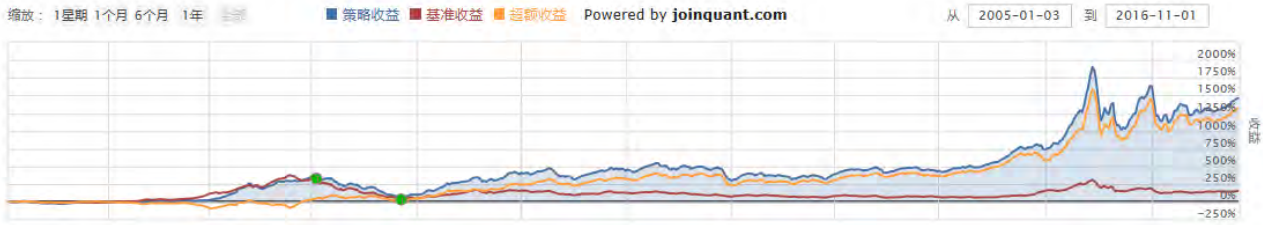
亏损次数	最大回撤
1110	65.447%(2008-01-15,2008-11-04)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1475.01%	27.13%	151.25%	0.203	0.648	0.900	1.162	0.742	0.257	0.276	0.637	0.554	2.066	1516

亏损次数	最大回撤
865	66.204%(2008-01-15,2008-11-06)



通过中证指数有限公司的网站我们看到2016年11月09日A股全市场BP（市净率倒数）确实存在者行业差别，因此可能在行业内部BP于股票收益呈现有序相关。混合在一起的所有股票中BP与回报率既不是简单的线性关系也不能很好的体现重置逻辑，因此难以简单的适用于均值（BP均值）回归的逻辑。最大1%策略可能的解释是在整体BP值较高的行业中BP值体现出一定的低估值反弹现象，但最小1%策略可能的解释是在低BP行业中（最低为软件与服务行业），行业整体发展和回报领先于市场平均水平。也有部分分析认为BP变动可以看作一个更好的指标，这可能就涉及到有效市场理论了。公司的权益资产是可以看作客官指标的，市场价格的偏离程度在什么程度上反映信息，而这样的偏离是系统性的还是随机性的在不确定市场有效性边界时两者难以分离，那么对于BP的变动到底是会出现回归还是保持趋势就难以判断。

静态市盈率 | 滚动市盈率 | 市净率 | 股息率

行业代码	行业名称	最新市净率	股票家数	其中净资产为负家数	最近一个月平均市净率	最近三个月平均市净率	最近六个月平均市净率	最近一年平均市净率
00	▼能源	1.27	75	2	1.26	1.25	1.22	1.24
01	▼原材料	2.82	489	7	2.84	2.8	2.71	2.71
02	▼工业	2.75	752	1	2.79	2.75	2.67	2.82
03	▼可选消费	3.28	481	3	3.38	3.38	3.31	3.49
04	▼主要消费	4.04	191	0	4.11	4.11	4.05	4.13
05	▼医药卫生	5.05	207	0	5.24	5.22	5.03	5.32
06	▼金融地产	1.24	199	1	1.27	1.28	1.25	1.32
07	▲信息技术	5.22	416	0	5.35	5.45	5.52	6
0701	▼软件与服务	6.91	167	0	7.11	7.39	7.66	9.04
0702	▼技术硬件与设备	4.59	223	0	4.67	4.7	4.7	4.88
0703	▼半导体产品与设备	3.31	26	0	3.47	3.58	3.64	4.03
08	▼电信业务	3.48	60	0	3.56	3.48	3.37	3.49
09	▼公用事业	1.95	89	0	2	2	1.94	2.07

指标说明：

1. 股票家数：行业内剔除暂停上市股票后剩余的股票数量；
2. 净资产为负股票家数：最新净资产为负的股票数量；
3. 最近一个月(三个月、六个月、一年)平均市净率：以最近一个月平均市净率为例，2012年2月9日公布的最近一个月平均市净率=该行业2012年1月9日至2012年2月8日时间区间内每日市净率的算术平均值，计算时只统计交易日天数。最近三个月、六个月、一年平均市净率的计算方法与此相同。

2盈利收益率 $\mathrm{EP} = 1/\mathrm{pe\_ratio}$

盈利收益率由盈利除以股票市值得到，在计算中我们使用了动态市盈率的倒数，表示的是收益在成本中的占比。由于该指标为比值性指标，因此得到的个股和整体指标应该相一致。这里成本一词可能不是很恰当，因为在理解成本时往往隐含着成本折旧的概念。更广泛的金融资产中，投资的成本可以理解为机会成本，那么EP的分母为机会成本的本金。说的过于抽象不是很好理解的话我们举一个例子来说明。100块钱如果存在银行定期存款每年能得到R元的利息，那么利率简单说年化R%。如果每股每年得到E元，每股价值相当于$\mathrm{E}/（\mathrm{R}\%）$元定期存款的作用，那么平均的利润水平回报的公司实际的市值就可以被估算粗来了。EP作为价值的一个指标，单个公司EP值大于（小于）平均EP值，就可以直观理解为该公司具有平均水平以上（以下）的盈利能力，股票价格可能被低估（高估）。当然，基于CAPM理论，中间还要考虑回报的风险变动什么的，这里都简化掉了。

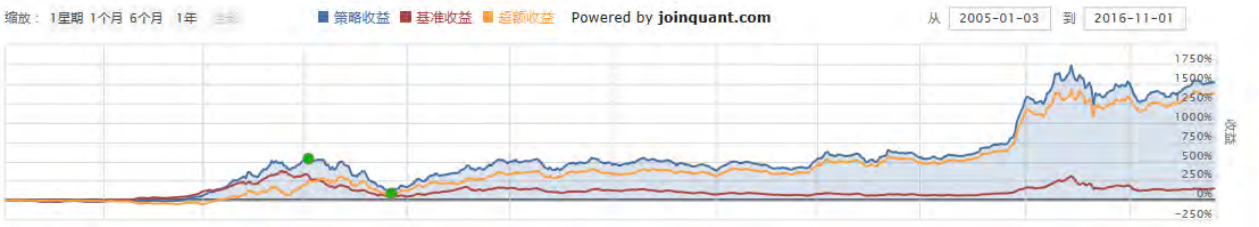
当然，这里要重点强调，框架逻辑十分粗糙。具体的EP于股价变动要是这么简单搞定，老么多人就没饭吃了。我们看一下EP指标在实际情况中的反应能发现，最大1%策略和最小1%表明了“好像什么都没表明啊”。这样的情况说明这个因子对于回报的影响不是简单线性关系，而且本文中很多指标好像都是这样。

可以分开两边理解EP指标，EP大说明相对于股票投资成比例回报较高，即单位资金可能回报较高。即使不分红的话，如果不存在极端情况个股收入都是负的比谁负的更少，那么至少有较高盈利能力的个股应该有较为稳定的基本面不至于出现大幅跌损。因为在当前的机制下，明显的估值偏离会被市场主体的获利趋势的行为纠正。同时，可能由于分红配送等利好行为推动股票价格上升，或者本身就存在巨大回报股票较低估值的抄底可能。那么这样的策略带来了还不错的收益。小EP则表明该股票的市场认可，即使单位金额带来的回报没有那么高，但是大家仍然认为它值那个价钱。这样的股票很多时候体现出在估值过程中成长性因素的综合考虑，即使当前EP值很低，但是到未来挣钱增长快会把EP值拉到正常水平甚至更低。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1522.98%	27.47%	151.25%	0.196	0.883	0.765	1.024	0.905	0.307	0.276	0.590	0.496	2.305	318

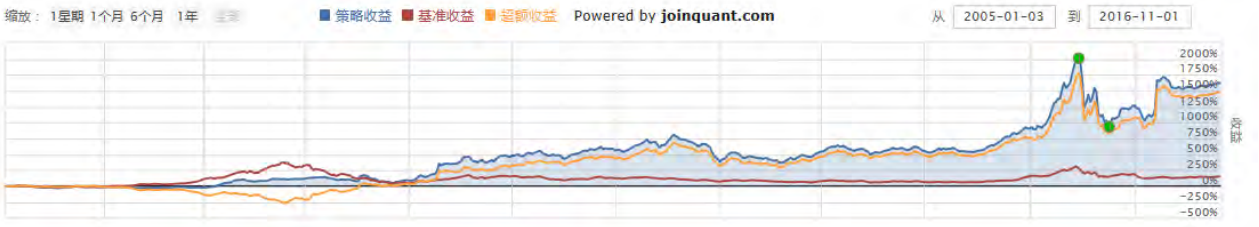
亏损次数	最大回撤
221	69.098%(2008-01-15,2008-11-06)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1626.99%	28.16%	151.25%	0.215	0.615	0.665	1.118	0.562	0.363	0.276	0.613	0.519	1.722	274

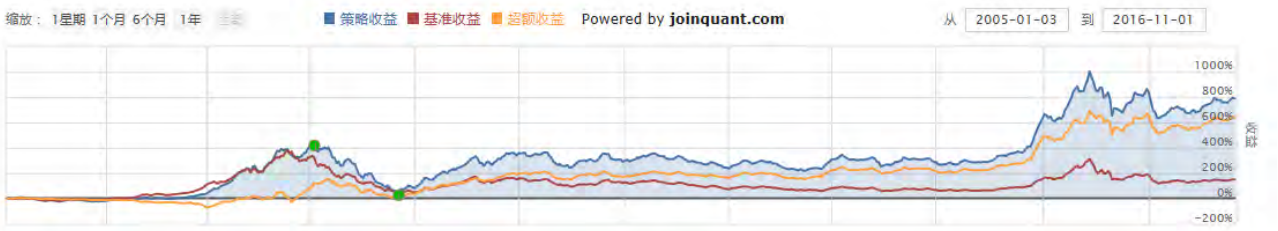
亏损次数	最大回撤
173	52.857%(2015-06-15,2015-09-30)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
791.53%	20.99%	151.25%	0.125	1.023	0.556	0.699	1.005	0.306	0.276	0.608	0.524	1.915	1618

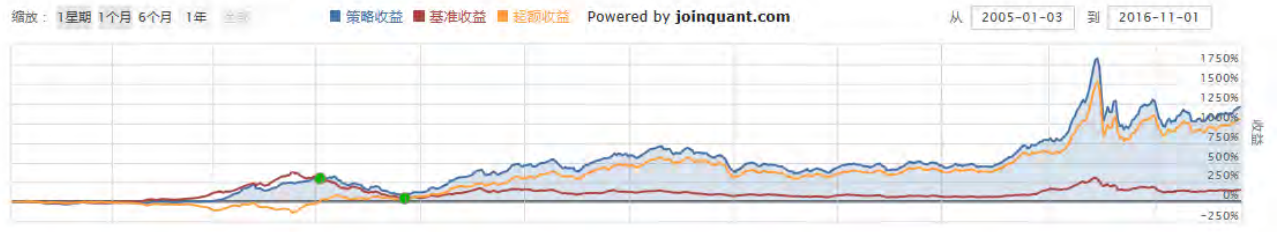
亏损次数	最大回撤
1044	74.380%(2008-01-15,2008-11-04)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1205.48%	25.07%	151.25%	0.178	0.753	0.761	0.953	0.737	0.277	0.276	0.600	0.561	1.869	1462

亏损次数	最大回撤
973	59.892%(2008-01-16,2008-11-06)



对比2008年前的一波行情，低EP高成长的股票没有体现出高EP股票的收益，而在2015年中股票行情中则正好相反。可能市场对于不同股票追捧热点的转移是一个解释，也可能在早期对于高成长股票的发现和认识不足是另一个解释。有兴趣的话，各位看官可以在社区搜索风格和板块，进行补充阅读。包括整个社会产业分工发展轮转，适应条件变迁可能行太多太多。下面同样放一个从中证指数有限公司网站上得出2016年11月09日A股全市场的动态市盈率分行业数据。EP最高的是金融地产业，超过10%，而最低的信息技术不到年化2%。同时通过变动情况我们能看出信息技术的公司EP值在一年平均，半年平均和当前最新水平下逐步增大，可能也是市场对于该行业的一个态度吧。

静态市盈率 | 滚动市盈率 | 市净率 | 股息率

行业代码	行业名称	最新滚动市盈率	股票家数	其中亏损家数	最近一个月平均滚动市盈率	最近三个月平均滚动市盈率	最近六个月平均滚动市盈率	最近一年平均滚动市盈率
00	能源	42.08	75	30	43.52	41.18	37.37	33.17
01	原材料	46.06	489	119	51.42	52.53	51.77	48.91
02	工业	30.82	752	99	31.11	30.62	29.76	30.49
03	可选消费	27.27	481	60	28.03	28.08	27.51	28.16
04	主要消费	28.34	191	33	29.21	29.5	29.49	30.14
05	医药卫生	43.98	207	9	45.77	45.73	44.35	45.77
06	金融地产	9.6	199	22	9.61	9.51	9.14	9.11
07	信息技术	56.4	416	38	59.63	61.06	61.4	63.95
08	电信业务	46.67	60	14	47.26	46.67	46.18	46.24
09	公用事业	17.29	89	10	17.51	17.02	16.06	16.43

指标说明：

1. 股票家数：行业内剔除暂停上市股票后剩余的股票数量；
2. 亏损家数：行业内过去4个季度累计净利润为负的股票数量；
3. 最近一个月（三个月、六个月）平均市盈率：以最近一个月平均市盈率为例，2012年2月9日公布的最近一个月平均市盈率=该行业2012年1月9日至2012年2月8日时间区间内每日滚动市盈率的算术平均值，计算时只统计交易日天数。最近三个月、六个月平均市盈率的计算方法与此相同。

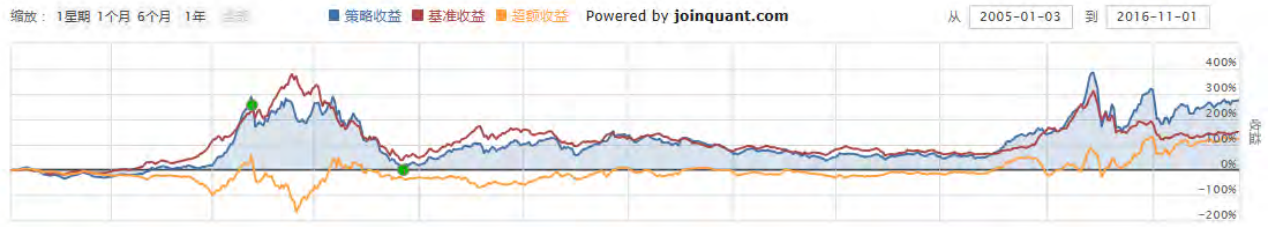
3PEG $\mathrm{PEG} = \mathrm{pe\_ratior} \cdot \frac{\mathrm{eps}t-\mathrm{eps}t-1}{\mathrm{eps}_{t-1}}$

PEG还用再说么？我们量化课堂里有啊——彼得·林奇的成功投资。还是再说两句，这个指标注重的是公司的成长性，PEG在筛选掉负的市盈率和收益增长率的公司后，与公司的成长性呈负相关。也就是说指标小的比较好，当然也不是绝对，因为计算收益增长需要较早期历史数据的原因，这个指标可能对未来不一定适用。不过可以看出排除负市盈率和收益增长率后，最小PEG明显高过较大PEG股票，但因为去掉了负值影响股票，PEG较大的股票仍然能跑赢大盘不少。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
279.30%	12.31%	151.25%	0.039	1.016	0.224	0.277	0.276	0.371	0.276	0.587	0.515	1.367	376

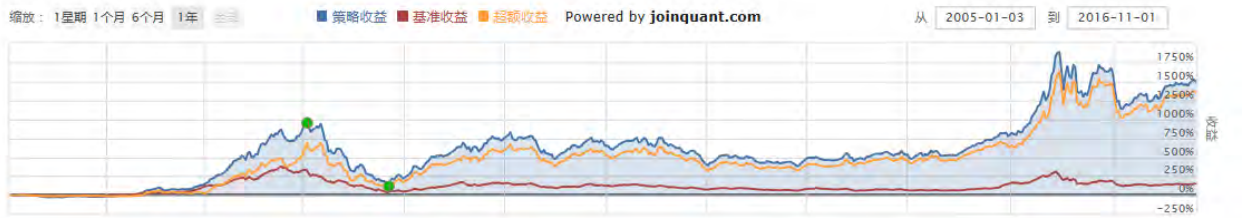
亏损次数	最大回撤
265	73.611%(2007-05-25,2008-11-06)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1506.43%	27.35%	151.25%	0.185	1.104	0.622	0.801	0.878	0.375	0.276	0.596	0.533	1.579	391

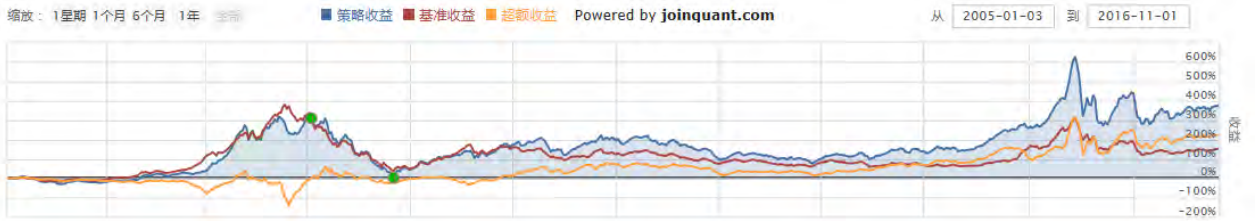
亏损次数	最大回撤
265	79.079%(2008-01-15,2008-11-04)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
376.49%	14.56%	151.25%	0.061	1.030	0.316	0.378	0.418	0.335	0.276	0.595	0.544	1.402	1879

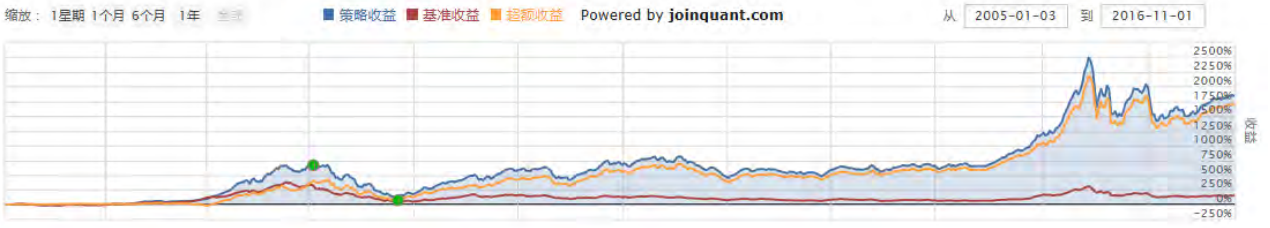
亏损次数	最大回撤
1277	73.705%(2008-01-15,2008-11-03)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1850.76%	29.52%	151.25%	0.207	1.119	0.744	0.919	1.299	0.343	0.276	0.617	0.566	1.759	1934

亏损次数	最大回撤
1203	75.892%(2008-01-15,2008-11-04)



4股息率 $\mathrm{DP} = \mathrm{dividend\_payable}/(\mathrm{market\_cap} \cdot 100000000)$

股息率是公司应付股利除以总市值。与EP指标相似表示，分子指标不同。与EP不同的是，EP表示公司客观而言盈利的能力（或对于盈利而言估值的水平），而DP择时实际的投资收益比例。如果某投资者年初持有该股票并持有一年，理论上应该等于该股票的年内变动加上股票总派息。但是，在一般财务指标中适用的是总派息除以当时市价作为投资收益的简化指标。

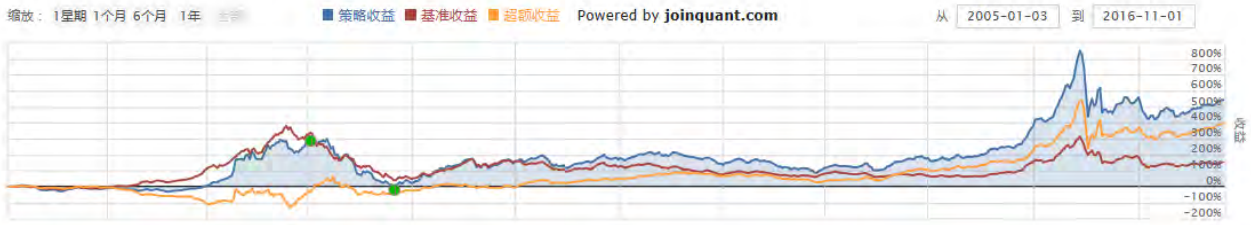
股息的多少除盈利外，还与公司的资金结构和计划安排相关。如果公司通过当期的收入调整负债比例或流动性资金比例，则可能导致相同EP公司的DP值不同。相似，如果工资对于未来计划中包含扩大再生产（投资）内容，很可能通过减少收入中的派息占比。理论上，有效市场的话EP和DP在长期可能两者关系恢复均值水平，即盈利的固定比例派发股息。总体而言，公司收入水平高可发股息的空间较大，而盈利水平低或负值还想要啥自行车啊。对于股权投资型机构而言，由于交易过程成本过高和反身性问题，可能对于股息更多偏好，这也是股息派发不能忽视的影响因素。综上，公司需要平衡发展潜力和长期资金支持，股息不能发放太少也不能发放太多。

但公司DP值与EP值面临着相似的问题，就是分母中的股票价格。对于成长性企业，由于高预期带来的高估值影响导致DP过低，尤其是具有高预期的公司可能处于成长初期，自身盈利水平有限，也会在另一方面降低DP指标。我们可以看出最大1%的策略带来了高于基准很多的收益，长期收益水平看起来还不错。尤其是经过一段市场波动洗礼后，高DP策略盈利水平逐渐稳定，可能代表了市场对其的认可增高（用脚投票带来的估值增高）。但低DP策略反映的是高估值选股策略，与低EP相似甚至同期收益率比低EP更高。也不排除低DP股票有更大的再投资安排，从而导致较好的增长情况出现。对比EP和DP可以保守的推断一下可能性，EP和DP同时偏低的股票可能反映的是高估值股票，高估值可能是增长预期推动，具有较好的收益率。但高DP不如高EP一样反映公司更多的盈利能力。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
542.03%	17.58%	151.25%	0.088	1.094	0.357	0.464	0.493	0.380	0.276	0.594	0.515	1.704	337

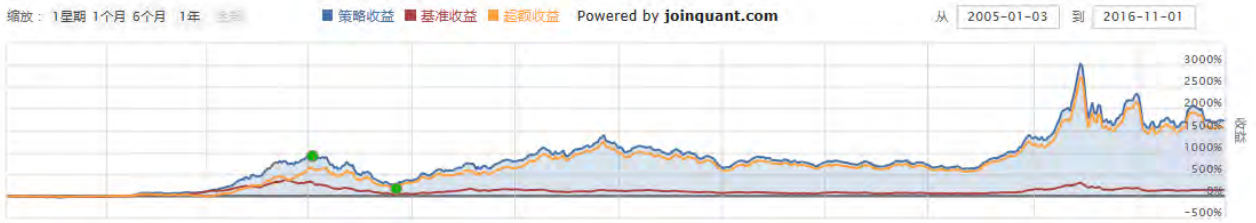
亏损次数	最大回撤
230	76.531%(2008-01-07,2008-10-29)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1732.25%	28.82%	151.25%	0.204	1.018	0.704	0.903	0.925	0.352	0.276	0.583	0.529	1.450	360

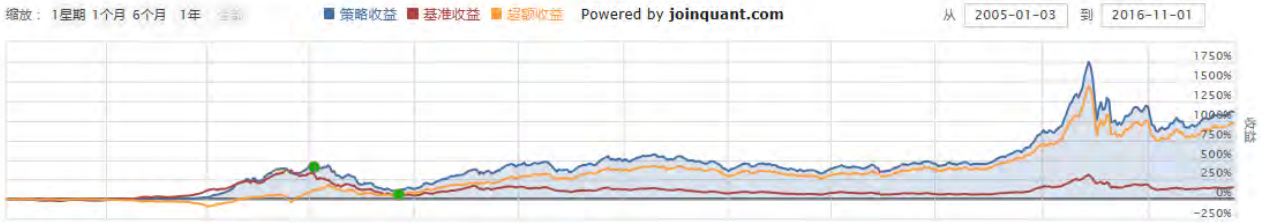
亏损次数	最大回撤
258	70.531%(2008-01-15,2008-11-04)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1113.53%	24.28%	151.25%	0.159	1.005	0.645	0.791	1.003	0.314	0.276	0.605	0.559	1.831	1698

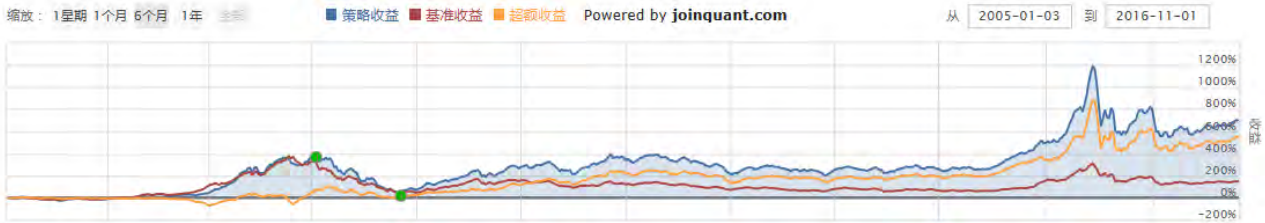
亏损次数	最大回撤
1108	66.904%(2008-01-15,2008-11-04)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
702.60%	19.88%	151.25%	0.114	1.040	0.487	0.590	0.750	0.326	0.276	0.591	0.561	1.612	1634

亏损次数	最大回撤
1131	72.941%(2008-01-15,2008-11-04)



同样补充从中证指数有限公司网站上得出的2016年11月09日A股全市场股息率。可以看出股息率最高的时金融地产和公共事业行业，如果不算当期的股票价格变动，年化收益率与一年期定期存款相似。

静态市盈率 | 滚动市盈率 | 市净率 | 股息率

行业代码	行业名称	最新股息率	股票家数	未分红家数	上月股息率	最近三个月平均股息率	最近六个月平均股息率	最近一年平均股息率
00	能源	.99	75	40	1	1.17	1.59	1.87
01	原材料	.49	489	208	.5	.51	.55	.58
02	工业	.89	752	202	.91	.92	.87	.82
03	可选消费	1.54	481	148	1.57	1.6	1.48	1.31
04	主要消费	1.83	191	70	1.86	1.78	1.56	1.3
05	医药卫生	.89	207	53	.9	.94	.9	.75
06	金融地产	3.01	199	58	3.06	3.03	3.02	2.9
07	信息技术	.41	416	114	.42	.43	.41	.35
08	电信业务	.8	60	21	.8	.84	.76	.6
09	公用事业	3.11	89	26	3.08	3.08	2.73	2.41

指标说明：

1. 股票家数：行业内剔除暂停上市股票后剩余的股票数量；
2. 未分红股票家数：在最近12个月中未进行现金分红的股票数；
3. 最近三个月、六个月、一年平均股息率：以最近三个月平均市净率为例，2012年7月30日公布的最近一个月平均股息率=该行业2012年4月30日至2012年7月29日时间区间内每日股息率的算术平均值，计算时只统计交易日天数。最近六个月、一年平均股息率的计算方法与此相同。

5现金收益率 CFP $= 1/\text{pcf\_ratio}$

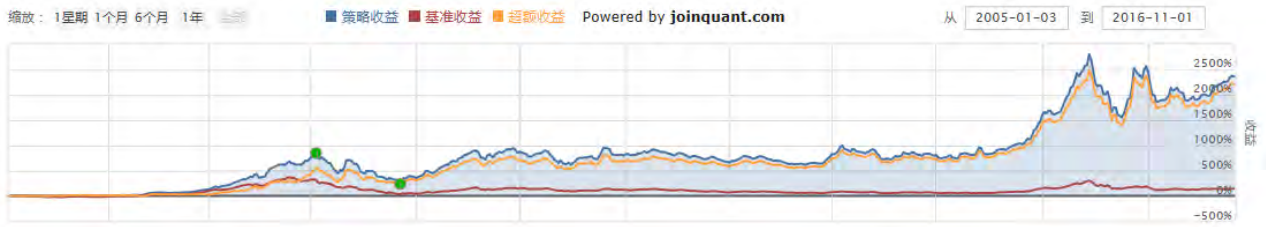
现金收益率是公司现金及等价物与股票市值的比例。据体数据处理过程中使用的是聚宽量化平台的"市现率"指标的倒数，市现率指每股市价除以过去12个月每股现金流量的倍数。这一指标成比例形式呈现，因此总量计算和每股计算得到的相同。该指标与之前提及的EP和DP有相互关联之处，同时也反映了其他方面信息。相似之处在于几个指标同时作为估值因子，是股票价格的基本面支撑，反映企业的盈利能力，同时反映市场对其估值的态度。不同之处在于CFP同时反映收入结构问题，即收入中现金和等价物的占比。举个粒子？离子？栗子？如果公司收入中有追缴的其他公司的欠账，但收回的欠账确是一堆电脑，不能简单变现。那么，公司该期核算时折旧、税费和其他支出时，不能用电脑的净值进行抵消，而需要在现金中额外准备一部分来填补差额。因此，虽然非现金部分收入在整体计算过程中导致公司账面还过得去，但实际上可能没有看起来那么美丽。因此，现金等价物的收入在所有收入中是个比较过硬的指标，同时可以经过简单处理得出现金等价物收入占公司营业收入的占比，这些进一步处理的指标对于特定情景的分析都十分有用。

在BP、EP和DP等指标中出现过的的问题也出现在这个指标中，就是并不是只有更大就更好。较低的现金流入很可能是由于公司进行了一些长期投资，在一般逻辑中久期的不确定性和非流动性问题会带来更高的平均回报，那么一期的较低的现金流可能并不能说明公司就不好。一个因子如果很漂亮，那么应该是同时在两段极值处都有很好的区分度，同时两侧的解释一致自洽。不过通过下面的回测我们可以看出，这样的因子在单独使用的情况下都有这么好的表现，我们也不好批评它了。最大1%策略中，CFP应该是收益较好的指标，毕竟人家公司现金流入占市值比例高，说明估值可能不算太高，盈利有保障，同时盈利中真金白银比较多。但我们也看到最小1%策略中，收入也不错，这就可能源自上述分析中的长期投资回报问题。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
2371.29%	32.22%	151.25%	0.237	1.044	0.820	1.092	1.166	0.344	0.276	0.586	0.504	1.910	346

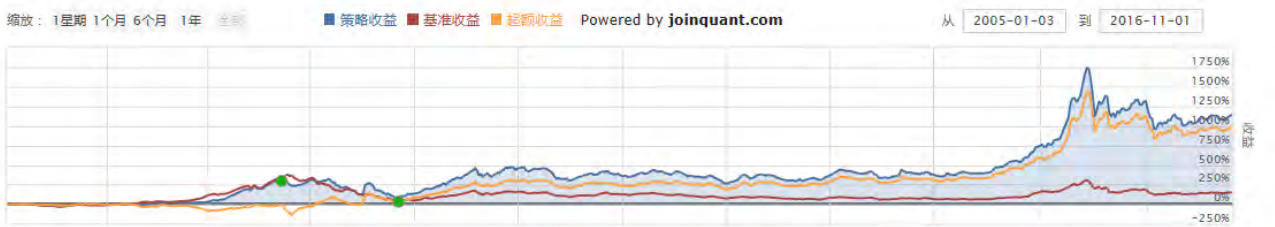
亏损次数	最大回撤
244	63.413%(2008-01-15,2008-11-04)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1142.75%	24.54%	151.25%	0.165	0.931	0.591	0.843	0.682	0.348	0.276	0.603	0.505	1.916	340

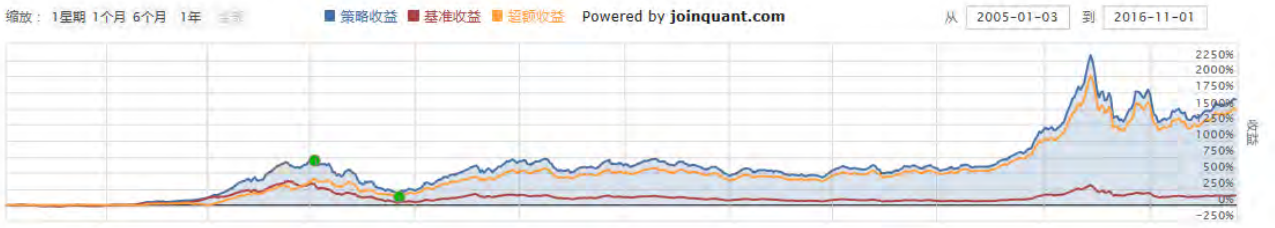
亏损次数	最大回撤
224	62.353%(2007-09-18,2008-11-04)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1645.91%	28.28%	151.25%	0.197	1.053	0.770	0.964	1.454	0.315	0.276	0.602	0.567	1.819	1752

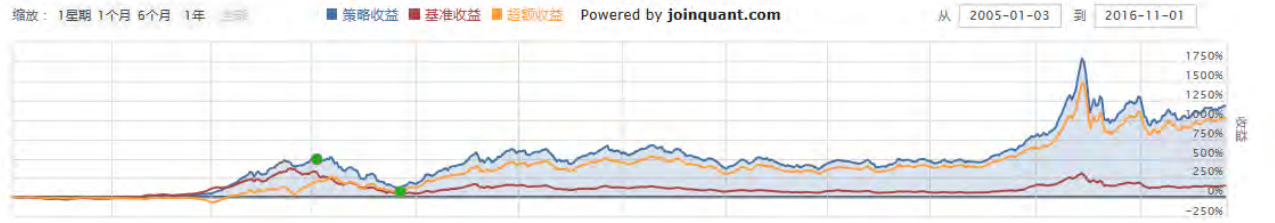
亏损次数	最大回撤
1156	69.362%(2008-01-15,2008-11-04)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1182.67%	24.88%	151.25%	0.164	1.038	0.644	0.788	1.024	0.324	0.276	0.581	0.559	1.718	1714

亏损次数	最大回撤
1237	68.956%(2008-01-15,2008-11-04)



6市销率 $\mathrm{PS} = \mathrm{ps\_ratio}$

市销率是指股票价格与每股销售收入之比，市销率越小，通常被认为投资价值越高。上述指标都是根据潘凡的材料中表述构建的，同样的BP指标也经常用作PB指标，一个指标的不同表达方式而已，可能一些同志更习惯其中的某一种形式。潘凡材料中前五个因子都是用的某指标比市价的表达方式，这个就转化为了市价对指标比例构建因子。当然，真正的高手对于这些因子及其影响应该从熟悉转到内化，无论正反都不加思索凭直觉给出影响结论。

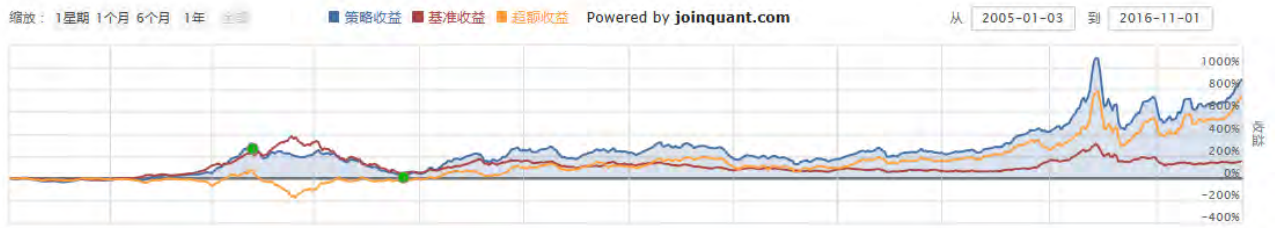
这里应该指出，对于EP、BP和CFP等指标而言，一些处于成长期的公司或者业务拓展期的公司，其实市场能力很强，但可能短期没有实现盈利，或盈利水平较低，那么前几个指标虽然不太好看，但是可以预期这样的企业在未来的发展不错。对于这些企业来说，PS指标可以在一定程度上反应成长状况，毕竟，产品卖的好，反正公司不应该太差。那么作为估值指标尤其是可以把前几个指标没表现出来的信息提供出来，这就是PS的作用。但是这个指标的缺陷在于，销售不包含成本问题，就是看着热闹不盈利的情况，比如说大量的补贴带来的销售；此外也有行业差异性存在，不同行业比较难以得出有意义的结论。

通过最小1%策略回测我们可以看出，较小的PS比值可能筛选出具有较好发展（较大销售额度）的公司，所以最终表现的还不错。相对最大1%结果也跑的比大盘好很多，得到的回报率尤其是在近年表现抢眼，在最近大势不是太理想的情况下超额收益率仍然逐步放大。那么问题来了，可能确实存在行业差别导致PS在跨行业的比较中并不是有效。可能，也就是可能，公司的各项能力没有实现销售转化，所以即使PS很大仍有不俗表现。此外预期和市场偏好推动估值过高带来了反向影响，导致虽然销售不低，但是PS更高。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
906.02%	22.27%	151.25%	0.153	0.692	0.608	0.807	0.518	0.300	0.276	0.629	0.526	1.877	292

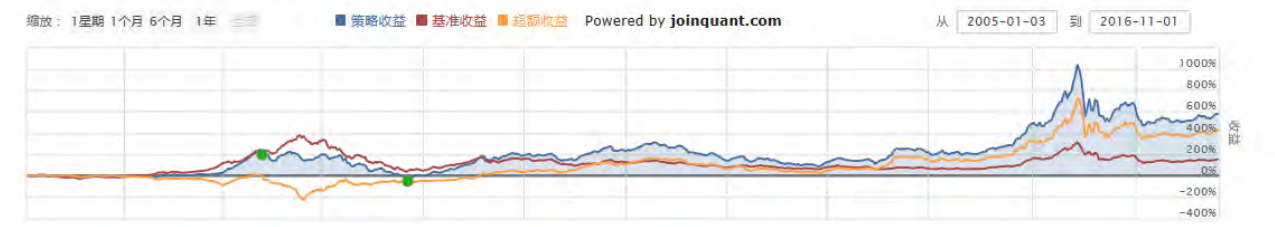
亏损次数	最大回撤
172	70.606%(2007-05-29,2008-11-04)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
575.03%	18.09%	151.25%	0.093	1.101	0.379	0.472	0.543	0.372	0.276	0.566	0.521	1.738	290

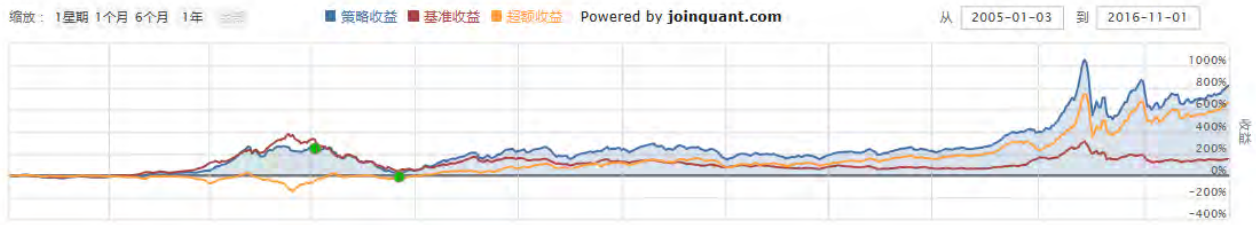
亏损次数	最大回撤
222	79.739%(2007-05-29,2008-11-04)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
828.49%	21.41%	151.25%	0.137	0.844	0.613	0.746	0.690	0.284	0.276	0.626	0.567	1.896	1540

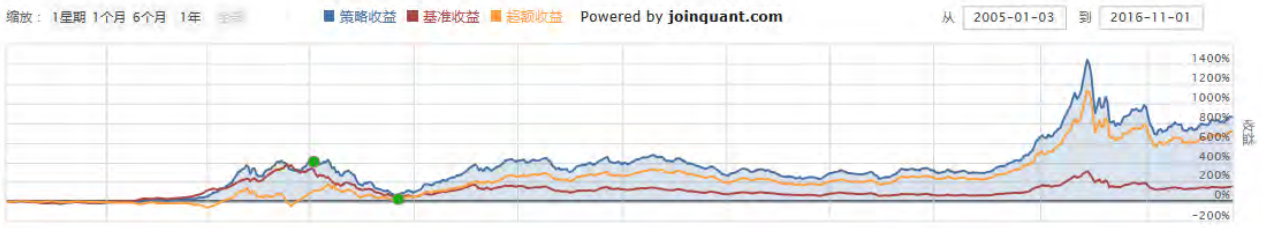
亏损次数	最大回撤
920	70.257%(2008-01-15,2008-11-04)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
863.34%	21.80%	151.25%	0.132	1.068	0.540	0.652	0.894	0.330	0.276	0.566	0.542	1.854	1419

亏损次数	最大回撤
1088	72.958%(2008-01-15,2008-11-04)



7资产负债率 $\mathrm{ARL} = \mathrm{total\_liability}/\mathrm{total\_assets}$

资产负债率指总资产除以总负债，结合我们聚宽的财务指标分别使用“负债合计”和“资产总计”，这一指标反映了总资产中有多大比例是通过借债来筹资的，也可以在极端情况下分析破产清算时股东和债权人各自的损失情况。经常被提及的资不抵债讲的就是ARL大于1的情况，这里一般不考虑隐性因素影响（包括信用、能力等）。在没有爆发破产前，这个因子作为资本结构因子，往要是衡量公司风险的因子，例如某公司在进一步发债投资过程中该指标被衡量其清算能力。进一步延展，高ARL公司可能在未来发债过程中面临者更高的成本，即其再投资成本将上升，算是一个负面影响。从公司角度出发，只要融资成本低于其盈利水平，那么融资成本上升带来的时回报率变动的边际变化（即一阶导数的变化情况），回报绝对额还是上升的，因此对于股东而言回报率还是上升的，股价按理说也受正向影响。

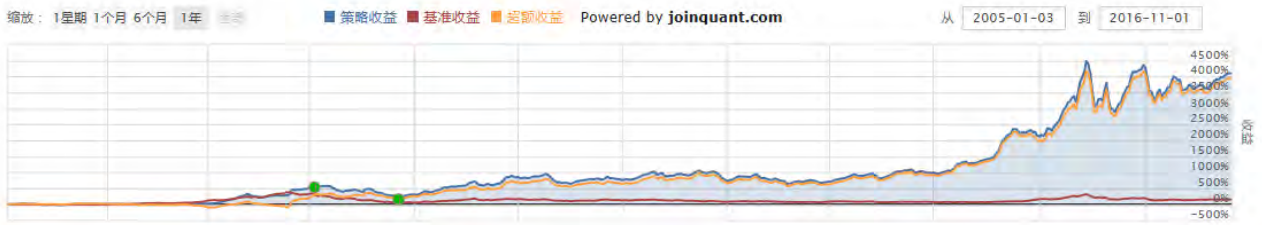
若要进一步深入分析该因子，则应该细分债务的构成情况。比如长期负债较高则公司可能再债务方面不需要进行更多频繁处理，短期负债较多则不利企业保持稳定的经济环境。其次，应该注重负债比例的变动，一方面扩大生产增大再投资过程中债务比例可能会上升，另一方面自由资产减值也会造成债务比例上升。对于不同情况应该有不同的其他因子组合进入策略，以使得选股策略能够真正进行有效识别。

有效识别在潘凡的材料中也有提及，能稳定的跑赢或跑输大盘。如果进一步处理，希望能分清大盘在好的时候和坏的时候分别或同时稳定跑赢（或跑输）大盘。如上述大部分因子一样，我们能看到ALR因子无论最大或最小策略胜率都不低，比较的问题从好坏变成好于更好的区别。ALR最大1%的策略市现率夸张的11年间年化38.5%的收益水平。最小1%ALR的收益其实也还不错了，那么两段收益的逻辑是什么？正常的公司应该保持一定比例负债，一个是为了资金流动的考虑，一个是防止占用大量自由资金的目的，其实两个是一个意思，所谓一体两面。负债太小，如果不是别人不借钱给它，那就很可能被认为开拓不足，没有进一步发展的计划安排。负债太大清算风险问题，另一个就是在不稳定情况下应急举债也受到限制，第三是没有合理控制债务会被视为收支平衡间出现问题。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
4116.24%	38.51%	151.25%	0.322	0.528	1.063	1.843	0.804	0.325	0.276	0.638	0.530	2.930	278

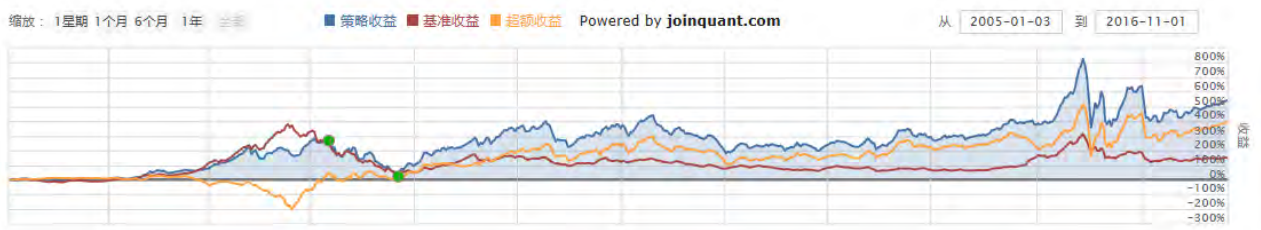
亏损次数	最大回撤
158	55.750%(2008-01-15,2008-11-06)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
551.48%	17.73%	151.25%	0.091	1.061	0.362	0.452	0.486	0.379	0.276	0.612	0.520	1.619	339

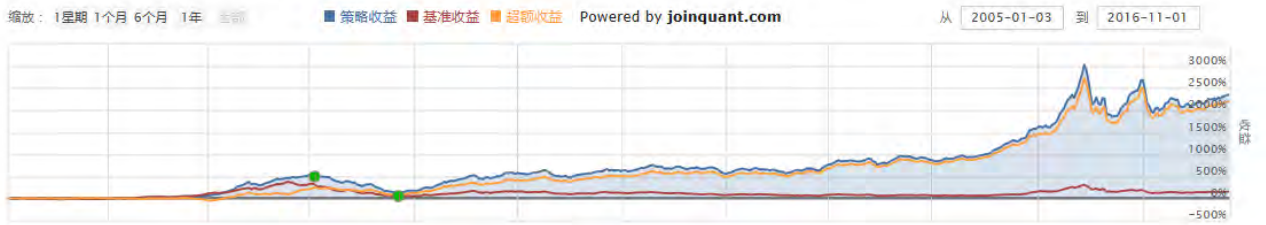
亏损次数	最大回撤
215	66.723%(2008-03-05,2008-11-04)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
2364.37%	32.19%	151.25%	0.250	0.729	1.159	1.531	1.219	0.243	0.276	0.663	0.558	2.581	1527

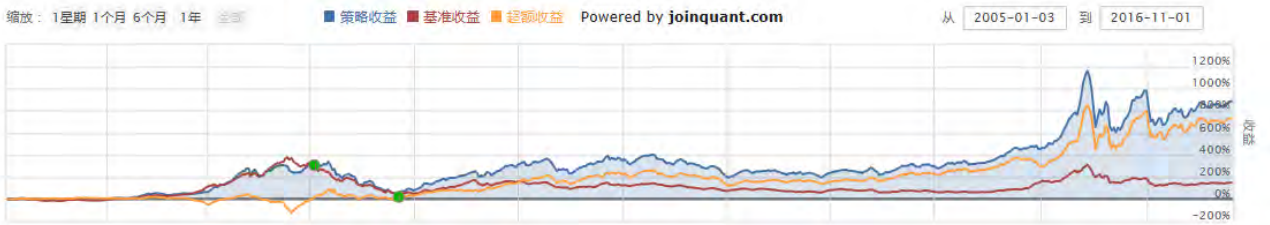
亏损次数	最大回撤
775	68.620%(2008-01-14,2008-11-06)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
898.30%	22.18%	151.25%	0.137	1.033	0.545	0.657	0.793	0.334	0.276	0.637	0.557	2.155	1612

亏损次数	最大回撤
919	68.654%(2008-01-15,2008-11-04)



不贴图了，凤凰财经给出了稍早一点的中国A股负债讨论的文章，虽然中石油绝对量超10亿的负债夺魁，ALR也才43.8%。对应高ALR非st*最高将近*100%，*而*st中有些高的惊人的ALR超300%。上边两个图仅使用上证股票做回测，与凤凰的池子有些差别。

8固定资产比例 $\mathrm{FACR} = \mathrm{fixed\_assets}/\mathrm{total\_assets}$

固定资产占比给出的是公司固定资产比上总资产的比例。资产总计是指企业拥有或可控制的能以货币计量的经济资源，而固定资产是指企业为生产商品、提供劳务、出租或经营管理而持有的，使用寿命超过一个会计年度的有形资产。固定资产比例是个老生常谈的问题，要是追溯起来，那么从古典前期还称不上经济学流派的经济学思想年代对于这个问题就开始讨论了，但资本和资产的影响并没有有效区分。

回到正题，当前的研究中关于固定资本对于企业影响主要从资金周转的角度讨论，当然这个视角从马克思（或更早）的年代就开始了。如果不考虑固定资本仅把资本定义为生产材料，那么周转一圈赚一圈的钱。那么长期而言固定资本比例过高影响了平均的资本周转速度，可就影响人家赚钱了。比如说百度百科上固定资产比率的表述。举个例子，如果企业的资本为A，非固定资本的平均收益率为R，并且由于固定资本的流动性低，所以产生的收益仅为非固定资本的一半，为 $0.5R$（注意这个 0.5 只是一个例子，实际系数根据企业运营模式而变）。那么企业当期的回报为 $A \cdot FACR \cdot 0.5 \cdot R + A \cdot （1 - FACR） \cdot R$，其中前一半为固定资本带来的回报，后一半为可变资本带来的回报。可以计算回报率为 $（1 - 0.5 \cdot FACR） \cdot R$，FACR为50%的企业回报率为 $0.75 \cdot R$，而FACR为80%的企业回报率为 $0.6 \cdot R$，可以看出FACR 更低的情况下回报率更高。

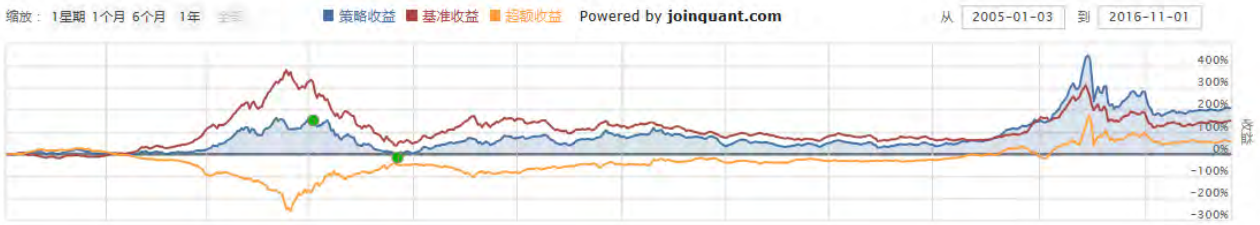
又提到异质性问题了，不同行业产业的不同情况。一个产业的附加值更多产生于固定资本还是非固定资本，或者说固定资本和非固定资本的回报率哪个更高？不同的固定资本和非固定资本的结合构成对平均收益率R是否有影响？懵了还是看数据结果吧，数据结果是平均水平上的一个结论展示。

最大1%FACR的回测结果是仅仅跑过大盘。我们可以看到较早的大部分时候其收益都不算太好，但2015年中之后波动较大的情况下反而体现出一些优良的性质。在理论中，较大的FACR不利于资金流转盈利。从相反的方面讲，固定资本占比大也是当前资金使用充分的一个表现，同时较大的固定资本在贷款申请等方面也会产生帮助，因此可能对于抵御波动行情有些许帮助。对应最小1%策略回测中在不同的大盘行情下，其都体现出稳定的超额收益率增长，这与一般的理论预期是相符的。相对于固定资本占用大量资金的企业，轻量型企业具有更好的平均收益水平。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
208.37%	10.30%	151.25%	0.021	0.964	0.195	0.240	0.176	0.324	0.276	0.576	0.505	1.640	293

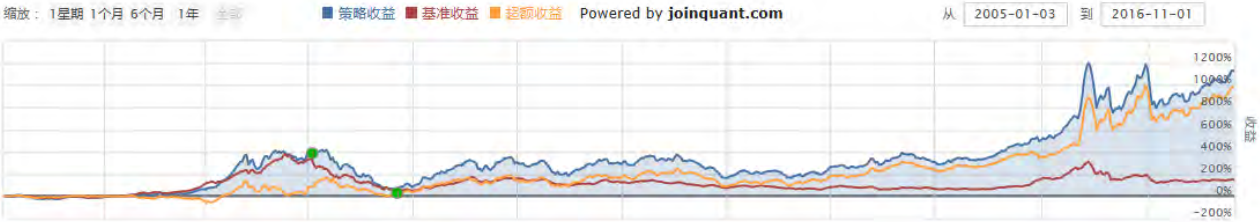
亏损次数	最大回撤
216	66.034%(2008-01-14,2008-11-06)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1135.12%	24.47%	151.25%	0.159	1.060	0.535	0.695	0.703	0.383	0.276	0.585	0.520	2.014	300

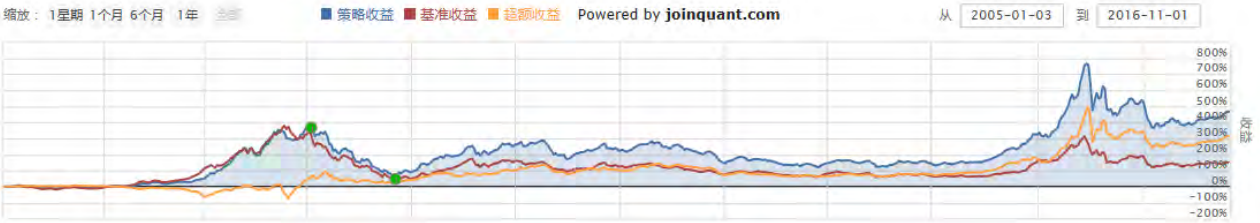
亏损次数	最大回撤
213	73.955%(2008-01-14,2008-11-06)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
467.95%	16.33%	151.25%	0.081	0.967	0.412	0.502	0.575	0.299	0.276	0.595	0.543	1.741	1448

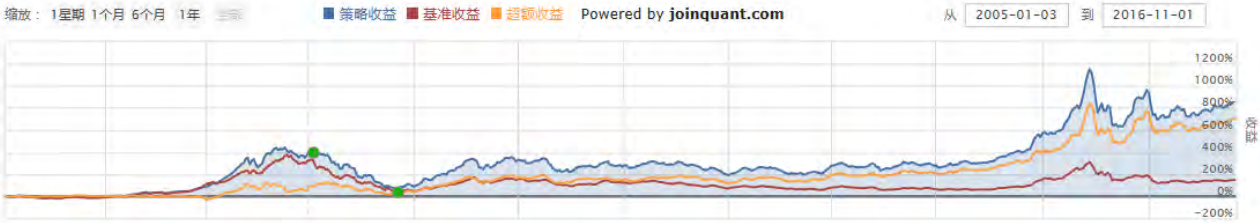
亏损次数	最大回撤
986	68.624%(2008-01-15,2008-11-04)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
857.24%	21.74%	151.25%	0.131	1.069	0.543	0.695	0.926	0.327	0.276	0.612	0.541	1.899	1520

亏损次数	最大回撤
962	72.011%(2008-01-14,2008-11-04)



9流通市值 CMC = circulating_market_cap

流通市值（亿元）指在某特定时间内当时可交易的流通股股数乘以当时股价得出的流通股票总价值。实际上上市公司的股份结构中不止有流动股，其中一部分是不能流通的，这就涉及到一些历史成因的问题。当年的逻辑就是要上市流通，盘活周转，但是不能国有资产流失啊，中间就产生了一系列的奇奇怪怪的乱入。有段子说，不知真假，有人当年打算薅羊毛，结果个人囤了很多国有股然后烂在手里不能流通买卖。

一般而言，如果流通股市值较大，那么较小的资金量对其操作带来的反身性就很小，也就是说交易行为和股票价格不会互相产生太大的影响。对于投资者来说，较小的反身性带来了正负两方面影响：因为不易炒作，所以很难出现暴涨机会，因此收益不会太高；反之，当股票被大量空单冲击时，造成暴跌的机会更少，便于控制损失。那么较小流通市值的股票就体现了对立的属性。对于量化投资而言，反身性是比较讨厌的东西，因为会造成较大的冲击成本，导致实际交易和回测结果有很大偏差。

最近我们的社区中有很多讨论小市值的策略研究，比较学院派的内容可以参看Fama-French三因子火锅和Fama-French五因子模型。在这两个模型中，可以把小市值解释为：小规模的公司面临更大的风险，因此它的股票理应有更高的风险补偿。

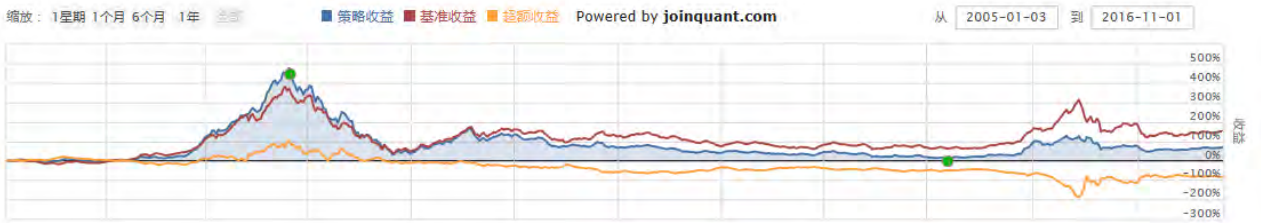
对比CMC最大1%和最小1%可以看出，这个指标相对与前边诸多个，表现出出奇的优良特性，也就是它和超额收益有较好的线性关系，大市值和小市值的超额收益基本是相反的。通过对比其实展现的内容，每个人可能解读有所不同。在我看来，2008年前的一波行情可能真的是波大行情，连大盘股都能带动领涨。但是后来，小市值逐渐显露出特别猛的尽头。一个原因，后来逐渐追捧高成长预期，推动估值增高，这样小盘股这样情况下逐渐被挖掘出来。另一个就

是炒作和情绪化的产物，依稀记得有些行情中价格较低的一个个被抬起来，其实很多都没有太好的基本面支撑。小盘股的灵活性被充分调用，造就了小盘股的神话。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
67.58%	4.60%	151.25%	-0.035	0.952	0.020	0.026	-0.167	0.306	0.276	0.541	0.466	1.174	271

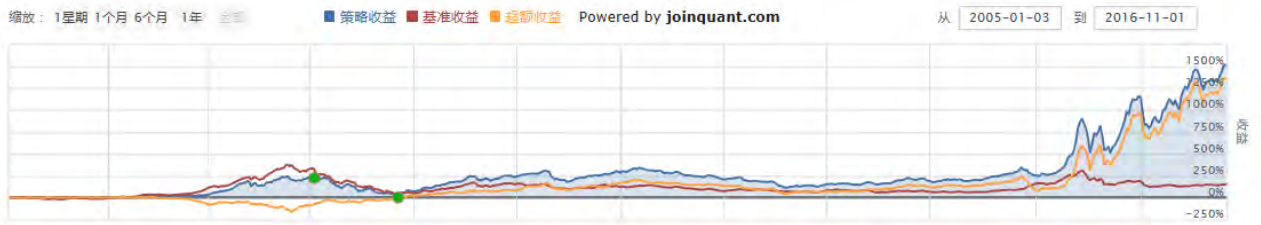
亏损次数	最大回撤
230	80.712%(2007-10-31,2014-03-12)



最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1519.15%	27.44%	151.25%	0.194	0.928	0.664	0.833	0.765	0.353	0.276	0.638	0.522	2.366	329

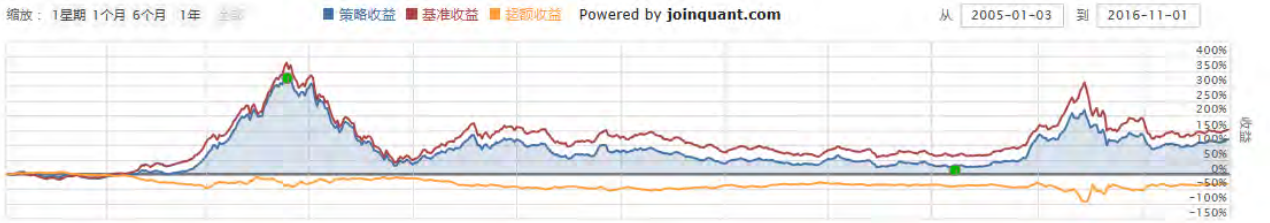
亏损次数	最大回撤
187	66.432%(2008-01-14,2008-11-04)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
118.93%	7.06%	151.25%	-0.013	0.999	0.106	0.140	-0.100	0.288	0.276	0.556	0.479	1.268	1344

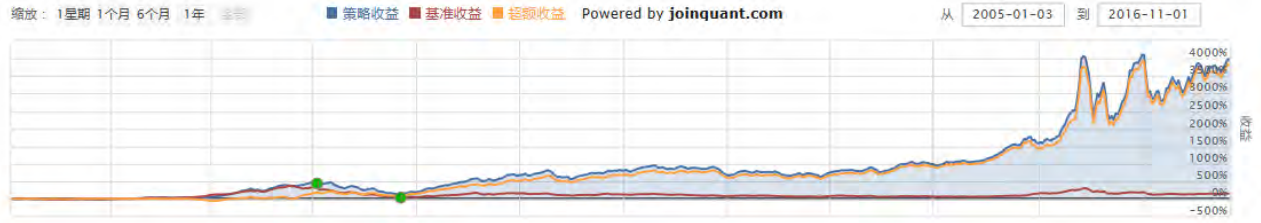
亏损次数	最大回撤
1073	73.359%(2007-10-16,2014-03-20)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
4030.18%	38.27%	151.25%	0.304	0.890	1.096	1.331	1.299	0.313	0.276	0.666	0.575	2.797	1583

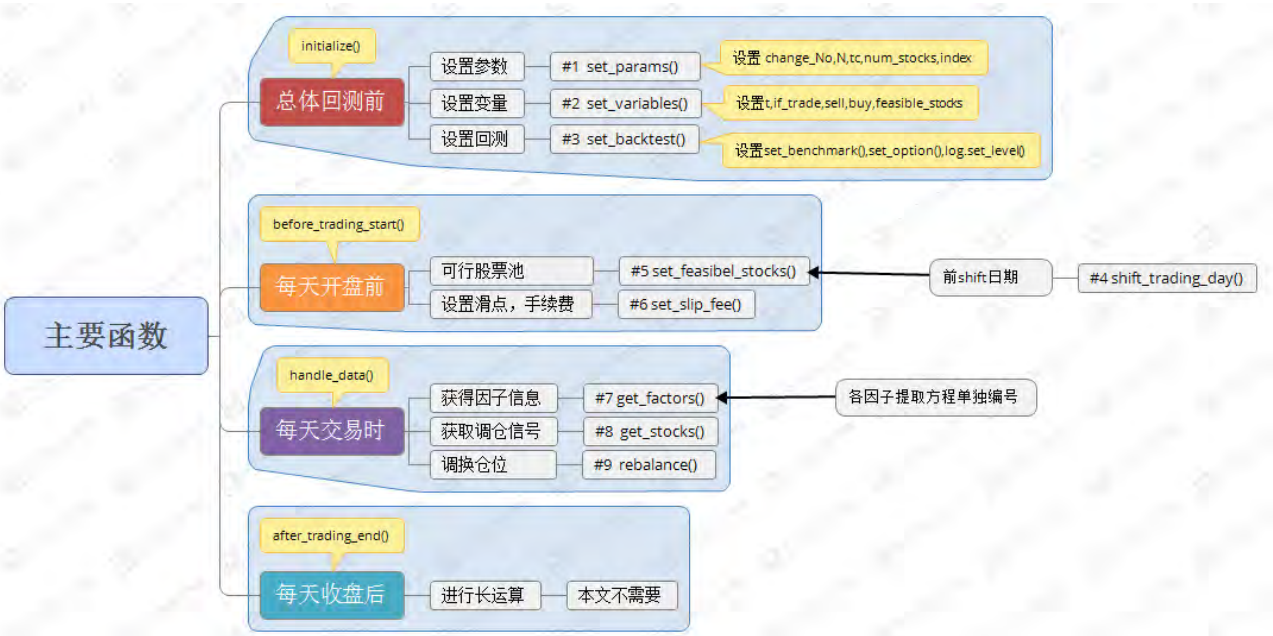
亏损次数	最大回撤
795	65.380%(2008-01-15,2008-11-04)

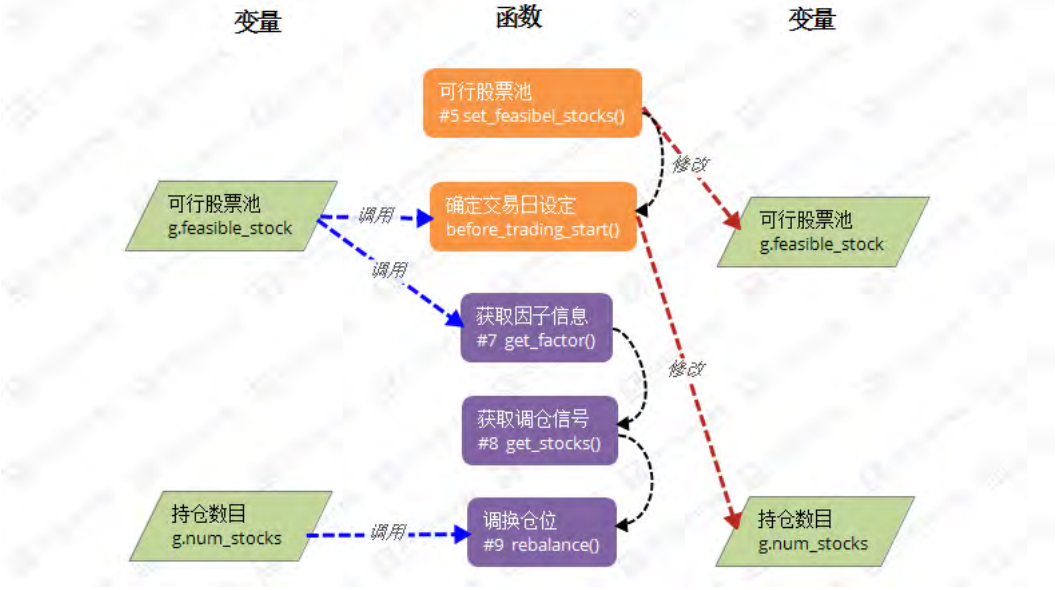


一个小结

可以看出，很多财务指标在对于收益率影响方式上都不是线性的，至少全部行业混在一起是不是简单的线性关系，这需要在具体策略实际构建时进一步展开分析。本系列的下一篇中我们会展示十个成长因子：ROE、ROE变动、ROA、ROA变动、EPS增长率、主营业务增长率、主营业务利润占比、营业利润比、收入净利率和收入净利率变动，敬请期待。

函数和变量说明书





v1.0，2016-11-15，文章上线
v1.1，2016-11-15，添加5%对比结果
v1.2，2016-11-30，补充函数和变量说明书，调整代码中函数编号

【量化课堂】因子研究系列之二 -- 成长因子

导语：不同因子的作用折射出每个宽客各自对于金融市场的理解，本文目的在于展现不同因子与市场表现帮助各位看官查看是否有所遗漏，同时提供因子提取和使用的简易实现代码。

作者：Sunx
编辑：肖睿

本文是一系列因子研究中的第二篇文章。本系列的文章有：

1. 【量化课堂】因子研究系列之一 -- 估值和资本结构因子
2. 【量化课堂】因子研究系列之二 -- 成长因子
3. 【量化课堂】因子研究系列之三 -- 技术因子
4. 【量化课堂】因子研究系列之四 -- 市值与行业的中性化

写在前边的话：

本系列文章的上一篇介绍了估值因子和资本结构因子，本篇则要介绍成长因子和财务指标因子。因子的分类的益处在于，组间的相同特性可以增进对同类因子的整体理解，对于组间的差异可以增进对于不同因子作用差别的异质性理解。

在因子的计算方法和逻辑之外，文中还将展示因子最大的和最小的 1% 和 5% 的股票的回测结果。一般来讲 1% 百分位属于极端样本，它的回测包含了一些有用的信息，但不具一般性的参考意义，更能表现因子效果的还是5%百分位的回测。因为部分指标生成需要上一期数据，本文回测从2005年6月开始。

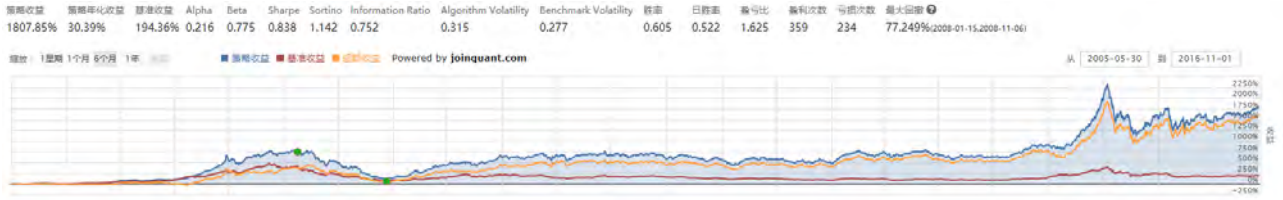
10净资产收益率 $\mathrm{ROE} = \mathrm{roe}$

ROE (Rate of Return on Common Stockholders'Equity) 中文名称为净资产收益率，也被称为净值报酬率/权益报酬率/股东权益报酬率/净资产利润率或股权利润率，该指标反映了股东权益的收益水平。据说ROE的思路出自杜邦公式（DuPont formula），在分析时针对同行业更加好用，一般认为 15-20% 的 ROE 水平就很不错了。看名称与【量化课堂】因子研究系列之一 -- 估值和资本结构因子 中的很多概念很像，例如 EP。两者中 EP 作为估值因子原因在于它表明净利润与当前股票价格（估值）的相对大小。而ROE是公司净利润与公司股东净资产之比，指标值越高说明投资带来的收益越高，体现了自有资本获得净收益的能力。需要特别指出的是，这里的净利润指的是"归属于母公司股东的净利润"，特别剔除利润中归属于子公司的部分。 ROE 有两种计算方法：一种称为"全面摊薄净资产收益率"，使用净利润除以期末净资产；另一类是聚宽财务数据中的"加权平均净资产收益率"，等于报告期净利润除以当期平均资产（期初与期末净资产平均值）。

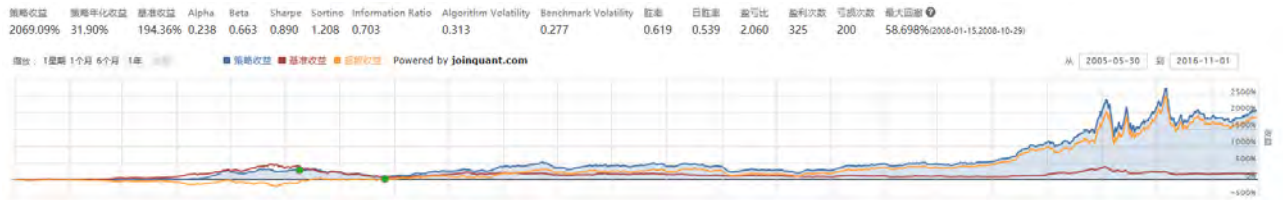
ROE 指标有几个比较明显的缺点。第一是，在没有结合债务指标的情况下，它无法反映出股票定价的合理性。举个例子来说，假设两个公司的净资产都是 100 万并且当年的收益都是 50 万，那么 ROE 同为 50 %。假设其中一个公司的经营不需要额外贷款，而另一个公司有额外贷款 100 万，那么第一个公司的盈利水平更高并且风险更低，所以股票定价应该更高，但 ROE 无法反映这个情况。第二是，当期如果公司支出过多， ROE 可能表现出较大比例负值。如上述 100 万净资产公司，当期营业收入为 50 万，但推广费用开支为 60 万， ROE 为 -10% 。因为 ROE 的指标特性， ROE 因子常与其他因子联合使用。

通过历史回测了解到，最大 5% ROE 的策略整体表现很好，尤其实在 2008 和 2015 年两拨动荡情况下虽然有获利回吐，也有大幅回撤，但是超额收益绝大部分时间为正。通过观测回测中持仓情况我们可以看到，虽然我们进行了每月调仓，但是实际情况为持有总市值在有限的几支股票间进行调整，以使得当期各股每股市值相同。这样的现象是因为 ROE 指标变动较慢，每月调整策略和指标的季报周期有一定差异。最小 5% ROE 也表现出远超大盘的回测结果，但是在 2014 年中后期才开始发力，这样的现象可能有两个方面解释：第一、ROE 被指责的一个缺点在于不能结合市值变动反映公司当期股价，低 ROE 股票很可能由于 ROE 因子数据不理想导致股价被低估，这也是低 ROE 指标可以有较好收益率的一个合理可能；第二、当期的净收益影响该季度 ROE 指标，ROE 较低可能由于当季度支出（投资性）增加导致（例如 2016 年第三季度大智慧的 ROE 大幅减少的情况），这样的投资性行为可能被视为股价利好刺激。

最大1%



最小1%



最大5%

最小5%

通过 5% 策略，我们可以看出策略选择池数量扩大，利润水平趋于"平庸"。ROE 的相对大小具体企业相差远比行业要大，但是在长时间大量个股回测中，应该也体现了一定的平均水平。那么平均水平上，ROE 高的回测结果表现还是要好一些，银行业的 ROE 稳定且较高，电子行业 ROE 相对波动大且较低。

11 ROE变动 $\mathrm{ROEC} = \mathrm{ROE} < em > \text{上一期} - \mathrm{ROE} < /em > \text{当期}$

我们看到了 ROE 指标的不好一面，那么自然想到一个在既有的指标上构建和完善得到更多的有益信息，减少不利的影响。一个直观理解是当期的公司好不一定以后还好，那么推动公司股价的因素可以理解为公司不但现在好，而且公司处于上升通道中——公司在不断越来越好。结合 ROE 指标就有了 ROE 指标的变动（这里用 ROEC 来表示）。相似也可以构建 ROE 的增长率，但是 ROE 的增长率可能出现的情况是并不单调（好坏在数值上区分不出来）。满篇随处可见的栗子，这里再来一个，看下图，期中t表示当前期，t-1 表示上个季度财报指标，C 表示不同的六种假设情况。其中 $\mathrm{ROEC} = \mathrm{ROE} < em > \text{上一期} - \mathrm{ROE} < /em > \text{当期}$， $\mathrm{ROEG} = \frac{\mathrm{ROE} < em > \text{当期} - \mathrm{ROE} < /em > \text{上一期}}{\mathrm{ROE}_{\text{上一期}}}$。对比 C1 和 C2，看出 ROE 正值变小和负值减小显然第一个是变得不好而第二个变得更好， ROEC 更够反映出方向，但是 ROEG 不能区分。对比 C2 和 C4 两种情况，可以看出 ROEC 不能区分"负的更少"还是"正的更多"。这也是本文后部分跨期指标使用变动而非增长的原因，因为部分指标分子分母都可能出现负值，造成指标含义难以解读。

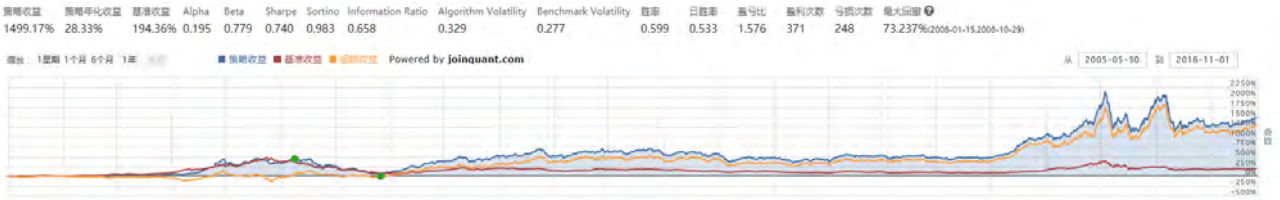
表 1 ROE 变动与 ROE 增长

	时期 t-1	时期 t	好/不好	ROEC	ROEG
C1	30%	20%	不好	10%	-33%
C2	-30%	-20%	好	-10%	-33%
C3	30%	-20%	不好	50%	-167%
C4	20%	30%	好	-10%	50%
C5	-20%	-30%	不好	10%	50%
C6	-20%	30%	好	-50%	-250%

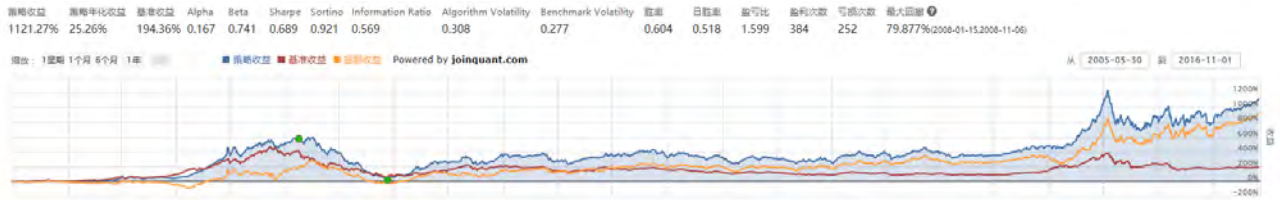
我们生成的 ROEC 使用前 90 个交易日前数据与当前数据的差值，90 个交易日为一个季度的粗略假定。通过图中的例子可以看出，我们生成的 ROEC 数值越小表示公司财务状态变好。ROEC 的问题在于不能识别“不好变到好”还是“好变得更好”，但两种情况对于股票价格冲击是同方向的。同时我们给出这一指标不足的一面，在实际情况中，包括在理论中，ROE 保持持续的提高是不现实的。一旦某一市场出现较高的 ROE 时，若真正存在高净资产收益，则新企业进入、竞争加剧和利润下降将会随之而来。如此而言，一个市场中不但难以实现持续的 ROE 增长，同时长期高 ROE 也是难以实现的。通过财务数据我们可以看到，不少公司的季报在月末相继发出，其中 ROE 变动还挺大。一般而言公司不出现重大变故，公司股东权益一般变动不会太大，那么 ROE 变动主要来源于当期的净收入水平变动，包括当季度收入的和支出两个方面的影响。

回测数据显示，5% 的最小策略回测体现了对于ROE增大的股价回报，好的成长面信息推动股票市场价格上涨。ROEC 的最大策略表明在 ROE 下降明显的情况下，股价也有上升的刺激。那么影响 ROE 下降的过程中收入下降、支出上升和权益净值增加中后两者可能具有对于股价利好的刺激。对比最小1%的情况，可以发现，当取股票较少时，极端值和异常值可能都选入持仓，但当所选的股票数量增加后，趋势趋于合理化——ROE 变得更好的股票带来整体回报变得更高（最小 1% ROEC 可能反映了不推动股价上涨的一些意外情况，但最小 5% ROEC 表现出来市场对于指标的认可），而 ROE 变差带来的回报下降很多。在具体策略逻辑中，到底要少取股票数量建仓意图捉妖（股），还是要增大选股数量以减少风险，都看个人偏好。但是对于因子的作用确实需要对于数据进一步的挖掘和理解。

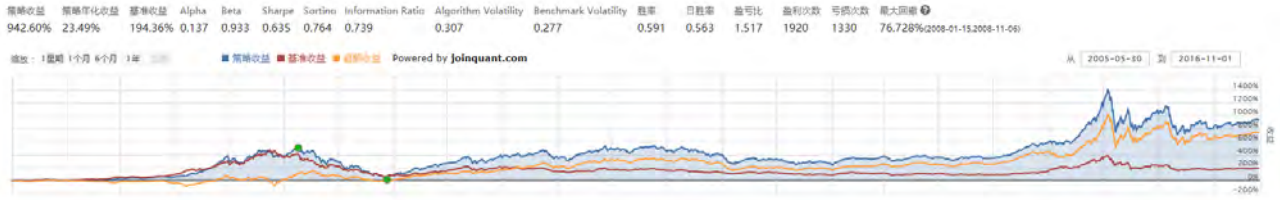
最大1%



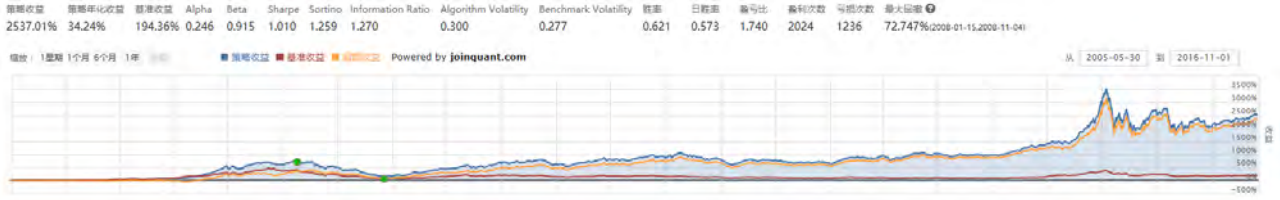
最小1%



最大5%



最小5%



例子（仅为个例说明一个可能的情况，不代表普遍性），最大5%策略在最后一期调仓中选入了大智慧。我们可以通过公开信息查询到大智慧 2016 年 10 月 29 日发布了季报，加权 ROE 从上季度的 -9% 变为 -32%，那么 ROEC 为 23%。我们可以看出就单项指标而言这表明该公司的盈利能力明显下降。从季报出来后到截止 11 月 15 日波动是有的，但股价基本持平，也即 ROE 指标的恶化情况没有明显反映到股价中。通过相关新闻和季报数据结合我们可以知道，当期的ROE减少主要由于当期投资于互联网平台项目的支出增加。对于这样的一个投资性行为，不同投资者可能会给予不同的解读，但是确定性的结论很难达成，市场上的股价反映了市场对该投资的一个期望。

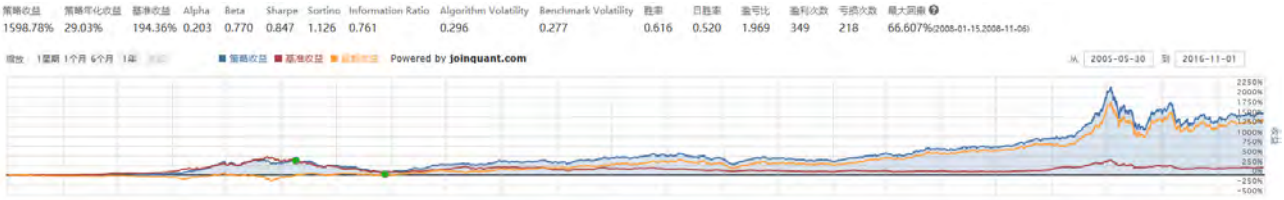
12资产报酬率 $\mathrm{ROA} = \frac{\mathrm{net\_profit+income\_tax\_expense+interest\_payable}}{\mathrm{total\_sheet\_owner\_equities}}$

资产报酬率是衡量公司是否能够有效利用总资金的指标。我们计算过程中资产报酬率等于公司的“净利润”、“所得税费用”和“应付利息”合计除以“负债和股东权益合计”。在实际的财务指标理解中，ROE 带来的是与 ROA 直觉的相似性和具体细节的差异性。在实际应用中，ROE 中分子使用的是“归属于母公司股东的净利润”，而 ROA 使用的收入减去负担的一些非借贷资本成本。另一点不同在于，ROE 分母使用的是期初和期末平均的归属于母公司的净资产，ROA 分母使用的是期初和期末平均的“负债和股东权益合计”。分子的差异中 ROA 把借贷资本的成本计入回报之中，而在分母中，ROA 同样把负债纳入指标计算，这样的结果就是公司的财务杠杆的成本和收益统一纳入衡量体系。可以这样理解，无所谓资金是否是借贷而来，所有的自由资金和负债资金的整体运营盈利能力就是 ROA。

两者都是针对公司的运营能力进行统计的指标，两者主要的区别在于对于股东权益和整体公司的侧重，以及财务杠杆作用。ROA 不同于 ROE 之处在于其讨论公司盈利能力时同时，除了股东利益外同时把债权人考虑了。一个例子为，假设公司有 10% 的资产收益率（ROE） 并且没有负债，当它增加 1 倍财务杠杆时，便可以使用两倍于本身资本的资金进行经济活动，得到的收入扩大一倍，那么在盈利能力不变的情况下 ROE 变为 20%。但是，在简化条件下，ROA 的计算中考虑了借入资金，所以在增加财务杠杆后 ROA 仍然等于 10%。由此可见 ROA 可以更合理地反应公司的盈利能力。

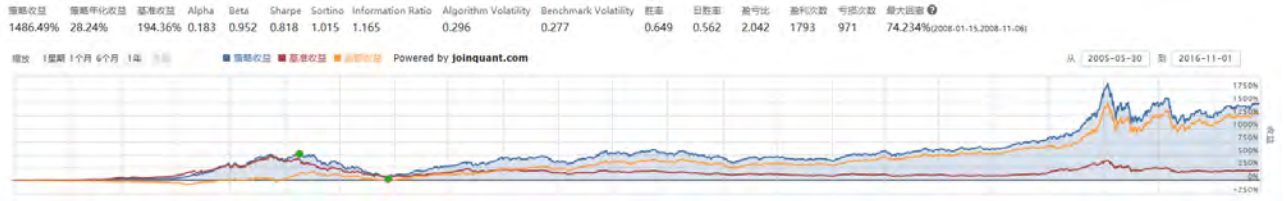
ROA 从回测的数据有效性而言，要好于 ROE。这里指的有效性是指，当该指标表现的好时其回测表现好，而当该指标不好时，回测指标相对较差。对应到回测图中，我们可以看出高 ROA 指标（最大 5% 策略）的回测好于低 ROA 指标（最小 5% 策略）的回测结果。对比 1% 和 5% 策略中高 ROA 指标我们可以看出，当包含了一些次优 ROA 指标股票进入持仓后结果变差，说明在高 ROA 处 ROA 指标具有较好的区分回测回报率高低的能力。ROA 在不同行业体现出较大差异，即需要结合其他原因综合考虑才可以，进一步的处理方法可以利用中性化等处理行业因素影响的方法。

最大1%



最小1%

最大5%



最小5%

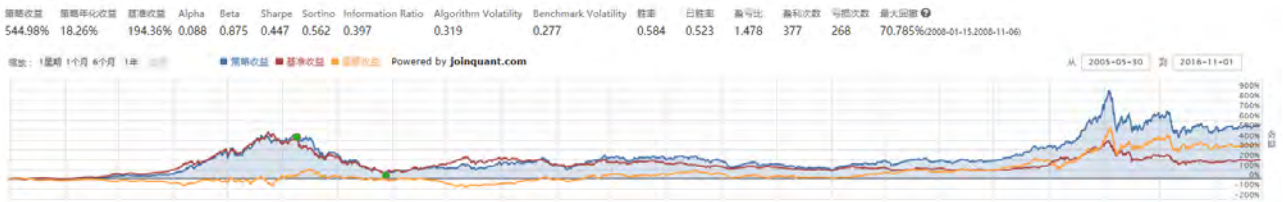
13ROA变动 $\mathrm{ROAC} = \mathrm{ROA} < em > \text{上一期} - \mathrm{ROA} < /em > \text{当期}$

ROA 变动计算方法为使用上一期的 ROA 减去当前期的 ROA，一期为 90 个交易日。与 ROE 变动相类似，ROA 变动也表示企业盈利能力的变动，当然也存在增长和变动两种指标的问题，所以这里使用的时 ROA 变动指标。这个指标的说明也与 ROEC 相似，如果前一期 ROA 高于当期 ROA 则 ROAC 为正；反之，前一期 ROA 小于当前期 ROA，ROAC 为负。ROAC 指标越说明企业的运营能力变得越差，而 ROAC 越小表示变得越好。

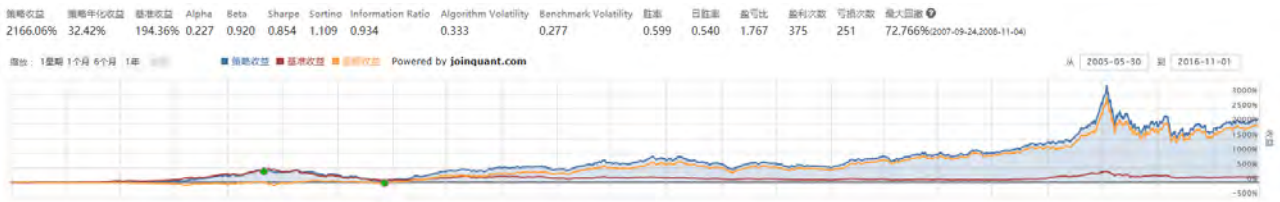
当前这个简单指标 ROAC 也存在着手写例子中的相似情况，原例子里 c1 和 c5 一个是获利变少一个是赔得更多，但对应的指标值可能相同。能更细致区分会感觉更好一些，不过市场逻辑在于价格是惯性的，每一个冲击带来价格变动的结果。如果前一期的估值是合理的，那么当前期的 ROAC 变动无论是获利更少还是赔得更多，股票市场价格变动的反馈方向是一致的。当然，所谓的量化仅能保证定性结论的正确性是不能满足大家的。多因子和该因子的改良处理，这是后续优化结果的两个选择。

来看数据，数据与基本逻辑有一些相符，ROAC 小的会得到很好的市场回报，因此最小 5% 相较于最大 5% 策略体现出较高的回报率。最小 5% 体现出，ROAC 指标变得更好会推动股票价格上涨，因此带来了回测中的获利。相比最小1%与5%策略，更大的选股数量使得策略获利水平趋于均值。ROAC 大的也会有很好的回测结果，虽然没有 ROAC 小的结果好，但是也是跑赢大盘很多，对此我们需要深入到个股查看行情数据。当期的投资性行为降低 ROA 分子中的资产回报，当期的融资性行为增大 ROA 分母，都会导致 ROAC 变大，但对于公司而言可能是正向影响。

最大1%

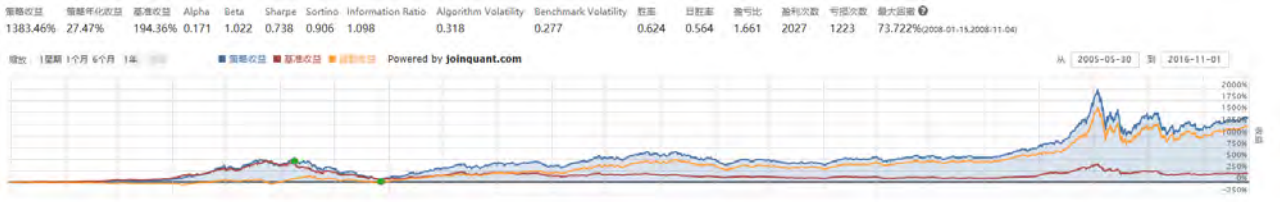


最小1%



最大5%

最小5%



14主营业务利润占比 $\mathrm{OPTP} = \frac{\mathrm{operating\_profit}}{\mathrm{total\_profit}}$

主营业务利润占比，直接从名字就可以看出表达的内容，具体的计算方法时使用营业利润除以利润总额。使用营业利润代替主营业务的原因我们已经在上一个因子描述中给出了原因。主营利润占比较高，说明企业主要工作中心在主营业务上，两方面影响：从好的一面说，主营业务占比较高的企业工作重心集中，在有限的企业资源条件约束下集中，企业应该主要依托主营业务发展；不好的一面而言，主营业务依赖度较高，对于行业层面的系统性风险抵抗能力较弱，需要依赖行业良好的发展条件企业自身才有较好的发展空间。

通过数据我们看到主营业务占比最大的 1% OPTP 在整体回测下具有良好的表现，当选股选取股票池前 5% OPTP 时，仍然不错。对比主营业务占比最小的 1% OPTP，表现明显不如高 OPTP 策略，可见在分散化业务情况下，市场没有给予充分的认可。当然，反过来的表述可能更接近事实，分散化业务的企业本身没有更好的自身发展，导致股票没有反映出市场的认可。应该额外指出的是，作为财务指标，策略虽然每月调仓，但是可能在季度中持仓仅进行仓位调整而非持仓股票调整。因此 OPTP 指标反映的一段时间内的回报情况，而不单单是较短时间的市场反映。公司的主营业务策略对于公司的影响超过季度财报的周期，那么对于持仓调整周一为1月的策略，不能良好的捕获OPTP指标带来的中长期市值增幅回报，也是有可能的。

最大1%

最小1%

最大5%

最小5%

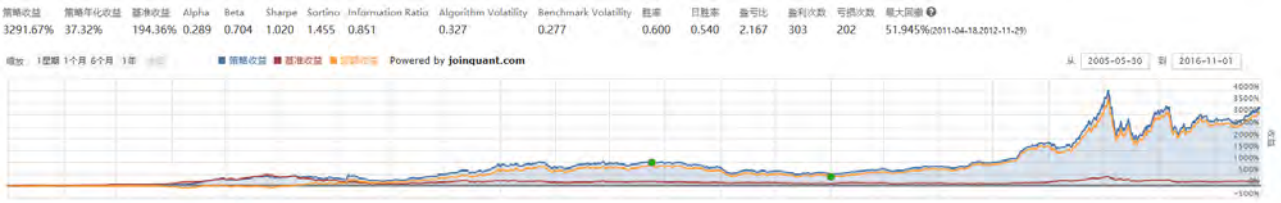
15主营利润占收入比例 $\mathrm{OPOR} = \frac{\mathrm{operating\_profit}}{\mathrm{operating\_revenue}}$

主营利润占收入比例与上一个主营利润占比相似，分子都为主营业务收入。OPOR 与与之前的 OPTP 差异在于，OPOR 使用的分母使“营业收入”，而 OPTP 分母使用的使“总利润”。对比财务细节和可以看出，两者的差距主要在于总利润与营业收入的范围差异。利润总额指企业在生产经营过程中各种收入扣除各种耗费后的盈余，反映企业在报告期内实现的盈亏总额。但是营业收入中包含了没计算在利润中的很多成本和支出。OPOR 和 OPTP 两个指标来自于【量化课堂】Foster Friess 积极成长策略，期中有具体的讨论。

我们看到 1% 的最大和最小两个策略反映出了很多极端情况，因为这部分极值点选股策略的回测情况与 5% 最大和最小两个策略的差异巨大。这个 OPOR 最大策略情况与 OPTP 最大策略回报率较为相似，但是最小策略中出现了反常现象，最小的 OPOR 居然可以实现比最大 OPOR 更高的收益水平。营业收入包括主营业务收入和其他业务收入，一般而言其他业务收入较低。营业收入与利润中间的差值包含也营业成本和业务税金及附加，OPOR 较小意味着差值部分较大，那么这样的公司可能本身利润水平不算高，通过高速的经营周转速度带来较高最终利润水平。那么可以看出 OPOR 反向选取的很可能出现类似零售业等高周转低销售利润率（单位销售物品的利润率）的产业。

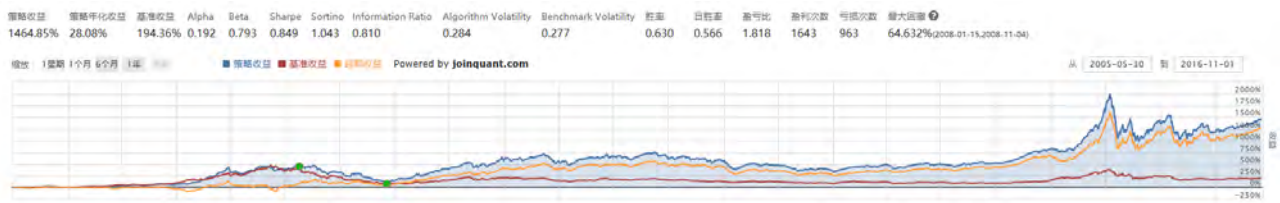
最大1%

最小1%



最大5%

最小5%



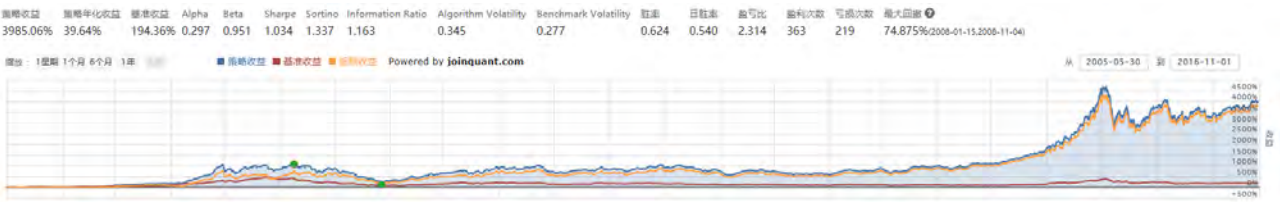
16收入净利率 $\mathrm{NPOR} = \frac{\mathrm{net\_profit}}{\mathrm{operating\_revenue}}$

收入净利率计算方法为净利润除以主营收入，与上一个指标OPOR相比主要差别在于分子选用了净利润而不是主营利润。相比于 OPOR 分子使用的营业利润，净利润指标表示的更加纯粹，剔除了营业外费用的影响。可以看出增长因子中很多对于财务数据指标的选取都大同小异，主要在于不同的指标可能有有不同的适用场景。以上的几个合成指标同时调用了连个或两个以上的指标，具体变动情况会更加复杂，需要在使用时对指标有清楚的认识。

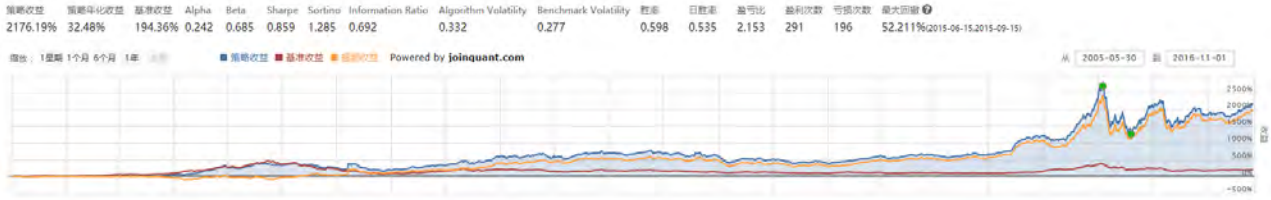
由于指标的构建选取的财务数据于 OPOR 相似，NPOR 具有与 OPOR 相似的分析情景，当 NPOR 较高时企业营收能力较强，因此应该有较高的回报。NPOR 较低时出现相反的情况，对于股价有向下的刺激。

通过数据我们可以看出，实际情况于 OPOR 相似当也存在区别。在极端情况下，1% 最大 NPOR 带来了非常漂亮的回报率，因此我们可以推断，领先全市场的实际利润水平会被市场追捧推动股票价格上涨。但是 OPOR 没有这样的体现因为营业收入不如净利润这般纯粹凸显收入能力。但是极端低值 NPOR 没有 OPOR 反映好，因为权衡企业周转、财务杠杆这些因素之外，本身指标过低还是会给估值留下减分的印象。5% 策略就没有这么夸张了，但应该更反映市场的一般情况。同样，最高的 NPOR 回报好于 OPOR，低 NPOR 回报差于 OPOR。在之后的深入研究中，我们不难推断，两个指标会得到很高的相关系数得分，进而在实际选取因子时很可能会有一个被当作冗余因子被删去。

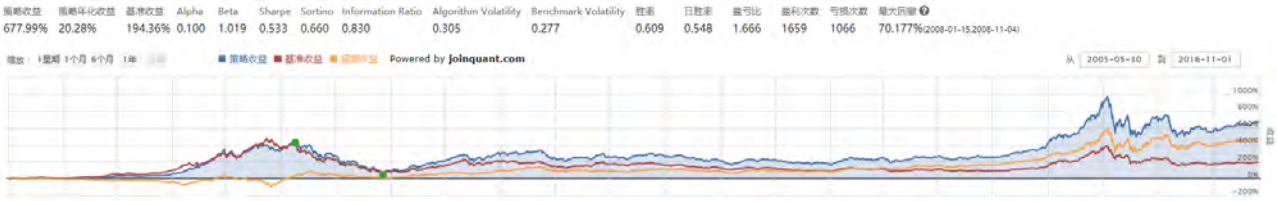
最大1%



最小1%



最大5%



最小5%

17收入净利润占比变动 $\mathrm{NPORC} = \mathrm{NPOR} < em > \text{上一期} - \mathrm{NPOR} < /em > \text{当期}$

周如净利润占比变动时 NPOR 上期指标减 NPOR 当期指标的值，根据我们上边表述多边的分析，NPORC 越小越好，越小表示 NPOR 上升，越大表示 NPOR 下降。这一指标有一个好处在于可以一定程度消弱 NPOR 行业产业异质性。有一些需要高周转（带来高费用）的产业，不一定表示其变动也与 NPOR 指标的相对大小排序相似。

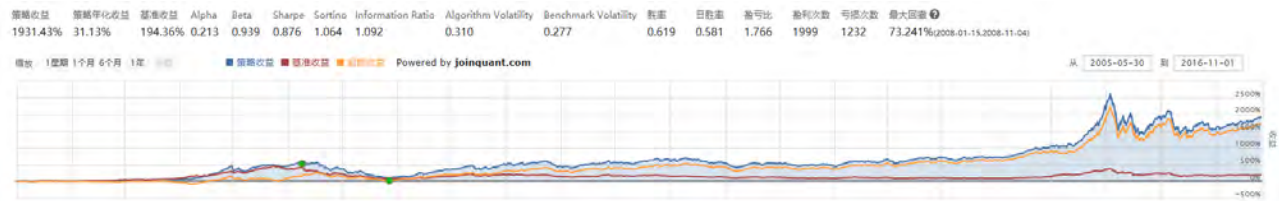
看数据我们看出，确实 NPORC 很小的时候回测回报率很高，远高于 NPORC 很大的情况。最小 1% 和最小 5% 相比，5% 减小回报但是应该会更加稳健（最大回撤率也稍有下降）。与前几个变动指标相似，相反向的变动指标很可能反映出公司的重大变动，因此会有超过大盘的回报率。

最大1%

最小1%

最大5%

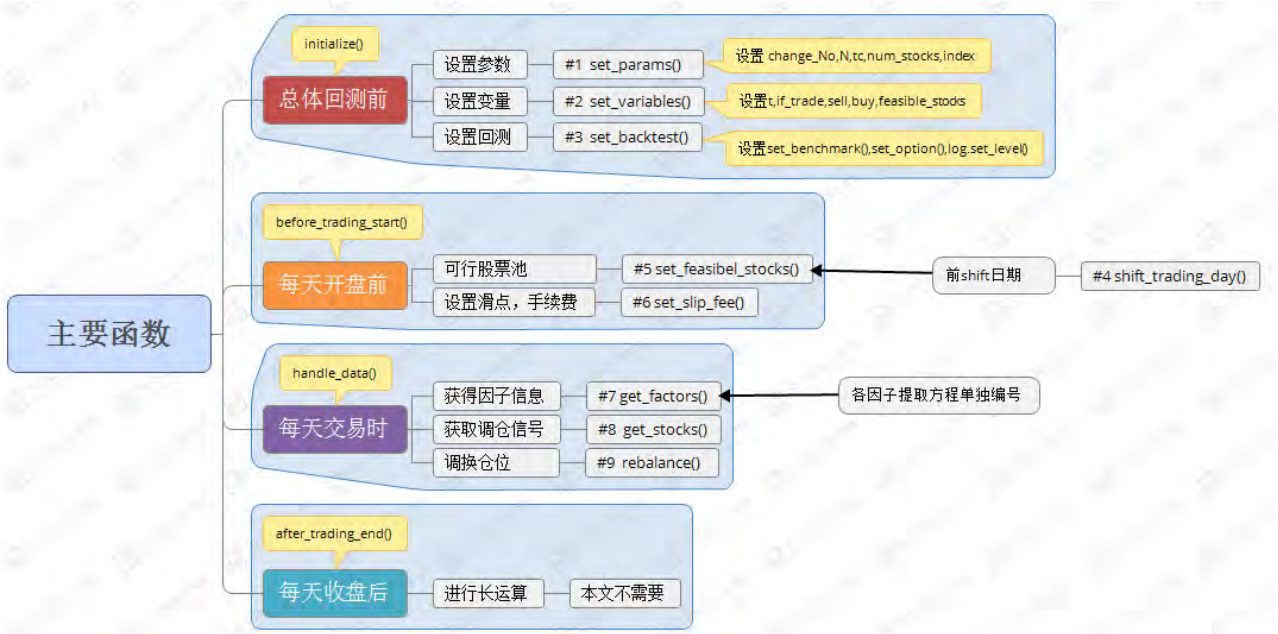
最小5%

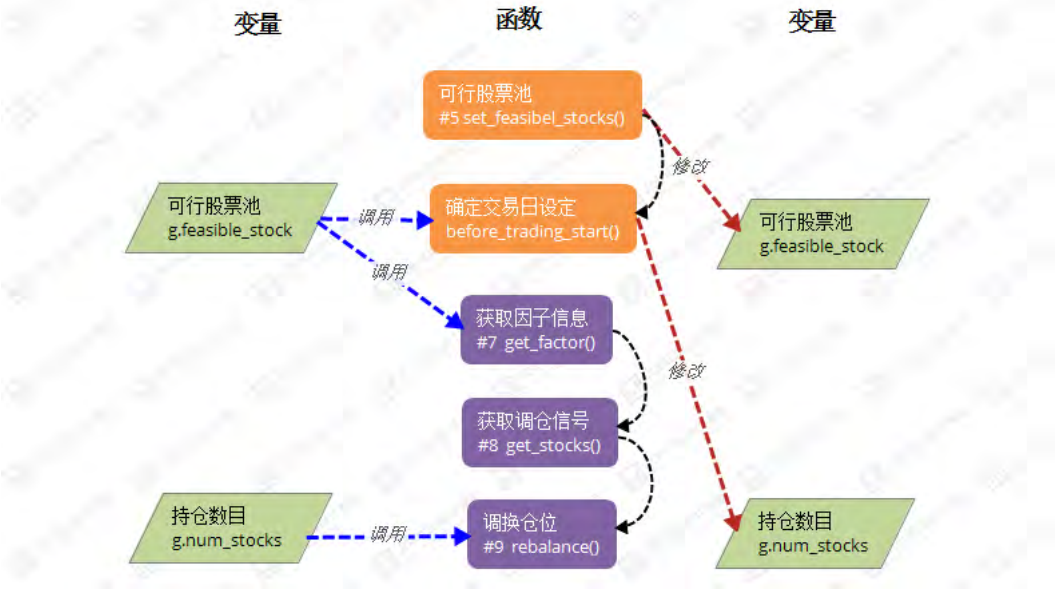


小结

这一组因子主要是成长因子，其实都是偏向于公司盈利能力的指标。观测完之后我不由得开始思考人生，这样的指标条件是要躺着赢钱的节奏么？市场是否对于“平庸”具有惩罚，对于无论是好的还是坏的刺激点都有明显回报？还是说因为每月初调仓，实际上所有的回报率都体现出了择时机制？下一次我们将会带来技术指标因子（一个月动量、三个月动量、六个月动量、12 个月动量、换手率、换手率变化、VAR、VAR 变化和震荡），敬请期待。

函数和变量说明书





v1.0，2016-11-22，文章上线
v1.1，2016-11-30，添加变量和函数说明书，调整代码中函数编号

【量化课堂】股指期货对冲策略

前言：本文介绍如何在一个量化策略的基础上使用股指期货进行对冲，以此抵消系统性风险，并获得策略的超额收益。

作者：swlaw
编辑：肖睿

本文由JoinQuant量化课堂推出，难度为进阶（上），深度为level-0。

Alpha 对冲背景

作为普通的投资者，大部分投资者都会选择基金这一投资渠道。而面对眼花缭乱的基金产品，投资者如何选择出符合自己风险偏好的基金就显得十分重要。这一选择的过程可能就需要我们对基金产品做一个绩效评价，选出其中表现最好的作为我们的投资组合。而最常见的三个评价标准有下述三个：

1、夏普 (Sharpe) 指数：
美国经济学家威廉•夏普于 1966 年发表《共同基金的业绩》一文，提出用基金承担单位总风险（包括系统风险和非系统风险）所带来的超额收益来衡量基金业绩，这就是夏普指数。夏普指数通过一定评价期内，基金投资组合的平均收益超过无风险收益率部分与基金收益率的标准差之比来衡量基金的绩效。计算公式为：

$$S_p = \frac{R_p - R_f}{\sigma_p}$$

其中，S_p 为夏普指数，R_p 为基金组合的实际收益，R_f 为无风险收益率，σ_p 为基金收益率所对应的标准差（σ_p 里既包括了系统风险也包括了特有风险）；也可见 CAPM 模型。
其直观含义是资产组合承担一单位的风险能够获得多少单位的额外收益。因此，可以认为，夏普业绩指数越大,基金绩效就越好；反之，基金绩效就越差。我们也可以通过图形做一个说明，以 CML（资本市场线）为例：在资本市场线以上的组合，其夏普比率较高，也就意味着承担单位的风险能获得更高的收益。

2、特雷诺 (Treynor) 指数简介
杰克•特雷诺 1965 年发表《如何评价投资基金的管理》一文，认为足够的证券组合可以消除单一资产的非系统性风险,那么系统风险就可以较好地刻画基金的风险，即与收益率变动相联系应为系统性风险。因此，特雷诺指数采用在一段时间内证券组合的平均风险报酬与其系统性风险对比的方法来评价投资基金的绩效。这就是特雷诺指数，它等于基金的超额收益与其系统风险测度β之比。
计算公式为：

$$T_p = \frac{R_p - R_f}{\beta_p}$$

其中，T_p 为特雷诺指数，β_p 表示基金投资组合的 β 系数，是投资组合要承担的系统风险。
特雷诺业绩指数的理论依据也是资本资产定价模型（CAPM 模型），但是是以证券市场线（SML）为评价的基点，当市场处于平衡时，所有的资产组合都落在 SML 上，即 SML 的斜率就表示市场证券组合的特雷诺指数。当基金投资组合的特雷诺指数大于 SML 的斜率时， 该投资组合就位于 SML 线之上，表明其表现优于市场表现；反之，当基金投资组合的特雷诺指数小于 SML 的斜率时，该投资基金组合位于 SML 线之下，表明其表现劣于市场表现。所以，特雷诺业绩指数越大,基金的绩效就越好；反之，基金的绩效就越差。

3、詹森 (Jensen) 指数简介
美国经济学家迈克尔•詹森 1968 年发表《1945-1964年间共同基金的业绩》一文，提出了一种评价基金业绩德绝对指标，即詹森指数。他认为,基金投资组合的额外收益可衡量基金额外信息的价值,因而可以衡量基金的投资业绩，其计算公式为：

$$J_p = R_p - [R_f + \beta_p \cdot (R_m - R_f)]$$

这里 J_p 为詹森指数，其他符号如之前所定义。
詹森业绩指数,又称为 α 值,它反映了基金与市场整体之间的绩效差异。詹森指数也以资本资产定价模型为基础，根据 SML 来估计基金的超额收益率。其实质是反映证券投资组合收益率与按该组合的 β 系数算出来的均衡收益率之间的差额。当然，差额越大，也就是詹森系数越大,反映基金运作效果越好。如果 α 为正值，则说明基金经理有超常的选股能力,被评价基金与市场相比，高于市场平均水平，投资业绩良好；α 为负值则说明基金经理的选股能力欠佳,不能跑过指数,被评价基金的表现与市场相比较整体表现差；α 为零则说明基金经理的选股能力一般,只能与指数持平。

在策略中的应用

基于上面的逻辑，我们可以先选出一个 Alpha 较高并且稳定的交易策略组合，然后用沪深 300 股指期货进行对冲，以此以抵消掉系统性风险 β，并只剩下 α 的部分，使得我们的组合无论在牛熊市都能够获得稳定的收益。思路如下：假设策略选出的股票组合的超额收益为 α，组合收益 R_p 与指数收益 R_m 的线性相关系数为

$$\beta_p = \frac{\mathrm{Cov}(R_p, R_m)}{\mathrm{Var}(R_m)},$$

（具体可见 CAPM 公式。）组合的收益可以用公式表示为：

$$R_p = \alpha + \beta_p \cdot R_m.$$

所以对于一单位的股票组合，卖空 β_p 单位的股指期货可得到的收益为 α，按照这样的操作即可获得 α 的超额收益。

根据这个思路，我们构建一个策略：
(1) 首先找一个 Alpha 比较高的投资组合（此处我们以因子研究系列中的 ALR 因子为例）
(2) 设置两个仓位，一个用来交易股票，另一个用来交易股指期货。此外由于期货合约采取每日结算，并且当日卖出股票得到的现金要第二日才能转入期货账户，因此期货仓位中的钱除了满足保证金要求外，还要预留出能够扛两个涨停日的准备金。
(3) 设置调仓频率，根据资产负债率选择 ALR 指标最大的 3% 的股票作为新的股票组合。
(4) 每日跟踪期货账户准备金情况，如果发现准备金不足，应卖出一定量的股票来补充准备金并调整对冲的仓位。

回测结果

先看看我们不做 Alpha 对冲，回测的结果：

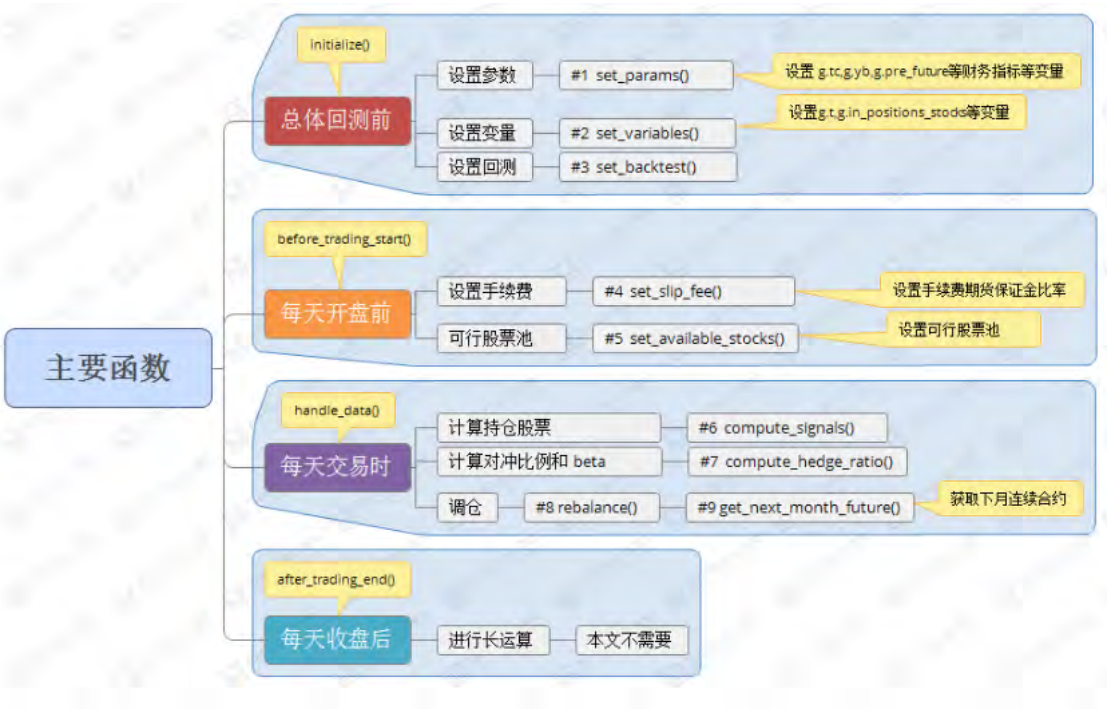
可以看出从 10 年 5 月到 16 年 11 月这一阶段，不做 Alpha 对冲可得的收益为 278%，但回撤较大，有 40%。做过对冲之后的结果如下：

可以发现后者放弃了很多的收益，但做过对冲后系统性风险基本为零，并且回撤回测明显减小，仅为 13%，而且完全没有受到股灾的影响。

结语

股指期货的规则和对冲的细节实际上比较复杂。比如说当指数发生变化之后，仓位里股票和股指期货空单就会分别发生一涨一跌的情况，对冲比例发生变化，应该及时调整仓位。又比如说任何时间点都有四个不同交割日期的沪深300期货合约，它们有着不同程度的折价或溢价，选择不同的日期的合约会带来一些差异。本文中没有考虑太多的细节，但这些都是值得考虑的，所以还有很大的提升空间。

函数和变量说明书



文章更迭记录：
v1.0，2016-11-22，文章上线

【量化课堂】因子研究系列之三 -- 技术因子

导语：不同因子的作用折射出每个宽客各自对于金融市场的理解，本文目的在于展现不同因子与市场表现帮助各位看官查看是否有所遗漏，同时提供因子提取和使用的简易实现代码。

作者：Sunx
编辑：肖睿

本文由JoinQuant量化课堂推出。难度为入门，深度为level-0。

本文是一系列因子研究中的第三篇文章。本系列的文章有：

1. 【量化课堂】因子研究系列之一 -- 估值和资本结构因子
2. 【量化课堂】因子研究系列之二 -- 成长因子
3. 【量化课堂】因子研究系列之三 -- 技术因子
4. 【量化课堂】因子研究系列之四 -- 市值与行业的中性化

技术因子简介

技术因子由技术指标转化而来。一般而言，技术因子的提出是对应成长因子而言。不同于成长因子结合财务数据等基本面指标，技术因子侧重于根据市场行情中股价和换手率等指标反映市场行为反映，用以推断价格变动趋势。量化课堂中双均线策略、海龟策略、均值回归入门和羊驼交易系统给出了一些简单利用技术指标的策略，基于协整的搬砖策略、多头趋势回踩策略和斗牛蛋卷二八轮动原版策略实现是基于技术指标（一般是价格）构建的更为复杂精巧的策略，而Hurst指数与应用（单股票）则是基于既有技术指标生成复杂的技术因子的策略。同时，大家也可以进一步查看 beta 和夏普率等相关内容，这些也都是基于往期价格生成的技术指标。

动量（ Momentum ）

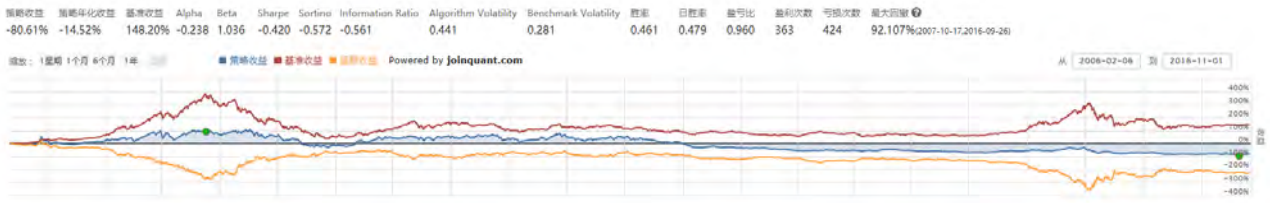
动量指的是一段时期内证券价格向某一方向持续变动的趋势。以动量因子为主的单因子策略一般被定义为动量策略（Momentum Strategy），根据较早一定窗口期内资产收益情况做多表现优于同类的资产，做空表现差于同类的资产。单纯的动量策略使用往期回报数据来预测未来截面回报数据，所谓未来截面回报（ Cross Section of Future Return） 讲的是一个时间点上不同投资产品的预测回报率。这东西怎么说呢，不少人在不同时间、不同市场上的不同产品上都发现了动量策略还挺好用的，但是据说中国不是这样的，至少我们之后会在下面的回测中看到一些结果表明我们股票市场好像这招不大好用。但是，即使他们观测到了动量还不错，然而也观测到了动量策略时不时崩溃了，而且一般连着几个月出问题（有兴趣阅读 2016 年 6 月 Journal of Financial Economic 上的文章 Momentum Crashes ）。

简述一下动量策略就是在上一个观测的窗口期计算回报率（一般为观测期最后一天和第一天的收盘价差值再除以前一天的收盘价），然后计算排序，通过排序选择回报率最高的几只股票并持有。当然，“动量”和“反转”是一对绕不开的冤家，如果上述策略中选择的是回报排名最低几只就变成反转策略了，反转的直观理解就是跌的太多了要反弹。这里面就涉及到很多问题了：首先，观测的窗口期定多长时间；其次，到底选择多少只股票；第三，动量还是反转是个纠结的问题，那么到底选择最高的几只还是回报最低的几只；第四，调仓周期选择；以及其他，像什么是否动态调仓、每只股票仓位选择、调仓时入场时间和价格判断和出场条件等等。本文作为描述不同因子的基本表现，简化策略为对应 1 月、 3 月、 6 月和 12 月窗口期，月初调仓平均分仓，持有一个月内没停排的股票中 1% 或 5% 的股票。因为聚宽股票价格数据起始于 2005 年初，为了可比性我们统一使用 2006 年 2 月开始回测（要不 12 个月动量没法算了啊）。实际使用的动量指标为收盘价格在窗口期的增长率，计算公式为 $\mathrm{MTM}=\frac{\text{收盘价当期}-\text{收盘价}\text{一月前}}{\text{收盘价}_{\text{当期}}}$，越大表示观测期间股票收盘价格增长百分比越多。

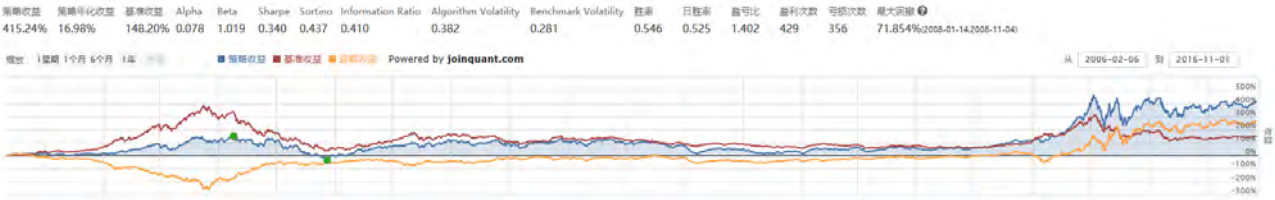
20 一个月动能 $\mathrm{MTM1}=\frac{\text{close当期}-\text{close}\text{一月前}}{\text{close}_{\text{一月前}}}$

通过回测我们观测到，好像市场体现出来的不是动量惯性而是反转现象。上一个月收益增长最高的股票在下一个月都跌回去，而上个月跌的最多的股票，在下一个月带来了不错的回报。 5% 最大一个月动量的策略回测结果表明，只有在大盘情况非常好的时候，上证股票才会体现出连续两月的上涨惯性，即第一月上涨后第二月仍然上涨。但是可以看出，这样涨幅的回报一直跑不赢大盘，在整体大盘形式不理想时表现出负收益。 5% 最小策略表现出，上一个月下跌最大的股票下个月带来了不错的回报率。结合最小 1% 和最小 5% 一个月动量一致反映出的内容，我们可以发现，在 2008 年前后的行情中，回报率对于反转信号并不十分敏感，但是随着 2008 年的市场洗礼后，市场对于一个月反转信号变得愈发敏感，似乎放大了信号的作用。这样的条件下，大盘走势较好时反弹更大，而大盘走势不好时，根据反转信号买入的股票跌幅远超大盘。

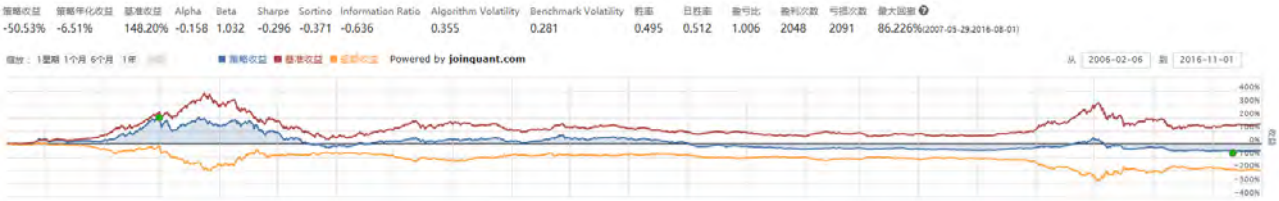
最大1%



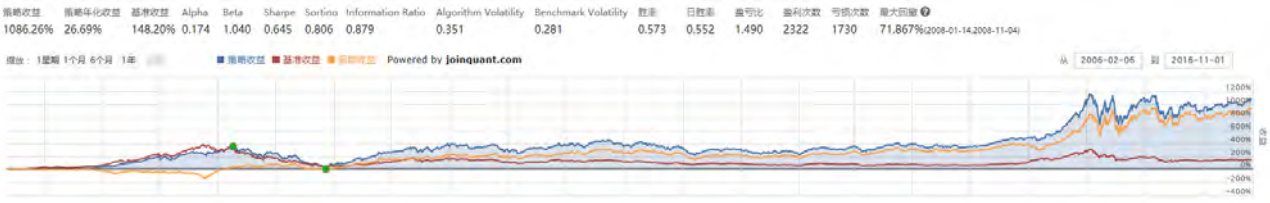
最小1%



最大5%



最小5%

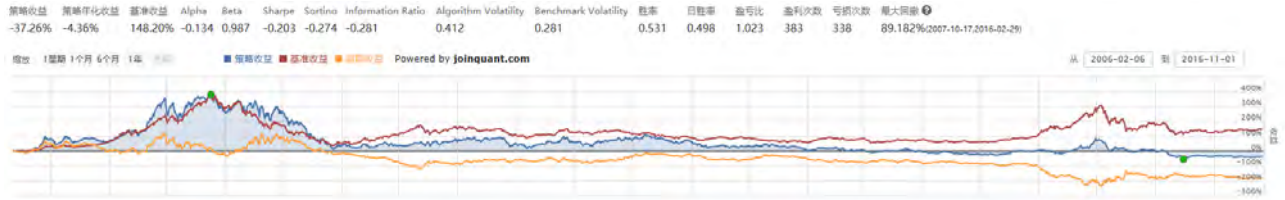


21 三个月动能 $\mathrm{MTM3}=\frac{\text{close当期}-\text{close}\text{三月前}}{\text{close}_{\text{三月前}}}$

在一些对北美市场的分析中，较长的观测期的动量指标对于当前股票市场回报的预测也较为有效，很多文献表示过去一年的动量是有效的。相反，反转信号的传递时间较短，一般为一个月反转。但是我们看到，当然之前不少国内研究也都发现了，A股中观测期超过一个月的长窗口动量仍然没有效果（其实是有效果的，不好的效果）。而反转的效果在观测期超过一个月后，仍然好用，而且比一个月反转的信号带来的回报更高，回测等技术指标也稍有改善。

进一步观测回测结果细节，可以发现，三个月的动量其实比一个月的动量表现稍微好一点，或者说三个月的动量表现虽差，但是没有一个月动量差。观测最小 5% 的三个月动量策略，即三个月反转策略，其回报率始终超过大盘，同时在 2015 年之后表现出了与一个月反转相似的情况，即收益增加同时波动幅度变大。

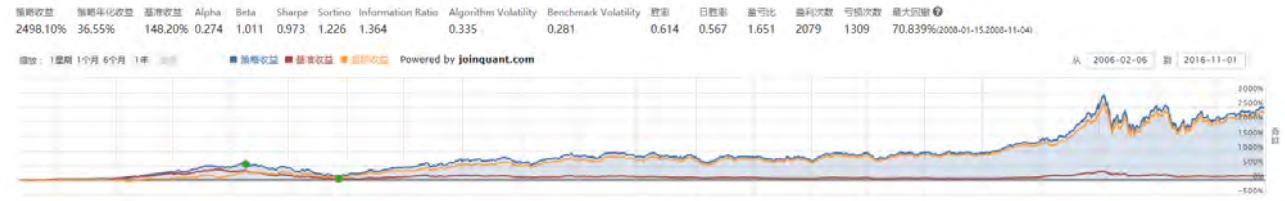
最大1%



最小1%

最大5%

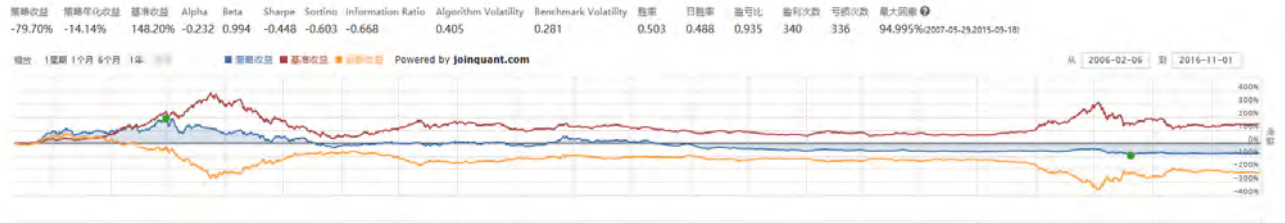
最小5%



22 六个月动能 $\mathrm{MTM6} = \frac{\text{close当期} - \text{close六月前}}{\text{close六月前}}$

对比三个月反转（ 5% 最小三个月动能）的超高回测回报率，六个月的最小动能表现没有那么惊艳。没有逐一测试一月与六月最小动能中其他时间节点的回测结果，但我们可以保守的推断，窗口长度为一个月到六个月之间存在一个时间长度，使得该长度的窗口期最小动能策略达到回测回报率最大。这是针对一个月持仓周期策略发现的性质，如果调仓周期变了，不同时间窗口的影响变动也会随之变动。

最大1%



最小1%

最大5%

最小5%

23 十二个月动能 $\mathrm{MTM12} = \frac{\text{close当期−close十二月前}}{\text{close十二月前}}$

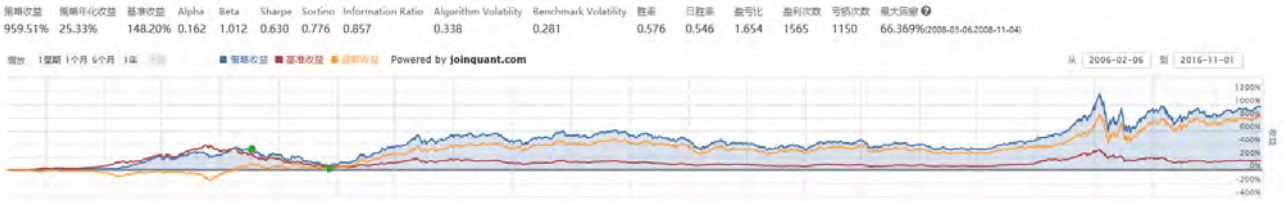
对比一个月动量、三个月动量、六个月动量和十二个月动量表现可以看出，动量基本上表现都不好，但是随着观测时间的拉长，长期动量最大的股票变现稍有好转。这与逻辑相符，长期股票回报率较好的公司，应该会由较大几率带来高回报。针对没有出现超过大盘的回报率这一结果，可能由于持仓期等设定与长期观测动量策略的矛盾导致。对比一个月反转、三个月反转、六个月反转和十二个月反转，可以看出 5% 最小动量策略回测收益率率先增大再减小。反转对于一个月持仓的回报率，其最有效的观测期可能存在与 1 到 6 个月之间。

最大1%

最小1%

最大5%

最小5%



动量策略有点“追涨杀跌”的意味，而大家直觉的反映应该都是比较短期的，国内的研究也发现国内股票市场中动量有效性在几周内还有点，长了就看出不了。这些文献呢，推测这种现象可能是由于我们的市场投资的行为模式与经典文献中北美市场里不同导致的。另一点需要突出强调的是，在经典文献中一般给出间隔月（Gap Month）的概念，例如使用的是去年 1 月初到 11 月底之间的动量，空过12月，分析这个 11 月动量与第二年 1月份回报的关系。因为部分文献表明这一个月的引入会带来短期反复现象（ Short-term Reversal），但在本文中简化的观测回测中，我们没有进一步处理这个问题。

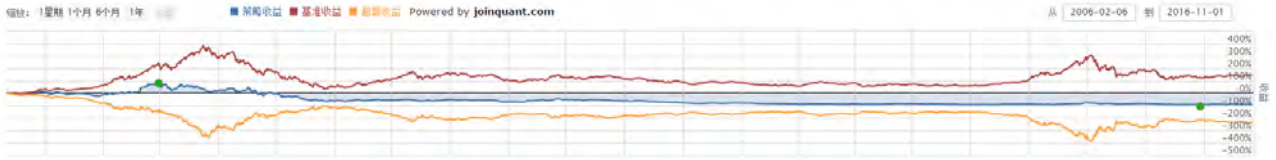
24 换手率 $\mathrm{TR} = \frac{\sum \text{volume}}{\text{circulating\_cap}\cdot 10000}$

换手率指在一定时间内市场中股票转手买卖的频率，是反映股票流通性强弱的指标之一。换手率的计算方式是一个时间窗口内（一个月）的股票交易数量除以期末的的流通股票数量。最明显的现象就是，如果一支股票的换手率较高，说明这支股票表现的比较活跃，投资者对它的参与（关注）度的较高。也有一些研究专注于分析换手率在什么情况表明由主力机构进入（或撤了），什么样的换手率情况表明由机构持仓运作等等，这里不展开讨论。

从交易的逻辑上而言，供过于求则价格上涨，供小于求则价格下降。那么一只股票的换手率过低表明交易双方参与较少，可以理解为这样的一个价位上，价格较为被市场认可，因此没有过多的交易产生。当然，这要在市场流动性充足，不存在大量提取流动性的假设下。通过回测我们可以看出，最大5%换手率的策略除了在 2008 年之前的行情中还略有正向收益，在 2008 年之后很少能维持正的收益。这表明，在买入的时间节点中，换手率最大的情况可能隐含着不好的因素。或者，在前一个期限内股票利好已经完全释放，表现反映在较大的换手率中，那么之后的时间中该股票的回报率将不会很大。对比而言换手率最小的 5%，很可能是在热门板块（或个股）轮动过程中获益。前期没有被关注的股票，在下一期被炒作带来的回报。这一逻辑在牛市时较容易被情绪化交易引起，可以看出回测结果仅在 2015 年行情过程中带来超额回报的扩大。

最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
-89.62%	-19.48%	148.20%	-0.294	1.160	-0.504	-0.674	-0.696	0.466	0.281	0.446	0.485	0.831	316	392	95.660%(2007-05-29,2016-05-18)

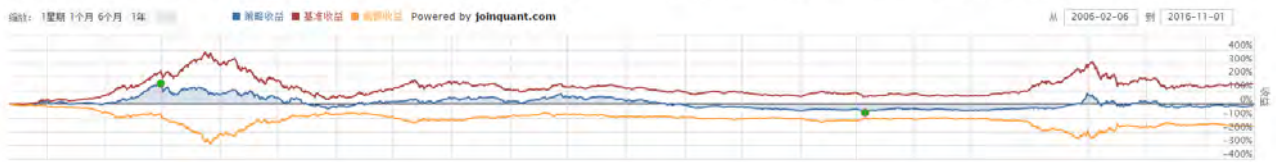


最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
361.48%	15.75%	148.20%	0.079	0.748	0.450	0.590	0.316	0.261	0.281	0.531	0.470	1.682	275	243	57.379%(2008-01-15,2008-11-06)

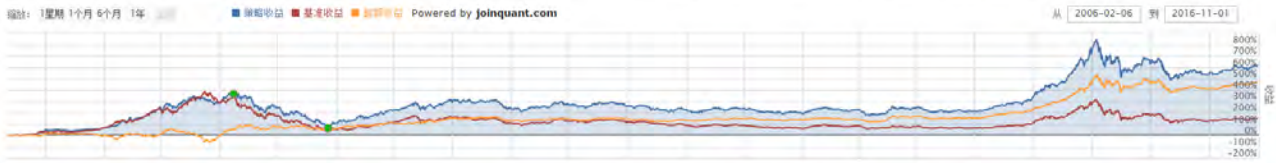
最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
-10.05%	-1.01%	148.20%	-0.106	1.096	-0.130	-0.166	-0.266	0.385	0.281	0.506	0.523	1.034	1630	1590	81.275%(2007-05-29,2013-06-28)



最小5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
611.91%	20.65%	148.20%	0.124	0.840	0.647	0.812	0.844	0.257	0.281	0.593	0.507	1.814	1573	1081	63.745%(2008-01-10,2008-11-04)



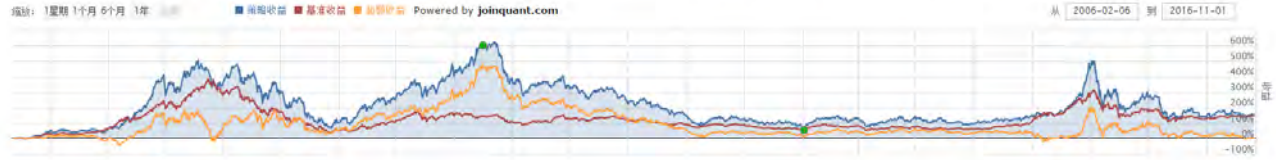
25 换手率变动 $\mathrm{TRC} = \mathrm{TR} < em > \text{前一期} - \mathrm{TR} < /em > \text{当期}$

换手率变动使用上一期的换手率减去当期换手率，如果 TRC 越大表示一个月期间换手率下降越多，相反如果 TRC 越小表明一个月中换手率增长越大。对比换手率的回测结果，我们看到换手率变动与换手率展现了相似的逻辑。最大 5% TRC获得超过大盘的回报率，最小 5%TRC 获得负的收益率。这表明当一个月中换手率变小了，可能还能获得与大盘相似的平均回报率。但是当这个月该股表现出放量情况，很可能导致后续一个月内负的回报率出现。

同样，对于换手率变动需要结合个股的具体情况，作为陪家长看股票节目多年的我，听到什么"底部放量"之类的表述磨的耳朵都起茧子了。但是对于换手率变动，对应的以及其变动，确实应该结合更多的其他因子一起分析。例如，没有其他因子支撑的换手率变动，看起来很异常不是。

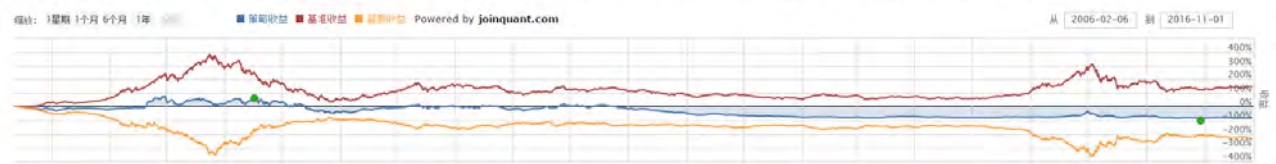
最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
153.70%	9.31%	148.20%	-0.005	1.141	0.127	0.162	0.187	0.419	0.281	0.527	0.508	1.129	416	373	77.250%(2010-03-03,2012-12-03)



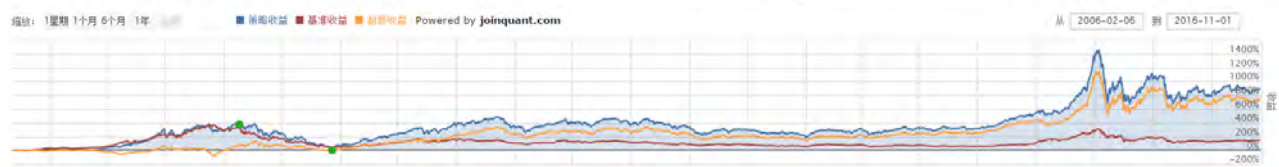
最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
-82.93%	-15.55%	148.20%	-0.252	1.106	-0.425	-0.563	-0.555	0.460	0.281	0.446	0.480	0.918	354	440	91.625%(2008-03-05,2016-05-18)



最大5%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
893.43%	24.56%	148.20%	0.149	1.111	0.560	0.678	0.822	0.367	0.281	0.567	0.560	1.369	2244	1717	72.946%(2008-01-15,2008-11-04)



最小5%

26 波动率 $\text{Volatility} = \text{VAR}(\frac{\text{close}t}{\text{close}t-1} - 1)$

波动率是过去的一个月中股票的日收益率的方差，期中日收益率为当前收盘价除以上一个交易日收盘价后减 1 ，或者表述为当前收盘价与前一天收盘价之差除以前一天收盘价。需要注意的是，这里使用的是收益率而不是股票价格，收益率的好处就是可以排除股票价格本身的影响。例如，100 块的股票变动 10 块，而另一个股票从 10 块涨到 20 块。如果按照价格方差计算两者可能差不多，但是按照收益率方差后一种情况就能明显区别于前一种情况得到较高的因子值。波动率一般用来测度股票价格的风险，实际上两日的收益率可以看作价格变动的一阶关系描述，那么波动率就是价格变动的二阶描述。我们可以看出收益率表述的价格变化本身，而二阶条件描述的是价格变动的变化情况，比如说一月股票日收益率方差是股票价格过去一个月变动的变动情况的概括。通过上述的描述可以看出，价格变动的累计加总（累计正负两个方向的价格变动）描述了股票价格的总体不确定性，而总体不确定性被用术语“风险”表述。当然，也有更加具体的梳理推导表达了价格均值和风险与股票估值的关系，较常见的就是均值方差模型。在本文中，该因子取一个月的日收益率方差，粗略计算使用的是用前22个交易日收盘价计算的21个日收益率，并以这组数据为样本来计算方差。因此，该指标越大表示在过去的约一个月中，该股票的价格波动越大，而 Volatility 越小表明该股票上月价格波动越小。

通过回测我们可以看出，风险（Volatility 因子）大的股票收益率很差，而 Volatility 因子较小的回测收益率还不错。既有的很多研究也表明和我们的回测结果相同的现象。较早有理论说，你这风险这么大，不应该给出来一点风险补偿不是，要不没人买你们家股票啊。但是，大量的经验证明，包括很多北美市场的经验（不只股票市场）也包括我们在 A 股的回测，嘿嘿，风险没有带来溢价，而低风险股票可能有更好的回报。CAPM 理论给出了一个解释，就是证券的很多风险都是可以靠合理配置被中和掉的，那么既然能被中和掉就不应该产生收益，而真正产生风险补偿的是不能被中和的部分。基于这个理论的一个猜想就是，高波动率的股票不仅承受了太多“多余”的风险，也没有获得风险补偿，所以收益自然就烂了。专业的认识给出了好多种其他猜测，也可以参见用户 Gyro 发表的量化投资趣谈：低风险股票回报更低还是更高？。

最大1%

最小1%

最大5%

最小5%

27 方差变动 $\text{VARC} = \text{VAR} < em > \text{上一期} - \text{VAR} < /em > \text{当期}$

方差变动（ VARC ）这个因子表示股票两期收益率方差的变动，是日收益率方差的变动，表述为价格变动的变动的变动，乱么？已有的研究已经开始构建更加复杂的指标了（比如一篇比较有代表性的研究，Martellini 和 Ziemann 发表在 Review of Financial Studies 上的 Improved Estimates of Higher-order Comoments and Implications）,据说因为收益率表现的不是正态分布，那么锋度（Kurtosis）和偏度（Skewness）（分别是一组数据样本的四阶条件和三阶条件下的指标）的指标可能会提供更好的预测。但是人家流氓有文化，使用复杂技术指标有理论（数学模型）推导。我们这里呢，仅构建一个方差变动指标，展示其回测现象，近给出一些逻辑描述。 VARC 较大说明收益率的方差变大了，可以理解该股风险增大，VARC 较小表明该股收益率波动变小。

通过回测我们可以发现，在最小 1% 处，无论是 VAR 变动最大和最小的股票收益率都不怎么样。可以简单理解为股票收益率上升或下降的变动在上一期已经基本完成，很可能接下来的一期该股票进入盘整阶段，那么对应下一期收益率当然会较差。对比 5% 的策略，其最小和最大策略基本上都略强于大盘，但是这样的趋势仅在最近一些年份才体现出来，在 2008 年或更早的年份就很少体现出这样的现象。对此，或许近年来对于上期的部分波动较大，但不是最大部分的股票，在下一期可能还在大家关注的范围内，利好或利空没有完全释放，在新热点出现资金转移前，仍然带动原有焦点对象有些许上涨。当然，上述的部分解释仅为合理推测，可能该因子没啥用，或者可能有用但具体的使用方法并不是简单的单因子排序策略，有兴趣的宽客可以进一步挖掘该因子的潜力。

最大1%

最小1%

最大5%

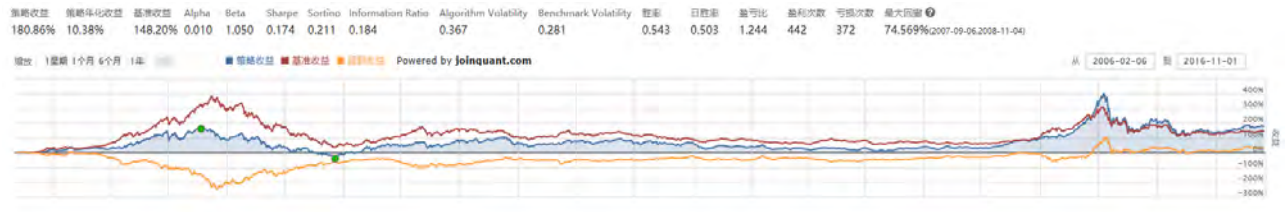
最小5%

28 震荡 $\mathrm{FLUC} = \frac{\mathrm{closemax-closemin}}{\mathrm{close期初+close期末}}$

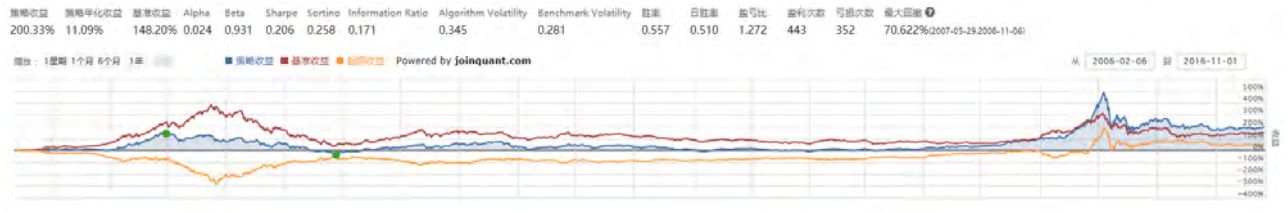
震荡变量是潘凡的一份因子分析材料中提到的技术因子，但是在其进一步分析因子有效性，以及对冗余因子进行剔除后，该指标也没有纳入其最终使用的有效因子之中。我们根据该材料给出的指标对这个技术因子进行计算。这个指标在该材料中的表述，“根据市场经验：横有多长，竖有多高。”就直觉而言，可以简单表述为月波动范围除以月平均价格乘以 2，感觉上是方差与动量两者的结合产物，。不过这个因子表现一般，这里简单展示一下。我们根据公式可以看出，该因子一定大于 0，那么该因子越大表明本月震荡幅度越大，越小表明本月震荡越小。当然这里的震荡仅是一个概述，具体月中波动的很多详情信息在这个因子中难以反映。

可以看出无论是最大还是最小策略，FLUC 因子表现的都不怎么样，就 5% 策略而言，我们看到最大 5% 策略在十年中稍微跑赢大盘一点，对应最小 5% 也差不多，本来该单因子对于回报的预测就很一般，这个指标表现不好的另一点在于最大和最小策略出来的结果没有区分度。

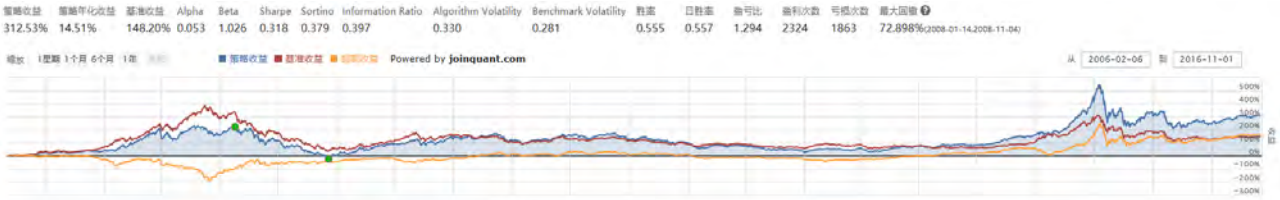
最大1%



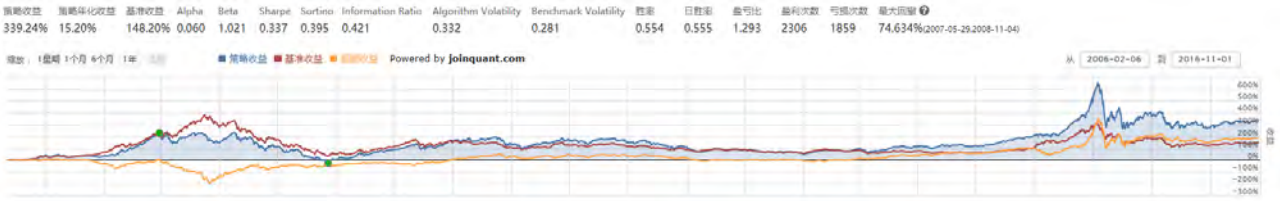
最小1%



最大5%



最小5%



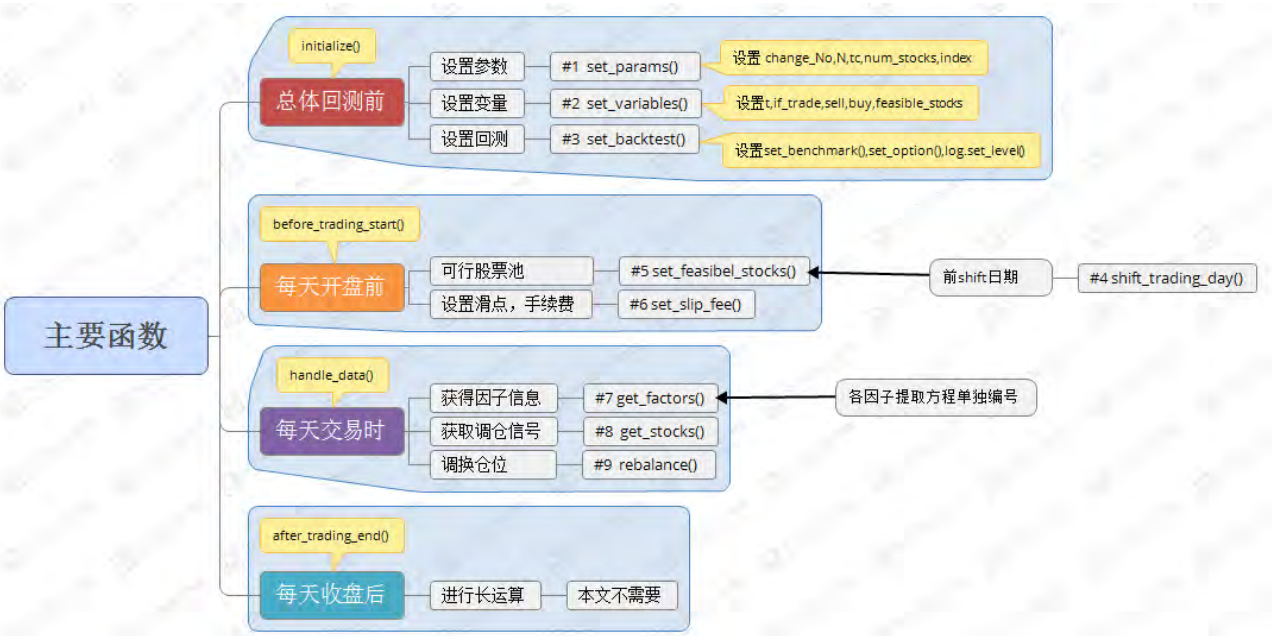
小结

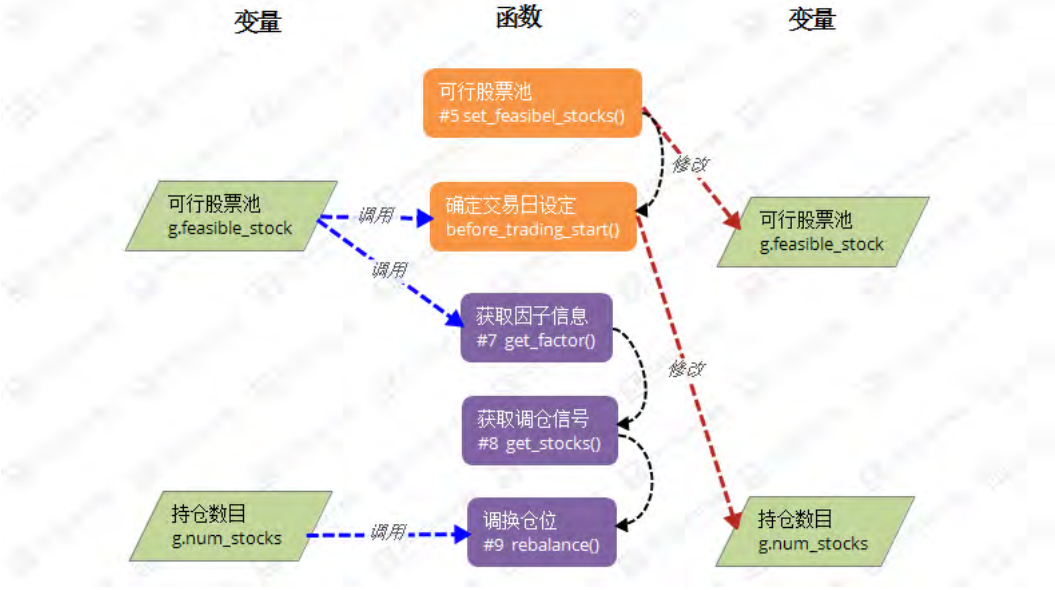
因子提取与单因子回测的系列内容这里第一个阶段已经结束，为大家展现了一些可能大家知道或不知道的现象。在代码中我们给出了每种因子的提取方式，进一步也给出了每个因子排序的指标，同时给出了多因子合并成一个DataFrame的方法，只要简单修改就能组成三篇文献中所有因子和因子排序数值的数据（可能因为避免程序矛盾部分原始因子值没有保留在最后合并DataFrame中，寻找到对应因子，删去'del'对应的语句就解决了）。

在进行系列材料整理的过程中，作者也认识到更多之前没有见到过的因子，但没有一一加入。我们会在稍后整理一下社区中较好的因子相关材料做一个汇总，各位朋友可以在本文后留言附加链接推荐自己或他人的因子相关内容的帖子。从数据而言，聚宽仅股票财务数据就包括市值数据（12项）、资产负载数据（84项）、现金流数据（57项）、利润数据（41项）和财务指标数据（34项），这些都可以作为因子，而且可以根据个人理解生成其他因子，有兴趣的朋友可以广泛尝试。

单因子的回测结果暴露出单因子策略的很多问题，比如很多因子的识别过程可能和产业、板块或市值相结合，再比如很多因子的表现并不是线性的，仅知道因子最大和最小的 5% 的回测效果并不能认识它的全貌。这也是不少宽友们给出的建议，我们会在之后的系列文章中，结合当前的一些主流处理方法，进一步展现因子在历史数据中的表现细节。以前留言的内容我们已经记录，在下一阶段的因子处理中各位朋友还有什么建议可以在本文后留言，我们在能力范围内尽量实现；）。

函数和变量说明书





v1.0，2016-11-30，文章上线

【量化课堂】基于修正TD指标的指数择时策略

前言：基于修正TD指标的指数择时策略，TD指标策略是大型投资基金Tudor的执行副总裁（Thomas R. DeMark）于80年代中期为了发现走势欲转折区域而设计的技术分析方法。因为其原理简单且预测精度高，而得到了广泛的使用。

本文由JoinQuant量化课堂推出，难度为入门，深度为 level-0。

作者：肖睿，swlaw
编辑：肖睿

TD指标基本原理：

TD指标策略分为TD序列和TD组合两种，原理基本相同但在细节上有一些差异，在应用中也是相互独立的，孰好孰坏并不好说。它们的基本原理如下:市场由买方卖方共同作用形成，当买方力量大于卖方力量时，指数表现为上涨，当卖方力量大于买方力量时，指数表现为下跌。
在走势上涨一段时间后，之前占优的一方现在逐渐走向疲软，从而转向下跌。TD指标正是为了发现走势转折区域而设计的。
TD序列和TD组合分别由启动和计数两个阶段构成。其中启动阶段的作用是识别股价的涨势或者跌势，在确认趋势之后开始统计上涨或下跌的天数，并且当计数达到一定程度之后我们认为趋势邻近终结，反转即将发生，所以做出相应的买入或卖出。
经过一定的实证分析可以得出，传统的TD指标并不是很适合我国市场预测，因此直接运用于我国市场时需要做一定的调整。传统TD指标最大的意义也在于为我们提供了一个识别股市走势的有效方法，我们可以根据要预测的对象的不同而对参数做一定的调整，接下来介绍的是经过修正并且可以适用于A股市场的版本。

修正TD指标详细逻辑：

启动阶段：

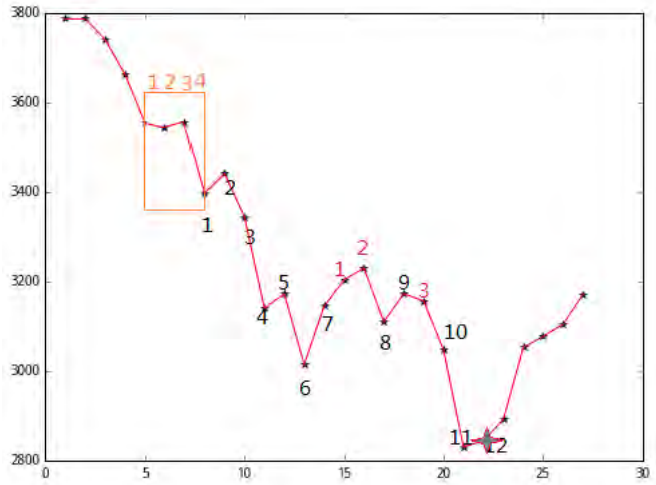
TD序列和TD组合的启动阶段是相同的。首先固定一个间隔参数T和统计天数N，当连续N个交易日的收盘价都比T天前的收盘价低的话，我们将它识别为一个下跌趋势，这时启动买入计数；反之如果连续N个交易日的收盘价都比T天前的收盘价高，是一个上涨趋势，那么启动卖出计数。

计数阶段：

当启动阶段确认了趋势之后即开始计数阶段，这个阶段的计数方法是TD序列和TD组合的差异所在，先来看TD序列的计数法。设一个参数M，并且每个交易日进行两组计数：
买入计数1：若当前K线的收盘价比之前第二根K线（也就是隔一天）的收盘价低，则第一个买入计数加1。
买入计数2：若当前K线的收盘价比之前第二根K线的收盘价高。则第二个买入计数加1。
当买入计数1达到M的时候，说明股价在确认下跌趋势之后又下跌太多，应该很快就会发生反弹，所以这时应该买入。或者当买入计数2达到M/2的时候，我们认为这些上涨的天数意味着之前的下跌趋势已经变得疲软，股价应该开始反转向上，所以这时也应该买入。因此，买入点正是计数1先达到M或者计数2先达到M/2的时候。
相应的，当启动阶段识别了上涨趋势之后，我们开始卖出计数，和买入计数正好相反
卖出计数1：若当前K线的收盘价比之前第二根K线的收盘价高。则第一个卖出计数加1。
卖出计数2：若当前K线的收盘价比之前第二根K线的收盘价低。则第二个卖出计数加1。
并在计数1先达到M或者计数2先达到M/2的时候卖出股票。

TD组合的计数规则要更复杂一些，首先我们将计数阶段的第一天的收盘价记录为P，然后每个交易日当下面a、b、c条件之一满足时在相应的计数上加1，如果计数1和计数2都有条件满足，那么只增加计数1不增加计数2：
买入计数1：a.收盘价小于或等于之前第二根K线最低价；b.最低价小于之前第一根K线的最低价；c.收盘价小于P。
买入计数2：a.收盘价大于或等于之前第二根K线最高价；b.最高价大于之前第一根K线的最高价；c.收盘价大于P。
卖出计数1：a.收盘价大于或等于之前第二根K线最高价；b.最高价大于之前第一根K线的最高价；c.收盘价大于P。
卖出计数2：a.收盘价小于或等于之前第二根K线最低价；b.最低价小于之前第一根K线的最低价；c.收盘价小于P。
和序列版一样，当计数1达到M或计数2达到M/2的时候执行相应的买入或卖出。

示例：



我们以TD序列为例，如上图所示，假设启动阶段的参数为间隔T=4以及N=4，图中橙色方框是连续四天里收盘价都比四天前的要低，于是判定为下跌趋势并且开始计数阶段。在那之后的每一个黑色数字是买入计数1，也就是收盘价低于隔一天前的收盘价（注意要隔一天）；红色数字是买入计数2。如果我们设M=12，那么在图中大四角星的位置正好满足计数1的条件，此时执行买入；如果是红色数字先达到M/2=6的话，也在那时进行买入。

参数的选择和止损条件：

你也许想问，上面为什么把M设为12，如果设成13的话不就赚不到钱了吗？的确是这样的。但我们应该认识到，技术分析并不能保证完全正确的判断，它可以提供的是一个可能有良好胜率的分析思路，具体参数定为多少应该通过历史回测去尝试（比如原版TD指标的参数就适用于美股市场但在A股回测很差），并且可以设置一些止损方案来应对判断错误的情况。

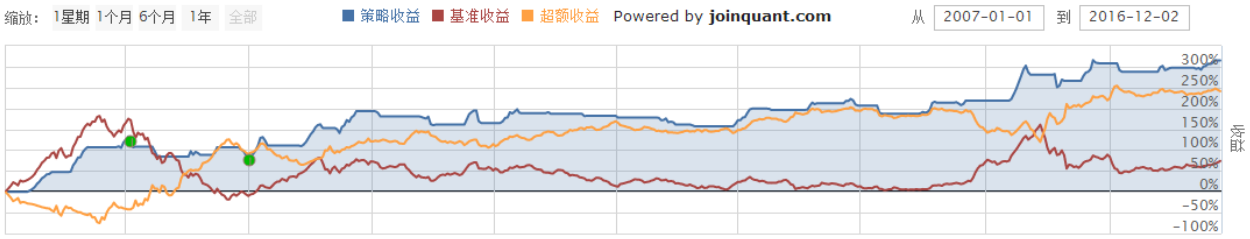
通过各种参数的尝试，我们发现设T=4、N=6以及M=28，从07年到16年12月可以持续提供稳定的收益率。并且，为了使策略表现的更加稳健，我们增加了一个止损条件。其思路较为简单，即在进行买入前的计数阶段里，记录下在计数周期内的最低交易价格，作为止损触发条件。在我们达到安全买点并持仓后，每个交易日进行一次判断，看上一日的K线最低价是否跌破了这一最低价。如果跌破了，则为了避免更大的损失，我们提前将所持仓位平仓以避免损失。

回测结果：

（一）先以TD组合为例，以沪深300指数作为交易对象：

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比
314.89%	15.89%	72.90%	0.114	0.247	0.788	1.086	0.211	0.151	0.304	0.581	0.506	3.104

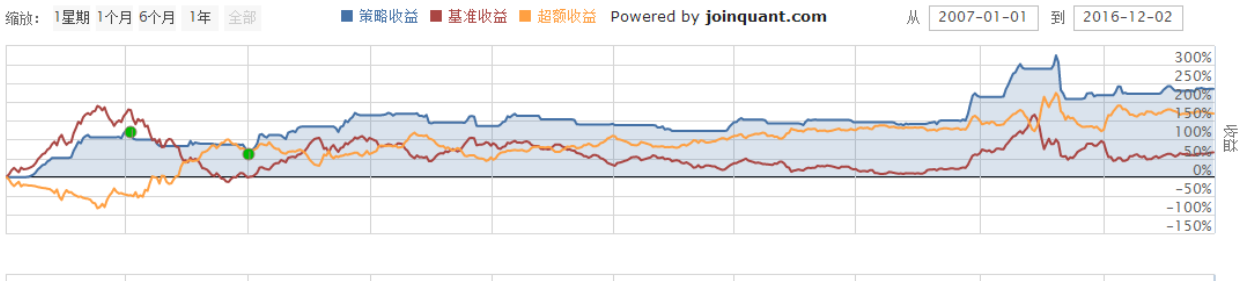
盈利次数	亏损次数	最大回撤
18	13	24.851%(2008-01-14,2008-12-31)



整体来看，TD组合策略近10年年化益率达到了近16%，而最大回测仅为24.8%，且避开了07年和15年的大熊市，表现出了不错的稳定性。
下面以深成指为例进一步说明：

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比
235.44%	13.37%	64.17%	0.090	0.269	0.558	0.756	0.130	0.168	0.323	0.581	0.486	2.180

盈利次数	亏损次数	最大回撤
18	13	30.191%(2008-01-14,2008-12-31)

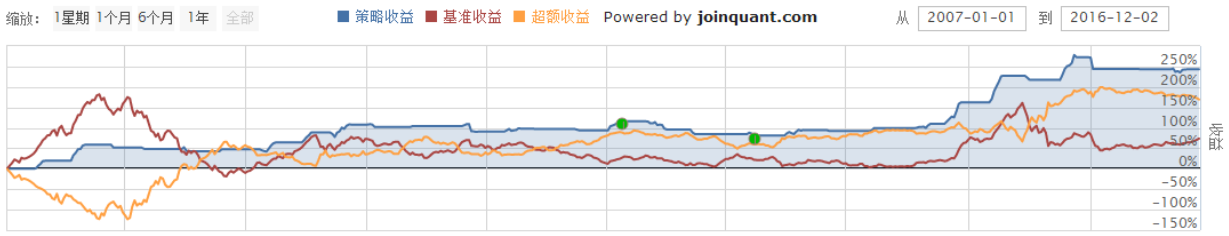


回测比沪深300指数大了一些，但在熊市的表现整体还令人满意，结果还比较能够让人接受。但需要注意的是对于其他一些指数或者股票组合可能上述参数并不是最优的，需要自己去调参以寻找最优的参数组合。

（二）下面看一下TD序列回测的结果，先以沪深300为例（TD序列代码）：

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比
243.18%	13.63%	72.90%	0.093	0.155	0.804	1.209	0.114	0.120	0.304	0.500	0.472	3.087

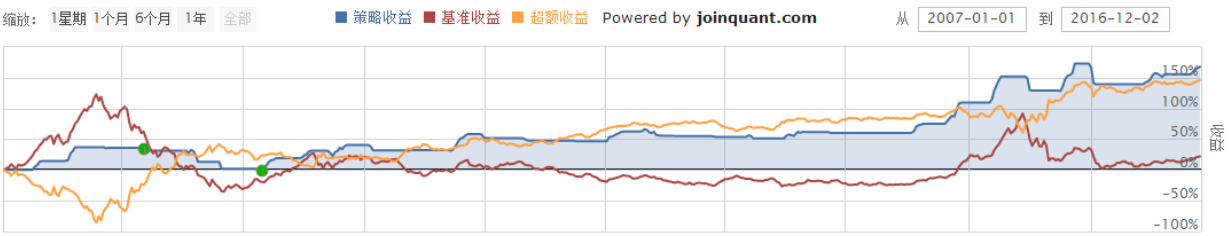
盈利次数	亏损次数	最大回撤
15	15	16.177%(2012-02-21,2013-04-01)



虽然收益不如TD组合的高，年化收益为13.6%，但优点是最大回测小，10年间最大回测为16%，且经受住了两次牛熊市的考验。再以上证指数为例：

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比
166.06%	10.67%	21.24%	0.070	0.146	0.611	0.874	0.177	0.109	0.285	0.500	0.462	2.252

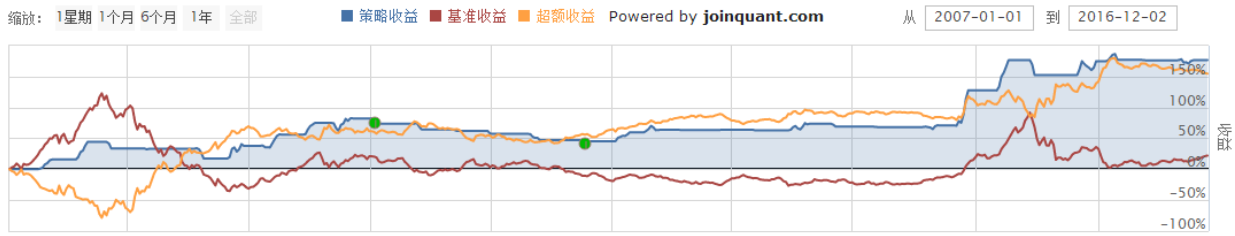
盈利次数	亏损次数	最大回撤
15	15	28.942%(2008-03-03,2009-03-03)



回撤比沪深300要大，但总体表现还能够接受。为了更贴合实际，我们用华夏上证50ETF（510050.XSHG）代替上证指数，再做一次回测：

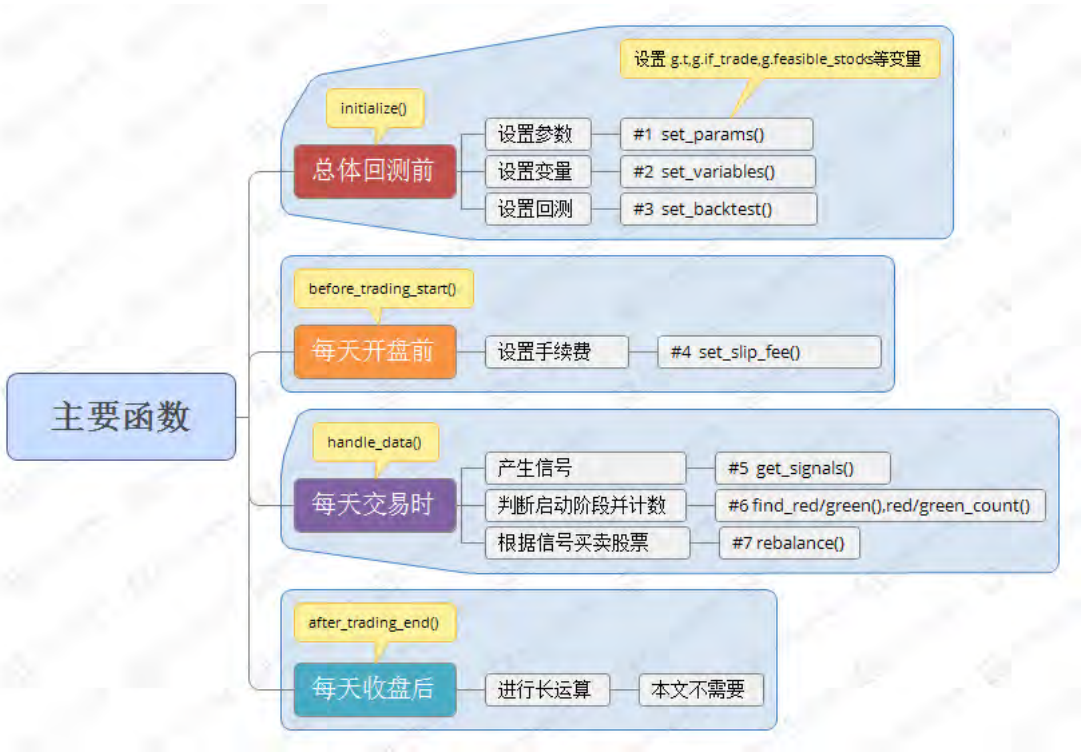
策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比
176.78%	11.13%	21.24%	0.075	0.191	0.525	0.831	0.206	0.136	0.285	0.516	0.471	2.920

盈利次数	亏损次数	最大回撤
16	15	21.589%(2010-01-12,2011-10-10)



可以发现在以上证50ETF为交易对象时，取得了更优异的表现，10年间最大回测仅为21.6%。

TD序列/组合函数示意图：



变量说明图：

文章更迭记录：
v1.0，2016-12-05，文章上线

【量化课堂】股指期货跨期套利策略

概述：本文章介绍使用同一标的不同交割日的股指期货的价差进行跨期套利的策略。

本文由JoinQuant量化课堂推出，难度为进阶（下），深度为 level-0。

作者：swlaw
编辑：肖睿

建议读者掌握协整和线性回归的相关知识。

引言

沪深 300 股指期货推出已久，利用它来进行套利交易是相关理论研究者和实际工作者非常关注的问题。N.Burgess 利用协整模型对 FTSE100 指数进行套利取得了良好的效果， Board 和 Sutcliffe 利用协整方法对大阪、新加坡和芝加哥的日经 225 指数合约之间的价差套利研究表明也存在套利交易空间。同时协整分析在股票指数间的长期平稳关系或期货保值中也得到广泛应用。

股指期货的两个不同期货合约因为对应的同一个股票指数，所以存在着长期协整关系的基础。依据配对交易的思想，基于协整的股指期货套利的核心在于准确发现价差交易出现的时机和概率，而本文可应用协整方法来构建不同到期月份合约价格序列的长期均衡关系，估计价差序列的分布，从而制定恰当的价差交易策略。

正是由于上述原因，本文利用二元协整的跨期套利策略对沪深 300 指数期货做相关的跨期套利研究，结果表明，本文提供的跨期套利策略具有可以发现更多的套利机会且风险可控等方面的优势。

股指期货及跨期套利策略与模型

股指期货指以股票价格指数为标的物的期货合约。由已有的金融文献中持有成本定价原理知，股指期货定价公式为:

$$F = Se^{(r-q)}(T-t) \qquad (1)$$

其中: F 表示股指期货在时间t时的价格; S表示现货指数在时间t时的价格; r 表示以连续复利计算的无风险利率; q 表示股息收益率; T 表示期货合约到期时间(年); t表示现在的时间(年)。

一般来说，基于相同标的股票指数的股指期货在市场上会有不同交割月份的合约同时交易。而股指期货的跨期套利，就是指利用基于同一股票指数的两个不同交割月份的股指期货合约之间的价差进行的套利交易。严格来讲，跨期套利不是无风险套利，它实际上属于价差套利交易，所以其操作重点在于判断不同交割月份合约的价差将来是扩大还是缩小。

依据对不同交割月份合约价差未来走势的判断，可将跨期套利的策略划分为三种:

` 牛市(多头)跨期套利:即判断远期合约相对近期合约被低估，（设远期合约价格为 F_2， 近期合约价格为 F_1，则 $F_2 = S_0 \cdot e^{(r(T_2-T_0))}$，$F_1 = S_0 \cdot e^{(r(T_1-T_0))}$， 此时 F_2 偏小，则存在套利空间，在套利的驱使下，F_2 将回归正常，即 F_2 将变大）价差将扩大，我们可以买入远期合约的同时卖出近期合约。

` 熊市(空头)跨期套利:即判断近期合约相对远期合约被低估，价差将缩小，我们可以买入近期合约的同时卖出远期合约。
在本文中讨论多头与空头策略，而上述策略的实现则需要跨期套利模型来提供精确的标准来进行。

传统的股指期货跨期套利依据股指期货的持有成本定价公式产生的无套利空间确定跨期套利策略。如现有近期合约 F_1 和远期合约 F_2，则依据式 (1) 有:

$$\ln F_2 - \ln F_1 = (r-q)(T_2 - T_1) \qquad (2)$$

如不考虑交易费用， 则 $(r-q)(T_2-T_1)$ 是两对数序列的价差的平衡点，则当这两对数序列的价差大于 $(r-q)(T_2-T_1)$ 时，说明价差偏大，价差将缩小，则实行空头策略，反之，则实行多头策略。然而这一模型存在如下缺陷:
①基于持有成本理论中的股息收益率 q 不易确定，限制了其应用;
②基于持有成本理论的套利往往需要经历较长时间才能完成，因为它的价差会一直处于偏高或偏低的状态，只有在合约临近到期时才有回归的压力。

使用协整模型

基于协整的跨期套利模型则可以避免上面的缺陷，并且可以充分利用已有的市场交易数据所提供的最新信息，从而发现更多的套利机会。其基本思路如下:

假设现有一段时间内的两期货合约序列分别设为近期合约 F_1 和远期合约 F_2，本文先将这段时间分成两段(前一段时间较长)，然后以第一段数据建立模型，第二段数据以第一段数据建好的模型为依据来进行交易。

一、在第一段数据中，先对两合约序列取对数为 $\ln F_1$， $\ln F_2$，则这两对数序列的协整关系处理可分为两步：首先，检验 $\ln F_1$， $\ln F_2$ 是否存在单位根；其次，若这两对数序列都存在单位根，那么就检验它们是否存在协整关系，即对它们建立回归方程（A, B 代表方程回归系数，$resid1$代表回归残差项）：

$$\ln F = A + B\ln F_1 + resid1 \qquad (3)$$

若 $resid1$ 不含单位根为平稳序列，则两对数序列存在协整关系。

二、接着将这一协整模型结果代入到第二段数据中，并设在这一时点满足上一段时间的回归关系，即:

$$resid2 = \ln F_2 - \mathrm{B}\ln F_1 - \mathrm{A} \qquad (4)$$

而第一段数据中建立的协整模型的残差为 $resid1$，其样本标准差为 $\mathrm{std}(resid1)$，则再设:

$$resid3 = resid2/\mathrm{std}(resid1) \qquad (5)$$

三、本文可以认为在第一段时间内建立的模型在第二段时间内依然成立，所以 $resid2$ 和 $resid1$ 有相同的分布，且都是均值为 0 的白噪声。由此可知，$resid2$ 时刻存在着向均值 0 回归的内在要求，而不像基于持有成本理论定价的套利策略只有在合约临近到期时价差才有回归的压力，这样的特性使得可以发现更多的套利机会。当 $resid2$ 的绝对值超过一个样本方差时（也就是当 Misplaced & 时），可以认为是一个较好的套利机会，但当 $resid2$ 的绝对值超

过两个样本方差（也就是当 Misplaced & ）这样的小概率事件发生时，则往往意味着这两对数序列的协整关系已经不再成立。综上所述，本文可将开仓平仓的标准如下设定:

` 当 $resid3$ 的绝对值超过 1 且小于 2 时开仓;
` 开仓后当 $resid3$ 回落到 0 值时平仓;
` 开仓后当 $resid3$ 的绝对值超过 2 时认亏平仓。

这样的开仓平仓标准可以确保两对数序列的价差在可控的范围内，而不像基于持有成本理论的套利策略那样:开仓后价差不在可控范围内，可能会向不利方向发展过大从而导致爆仓。

策略交易过程

我们以 10 年 7 月 1 日 14:21 这一时间点为例来介绍一下上述策略的执行过程:
（1）我们在每一个分钟数据上取过去 240 分钟的 IF1007 和 IF1008 的收盘数据F_1, F_2，对上述数据取对数，得到 $\ln F_1$ 和 $\ln F_2$。利用 ADF 检验判断 $\ln F_1$ 和 $\ln F_2$ 是否为平稳时间序列:

Unknown environment 'align'

由上可知 $\ln F_1$ 和 $\ln F_2$ 均为非平稳序列。

（2）对 $\ln F_2$ 和 $\ln F_1$ 做回归可得:

$$\ln F_2 = 0.49513029 + 0.93776831 \ln F_1 + resid1$$

对 $resid1$ 进行 ADF 检验:

Unknown environment 'align'

可以发现残差项此时的 p 值仅为 0.000412，远小于 0.01 的临界值。此时我们认为两个对数序列满足协整关系，接着我们可以利用这一协整关系来进行交易。

（3）接下来这段时间内我们可以利用上述选择的标准进行交易。首先我们认为这段时间内协整关系依然会保持下去。因此当两股指期货价格出现偏离时，其偏离最终会回归到 0。具体交易思路如下，在每个交易分钟开始时我们取上一分钟的两股指期货的收盘价的对数，记为 $\ln F_1$ 和 $\ln F_2$，将其带入步骤（2）中得到残差项 $resid2$:

$$resid2 = \ln F_2 - (0.49513029 + 0.93776831 \ln F_1)$$

将这一残差项标准化为 $resid3 = resid2/\mathrm{std}(resid1)$；则 $resid3$ 有向 0 均值回归的特性。根据这一特征我们设计了下面的交易策略:

1， 在 14:21时，如果 Misplaced & 且 $resid3 < 2$，说明远期的股指期货相对近期的股指期货价格偏高。因此进入近期期货合约的长头寸同时进入远期期货合约的短头寸。反之，如果 $resid3 < -1$ 且 Misplaced & 时，按照和上述相反的头寸进行建仓。由于过大的交易量在实际交易中会造成较大的冲击成本，为了反映现实情况我们将交易量限制在下一分钟两个期货合约交易量的十分之一。

2， 在 $1 < resid3 < 2$ 的情况下建仓后，如果当 $|resid3| < 1$ 时，我们认为 $resid3$ 向均值 0 回归，此时我们将在 (1) 中建立的仓位平仓。当 Misplaced & 时我们认为判断出现错误，$resid3$ 并没有按照预期回归到 0，此时为了避免更大的损失我们提前平仓止损。

3， 在 15 年 9 月 7 日之后因为交易数量有限制，最多只能交易 10 日，且平今仓手续费变为原来的的 100 倍因此我们此处严格按照这一限制进行仓位操作。

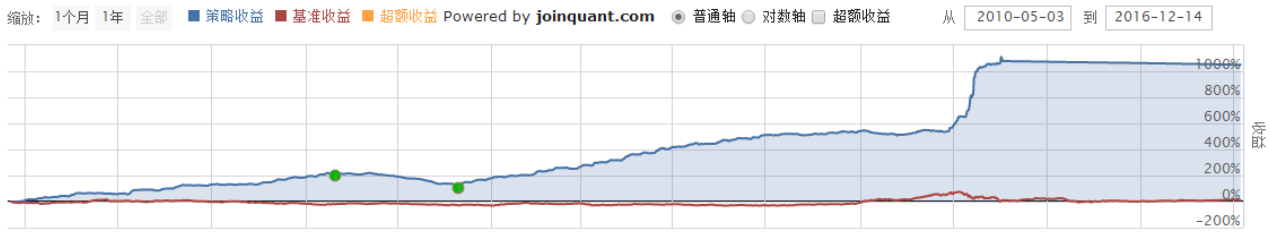
回测结果

在以下的回测中为了更贴近现实，我们回测中选取中国金融期货交易所中公布的手续费率，保证金比例以及平今仓手续费率。

首先看一下从 10 年 5 月到 16 年 12 月之间的回测结果:

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
1049.69%	46.15%	10.16%	0.421	-0.010	3.563	8.384	1.211	0.118	0.252	0.538	0.515	1.278	22530

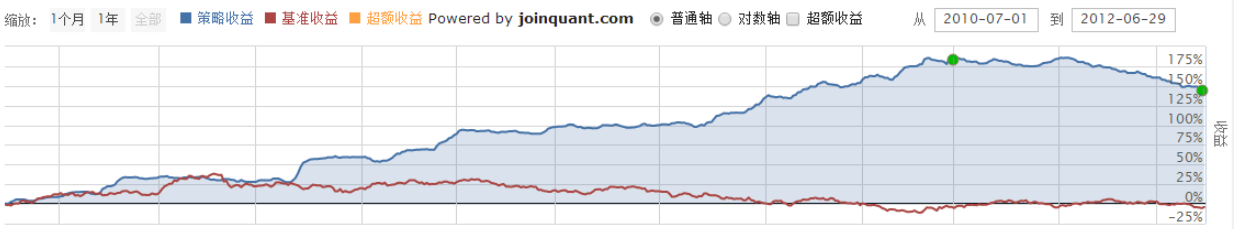
亏损次数	最大回撤
19358	29.470%(2012-01-31,2012-09-21)



可以发现，这一股指期货的跨期套利策略还是比较有效的，但 15 年 9 月政策收紧后，套利空间基本不存在了（尤其是由于平今仓手续费过高的原因实际已经接近 T+1 制度）。可以看出这一策略还是能准确的利用股指期货价格的偏离，将风险控制在一个较低的水平。再来分段看下:

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
148.18%	59.77%	-3.96%	0.556	-0.028	4.395	9.196	1.839	0.127	0.218	0.557	0.528	1.437	10075

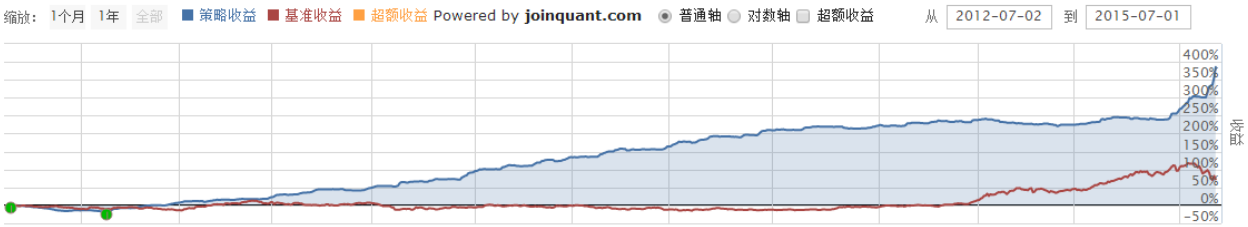
亏损次数	最大回撤
8006	13.989%(2012-01-31,2012-06-28)



可以发现随着市场的成熟，在12年1月到12年6月这段时间，套利的空间比较少，并且因为是高频交易产生了很大的手续费，因此出现了一定的回撤。

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
385.40%	71.90%	72.77%	0.674	0.030	5.832	15.713	1.288	0.116	0.238	0.523	0.539	1.247	11268

亏损次数	最大回撤
10264	18.101%(2012-07-02,2012-09-21)



从12年到15年政策收紧之前，整个策略还是有效的，三年间最大回撤为18%，且在15年这段时间内仍然有很大的套利空间存在。

需要注意的是在本策略中，因为是按分钟数据进行交易，交易的频率偏高，因此对手续费异常敏感，这也是政策收紧调采用调高平今仓手续费的原因之一。

函数说明图

文章更迭记录：
v1.0，2016-12-30，文章上线

【量化课堂】因子研究系列之四 -- 市值与行业的中性化

导语：本文给出以BP（账面市值比）为例的因子缩尾处理和对市值与行业中性化的处理实现代码，展现单因子不同分位处在不同处理方法下的表现。

作者：Sunx
编辑：肖睿

本文是一系列因子研究中的第四篇文章。本系列的文章有：

1. 【量化课堂】因子研究系列之一 -- 估值和资本结构因子
2. 【量化课堂】因子研究系列之二 -- 成长因子
3. 【量化课堂】因子研究系列之三 -- 技术因子
4. 【量化课堂】因子研究系列之四 -- 市值与行业的中性化

引言

因子系列文章首先展示了不同因子的构建与端点处的股票策略回测结果，本文借助【量化课题】多回测运行和参数分析框架中给出的分析方法和框架，展示对因子按照市值和行业进行标准化和去极值化的方法和回测结果。

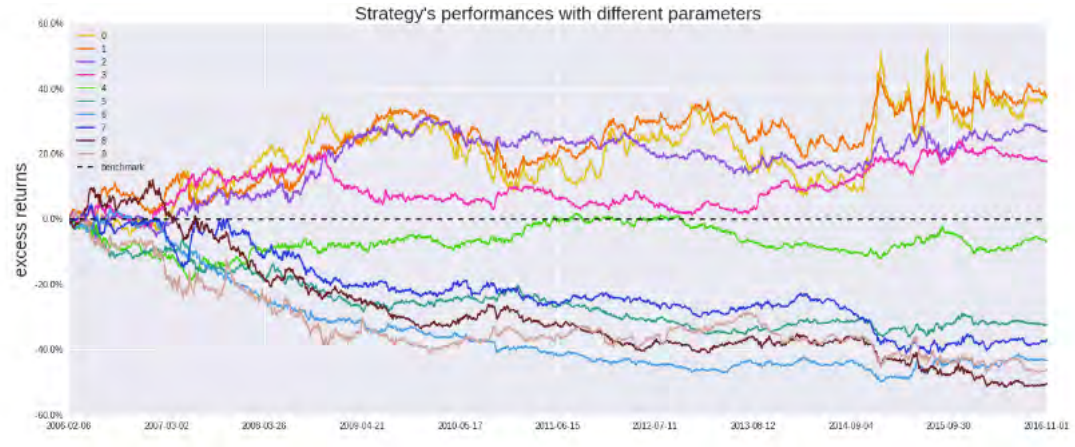
单因子的对比

如因子研究系列一~三篇中所示，在单因子策略中我们把股票池中股票按照某个因子从小到大排序，因为我们认为因子指标跟高或更低的股票会取得超额的收益。但很多时候用两支股票的同一因子指标在未经处理的情况下缺乏可比性。一个非常明显的例子就是 BP 值（账面市值比），银行股的 BP 一般平均大于 1，而互联网相关服务行业近一年平均 0.1。所以如果只是认为高 BP 就是估值过低，那么就只会买进银行股而不买科技股，但这很明显不是合理的投资思路。同理的，一般大盘股 BP 值都比小盘股要高，但我们并不能根据 BP 的估值方法来断论小市值股票的收益率低于大市值股票，毕竟事实是恰好相反的。

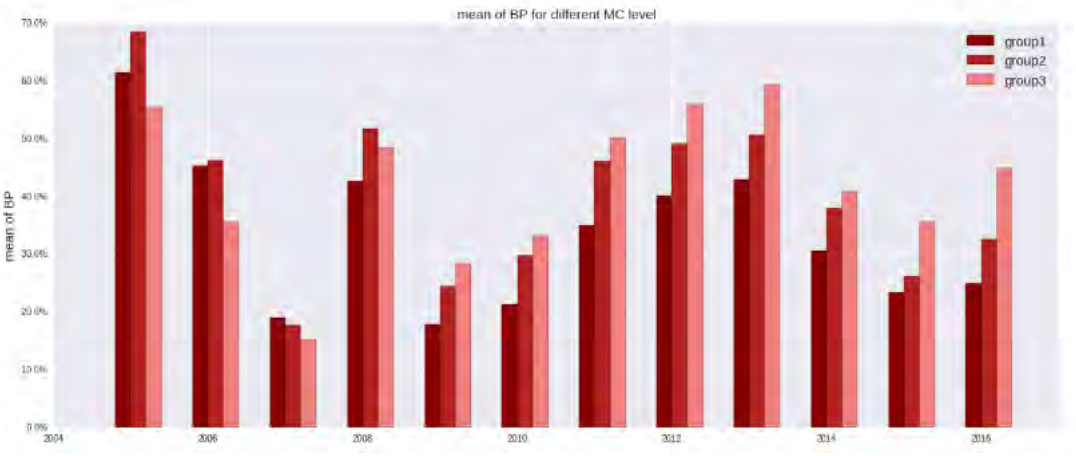
对单个因子进行分析，我们会发现有一些额外的因素影响了该因子的取值，比如在 BP 值的例子中，投资者对不同行业和规模的上市公司的成长预期以及市场的热点影响了 BP 的取值。我们要做的是把受到影响因素相同的股票归为一堆，然后在每一堆上进行中性化和去极值的处理，这样受到影响不同的股票之间就可以进行相互比较。接下来我们就以 BP 值为例介绍具体的处理方法，并可通过回测看出处理后的因子策略更为有效。

市值

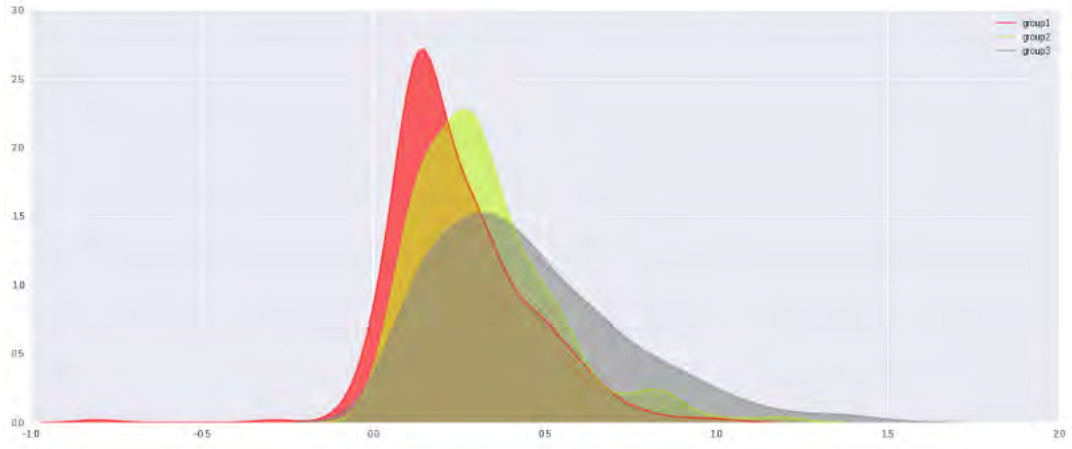
首先，我们注意到【量化课题】多回测运行和参数分析框架中给出的 BP 因子分位回测结果中，虽然从十年回测结果上看 BP 因子不同分位回报率还是比较有区分度，但是查看具体的历史走势显得凌乱混杂。下图是对比等权全指为基准的 BP 分位回报率超额收益（2006 年 2 月 1 日到 2016 年 11 月 1 日，两百万，持仓一个月每月月初调整，具体见链接）。



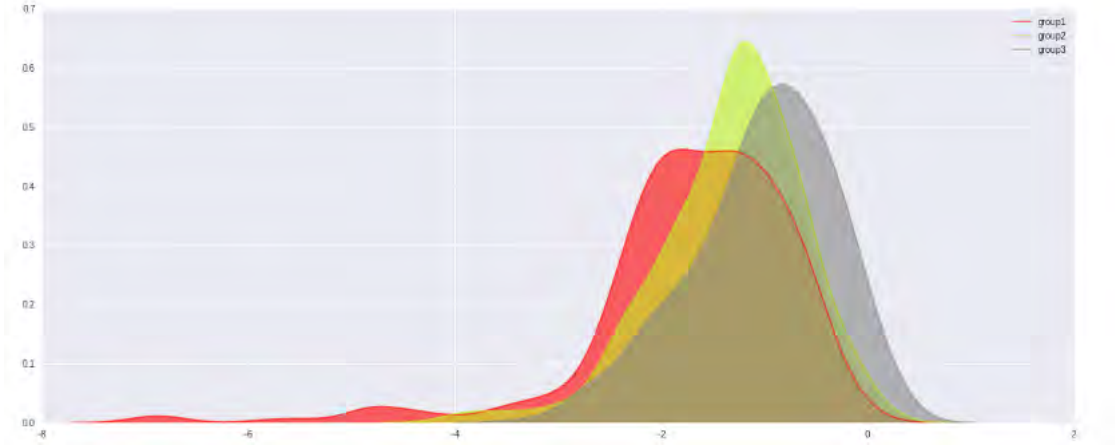
正如上面提到的，对于 BP 因子来说，常见的现象就是其分布在不同行业和不同市值的公司存在差异。下图展现的是从2005年到2016年每年年底按照总市值大小分成三种后每组的 BP 均值，其中 group1 是小市值股票，group2 是中市值，group3 是大市值股票。跨年比较的数据表明 BP 指标跟大盘的变动也挺相似，在牛市和熊市变动间不断转换，牛市平均 BP 低于熊市平均 BP。 同时我们可以看出，三个市值组合的平均 BP 相对排名在不同年份中是有变动的，在早年市场对于大盘股较为追捧，估值高 BP 相对小市值较低。从 2008 年之后，趋势发生了变动，小盘股更被关注，估值相对较高 BP 相对较低。感兴趣的朋友可以从相对变动的排名和 BP 值中挖掘处更多的信息。



以 2016 年为例，2016 年年底的 BP 按照不同的市值分组的分布情况，根据分布情况我们不止能得到均值的信息，而且能够分析出更多内容。根据分布情况我们可以直观的了解到不同市值分组的 BP 一、二、三、四阶矩都不同，反应为均值、方差、峰度和偏度都存在差异。看起来像是对数正态分布，但实际上我们试过对数转换后发现起始看起来离正态分布也差很远，因此没有使用对数值为基础进行随后的处理，而是使用 BP 的原始值。下图是使用默认设置的核函数绘制的不同市值水平 BP 的分布图。

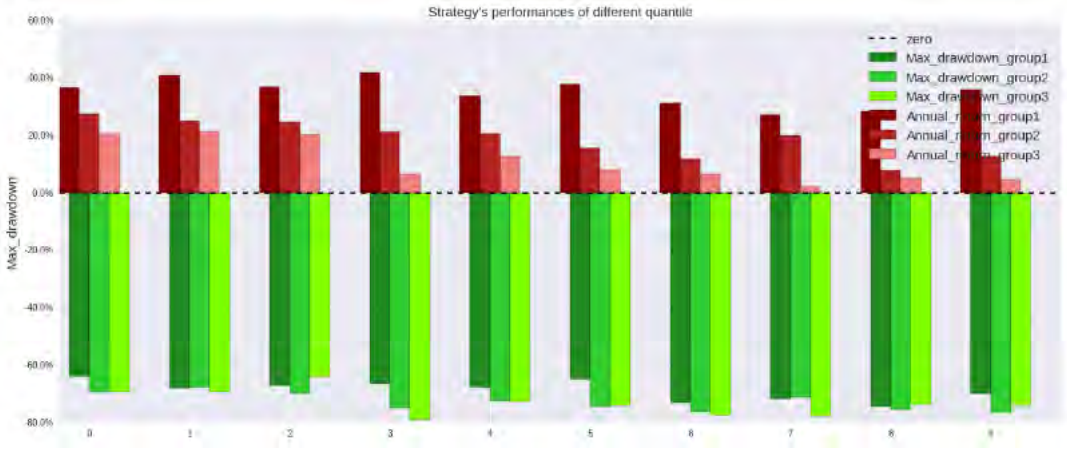


分布的对数处理后效果

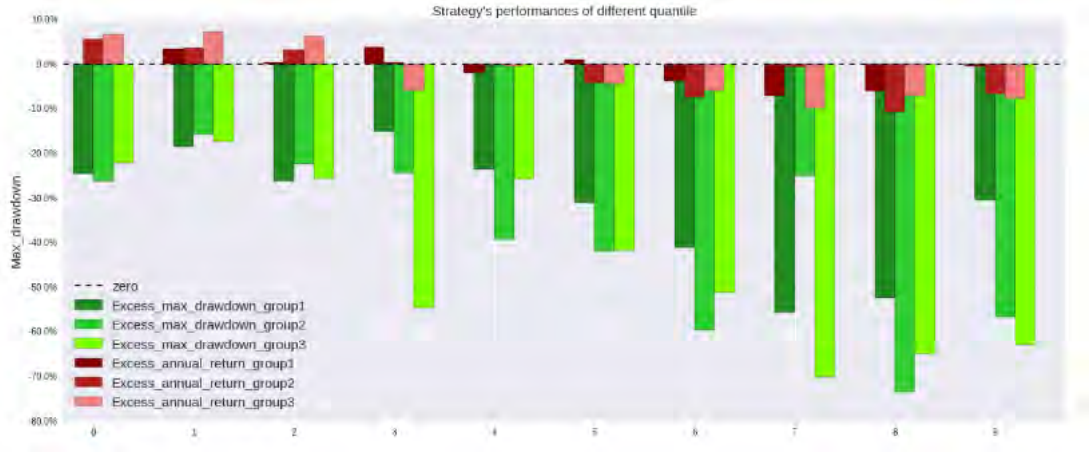


我们接下来要在大、中、小市值股票上分别对 BP 因子进行检测，为此应该分别对大、中、小市值的走势有一个评价标准，所以要构建三个分市值上证全股的回测作为基准。具体回测策略如下：按照当天的市值进行排序分三组，每月初进行调仓持有相同金额一个月上月没有停牌的该组内所有股票，从 2006 年 2 月 1 日开始到 2016 年 11 月 1 日，初始资金 2000000。下图我们可以看出三种不同市值股票的变动情况。从图形中三种策略开始区分我们对应到上图中几个组间 BP 相对排序变化的时点，在 2008 年之后开始，与普遍观测较为一致，小市值带来了很高的收益，尤其是相对于中等市值和大市值股票而言。

根据上图的三个回测结果为新基准，我们引入【量化课题】多回测运行和参数分析框架进一步分析。首先给出一个总体的表现汇总，每个市值的不同分位回报率和最大回撤的结果。策略为：按照当天的市值进行排序分三组，在每个总市值分组中再对 BP 按大小分10组，每月初进行调仓持有相同金额一个月上月没有停牌的该组内所有股票，从 2006 年 2 月 1 日开始到 2016 年 11 月 1 日，初始资金 2000000，使用的是上证的股票。下图中 Max_drawdown 和 Annual_return 分别表示最大回撤和年化收益率，group1、group2 和 group3 分别表示小、中和大市值回测结果。我们可以看出各种表现里小市值具有明显高于另外两组的收益，无论从哪一个 BP 分位而言，同时其也具有相对于其他市值分组更小的最大回撤。



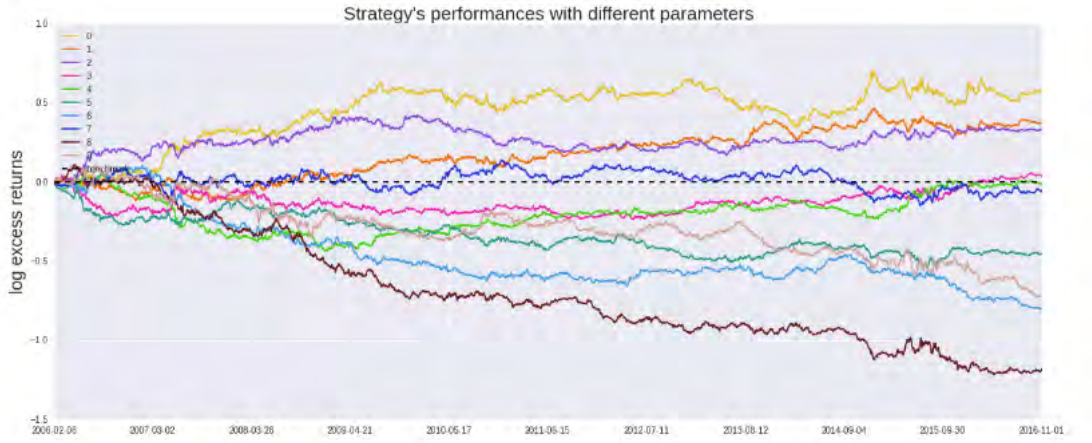
结合之前等权的分市值回测结果新基准，超额收益和超额收益的最大回撤率分别如下图所示。其中 Excess_max_drawdown 和 Excess_annual_return 分别表示超额收益率的最大回撤和年化超额收益率，group1、group2 和 group3 分别表示小、中和大市值回测结果。首先，可以看出不同市值水平下，BP 因子还是具有相似的逻辑，BP 因子越大回报率相对较高。其次，高 BP 股票结合新基准对冲之后的超额收益率具有较低的最大回撤，这是我们很高兴看到的。第三，结合不同市值分组的高 BP 因子分位表现，我们可以发现小市值结合 BP 后并没有特别出色的超额收益表现。这说明小市值股票整体具有较好的走势，而小市值中的 BP 因子却不能带来更好的超额收益。相反，在大市值股票中，BP 具有很好的超额收益。



下图的回报率曲线仅展示针对自定义基准超额收益对数轴的图例（超额收益和对数轴的细节请查看【更新说明】新超额收益 和 对数轴）。从上到下分别是小市值、中市值和大市值。

小市值

中市值

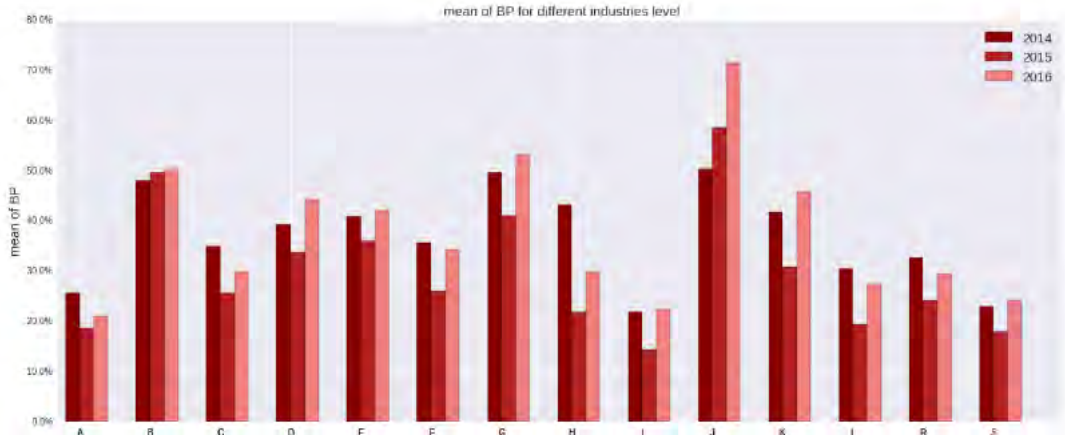


大市值

通过三个市值对数轴的超额收益我们可以看出，虽然不同市值最后的年化收益因子分位区分度较为明显趋同，但是具体的时序表现而言，不同市值因子表现差异巨大。BP 本质是估值因子，高 BP 分位策略就是持有 BP 因子较高股票，剔除持仓中 BP 因子较低股票，整体反映了价值投资的逻辑。对数轴超额收益最直观的解读是其数值大于零表示超额收益为正，超额收益对数值变大（变小）表示相对于基准收益差异变大（变小）。那么三个市值的超额对数收益率的表现如下：在大市值中高 BP 策略收益相对于大市值等权收益持续扩大；小市值高 BP 策略对于小市值等权收益的超额收益主要产生于 2011 年之前，后来没有扩大的趋势反倒发生了回撤，并且 BP 最大的小市值股票完全没有产生任何的超额收益；中等市值中大小市值中存在的现象并存。结合时间趋势可以看出，市场早期（2011 年以及之前）对于高 BP 的高额回报存在与大中小市值中，但中小市值的回报更多。这体现了早期市场中对于 BP 估值因子体现的公司价值投资的认可；在 2011 年之后的市场对于 BP 的高额回报更多体现在大市值中，小市值股票体现出了炒作影响，高 BP 策略已经难以带来持续的超过小市值等权带来的平均水平收益了。

行业

单因子结合行业的分析是另一种常见的分析框架。在聚宽平台中，我们分别有证监会行业分类和聚宽的行业分类。我们提取了 2014 年到 2016 年年底各行业的 BP 数据生成了下图，其中每个字母表示我们根据证监会行业分位的首字母把细分行业合并成上级行业后的行业分类（例如 A01 到 A05 五个次级行业归为一个上级行业 A）。这样处理后，我们给出了 14 个一级行业，分别为 A 农业，B 开采，C 制造业，D 能源，E 建筑，F 零售，G 交通运输，H 餐饮住宿，I IT相关，J 金融，K 房产，L 商务租赁，M 研发，N 公共产品，P 教育，R 文化，S 综合。删除没有行业给定的股票（以 2016 年年底为例，删除约占总股票数目的 9%），删除只有一只股票的行业（因为限制在上证上市股票可能最后出现这样的情况），同时删除同时具有两个行业标签的股票。



通过上图可以看出，在对行业区别对待 BP 值这种方法上，确实不同行业的 BP 体现了行业估值逻辑的差异性。简单总结图中的几点：第一，行业的估值相对水平基本上比较稳定，例如无论哪年金融都比别的行业 BP 高，而 IT 相关的行业相对 BP 较低；第二 BP 在不同行业都体现了大盘的波动，普遍大部分行业的 BP 水平在 2015 年都低于该行业 2014 年和 2016 年的行业平均水平；第三点是 BP 不遵守上述两点规律的表现体现了市场的投资风向变动，这同时源于不同行业在当时经济发展和制度政策的条件。

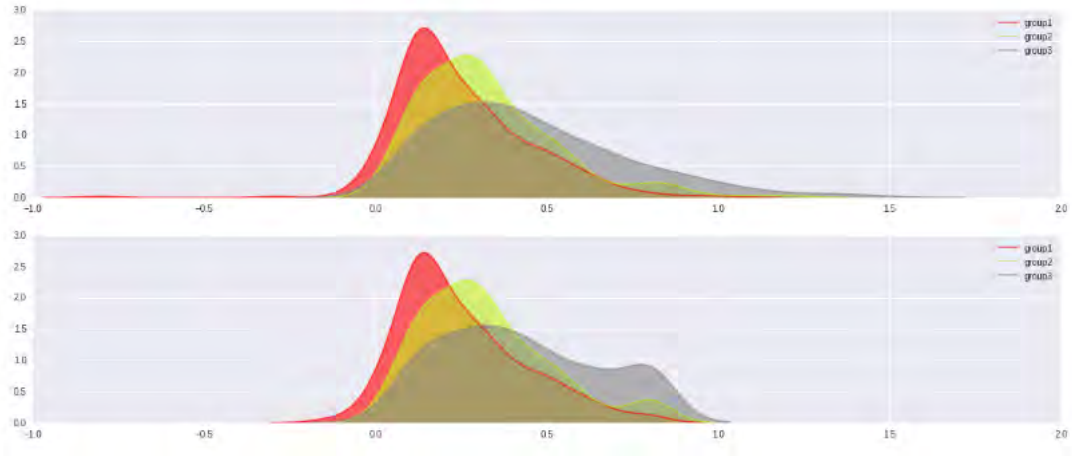
因为行业数量比较多，这里就不分别为每个行业进行分 BP 回测。在后面我们会展示按行业处理后的整体 BP 回测结果。接下来将介绍缩尾和中性化两个处理方法。

缩尾

缩尾和截尾是两种处理极值处指标的方法。所谓的 尾 是指上边分布图中拖拉很长但是每个横轴对应的分布概率又很小的部分。缩尾就是把超过阈值的指标值用阈值替换，比如大于 1.5 的 BP 值在后续处理中转化成 1.5 处理。截尾处理一般是把超过阈值的样本删除，然后在保留的样本中进行处理。

缩尾和截尾一般对应两种阈值生成方法，一种是指定分位，一种是利用概率分布的均值方差处理。分位的方式很直观，当一个指标最大的 1% 和最小的 1%（或其他百分位）被认为属于极值，那么直接设定该值为阈值就可以了。这样的一个阈值选取需要基于既有的经验或习惯。另一种可以称为数据驱动型，就是阈值是由于数据本身的属性生成的而非人为给定，例如均值标准差方法。结合概率论的表述也很好理解，我们使用均值和标准差构建了一个样本值范围，该范围为所有样本中样本在一选定样本的概率（例如本研究中使用均值上下两个标准差的概率选定为 95%）。

我们根据 2016 年底上证股票 BP 均值上下两个标准差进行缩尾处理，即把出现概率小于 5% 的极端值转化为阈值处的指标水平，然后按照市值分组画出处理前后的对比图。由于原始值的分布尾巴明显往右拖，因此缩尾处理后对于右部影响明显，明显把阈值外部分挤向中间。

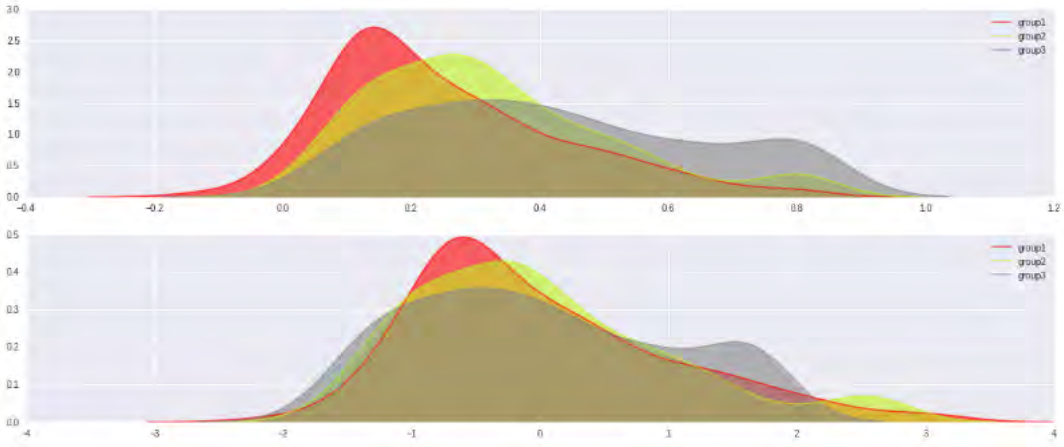


这两者对于尾的处理除了主要关注的值本身，还有对于样本的影响。即使不结合对于样本分布的影响讨论，样本量本身的变动差异可以看出，断尾处理与缩尾处理相比，减少了备选的可行股票池，具体减少多少根据预设处理参数决定。在实际的单因子策略中缩尾处理如果指定的标准影响的原始值数量不多，处理与不处理得到的结果影响不大，因为按照排序选取，择股只与相对大小有关而与原始值没关系了。此处就是缩尾与截尾对于结果影响的简单例子，如果使用其他比较复杂方法或模型，两者的差别会更加复杂。

中性化

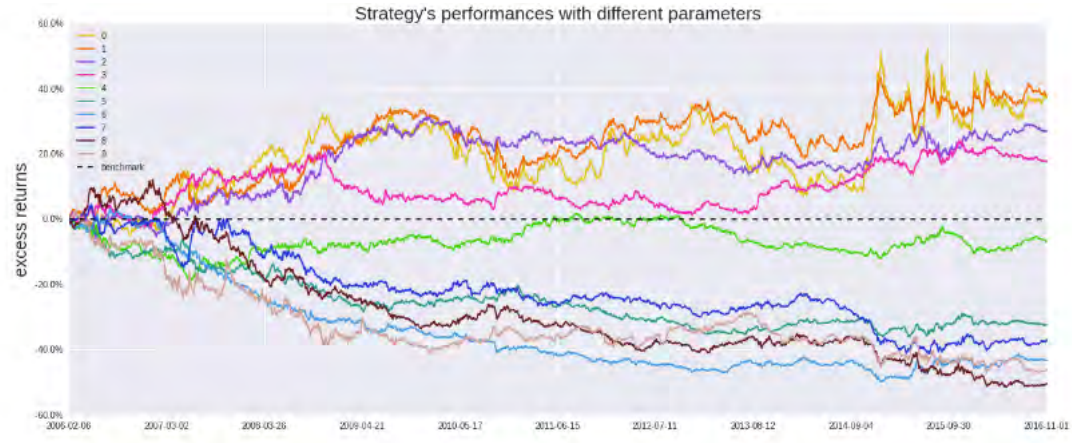
中性化一般的处理方法是把原始值减去均值除以标准差得到，转化后新的分布均值为 0，大于原始值一个标准差的值变为 1，表示与均值一个标准差距离。有的时候研究中给出了中性化、标准化或规范化三个定义，实际上差异主要存在于表述中。简单总结，中性化体现了针对某个标准（例如行业或者市值）规范化/标准化的结果。

转化的意义在于怎么把鸡和鸭一起比，比如说 3 苹果和 2 橘子哪个好？庸俗点转化成人民币标准就可以比了。不同行业（市值）的 BP 水平不一样，但我们假设每个行业最大 BP 的公司可能存在着相同的估值偏低水平，那么标准化就可以实现这样的效果了。上一幅图展现了数据在是否使用缩尾带来的差异，那么下图展现 2016 年年底 BP 数据在缩尾处理后是否对市值行业中心化的差别，使用默认核函数生成分布图。经过转化的图形看起来就比较正态相似了，那么可能每个不同市值组中提取相同 BP 值对应的股票，具有相似的可比性。



行业和市值中性化

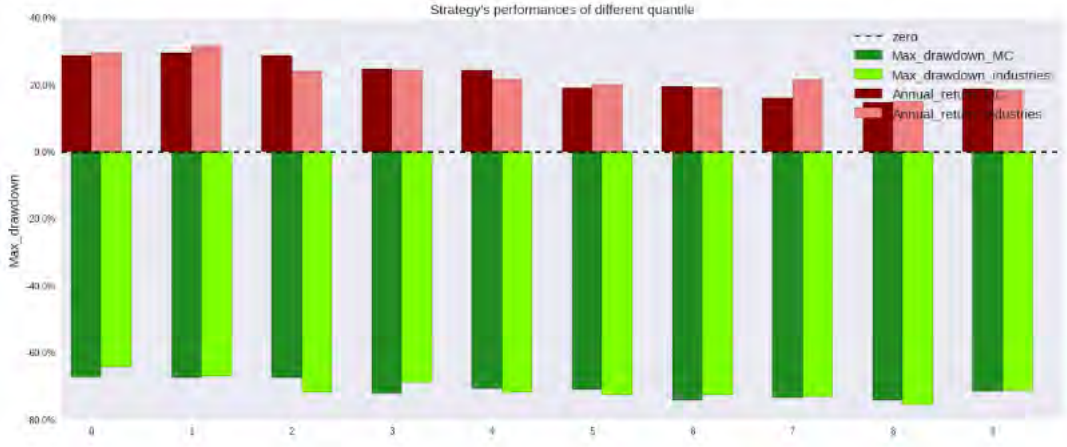
首先看一下市值中性化的结果，在此之前先回忆一下不处理市值中性化前 BP 分位回测结果。



对市值中性化处理后 BP 因子策略的分位回测结果显得具有明显区分度，并且在时序变化表现中具有较好的一致趋势。高 BP 策略针对于等权全指策略的超额收益率持续放大。超额收益率的最大回撤也在下面展示了，最小的最大回撤（好拗口，就是第二、三分位的最大回撤）都在 10% 左右，已经还不错了。

不同的行业中性化 BP 分位回测对比等权全指的对数超额回报率。相对于等权全指的，行业中性化的结果和市值中性化的结果都明显好于没有处理的结果，但是行业和市值中性化结果对比而言，不同 BP 回测结果的超额收益率对数表明其区分度更好。反而言之，对于 BP 不能在不处理时具有很好的区分度这一问题，同时包含了市值影响和行业影响：不同市值的 BP 在均值和分布上都有差异，不同行业的估值逻辑导致其 BP 水平具有较大差异。结合对比的回测结果可以看出，市值和行业的 BP 干扰中行业的问题更大。

行业中性处理和市值中性处理的回报率与回撤。

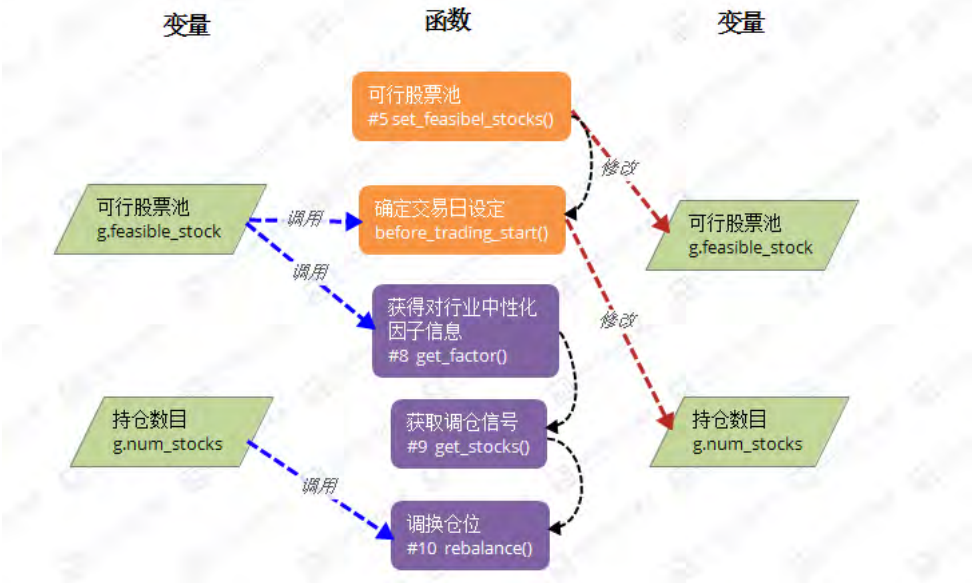


行业中性处理和市值中性处理的超额回报率与超额回报最大回撤。

小结

本文是对于之前调参框架的实用案例，也是对于因子数据处理方法的一个介绍。具体的股票分类和中性标准寻找，需要结合研究的因子而变，因为每个因子受到的影响因素存在差异。本文给出的市值和行业是因子分析中常用的特征指标，但并不是所有因子都这样。当找到合适的标准后，根据上文给出的分析方法，可以进行分组研究并进行中性化处理已得到更加合理科学的结果。

函数和变量说明书



v1.0，2017-01-19，文章上线

【量化课堂】PEG 动态选股策略

导语：

彼得林奇的 PEG 那真是大名鼎鼎，（如果对 PEG 不了解的同学可以移步这里把它当故事看一看，难度 -level 0 ，不看也成，下文会有介绍）但是当我们把经典PEG做成策略，写成代码后发现，欸，怎么就不好使了呢？所以小编决定学习我党共产主义中国化的先进经验，粗浅的琢磨了一个PEG投资策略的沪深化，望各位大神指摘。

本文由JoinQuant量化课堂推出 。难度标签为入门，理解深度标签：level-0

作者： 路安
编辑： 肖睿

PEG：

PEG 的全名我觉得应该叫 PE 除以 G 。PE， 市盈率，可以简单理解为公司的市值除以公司盈利能力得出的倍数。举个栗子，老张在村儿里卖猪肉，深得隔壁王寡妇欢心。邻村老李头觊觎王寡妇多年，百思不得骑姐，并觉王寡妇和老张的破鞋一定是因为猪肉的关系。于是乎大手一挥，买了老张的肉铺，霸了老王的寡妇。过程中，老李出资50万给张屠户，收购了他的肉铺，并发现肉铺每年还帮老李赚了10万块。结果最后一箭双雕，即收拾了邻村王寡妇，又开开心心赚到了钱。

故事结束，敲黑板敲黑板，看官朋友们，如果我们把肉铺想成是一家只发行了一股的公司，那么这里老李花的五十万除以每年盈利的十万，五倍，基本就是 PE ratio 市盈率的概念啦，即某种股票每股市价与每股盈利的比率。

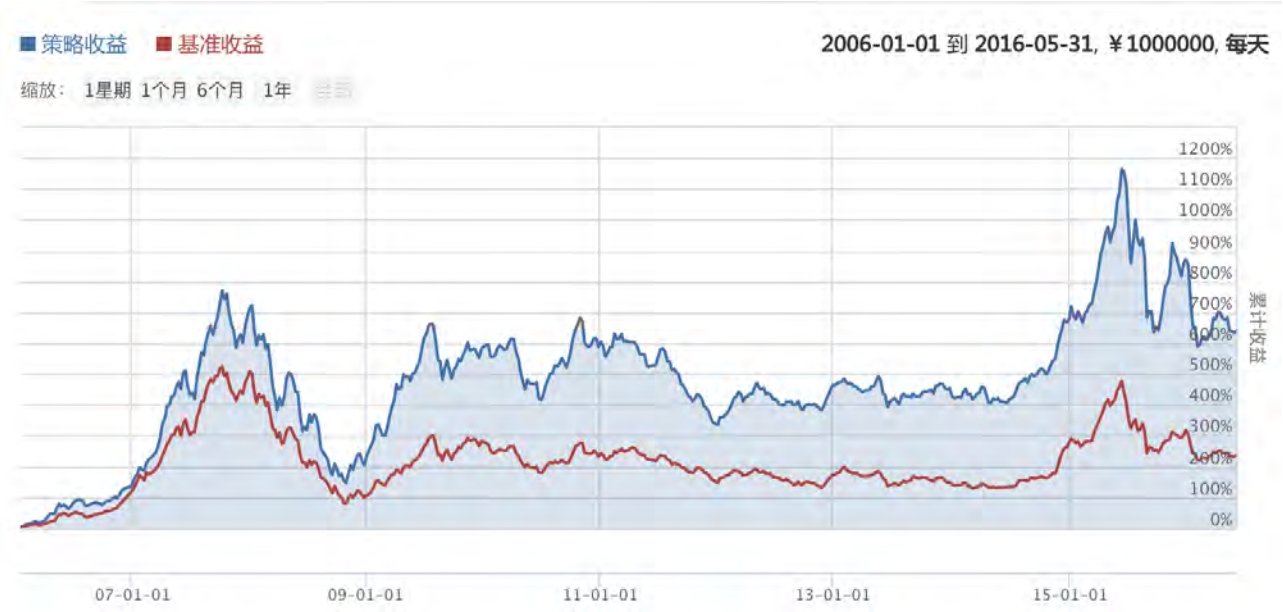
下面我们再说说这个 G，这个 G 可跟老李头与王寡妇婚后的性福生活就没有关系啦。主要是，老李接摊肉铺生意后，在王寡妇的帮助下生意越做越好，收益连年递增，头年挣10万，来年挣15万。

这时，肉铺生意的增长率 G（Growth Rate） 就是拿今年增长的部分：15-10 = 5，除以去年的利润：5 / 10= 0.5,即 G =50%。因此老李肉铺的 PEG 即可得出为 PE = 5， G = 50，PEG = PE ➗ G = 0.1

好啦，PEG和王寡妇的故事就给大家讲到这里，下面我们来看一下 PEG 的改进。

PEG 的中国化：

因为当年林奇老爷子认为，任何一家公司，股票如果定价合理的话，市盈率就会与收益增长率相等。所以他以 PEG =1为线，把 PEG 小于1的留位备选，大于1的剔除。（这只是个粗略的指导思想，当然会有例外）但林大爷叱咤风云的年代，那是几十年前的美国，放到今天，只能说既不天时，也不地利。所以我们代码实现 PEG 策略时，发现回测结果并不像老李头和王寡妇生活的那么幸福。



从回测结果看，PEG 策略虽然收益跑赢大盘，但是波动幅度巨大，最大回撤奇高，并且完美的栽倒在了中国股市的两次股灾面前。于是乎，聚宽团队决定小试牛刀，进行一次 PEG 策略的中国化。

策略收益	基准收益	年化收益	Alpha	Beta
652.59%	243.23%	22.08%	0.087	1.043
Sharpe	**Sortino**	**Information Ratio**	**Volatility**	**Benchmark Volatility**
0.519	0.674	0.635	0.348	0.304
胜率	**日胜率**	**盈亏比**	**盈利次数**	**亏损次数**
--	--	--	--	--
最大回撤				
72.845%				

通过研究发现，数据的时效性和合理适当的止损、调仓措施，对 PEG 回测结果的影响非常大。就我个人的理解， PEG 还是一个偏基本面的分析策略，而其中的关键，则是对公司盈利能力增长的判断，当 EPS 数据失效时，策略就会无所适从。于是乎，我们想到了使用其他因子对公司盈利能力的增长进行判断，并将 EPS 的增长率换成了利润的增长和销售能力的增长两个因子。基于这个想法，我们调出了上证和深成，两个指数构成的所有股票，并进行筛选。首先计算 PE 和 Profit Growth Rate 的比值,并以此为依据对所有股票进行升序排名（得出的商越小越好，所以值越低越排在前面），再用 PE 和 Revenue Growth Rate 的比值为依据重复上述过程。

这个过程如果不好理解的话，那我们举个栗子，好比烤鸭店选大肥鸭，要想做出香喷喷的烤鸭，原材料的选取上，一定是第一关。所以第一步，我们筛出体长 50cm 以上的鸭子，第二步在第一步的基础上进一步选出体重在 10kg 以上的，第三步再筛选羽毛必须洁白，没有杂毛的。第四部筛选身体健康，各项指

标都合格的，并把筛选出来的鸭子进行编号，生成列表：

【小白鸭1，小白鸭2，小白鸭3…】

通过这么一步一步的筛选，我们有理由相信，筛选之后烤制出的鸭子，和随便扒拉一只做出的烤鸭相比，前者，有很大的概率，会比后者，好吃。我们选择股票也是同样的道理。

之后，我们用前面介绍过的方法，可以得到两个股票列表，取他们各自的前五十名，并求交集，生成我们全新的备选股票列表。为了具象一点，我们再举个栗子，好比班里要选三好生，条件是成绩又好，体育又好。于是乎，我们把同学们的学习成绩排了个名，前五名分别是:

{小红，小明，小李，小张，小睿}

之后又把大家跑步的成绩排了个名，但是发现前五的名单有了不小的变化，分别是：

{铁蛋，二嘎，小睿，虎子，小明}

无奈之下，我们取个交集，小张，小李，小红惨遭淘汰，只剩下 {小睿，小明} 晋级三好学生，这样，我们有理由相信，小睿和小明有很大的概率都是品学兼优德智体美全面发展的好孩子。这就好比我们筛选出的备选股票池，经过各种指标筛选并排名再筛选后，我们备选池中的股票，大概率上应该都是根正苗红，嗷嗷待涨的好股票。

但是！ BUT！ HOWEVER! 我们能就此放心大胆的买入这些股票了么？当然是不能！原因就是我们抄的是中国股市，那熊市来的时候，甭管什么 PEG，APT ，KNN 还是 Fama-French，策略统统不好使，无论基本面多优秀，血淋淋的经验告诉我们，持仓冒险的结果只有跌跌跌。唯有清仓止损避一避，赚取一定的无风险收益才是最稳健最保险的。因此我们引入 MA 函数对股市趋势进行判断，虽然什么时候涨不好预测，但是跌势显现时，什么时候清仓什么时候跑的判断还是相对简单的，这样我们引入沪深三百，上证综和深成指指数数据（原因是基本覆盖了股市中绝大部分的股票）并对主要指数前一个月的收盘价取平均值，当今日收盘价低于 MA 平均值时，生成信号，执行清仓，并买入国债指数，引入无风险收益。

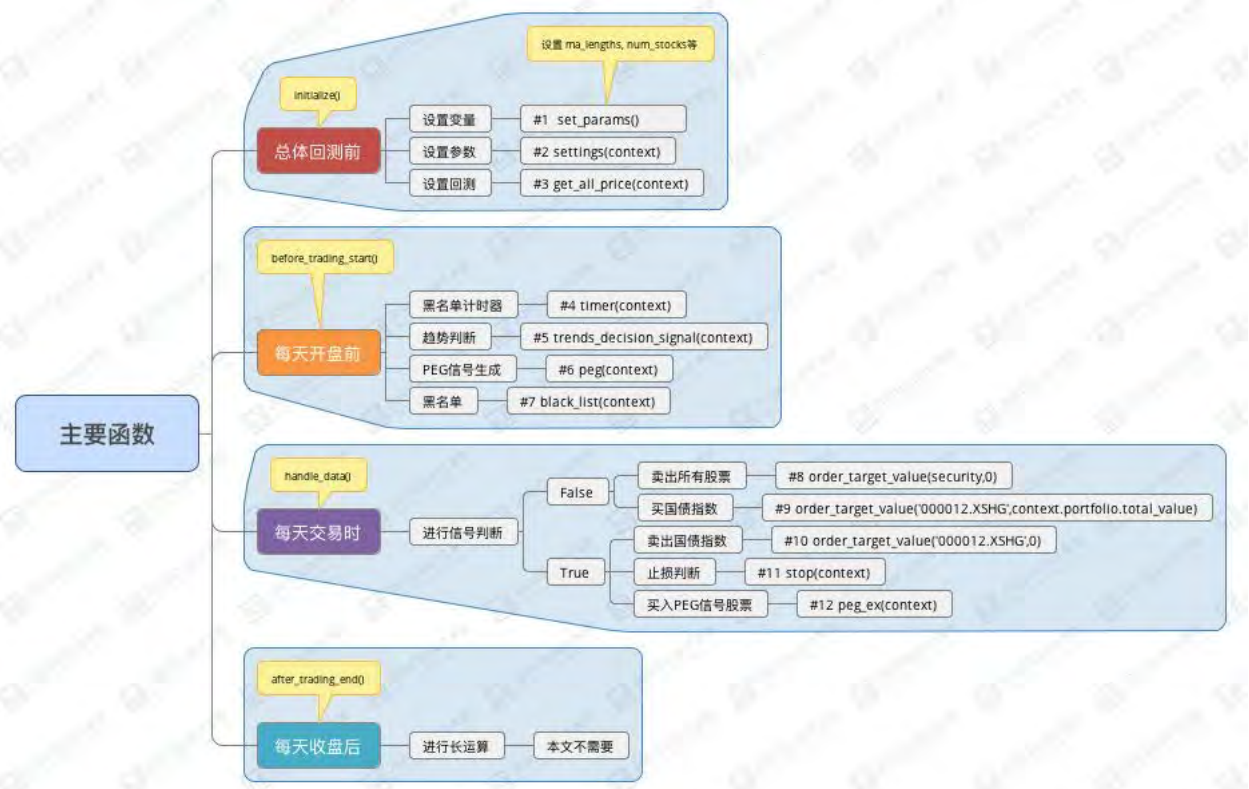
另一方面，我们要谈一谈关于止盈止损和调仓的措施。好不容选出了这么几个矬子中的大个，嗷嗷待涨的小股票，那么我们是不是就要一直持有着，等着他们一个个变成印钞机呢？答案自然也是 no no no, 因为止盈止损的执行在我们策略中担当的角色，还是任重而道远的。小编本人是一个非常保守的投资小白，所以为了保本儿，我们设计了两个制度，一个是最高价名单制度，另一个是抛出黑名单制度。最高价制度，故名思议，我们按分钟记录每一支持有股票的价格，并创建一个字典，存入他在持有期限中曾经达到过的最高价格，这个感觉有点像我们的体重，比如说，小张同学身高一八五，体重两百斤，造成的结果就是甘油三酯也高，胆固醇也高的，最后就是因为太胖，连三好学生也没拿到。于是乎，小张同学决定减肥。但因为身宽体胖吃对的多喝的多的缘故，所以其实体重可以描述成一个连续时间序列，在这过程中小张体重一直下降，但有时吃完饭或者喝完水之后体重会上涨一些，这是正常情况。但如果体重连续上涨了5%就不太正常了，要赶紧采取措施不然可能会减肥失败。这个5% 不是和开始减肥前的体重比的，而是和开始减肥后的最低体重比的。持有股票也同理，只不过上涨下跌要反过来。根据这个思路，我们取得的股价最高值也一样，如果以购买之后的最高价为标准，跌幅超过8%，策略就进行止损。同时我们设立一个止盈标准，如果当前股价超过成本价的100%，我们就进行止盈（多么贪心的一只小编，当然这些参数都是可以通过 set_para() 函数来实现用户自定义的）。止盈止损后卖出的股票会进入另一个黑名单系统，并且十个交易日内不再买入。

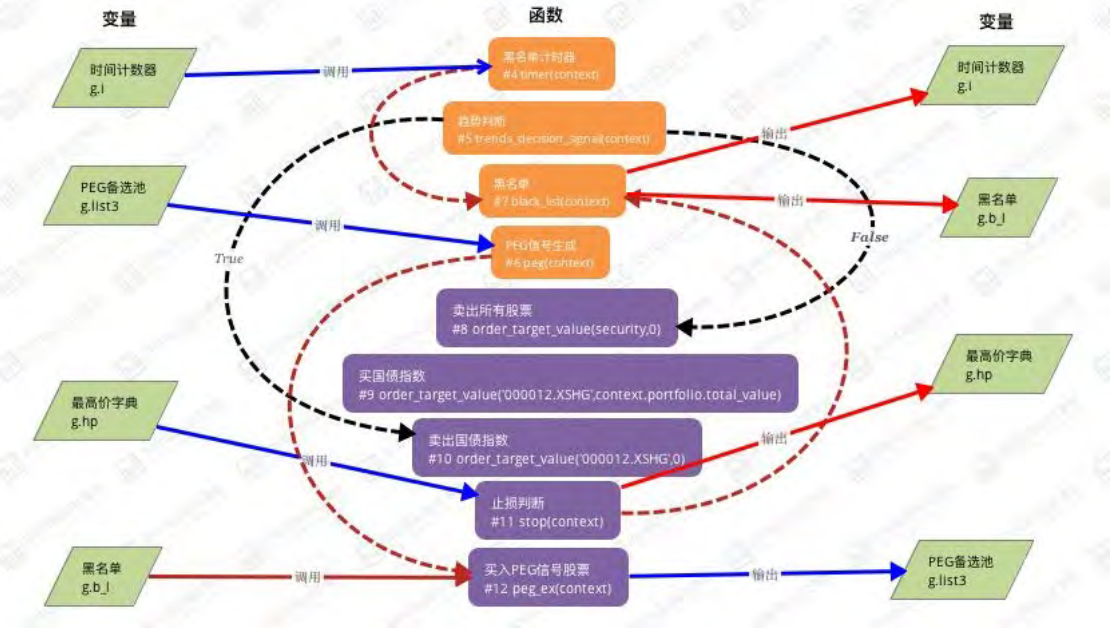
好，我们闲言废语不要讲，在这个结果导向型的节骨眼上，不拿出点儿真东西，怎么好意思在观众面前逼逼。我们上图，分钟回测结果成绩如下：

总结：

从 PEG 改进策略的分钟回测结果来看，该策略最大的优势在于其可以提供的长期、稳定的超额收益。可以在股指震荡的过程中，动态的筛选股票，并止损调仓。但略势同样明显，因为交易信号和交易条件确认的过程相对保守，所以当大盘牛市来临时，策略会有一个相对应的滞后，这也就是为什么在策略初期，我们的算法会稍微落后于大盘指数的原因。

函数和变量说明书





文章更迭记录：
v1.0，2017-03-17，文章上线

【量化课堂】多因子换档反转策略

前言：

（这期的前言有点长，不看前言对读懂策略没影响）亲爱的观众朋友们大家好，积日不见，甚是想念。聚宽量化课堂又和大家见面了。说实话，好歹小编也是个科班出身的博士，但不得不说在聚宽的这段时间，大家带给我的进步，那真是一日千里，突飞猛进，就像坐着窜天猴儿一样。在聚宽，给人最直观的冲击就是，每天都能感受到自己的进步。这其中的一部分来自于你对学过的内容，通过实操，进行复习，孔老头说温故而知新，大概可能就是这个道理。另一部分来自于每天要面对的，一大堆自己从来没接触过的新需求。这时就是聚宽体现他美丽的时刻了，聚宽有无数的小伙伴儿，乐于对你倾囊相授，敲黑板敲黑板，是倾囊相授！如果你要问了一个他们解决不了的问题，这些人恨不得现去自学，然后再特么给你讲。所以，如果你是一个注重过程，期待今天的自己总是比昨天好那么一点儿的人；如果你是一个对量化交易感兴趣，是一个对自己的能力所学知识有热爱，肯憧憬的人，那么请戳这里。 好了，说了一堆感言，下面开始进入正题吧。

本文由 JoinQuant 量化课堂推出 。难度标签为入门，理解深度标签：level-0

作者： 路安
编辑： 肖睿

导语：

之前量化课堂的同事出过一个因子研究系列一、 二、 三，小编读完了以后感觉就像是被人开了天灵儿盖一样，灵感咔咔的往外冒。所以推荐大家拜读一下。这个新策略是在原基础上经过筛查，选取 ARL，CFP 和 BP 三个因子(我们把很多其他因子的调用逻辑也写成了函数，并附在了代码的最后，以便大家的使用)，以二十天为调仓基准的换档反转策略。这个策略的缺点是年化收益并不像前几个，有着高的邪乎吓人的回报曲线。但其优点也是咔咔的，稳定且完备。妥妥的一名任劳任怨表现稳定的经济适用男。千分之三的滑点，双边交易手续费，佣金全都扣了，还保证个百分之二三十的年化回报。而且入市的时机没有什么要求，简单讲就是，大多数情况下都好使！接下来我们闲言废语不要讲，撇开那些光说不练的假把式，我们正式开始。

因子选取介绍：

如果大家知道 ARL，CFP，BP 都是啥，那可以直接跳过下面三个小标题，直接去看策略结构。

ARL ——资产负债率因子：

先抛砖引玉丢个公式吧：ARL=total_liability/total_assetsARL=total_liability/total_assets

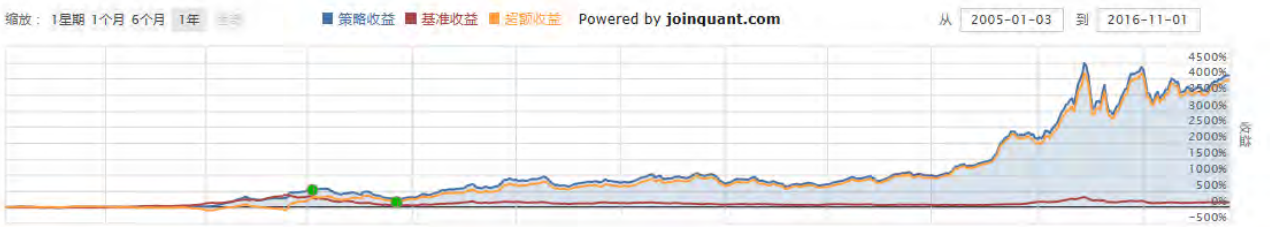
没啥难的，资产负债率指总资产除以总负债，结合我们聚宽的财务指标，这里我们使用“负债合计” total liabilities 和“资产总计”
tatal assets 来计算 ARL 。这一指标反映了总资产中，有多大比例是通过借债来筹资的。举个栗子，家住成都的老王手里有30万存款，然后看上了市郊一套售价100万的房子，无奈手里钱不够，又有刚需，因为没有房子讨不到老婆。老王遂从银行贷了70万，买下了一套市郊的两室一厅。这时，这套100万的房子就是老王的总资产，贷款70万是总负债。我们把老王看成一个公司，这时他的 ARL = 70万总负债/100万总资产 = 0.7 。

从回测结果能看出来，如果我们从上证综指选出 ARL 最高最低的1%和5%，抛开夏普率，波动率，最大回撤这些不谈，只看累计收益我们也能发现，ARL越高，回测收益越好。

ARL 最大1%，震撼不震撼 可是最大回撤也同样震撼

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
4116.24%	38.51%	151.25%	0.322	0.528	1.063	1.843	0.804	0.325	0.276	0.638	0.530	2.930	278

亏损次数	最大回撤
158	55.750%(2008-01-15,2008-11-06)



现金收益率： CFP

CFP=1/pcf_ratioCFP=1/pcf_ratio ， 这又是个啥呢？

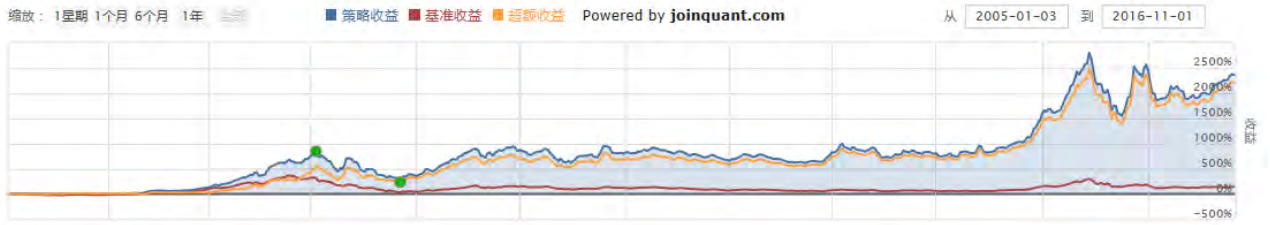
先给大家看个官方解释，现金收益率是公司现金及等价物与股票市值的比例。并且为了方便获取数据，我们使用聚宽量化平台的“市现率”指标的倒数，市现率指每股市价除以过去12个月每股现金流量的倍数。这一指标以比例形式呈现，因此总量计算和每股计算得到的相同。听着有点儿绕是不是，举个例子大家就明白啦，小陈和小睿两个人都在追求王翠花，为表衷心，俩人拿出了自己的存折，交给翠花过目，结果发现二人存款都是五十万。（都比讨不到老婆的小编有钱）这下可特么难坏了村妇王翠花，老娘到底嫁谁好呢？谁更有钱？跟谁在一起的生活质量会更高？婚后生活更幸福呢？困惑的翠花只能去问她的妈妈，菊花。这个姜还得说是老的辣，翠花妈妈直接让小睿和小陈上缴了银行流水，把近几年每个月的现金流入和流出都一五一十的打在了单子上。结果不比不知道，这一比还真吓一跳。虽然都是五十万存款，可老陈平均每月只有两万多的进账，不足三千的开销。就是为了追求翠花，近几月的花费才明显有了增加。小睿则正好相反，把每月的开销随便一加就是大几万，虽然每月没有稳定的进账，但笔笔进账后面都跟着好几个零。所以，跟谁在一起生活的质量高还用想么？高下立判。绕了个大圈子，只是想给大家讲明CFP的道理，用流水除以现值（所以 CFP 越大越好），用动态和静态的财务数据做对比，这样我们就能够比较清晰地分析出，哪家上市公司是真有钱，哪家上市公司是假大款。

为了让大家更直观的看到 CFP 因子的作用，我们把回测结果也 po 在了下面

CFP 最大1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
2371.29%	32.22%	151.25%	0.237	1.044	0.820	1.092	1.166	0.344	0.276	0.586	0.504	1.910	346

亏损次数	最大回撤
244	63.413%(2008-01-15,2008-11-04)



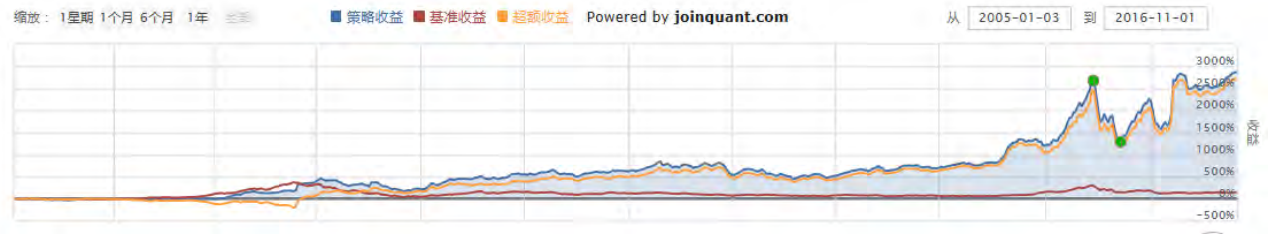
账面市值比：BP

最后讲讲 BP 这个因子，还是先丢个公式给大家 BP=1/pb_ratio
账面市值比为股东权益除以股票市值，在计算中这里使用聚宽财务数据中的 pb_ratio（市净率=每股市值/每股净资产）倒数计算得出。其实结果就是 BP = Equity / Market Cap，道理很简单，我就不赘述了，如果有感兴趣的朋友可以点这里获得更详细的内容

BP 最小1%

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
2888.39%	34.42%	151.25%	0.282	0.506	0.823	1.597	0.642	0.370	0.276	0.634	0.525	2.130	255

亏损次数	最大回撤
147	53.981%(2015-06-12,2015-09-15)



换档反转策略

其实我们的想法很简单， ARL，BP，CFP 这三个因子就好比三个动力强劲但是不很稳定的发动机，他们的表现时好时坏并具有周期性。因此我们运用反转的思想，如果三个因子中，某一个在前三个月的表现非常不好，那我们就期待他会在接下来的一个月有一个小的爆发。因此我们会在合适的时候进行换档，即因子转换，为了便于操作，我们这里设置的调仓时间为20个交易日，如果大家看着不爽可以改哈。

过程中，我们首先从备选股票池中过滤掉停牌股，然后按照前面大家看到的那几张图，以单因子的回测结果为依据，每天获取出 ARL，CFP 最大的前1%的股票和 BP 最小的前1%的股票，并通过研究与回测的联通模块（不会的同学请点这里，有详细讲解哦），来监视每个因子每天的表现。为了方便大家的应用，小编在这里已经把这个复杂的工作整理好了，大家只要下载下面三个因子的CSV文件(点我点我)，并存贮在自己账号，研发模块的根目录下，回测引擎就会自动调用啦。当然如果大家自己感兴趣的话，也可以从我们之前的教程中选取其他因子，比如 ROE，ROEC 等等。

关于CSV：

因为有很多朋友对CSV非常感兴趣，那小编就给大家详细介绍一下研究模块和回测的联通功能。这只是一个定性的讲解，如果想从到到尾的撸明白，那么可以点这里。下面，大家可以想象这么个场景，如果你是中国田径队男子100m的主教练，现在马上世锦赛了，你只能派一名队员去，可是有五个种子选手，你应该选谁呢?

抛开耽误腐败和女教练和男运动员这种特殊的关系不谈，答案应该是非常简单的，谁成绩好谁去！所以研究回测联通模块其实就是这么个选拔过程。比如我们有五名队员 A, B, C, D, E, 就相当于五个因子，平时让他们跑模拟盘，记录回测结果。然后用研究模块调用这个回测结果，并计算谁的成绩好。最后我们把这个研究模块的结果存成CSV文件放在研究模块的根目录下，回测引擎就可以直接调用这个结果了。说的再简化一点，这样做的目的实际上就是为了同时开好几个回测引擎，然后哪个结果好就用哪个。生成CSV文件后，我们在回测引擎中要把它转成DataFrame格式，这样就大功告成了！具体的格式嘛，从一个因子的 CSV 读出来的 DataFrame 只有一列，index 是日期，取值是这个因子在回测中的净值，这样就知道每个因子的历史走势了，然后我们就可以根据自定义的规则在因子之间进行换档。

==========牵割线==========

接下来，我们以60个交易日为一个节点，进行评测。计算出调仓前一个交易日和前六十个交易日之间，单因子策略的回报率（其实就是上面那六张图的切片）。表现最不好的那个单因子策略，就是我们期待着他能反转的那个。这里面的原因会涉及到一点统计学的知识，如果有感兴趣的同学，欢迎大家在评论中踊跃提问，小编一定知无不言，言无不尽。其实马上有同学就可能会问了，你这么整，他万一要是没转过来，我这二十个交易日的持仓，那岂不是赔的裤衩子都没了。为了避免这个问题，我们同之前的 PEG 策略 (点这里点这里) 一样，制定了最高价黑名单制度。小编在聚宽内部做了个小的样本抽样，发现大家对短期内的平均忍耐力大概是以买入价为基准，允许回撤18.799%。之后我们稍微改变了一下思路，如果股票先涨后跌，那我们的止损线，以20天持仓期间达到过的最高价为基准，允许20%的最大回撤。止损后剩余资金买入国债指数，获取无风险利率。为了尽量模拟实盘交易，滑点设为千分之三，交易手续费按照证监会要求和实际执行标准分段设定。接下来的两个回测是全真模拟环境下的收益截图。

但是因为量化课堂的统一标准，所以我们在下面的源码里把手续费 滑点设为了默认的0，请大家一定要注意哦。 废话不多说，光说不练的那是假把式，先来看看我们近一年的从2016.3.27到2017.3.30的收益和从05年以来的长期回报

2016. Mar-2017.Mar

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility
23.72%	24.04%	7.75%	0.169	0.823	1.122	1.219	1.016	0.179	0.129

胜率	日胜率	盈亏比	盈利次数	亏损次数	最大回撤
0.615	0.538	2.671	32	20	14.261%(2016-04-19,2016-06-24)

2005. Jan - Today

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Algorithm Volatility	Benchmark Volatility	胜率	日胜率	盈亏比	盈利次数
2740.79%	32.51%	158.31%	0.267	0.419	1.401	1.789	0.803	0.204	0.271	0.699	0.516	1.812	1505

亏损次数	最大回撤
647	33.658%(2008-01-15,2008-12-31)

总结:

小编对这个收益率还是比较满意的，毕竟量化课堂校长睿神有句名言，年化回报率离谱的策略，大部分都忽略了某些因素，跑实盘肯定会有问题。比如小编当年的PEG，回报率血高，但是其实并没有考虑滑点，手续费，佣金，停牌股票等一系列问题。这个多因子换档反转的策略考虑的因素非常多，而且回报率稳定，设置更接近实盘，所以在这里推荐给大家。

函数和变量说明书

文章更迭记录：
v1.1，2017-04-7 增加回测研究联通功能讲解
v1.0，2017-04-7，文章上线

数学课堂

【数学课堂】线性代数系列 -- 同构映射

导语

在线性空间中可以维持空间的线性特性的映射叫做线性映射，一个最简单的线性映射就是从一个线性空间 V 上的恒等映射 I，即对于所有 $v \in V$ 都有 $I(v) = v$。恒等映射告诉我们它的定义域和陪域是相同的线性空间，这并没什么稀奇的，因为它的定义域和陪域都是 V。但是有没有两个空间本来不是一样的，但是有着同样的线性构造呢？

举一个简单的例子，考虑现线性空间 $V = \mathbb{R}^2$，以及 $\mathbb{R}^3$ 的子空间

Missing or unrecognized delimiter for \left

这个子空间 W 和 V 其实是一样的，只不过多了一个一致是 0 的坐标，相当于把一个二维的世界嵌入到一个三维世界中，这个二维世界的 z 坐标一律是零，从这个二维切片上来看，它就是一个平面世界，和 $\mathbb{R}^2$ 一样，\textbf{同构}的概念让我们可以严格地描述这种性质。

同构

定义. 设 V 和 W 是两个 $\mathbb{R}$ 上的线性空间，并且设 $f : V \to W$ 是一个线性映射。如果 f 是可逆的，并且 $f^{-1} : W \to V$ 也是一个线性映射，我们说 f 是一个*线性空间的同构* (linear isomorphism)，并且 V 和 W 是*同构的* (isomorphic)，写作 $V \cong W$。

同构就是说是“相同的结构”，比如上面的例子中的 $V = \mathbb{R}^2$ 和 $W \subseteq \mathbb{R}^3$，如果想验证 V 和 W 是同构的，我们需要首先找到一个函数 $f : V \to W$ 并且需要它满足

1. f 是线性映射；
2. f 是可逆的；
3. f^{-1} 也是线性映射。

考虑下面这个函数 $f : V \to W$

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}.$$

很明显 f 是线性的，因为

$$f\left(r \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}\right) = \begin{bmatrix} rx_1 + x_2 \\ ry_1 + y_2 \\ 0 \end{bmatrix} = r \cdot f\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right) + f\left(\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}\right).$$

并且，f 是可逆的，逆函数是

$$f^{-1}\left(\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}\right) = \begin{bmatrix} x \\ y \end{bmatrix}$$

同样可以很轻松地验证 f^{-1} 也是现行的。所以，V 和 W 是同构的，它们的一个同构映射是 f。

上面用到的同构映射是非常符合直觉的，因为从 V 到 W 就是多了一个等于 0 的 z 坐标嘛，所以把 x 和 y 坐标延续下去就可以。但是我们要意识到，同构映射并不是唯一的，考虑下面这个函数

Unknown environment 'align'

这个函数是不是线性的呢？是的，因为

Unknown environment 'align'

那它是不是可逆的呢？是的，它的逆映射是

$$g^{-1}\left(\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}\right) = \begin{bmatrix} \frac{y}{2}\, x - \frac{y}{2} \end{bmatrix},$$

同样，也不难验证 g^{-1} 是一个线性映射，所以 g 也是一个同构映射。两个线性空间之间有两个不同的同构映射并不矛盾，实际上只要存在一个同构映射，就会同时存在无数个，我们之后也会介绍如何随意地生成同构映射。

逆函数的线性性质

在同构的定义中有三个要验证的性质，即 f 是线性的、f 是可逆的并且 f^{-1} 是线性的，但实际上其中的一个性可以被省略，这样在我们验证同构关系时就减少了工作量。

定理. 设 V 和 W 是两个 $\mathbb{R}$ 上的线性空间，并且设 $f : V \to W$ 是一个函数。如果 f 是线性映射并且 f 是可逆的，那么 f^{-1} 也是一个线性映射。

证明. 取 $w_1, w_2 \in W$ 和 $r \in \mathbb{R}$，有

Unknown environment 'align'

同样的，

Unknown environment 'align'

所以，f^{-1} 也是一个线性函数。 Undefined control sequence \square

所以，在检测线性空间同构的时候，只要看它是不是线性的并且是不是可逆的就行了。考虑下面两个 $\mathbb{R}^3$ 的子空间

Missing or unrecognized delimiter for \left

Missing or unrecognized delimiter for \left

从这两个空间的描述上就能看出

$$h\left(\left[\begin{array}{cc} x & y \\ x+y \end{array}\right]\right)=\left[\begin{array}{cc} x & 2x & y \end{array}\right]$$

是一个符合直觉的函数，读者可以\textcolor{red}{作为练习}来验证 h 是一个线性映射并且是可逆的，从而判定 V 和 W 是同构的。

反例

并不是所有线性空间相互之间都是同构的，如果是那样那同构的概念就没有意义了。我们考虑两个空间 $\mathbb{R}^2$ 和 $\mathbb{R}^3$，并证实它们不是同构的。为了反证，我们先假设 $\mathbb{R}^2$ 和 $\mathbb{R}^3$ 是同构的，存在一个可逆的线性映射 $f:\mathbb{R}^2\to\mathbb{R}^3$，并且设

$$v_1=\left[\begin{array}{ccc} x_1 & y_1 & z_1 \end{array}\right]=f\left(\left[\begin{array}{cc} 1 & 0 \end{array}\right]\right),$$

$$v_2=\left[\begin{array}{ccc} x_2 & y_2 & z_2 \end{array}\right]=f\left(\left[\begin{array}{cc} 0 & 1 \end{array}\right]\right).$$

那么 $\mathrm{Span}(v_1,v_2)$ 是 $\mathbb{R}^3$ 的一个线性子空间，因为是由两个向量张成的，但是 $\mathbb{R}^3$ 是三维空间，所以 $\mathrm{Span}(v_1,v_2)\neq\mathbb{R}^3$。取一个向量 $w\in\mathbb{R}^3\setminus\mathrm{Span}(v_1,v_2)$，用 f 的逆映射 f^{-1} 得出某个

$$\left[\begin{array}{cc} a & b \end{array}\right]=f^{-1}(w).$$

那么，

Unknown environment 'align'

所以 $w\in\mathrm{Span}(v_1,v_2)$，这和 $w\in\mathbb{R}^3\setminus\mathrm{Span}(v_1,v_2)$ 是矛盾的，所以最初 $\mathbb{R}^2\cong\mathbb{R}^3$ 的假设是不成立的，它们并不同构。

【量化课堂】单特征因子的隐马尔可夫模型在商品期货中的应用

导言

隐马尔科夫模型作为一种机器学习方法曾广泛应用于语音识别领域，目前又被大量地研究应用于金融市场当中。为了能够尽可能地挖掘该模型的应用潜力，本篇报告就尝试使用它在商品期货上面构建出的日间级别的 CTA 策略。

本文由 JoinQuant 量化课堂推出 。难度标签为进阶，理解深度标签：level-1

作者： YR
编辑： Mathematical23

模型简介

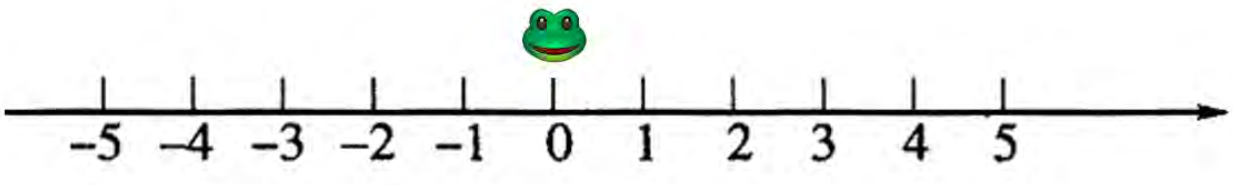
隐马尔科夫模型（ Hidden Markov Model, HMM）是被广泛使用在自然语言处理领域的一种统计学习模型。为了了解隐马尔可夫模型，我们首先来了解一下马尔科夫模型（Markov Model）。马尔科夫过程是指具有马尔可夫性质的离散时间随机过程。马尔科夫性质是指，在给定当前知识或信息的情况下，过去（即当前以前的历史状态）对于预测将来（即当前以后的未来状态）是无关的。

用数学公式表达就是：

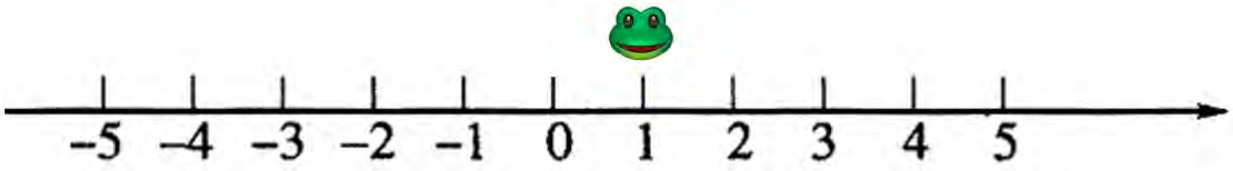
$$P(Xn+1=x\mid X_0,X_1,X_2,\ldots,X_n)=P(Xn+1=x\mid X_n)$$

举一个简单的例子：

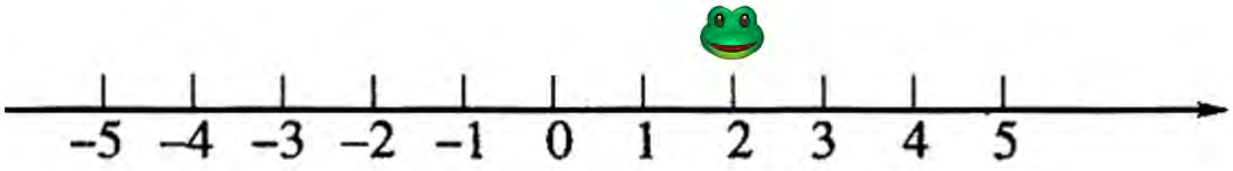
有一只青蛙，0时刻在数轴上的原点，他每一秒可以往左或者往右跳一个格子，并且往哪边跳是完全随机的。



那么在1时刻，它有可能在1，也有可能在-1。我们不妨设他1时刻跳到了1。



那么在2时刻，他有可能跳到了2，也有可能跳到了0。



而由于每一步的跳跃都是随机的，所以1->2时刻的跳跃与0时刻的位置没有关系，只与1时刻的位置有关。即：
$P(t_2 = 2 \mid t_1 = 1) = P(t_2 = 2 \mid t_1 = 1, t_0 = 0)$ 这个性质就是马尔科夫性：下一时刻的状态只和上一时刻有关，和上上时刻，上上上时刻……等再往前的时刻都无关。

为了简化问题，并且方便用数学语言描述，我们让青蛙改成在三个格子里跳跃。

假设它某时刻在格子1里，下一时刻跳到格子1，2，3里的概率分别是0.2, 0.3, 0.5；某时刻在格子2里，下一时刻跳到格子1，2，3里的概率分别是0.4, 0.2, 0.4；某时刻在格子3里，下一时刻跳到格子1，2，3里的概率分别是0.5, 0.4, 0.1。

那么青蛙跳格子这件事就可以用以下这个矩阵来描述：

$$(p_1 \quad p_2 \quad p_3)\begin{pmatrix} 0.2 & 0.3 & 0.5 \\ 0.4 & 0.2 & 0.4 \\ 0.5 & 0.4 & 0.1 \end{pmatrix} = (p_1' \quad p_2' \quad p_3')$$

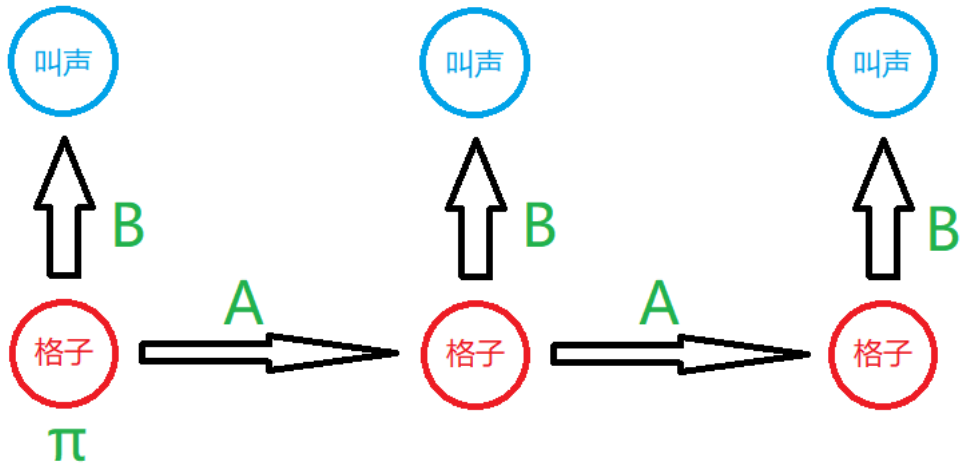
其中 p_1, p_2, p_3 是上一时刻青蛙在各格子的可能性，Misplaced & 则是下一时刻青蛙在各格子的可能性。假设上一时刻青蛙在格子1，那么 $p_1 = 1, p_2 = p_3 = 0$，乘一下就会发现，Misplaced & 恰好等于前面说的0.2, 0.3, 0.5。这个矩阵被称作转移概率矩阵，也就通过是每一时刻各点概率可以决定下一时刻各点概率，并且下一时刻各点概率仅与上一时刻各点概率相关。

以上的青蛙跳跃模型就是马尔科夫模型。那什么是隐马尔可夫模型呢?

我们给上面的青蛙增加一个叫的能力，每一时刻他都会发出一次叫声。并且它的叫声分两种，分别是叫声1和叫声2。而青蛙身处每个格子里时，发出各种叫声的概率是不一样的。假设青蛙在格子1中发出两种叫声的概率分别是0.2, 0.8；在格子2中发出两种叫声的概率分别是0.3, 0.7；在格子3中发出两种叫声的概率分别是0.6, 0.4。那么我们同样也可以用一个矩阵来描述这个过程：

$$(p_1 \quad p_2 \quad p_3)\begin{pmatrix} 0.2 & 0.8 \\ 0.3 & 0.7 \\ 0.6 & 0.4 \end{pmatrix} = (p_1' \quad p_2' \quad p_3')$$

现在，我们做一个重要假设：如果盒子被一块黑布盖起来，我们只能听到青蛙的叫声，不能看到青蛙的确切位置。现在的流程图大致变成了这样：



红圈代表每一时刻青蛙处在哪个格子，我们看不到（i,隐状态序列）；蓝圈代表每一刻青蛙会发出哪种叫声，我们听得到（o,观测序列）。但并不是每一时刻的叫声决定下一时刻的叫声，而是每一时刻的格子决定下一时刻的格子（A，转移概率矩阵），然后由格子来确定叫声（B，混淆矩阵）。当然，为了决定这个序列，我们需要知道青蛙最一开始处于哪个格子，或者处在各格子中的概率（π，初始状态）。记$\lambda = （A，B，\pi）$，这三者就决定了一个隐马尔可夫（HMM）模型。

HMM模型有三个基本问题需要解决。

1、评估问题。给定模型$\lambda = （A，B，\pi）$和观测序列$O = o_1，o_2 ..... o_t$，计算在模型λ下出现这个观测序列的概率。这个问题的求解需要用到前向后向算法，是HMM模型三个问题中最简单的。

2、学习问题。给定观测序列$O = o_1，o_2 ..... o_t$，估计模型参数$\lambda = （A，B，\pi）$，使得在这个模型下出现观测序列的概率$P（O|\lambda）$最大。这个问题的求解需要用到基于EM算法的鲍姆-韦尔奇算法，是HMM模型三个问题中最复杂的。

3、解码问题。给定模型$\lambda = （A，B，\pi）$和观测序列$O = o_1，o_2 ..... o_t$，求给定观测序列条件下，最可能出现的对应的状态序列，这个问题的求解需要用到基于动态规划的维特比算法，是HMM模型三个问题中复杂度居中的。

想了解更多与算法和原理相关的问题，可以参考CSDN上刘建平Pinard的博客 。

隐马尔可夫模型在Python 上的实现

Python 的第三方库 hmmlearn 可以高效地解决以上三个问题。具体使用方法同样可以参照刘建平Pinard的博客 。
本研究仅用到了该库的一个函数。

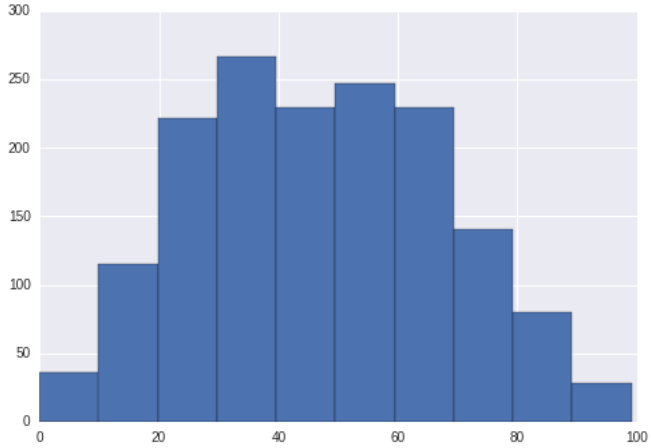
```
model = GaussianHMM(n_components=hid_sts, covariance_type="full", n_iter=1000).fit(X)
```

GaussianHMM 是用于解决连续型变量的。n_components 表示这个模型的隐状态个数，covariance_type 代表隐状态的方差矩阵的形式，n_iter代表迭代次数。.fit(X) 中的X是观测序列，也就是fit这个操作是让模型根据观测序列X去拟合最优的$\lambda = （A，B，\pi）$，并存储于model中。
具体关于该库的其他函数，可以参考官方文档 。

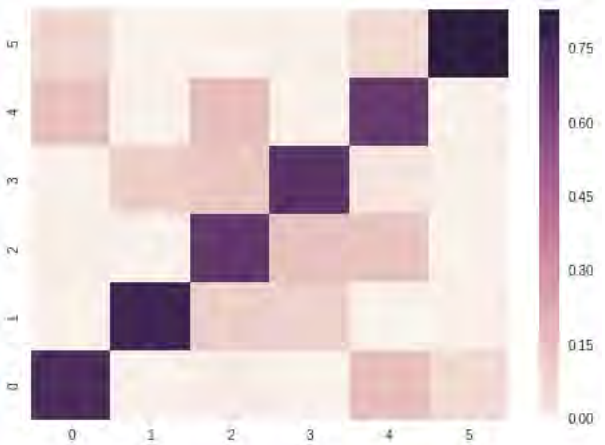
单特征因子的隐马尔可夫模型在商品期货上的应用

我们以国内商品市场上日均交易量最大、交易最为活跃的螺纹钢主力合约作为研究对象。研究时间选取2009-06-11至2015-12-31。
我们可以选取不同的技术指标作为观测序列，来拟合隐状态序列，进而使用每天隐状态和次日收益率的关系来对未来收益率进行估计。我们尝试使用RSI作为输入的技术指标。关于RSI的使用方法可以参考聚宽的技术指标讲解 。我们假定隐状态的个数为6个，并对模型进行拟合。

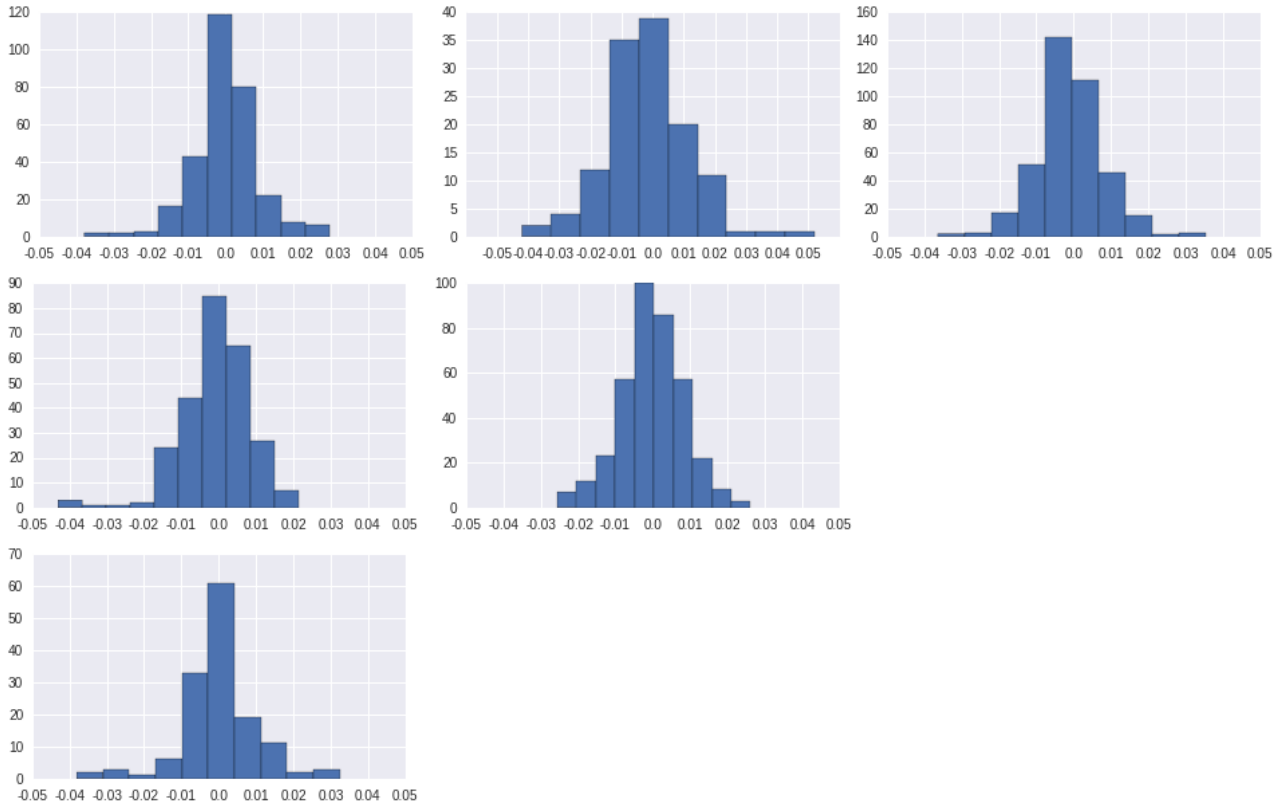
对输入的特征因子（RSI）绘制直方图，发现其大致符合0~100之间的正态分布。



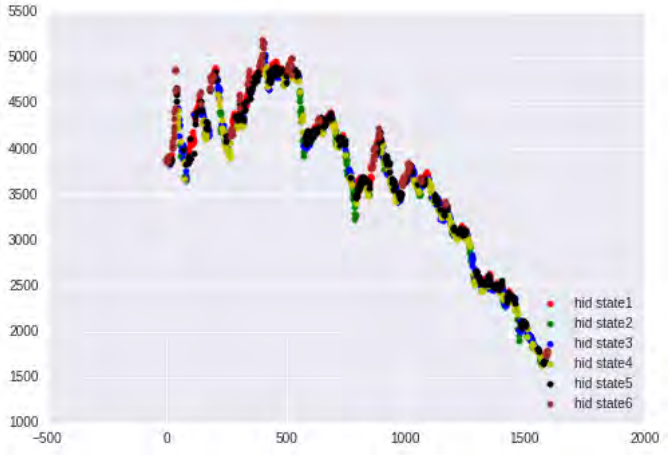
在得到转移概率矩阵后，绘制转移矩阵热力图。图中格子含义为隐状态由纵坐标所在格子转移至横坐标所在格子的概率。可以看出，大部分隐状态都是比较“懒惰”的，转移至其他隐状态的概率并不太高。



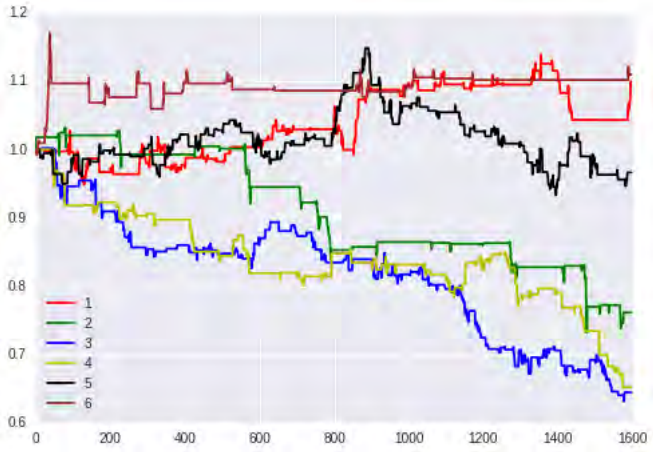
在得到隐状态序列后，我们需要了解每种隐状态对应该时刻RSI因子和对应下一时刻收益率的情况。以下6图为每一时刻的隐状态对应下一时刻收益率的分布情况。



将不同隐状态以不同颜色绘制在对应的收盘价上。



那在得到了每天的隐状态后，怎么根据隐状态来判断次日的交易方式呢？我们对每个隐状态都构建一个多头策略，即构建6个策略。每个策略都是仅在一个因子出现时做多，其他因子出现时空仓，因子的出现以模型拟合出的隐状态序列为准。例如，在1000天的观测中，第1天到第100天为状态1，其他交易日没有出现状态1，那策略1就是在第2天到第101天做多，其他日子空仓，其他策略类似。这6个策略如果跑完后的收益为正，就做多该因子；结果为负，就做空该因子。我们得到的各因子多头策略净值图为：



值得一提的是，在绘制上面两张图时，我们采用的隐状态序列都是每一天出现概率最大的隐状态。比如，在经过计算后某一天六种隐状态出现的概率分别为(0.1, 0.2, 0.2, 0.1, 0.3, 0.1)，那么模型就会认为这一天的隐状态是5号，但这样其实浪费了一些有关其他因子的信息。如果我们将每天的次日收益率与当日各隐状态出现的概率相乘，作为隐状态因子的“预期”收益率，再对每天的预期收益率构建多头/空仓策略并判断哪种因子做多、哪种因子做空，这样的方法可以反应更多模型的信息。这种方法操作相对而言较为困难，并且重要缺点是时间开销太大，感兴趣的同学可以自己加以尝试。提示：可以利用 hmmlearn 里的 predict_proba 函数。我们称这种方法为“平均预期”方法，而之前的以出现概率最大隐状态作为当日隐状态的方法称为“最大概率”方法。

本研究主要采取的还是最大概率的方法。在这种方法下，我们进行多头/空仓策略得到的结果是：在出现1号和6号因子的时候做多，出现2345号因子时做空。以此策略进行回测，可得到净值收益曲线和回撤曲线如下。蓝色曲线为净值收益曲线，对应y轴为左侧轴；红色为回撤曲线，y轴为右侧轴。

至此，我们的样本内回测就完成了。但这个回测方法其实存在一个问题：在预测隐状态序列时，我们其实用到了全体样本的特征因子。但事实上，我们如果想要真实使用这个方法，在预测下一日收益率时只有当日及之前特征因子和隐状态的信息，而没有之后特征因子的信息。所以接下来，我们要使用样本外回测。即：使用过去若干天（例：20天）的特征信息，拟合过去20天的隐状态序列，建立19对"前一日隐状态-后一日收益率"的关系，并利用这些关系构建多头策略以确定每种隐状态应做多还是做空；然后考察最后一天的隐状态，并以此决定下一交易日的交易策略。

另外，我们还希望将宏观数据纳入考量范围。由于研究对象是螺纹钢，所以我们采用了全国工业分行业增长速度中的黑色金属矿采选业增加值_同比增长(%)，具体可以参考聚宽宏观经济数据 。该指标为月度数据，所以为了将其与日度数据对齐，我们将每月的该数据对应进该月的每一个交易日中。由于rsi指标在数值较大时是买入信号，而黑色金属矿采选业增加值较大时也是买入信号，所以我们对二者进行处理，方法是：$rsi' = rsi * (1 + yoy/100)$，其中rsi'为新的特征因子，yoy 为黑色金属矿采选业增加值同比增长率。
同时，在进行样本内外回测时，我们希望能控制特征因子类型、样本外回测时向前回溯的天数和是否要采取归一化方法。归一化方法是指，我们在输入一个特征因子时可能对他的正态性或者偏态情况等不满意，就可以采取归一化的方式进行处理。在这个函数中，我们支持的归一化方法有 box_cox 方法，均值-方差标准化方法，以及映射至[0,1]区间的方法。
最终，我们构建的函数为：

```
backtest(func,delta_days=20,stdmtd=None,mac=None)。
```

其中，func 为特征因子名，如 RSI、ROC 等；注意，如果取的指标有多个返回值，需要在函数体内部进行修改。delta_days 为进行样本外回测时向前追溯的天数，stdmtd 为标准化的方法，mac 为宏观数据。

由于样本内回测用到了未来数据，所以样本内回测的结果都是比较好的。在多次进行回测后，我们发现使用 CCI 技术指标，使用均值方差标准化的方法得到的回测收益比较接近市场收益，其他方法大多很难跑赢市场。我们认为，单因子的方法可能会遗漏较多信息，并且我们选取的大多数指标都是比较流行的指标，所以很难取得超额收益。

func=CCI,delta_days=20,stdmtd='mean_std'

func=KD,delta_days=20,stdmtd='map01'

func=RSI,delta_days=20,stdmtd='mean_std',mac='black_metal_mining_yoy'

func=RSI,delta_days=20,stdmtd='mean_std'

问题和可能的改进方法

本研究仅考虑了使用单因子的情况，但就像研究结果显示的那样，我们的样本外回测结果并不好。日后可以尝试更多的因子，或者同时输入多个因子作为特征因子。

本研究没有考虑日内止盈和止损，以及加杠杆的情况。另外，也没有设置滑点。这些细节在未来进一步研究时都应该有所考虑。

参考文献

《基于单特征因子的隐马尔科夫模型在商品期货上的应用》，朱剑涛，东方证券
《Tutorial—hmmlearn 0.2.1 documentation》
《隐马尔可夫模型HMM》，刘建平Pinard，CSDNBlog

【数学课堂】线性代数系列 -- 线性映射

导语

一般在讨论数学对象的时候我们也会关注两个对象之间的映射，这些映射必须满足一些特定的性质以此来维持原有对象的性质，通过对这些映射的学习我们可以对数学对象有更好的了解。

在线性代数中，我们研究的最基本的数学对象是线性空间，在这些空间上特有的一种映射叫做线性映射。

线性映射

定义. 设 V 和 W 是两个 $\mathbb{R}$ 上的线性空间，如果一个映射 $f: V \to W$ 满足：

1. 对于 $v, w \in V$，有
$$f(v+w) = f(v) + f(w);$$
2. 对于 $v \in V$ 和 $r \in \mathbb{R}$，有
$$f(rv) = r \cdot f(v).$$

我们说 f 是从 V 到 W 的一个*线性函数* (linear function) 或*线性映射* (linear map) 或*线性算子* (linear operator)。

我们可以这么理解，线性映射 f 把 V 里的一个向量变成 W 里的一个向量，并且保持了它们作为向量的性质，也就是说：如果把 v 和 w 两个向量用 f 进行转换，那么先把它们相加再转换和先分别转换再进行相加最后的结果是一样的；并且，如果在转换前把 v 乘以实数 r，这和在转换后再乘以 r 的结果是一样。可以粗浅地理解为（不完全正确）：线性映射只不过是给 V 里的向量换了个名字，实际的线性性质不变。

本着这个“名变质不变”的理解，我们来看一个线性映射是把 V“变成”了 W 的一个子空间。不过，我们要证明，线性映射不会改变空间的零点。

命题. 设 V 和 W 是 $\mathbb{R}$ 上的线性空间，$f: V \to W$ 是一个线性映射，将 V 和 W 的零点（加法单位元）分别写为 $\mathbf{0}_V$ 和 $\mathbf{0}_W$，那么
$$f(\mathbf{0}_V) = \mathbf{0}_W.$$

证明. 根据线性映射保留加法的性质，可以计算

Undefined control sequence \square

定理. 设 V 和 W 是 $\mathbb{R}$ 上的线性空间，$f: V \to W$ 是一个线性映射，那么 f 的图像 $\operatorname{Im} f = f(v) : v \in V$ 是 W 的一个子空间。
证明. 我们检验线性子空间的三条性质：

1. 根据上边的命题，$f(\mathbf{0}_V) = \mathbf{0}_W$，所以 $\mathbf{0}_W \in \operatorname{Im} f$；
2. 取 $f(v_1), f(v_2) \in \operatorname{Im} f$，有
$$f(v_1) + f(v_2) = f(v_1 + v_2) \in \operatorname{Im} f;$$
3. 取 $f(v) \in \operatorname{Im} f$ 和 $r \in \mathbb{R}$，有
$$r \cdot f(v) = f(rv) \in \operatorname{Im} f.$$

这些性质证实了 $\operatorname{Im} f$ 是 W 的一个线性子空间。 Undefined control sequence \square

例子

到此为止我们讨论了线性映射的性质，但是还没有看到任何具体的线性映射的例子，接下来我们探索一个线性映射应该长成什么样子。先考虑最简单的情况，$V = W = \mathbb{R}^1$。假设 $f: \mathbb{R}^1 \to \mathbb{R}^1$ 是一个线性函数，那么它会把 1 映射到某个实数 a，也就是 $f(1) = a$，那么对于任何 $r \in \mathbb{R}^1$，由于 f 保留乘法性质，所以有
$$f(r) = f(r \cdot 1) = r \cdot f(1) = ra,$$

所以 f 就是乘以 a 的函数。如果 $a \neq 0$，那么 f 的逆函数是除以 a 的函数；如果 $a = 0$，那么 f 把所有的实数都映射到 0，是没逆函数的。

再稍微复杂一点，假设 f 是一个从 $\mathbb{R}^1$ 到 $\mathbb{R}^n$ 的线性函数。故技重施，设
$$f(1) = \left[\begin{array}{c|c} a_1\ a_2 & a_n \end{array}\right],$$

那么对于任何 $r \in \mathbb{R}$，有
$$f(r) = f(r \cdot 1) = rf(1) = r \cdot \left[\begin{array}{c|c} a_1\ a_2 & a_n \end{array}\right] = \left[\begin{array}{c|c} ra_1\ ra_2 & ra_n \end{array}\right].$$

也就是说这个函数就是把输入的值乘上一个固定的向量 $(a_1, a_2, \ldots, a_n)$，虽然 $\mathbb{R}^1$ 里有无数个元素，但我们发现只要知道了 f 在一个数上的映射，就知道了它在所有其他数上的效果。

最后我们考虑最复杂的情况，假设 f 是 $\mathbb{R}^n$ 到 $\mathbb{R}^m$ 的一个线性映射，首先考虑 $\mathbb{R}^n$ 里一些最基本的向量
$$e < em > 1 = \left[\begin{array}{c|c} 1\ 0 & 0 \end{array}\right], e_2 = \left[\begin{array}{c|c} 0\ 1 & 0 \end{array}\right], \ldots, e_n = \left[\begin{array}{c|c} 0\ 0 & 1 \end{array}\right],$$

这里 e_i 就是第 i 个位置是 1，其他位置都是 0 的向量。那么，f 会把 e_1 映射到 $\mathbb{R}^m$ 中的某一个向量，我们将它写作

$$f(e_1) = \left[a < /em > 1, 1 \; a < em > 2, 1 \vdots a < /em > m, 1 \right],$$

也会把 $e < em > 2$ 映射到 $\mathbb{R}^m$ 里的某个向量

$$f(e_2) = \left[a < /em > 1, 2 \; a < em > 2, 2 \vdots a < /em > m, 2 \right],$$

以此类推，在 $e < em > i$ 上有如下的映射

$$f(e_i) = \left[a < /em > 1, i \; a < em > 2, i \vdots a < /em > m, i \right].$$

那么如果我们取某个向量

$$\mathbf{x} = \left[x < em > 1 \; x_2 \vdots x_n \right] \in \mathbb{R}^n,$$

它可以被改写为

Unknown environment 'align'

那么可以推算出 f 在 $\mathbf{x}$ 上的映射为

Undefined control sequence \<

也就是说，只要我们知道了 f 在 $e < em > 1, e_2, \ldots, e_n$ 这 n 个向量上的映射，那么我们就可以轻松推算出 f 在任何其他向量上的映射，而 f 的全部信息都包含在

$$a < /em > i, j : i = 1, 2, \ldots, n; \; j = 1, 2, \ldots, m$$

这 nm 个数中。正因如此，有时我们会干脆将 f 表示为

Undefined control sequence \<

这就是下一篇中将要讲解的，充满神秘感却又莫名奇妙的方状物体，矩阵。

文章迭代记录
v1.0，2017-12-08 文章上线

【数学课堂】线性代数系列 -- 线性子空间的交集和加和

概述

我们知道一个线性空间 V 至少有两个子空间 $\mathbf{0}_V$ 和 V 本身，当然，它还可以有很多其他的子空间。你也许好奇这些子空间是否可以拼接或者拆解成更大或者更小的子空间，答案肯定的。首先我们来看，两个子空间的交集是一个子空间。

交集空间

定理. 设 V 是 $\mathbb{R}$ 上的一个线性空间，$W, U \subseteq V$ 都是 V 的子空间，那么 $W \cap U$ 也是 V 的子空间。

证明. 假设 W 和 U 都是 V 的子空间，我们使用线性子空间一文中的定理来验证 $W \cap U$ 也是 V 的子空间：

1. $\mathbf{0}_V \in W$ 并且 $\mathbf{0}_V \in U$，所以 $\mathbf{0}_V \in W \cap U$;
2. 如果 $v_1, v_2 \in W \cap U$，那么由于 W 和 U 分别都封闭于加法运算，所以 $v_1 + v_2 \in W$ 并且 $v_1 + v_2 \in U$，因此 $v_1 + v_2 \in W \cap U$;
3. 逻辑同 2，*留给读者作为证明练习*。
   由于满足三个性质，那么根据定理，$W \cap U$ 是 V 的一个线性子空间。 Undefined control sequence \square

接下来考虑一个例子。假设 $V = \mathbb{R}^3$ 是三维的欧氏空间，定义两个子空间 $W = (x, y, 0) \in \mathbb{R}^3$ 和 $U = (x, 0, z) \in \mathbb{R}^3$，它们分别是所有 z 坐标为零的向量和所有 y 坐标为零的向量所构成的二维空间。认真的读者应该自己尝试验证 W 和 U 的确是 V 的子空间。那么 W 和 U 的交集是

$$W \cap U = (x, 0, 0) \in \mathbb{R}^3,$$

是所有 y 和 z 坐标为零的向量。根据以上的定理，$U \cap W$ 是 V 的一个线性子空间，读者也可以根据线性子空间的定义来确认这个性质。

加和空间

当然，下面一个自然会被问到的问题就是：那么两个子空间的并集是不是一个子空间呢？答案是否定的，我们用上面提出的 W 和 U 就可以看见一个反例，这两个子空间的并集是

$$W \cup U = (x, y, z) \in \mathbb{R}^3 : y = 0 \text{ 或者 } z = 0.$$

很明显 $(0, 1, 0)$ 和 $(0, 0, 1)$ 都是 $W \cup U$ 的元素，但是它们的和

$$(0, 1, 0) + (0, 0, 1) = (0, 1, 1)$$

的 y 坐标和 z 坐标都不是零，自然不属于 $W \cup U$。

那么问题来了，我们怎么从已有的子空间中构建出更大的子空间呢？单单取它们的并集看起来不行，因为一个子空间的元素加上另一个子空间的加和不一定属于它们的并集里。那么如果把两组元素的加和也都算上，也许就可以是一个完整的子空间了。

\noindent \textbf{定义. }设 V 是 $\mathbb{R}$ 上的一个线性空间，W 和 U 都是 V 的线性子空间，我们定义 W 和 U 的\emph{加和 (sum)}（写为 $W + U$）为

$$W + U := w + v : w \in W, u \in U.$$

也就是所有 W 的元素和 U 的元素的加和的结果。

读者也许好奇，$W + U$ 真的包含 W 和 U 吗？答案是肯定的，因为 $\mathbf{0}_V \in U$，那么对于每一个 $w \in W$，都有

$$w = w + \mathbf{0}_V \in W + U,$$

所以 $W \subseteq W + U$。同理，也有 $U \subseteq W + U$。下面我们证明 $W + U$ 的确是一个线性子空间。

定理. 设 V 是 $\mathbb{R}$ 上的一个线性空间，W 和 U 都是 V 的线性子空间，那么 $W + U$ 是 V 的子空间，并且它是包含了 $W \cup U$ 的最小的子空间。

证明. 我们验证 $W + U$ 满足三个线性子空间的性质：

1. 由于 W 和 U 都是 V 的子空间，所以 $\mathbf{0}_V \in W \cap U$，那么
$$\mathbf{0}_V = \mathbf{0}_V + \mathbf{0}_V \in W + U.$$
2. 取 $v_1, v_2 \in W + U$，它们可以被写为以下形式
$$v_1 = w_1 + u_1,\ v_2 = w_2 + u_2;\ w_1, w_2 \in W,\ u_1, u_2 \in U,$$
那么
$$v_1 + v_2 = (w_1 + u_1) + (w_2 + u_2) = (w_1 + w_2) + (u_1 + u_2) \in W + U.$$
3. 取 $w + u \in W + U$，以及 $a \in \mathbb{R}$，根据线性空间的性质，有
$$a \cdot (w + u) = aw + au \in W + U$$
（因为 $aw \in W$ 并且 $au \in U$）。因为满足上面的三个性质，所以 $W + U$ 是 V 的子空间。

接下来我们证明 $W + U$ 是包含了 W 和 U 的最小的子空间。假设 $X \subseteq V$ 也是一个子空间，并且 $W \cup U \subseteq X$。取任意一个 $v \in W + U$，它可以被写为 $v = w + u$，其中 $w \in W$ 并且 $u \in U$。我们知道 w 和 u 都是 X 的元素，那么它们的加和，v，必定也是 X 的元素。由此判断 $W + U \subseteq X$。
Undefined control sequence \square

最后，我们在再用

Unknown environment 'align'

来举例。根据子空间的加和的定义，有

$$W + U = (x_1, y, 0) + (x_2, 0, z) : x_1, x_2, y, z \in \mathbb{R} = (x_1 + x_2, y, z) : x_1, x_2, y, z \in \mathbb{R}.$$

机智的读者一定已经发现，其实 $W + U$ 就等于整个 $\mathbb{R}^3$。

文章迭代记录
v1.0，2017-11-24 文章上线

【数学课堂】线性代数系列 -- 线性子空间

回顾

一个线性空间是由很多的点组成的，在这些点上具有加法和纯量乘法的运算，并且满足一些特定的性质。比如我们平时生活的三维世界就可以用线性空间 $\mathbb{R}^3$ 表示，但是在我们的三维世界里也包含着二维世界，比如一张纸上的二维世界就可以用 $\mathbb{R}^2$ 表示，而它也是包含在 $\mathbb{R}^3$ 里的。像这样的包含在一个大的线性空间里的一个小线性空间，叫做一个子空间。

线性子空间

定义. 设 V 是 $\mathbb{R}$ 上的一个线性空间，$W \subseteq V$ 是一个子集，并且 W 在继承了 V 的加法和纯量乘法的运算的情况下是一个线性空间，那么 W 是 V 的一个 *线性子空间* (linear subspace)。

这里解释一下"继承"某种运算的含义。V 中的两个元素可以进行相加，或者和 $\mathbb{R}$ 中的元素进行相乘，因为 W 是 V 的一个子集，所以 W 的元素也都是 V 的元素，那么自然可以用同样的运算规则进行相加和相乘的。当然，一个子集并不一定能继承这些运算，比如说，假设 $V = \mathbb{R}^2$，那么 $S = (1,2), (3,4)$ 是 V 的一个子集，但是它并没法继承加法运算，因为

$$(1,2) + (3,4) = (4,6) \notin S,$$

加法得出的结果不是 S 的一个元素，所以我们不能在 S 上进行加法运算。还可以考虑 $T = (x,x) \in \mathbb{R}^2 : x \geq 0$ 是所有两个坐标数值都一样并且大于等于零的点，如果 $(a,a), (b,b) \in T$，那么

$$(a,a) + (b,b) = (a+b, a+b) \in T,$$

所以 T 是可以继承加法运算的。但是（总有这么个但是）！如果我们尝试进行乘法运算的话，比如拿 $-1 \in \mathbb{R}$ 和 $(1,1) \in V$，有

$$-1 \cdot (1,1) = (-1,-1) \notin V,$$

乘法结果不属于 V，所以 V 是不能继承乘法运算的。

当一个子集不能继承线性空间的加法和乘法运算的时候，它自然不是一个子线性空间。

在子空间的定义中，我们要求子集 W 必须是一个线性空间，也就是说它必须满足线性空间定义中的六个性质，而如果我们每次都要验证这六个性质的话那会被累死的。下面我们看一个可以简化验证的工作量的定理。

定理. 设 V 是 $\mathbb{R}$ 上的一个线性空间，W 是 V 的一个子集，那么 W 是 V 的子空间当且仅当它满足下面的三个性质：

1. 加法的单位元 $\mathbf{0}_V \in W$；
2. 如果 $w, u \in W$，那么 $w + u \in W$；
3. 如果 $w \in W$ 并且 $r \in \mathbb{R}$，那么 $r \cdot w \in W$。

这个定理中的后两条 2 和 3 就是说 W 可以完整继承 V 的两种运算，而第一条实说 W 包含了加法单位元，其他的几条性质都不需要验证了。下面我们给出证明：

证明. ($\Rightarrow$) 如果 W 是一个子空间，那么它必定满足条件 1、2 和 3，这不需要证明。

($\Leftarrow$) 如果 W 满足性质 1、2、3，那么由于两种运算本身都满足交换性、结合性、分配性并且拥有乘法单位元，所以这些性质在被 W 继承后依旧满足，加法单位元的存在是条件 1，剩下只需要验证加法逆元素。设 $w \in W$，那么 w 在 V 中的加法逆元素是 $-w = -1 \cdot w$，根据条件 3，w 和实数的乘积是 W 的元素，所以 $-w \in W$，完成了证明。 Undefined control sequence \square

我们举个例子，设 $V = \mathbb{R}^3$，$W = (x, 0, 2x) \in \mathbb{R}^3$，那么很明显 $W \subseteq V$，我们来验证 W 是 V 的一个子空间：

1. 设 $x = 0$，可以看出 $(0,0,0) = \mathbf{0}_{\mathbb{R}^3} \in W$；
2. 如果 $(a,0,2a), (b,0,2b) \in W$，那么
$$(a,0,2a) + (b,0,2b) = (a+b, 0, 2a+2b) = (a+b, 0, 2(a+b)) \in W,$$
3. 如果 $(a,0,2a) \in W$ 并且 $r \in W$，那么
$$r \cdot (a,0,2a) = (ra, 0, 2ra) \in W.$$

由此可见 W 的确是 V 的子空间。

很明显，一个线性空间 V 也是它自己的一个子空间，并且要注意，V 的最小的子空间是只包含零点的空间 $\mathbf{0}_V$。很多初学者误认为空集 $\emptyset \subseteq V$ 也是一个子空间，其实不然，因为它不满足包含加法单位元（零点）的性质。

文章迭代记录
v1.0，2017-11-14 文章上线

【数学课堂】线性代数系列 -- 线性空间

前言

本篇是量化课堂数学课的第一篇正文，之后会陆续推出线性代数、数学分析、测度论、概率统计、随机过程等量化分析相关的数学课程。想对线性代数有概括性了解的同学可以参阅为什么学线性代数。

导语

线性空间和向量是线性代数中最基础的概念，任何学习理工科的人都绕不开的概念。“线性”中的“线”可以理解为实数线的“线”，只要我们用到实数的时候就涉及到了线性的概念。不夸张地讲，现实中的绝大多数数理概念都是线性的，即便不是，它在局部上也会有近似线性的结构。

一般来讲，线性空间的定义基于一种叫做*域* (field) 的代数结构。但是在实际应用中我们并不需要对一般性的域进行讨论，而只需要使用我们最熟悉的域 --- 实数 $\mathbb{R}$。为了不给读者带来混淆，数学课堂线性代数课程都只基于实数域，而对一般性域感兴趣的读者可以自行参阅任何抽象代数的入门教材。

线性空间

下面我们定义线性空间。

定义. 一个 $\mathbb{R}$ *上的线性空间* (vector space over $\mathbb{R}$) 是一个集 V，其具有加法 $+ : V \times V \to V$ 和纯量乘法 $\cdot : \mathbb{R} \times V \to V$ 两种运算，并满足以下性质：

- *交换性* (commutativity)：
  对于任何 $u, v \in V$ 都有 $u + v = v + u$;
- *结合性* (associativity)：
  $(u + v) + w = u + (v + w)$ 和 $a \cdot (b \cdot v) = (a \cdot b) \cdot v$ 满足于任何 $a, b \in \mathbb{R}$ 和 $u, v, w \in V$;
- *加法单位元* (additive identity)：
  存在一个元素 $0_V \in V$，有 $0_V + v = v$ 满足于任何 $v \in V$;
- *加法逆元素* (additive inverse)：
  对于任何 $v \in V$ 都存在一个 $w \in V$，满足 $v + w = 0_V$，写为 $w = -v$;
- *乘法单位元* (multiplicative identity)：
  $1 \cdot v = v$ 满足于任何 $v \in V$，这里的 1 就是实数 1;
- *分配性* (distributivity)：
  $a \cdot (u + v) = a \cdot u + a \cdot v$ 和 $(a + b) \cdot v = a \cdot v + b \cdot v$ 满足于任何 $a, b \in \mathbb{R}$ 和 $u, v \in V$。

线性空间里的元素都叫做*向量* (vector)。

出于书写的简便性，我们一般将乘法的点省略掉。比如有 $a \in \mathbb{R}$ 和 $v \in V$，可以将它们的乘积写作 $av = a \cdot v$。另外需要指出，虽然在线性空间的定义中没有减法运算，但是由于加法逆元素 $-w$ 的存在，我们可以将减法定义为 $v - w = v + (-w)$。

我们在实际应用中会用到的线性空间基本上都是 n *维欧氏空间* (n dimensional Euclidean space)，写为 $\mathbb{R}^n$，这些线性空间的元素都用 $(x_1, x_2, \ldots, x_n)$ 表示，其中每一个 x_i 都是一个实数。比如说，$(3, 5)$ 是 $\mathbb{R}^2$ 中的向量，而 $(1.5, 0.38, 8)$ 是 $\mathbb{R}^3$ 中的向量，这里 x_1, x_2, x_3 的数字也就是我们常说的 *坐标* (coordinate)。

为了确认 $\mathbb{R}^n$ 的确是一个线性空间，我们必须定义 $\mathbb{R}^n$ 上的加法和乘法运算，并逐一检测它的线性空间的性质。虽然这些性质检验起来比较简单，但数学是一门严谨的学科，只有在亲自证明之后才能理直气壮地下结论。

定义和定理. 在集合 $\mathbb{R}^n$ 上定义加法和纯量乘法如下：对于 $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \mathbb{R}^n$ 和 $a \in \mathbb{R}$，定义

$$(x_1, \ldots, x_n) + (y_1, \ldots, y_n) := (x_1 + y_1, \ldots, x_n + y_n);$$

$$a \cdot (x_1, \ldots, x_n) := (ax_1, \ldots, ax_n).$$

那么 $\mathbb{R}^n$ 是一个线性空间。

证明. 我们逐一检验线性空间的性质：设 $(x_1, \ldots, x_n), (y_1, \ldots, y_n), (z_1, \ldots, z_n) \in \mathbb{R}^n$ 和 $a, b \in \mathbb{R}$，

- 交换性：

Unknown environment 'align'

- 结合性：

Unknown environment 'align'

以及

Unknown environment 'align'

• 加法单位元：设 $\mathbf{0} < em > \mathbb{R}^n := (0, 0, \ldots, 0)$，那么对于任何 $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$，有

$$\mathbf{0} < /em > \mathbb{R}^n + \mathbf{x} = (0 + x < em > 1, \ldots, 0 + x_n) = (x_1, \ldots, x_n) = \mathbf{x},$$

所以 $\mathbf{0} < /em > \mathbb{R}^n$ 是 $\mathbb{R}^n$ 中的加法单位元。

• 加法逆元素：对于 $\mathbf{x} = (x < em > 1, \ldots, x_n) \in \mathbb{R}^n$，有 $\mathbf{y} = (-x_1, \ldots, -x_n) \in \mathbb{R}^n$，满足

$$\mathbf{x} + \mathbf{y} = (x_1 + (-x_1), \ldots, x_n + (-x_n)) = (0, \ldots, 0) = \mathbf{0} < /em > \mathbb{R}^n,$$

所以加法逆元素是存在的。

• 乘法单位元：很明显，

$$1 \cdot (x_1, \ldots, x_n) = (1x_1, \ldots, 1x_n) = (x_1, \ldots, x_n),$$

所以 1 是 $\mathbb{R}^n$ 的乘法单位元。

• 分配性：***留给读者作为证明练习。***

因为满足所有线性空间的条件，因此 $\mathbb{R}^n$ 的确是一个线性空间。 Undefined control sequence \square

例子

我们日常生活的三维世界就是一个线性空间，它也是我们对线性空间所有直觉的由来。我们现在请来中国人民的好朋友，小明同学，来进行一些示范。因为小明是中国教科书上最传奇的人物，所以他可以上天入地，如果我们发现小明做了一些超乎常理的事情（比如会飞），无需惊讶。

假设小明他站在地上，然后他向正右走了 5 米，向正前方走了 3 米，接着再向上飘了 8 米，又向左飘了 1 米，我们可以把这四个动作用 $\mathbb{R}^3$ 中的向量分别表示为 $\mathbf{x} = (5, 0, 0), \mathbf{y} = (0, 3, 0), \mathbf{z} = (0, 0, 8), \mathbf{w} = (-1, 0, 0)$，那么他现在的坐标是

$$\mathbf{v} := \mathbf{x} + \mathbf{y} + \mathbf{z} + \mathbf{w} = (4, 3, 8).$$

假设他把上面的动作再都按顺序重复两次，那么他的新坐标是

$$\mathbf{v} + 2\mathbf{v} = (4, 3, 8) + 2 \cdot (4, 3, 8) = (12, 9, 24).$$

然后小明从天上掉了下来，他的高度坐标清零，新坐标是

$$(12, 9, 0),$$

他的下坠过程用向量表示为

$$(12, 9, 24) - (12, 9, 0) = (0, 0, 24),$$

总共下坠了 24 米。

因为我们平时只能看到三维的空间，所以很难想象四维以上的空间，感觉那都像科幻故事里的事情。其实不然，高维度的空间其实是非常简单的 --- 只要你不去想它的长相。假设小明学校有 1000 个学生，并且学校里流行收集红、蓝、绿、黄色的玻璃珠，那么每个学生的四种玻璃珠的数量 $(\text{红}, \text{蓝}, \text{绿}, \text{黄})$ 即是四维空间 $\mathbb{R}^4$ 上的一个点，整个学校的学生代表 1000 个这样的点。

双胞胎小刚和小钢两人的玻璃珠数量一样，分别有 5 个红色、1 个蓝色、2 个绿色并且没有黄色，那么他们其中一人的玻璃珠数量可以表示为向量 $(5, 1, 2, 0)$，两个人的玻璃珠总和是

$$2 \cdot (5, 1, 2, 0) = (10, 2, 4, 0).$$

如果小红只有 100 个红色珠子，那么她的向量是 $(100, 0, 0, 0)$。而小明是学校的“万珠统领”，每种玻璃珠各有 2500 个，他的向量是 $(2500, 2500, 2500, 2500)$。小明、小红、小刚和小钢四个人的珠子加在一起是

$$(2500, 2500, 2500, 2500) + (100, 0, 0, 0) + 2 \cdot (5, 1, 2, 0) = (2610, 2502, 2504, 2500).$$

结语

线性空间是很基础也很容易想象的数学结构，但严格的数学定义和论证可能让初学者有些困惑，希望同学们努力尝试去理解这些概念，对以后的学习会有很大帮助。

文章迭代记录
v1.0，2017-11-02 文章上线

【数学课堂】为什么学线性代数

前言

量化课堂也有一阵子没更新了，从现在起将陆续推出一套数学课堂，内容将含括量化交易会用到的数学基础：线性代数、数学分析、测度论、概率统计、随机过程，还有为没经历过严格数学训练的同学准备的数学逻辑和证明方法基础。数学课堂目标是能让学员有能力理解量化分析相关文献中的数学和统计学论证。目标不小，咱慢慢来。

先从线性代数开始。

先从“为什么要学线性代数”开始。

导语

线性代数是数学中最基础却极其重要的一门学科，因为任何科学领域都绕不过线性代数，把线性代数称数学中最有必要掌握的一门也不为过。就拿其他数学分支来说，代数中的群表示论的基本是从任意群到矩阵群（一般线性群的子群）的同态映射；分析中的泛函分析是对无限维度的赋范线性空间和其上的线性算子的研究；几何中的流形是局部同胚于线性空间的拓扑空间，它们的切从是用线性空间来表示，这些线性的性质也延伸到上同调的理论；在离散数学中，代数图论是用邻接矩阵来表示一个图并用线性代数的技术展开研究，还有正如其名的矩阵胚，是有限域上的矩阵的一种延伸，在图论和组合学中有重要应用。

当然线性代数的作用远不局限于数学中，在物理、工程、计算机、金融、人工智能等科学领域中，线性代数以及上述需要线性代数的数学分支都有不计其数的应用场景。本篇介绍线性代数在各个方面的一些应用，覆盖面肯定不全，但足以展示线性代数的常用概念以及它对量化交易的重要性。

解决线性系统

线性最直接的应用应该就在于解决线性系统。大家一定都熟悉鸡兔同笼的问题。我们就举个例子，有一个笼子里有好多鸡和兔子和蚂蚁，三个物种分别有 2、4、6 只脚。鸡和兔子加起来一共有 22 只脚，兔子和蚂蚁加起来一共 46 只脚，鸡和蚂蚁加起来一共 60 只脚，请问三种动物各有多少。

设我们有鸡 x 只、兔 y 只和蚂蚁 z 只，那么上面的条件可以写为系统

Unknown environment 'align'

通过线性代数的符号我们可以将这个系统写为矩阵和向量的乘积

$$\begin{bmatrix} 2 & amp;4 & amp;0 \\ 0 & amp;4 & amp;6 \\ 2 & amp;0 & amp;6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 22 \\ 46 \\ 60 \end{bmatrix}.$$

使用 NumPy（或者手算，3×3 的矩阵并不大），我们计算出等式中矩阵的逆矩阵为

$$\begin{bmatrix} \frac{1}{4} & amp;-\frac{1}{4} & amp;\frac{1}{4} \\ \frac{1}{8} & amp;\frac{1}{8} & amp;-\frac{1}{8} \\ -\frac{1}{12} & amp;\frac{1}{12} & amp;\frac{1}{12} \end{bmatrix},$$

所以可以计算出

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & amp;-\frac{1}{4} & amp;\frac{1}{4} \\ \frac{1}{8} & amp;\frac{1}{8} & amp;-\frac{1}{8} \\ -\frac{1}{12} & amp;\frac{1}{12} & amp;\frac{1}{12} \end{bmatrix} \begin{bmatrix} 22 \\ 46 \\ 60 \end{bmatrix} = \begin{bmatrix} 9 \\ 1 \\ 7 \end{bmatrix},$$

有 7 只鸡、1 只兔子和 9 只蚂蚁。

空间的操作

线性代数中两个主要的数学对象是空间本身，叫做线性空间，和对线性空间的各种操作，叫做矩阵，它等同于线性映射，有时也叫做线性变换。比如说，一个二维的旋转矩阵的形式为

$$R < em > \theta = \begin{bmatrix} \cos\theta & amp;\sin\theta \\ -\sin\theta & amp;\cos\theta \end{bmatrix},$$

它的作用是将整个空间旋转 θ 的弧度（radian）。比如说，我们有一张图片（图片源：SciPy 数据库）

这个图片的像素为 1024×768，它的信息被储存为一个矩阵的数据

$$\left[c < /em > 1,1 \quad amp;\dots \quad amp; c < em > 1,1024 \vdots \quad amp; \ddots \quad amp; \vdots c < /em > 768,1 \quad amp;\dots \quad amp; c < em > 768,1024 \right],$$

$c < /em > x, y$的值代表这个图片在坐标 (x, y) 的颜色。我们想把这个图片旋转 60°，也就是 $\frac{\pi}{3}$，那么就要把这个图片所在的整个二维空间用旋转矩阵 $R < em > \frac{\pi}{3}$ 进行变换。一个坐标 $[\, x \; y \,]$ 在转变之后的新坐标是

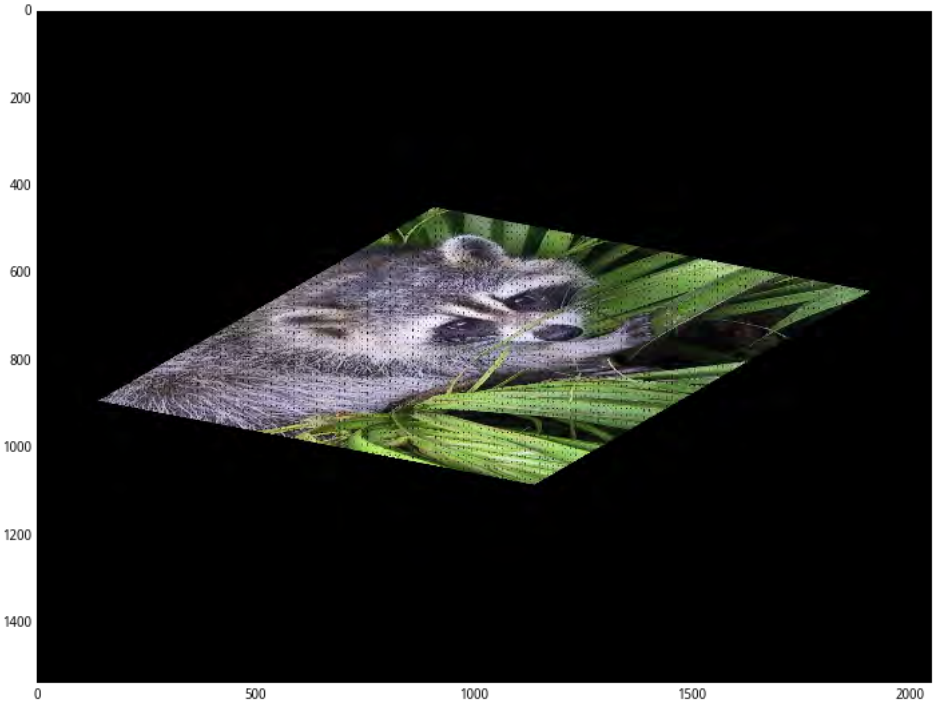
$$[\, x < /em > \text{new} \; y < em > \text{new} \,] = R < /em > \frac{\pi}{3} [\, x \; y \,].$$

那么就在一个新的画板上，把坐标 $(x < em > \text{new}, y < /em > \text{new})$的颜色定义为 $c_{x,y}$，得到的就是旋转 60° 之后的图像结果（因为旋转之后的坐标不是整数，整数化之后留下一些黑点，这需要其他的图像处理技巧来解决，这里就不延伸）

然后我们可以使用另一个矩阵

$$[\, 1.5 \quad amp; 0 \; 0 \quad amp; 0.5 \,]$$

来进行同样的操作，这次得出的结果是把图片按 x 轴拉伸为 1.5 倍并按 y 轴挤压为 0.5 倍，得出结果



线性回归

线性回归是统计学中一个常用的工具，给定一组二维空间上的点（后面图中的蓝点）

$$(x < em > 1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

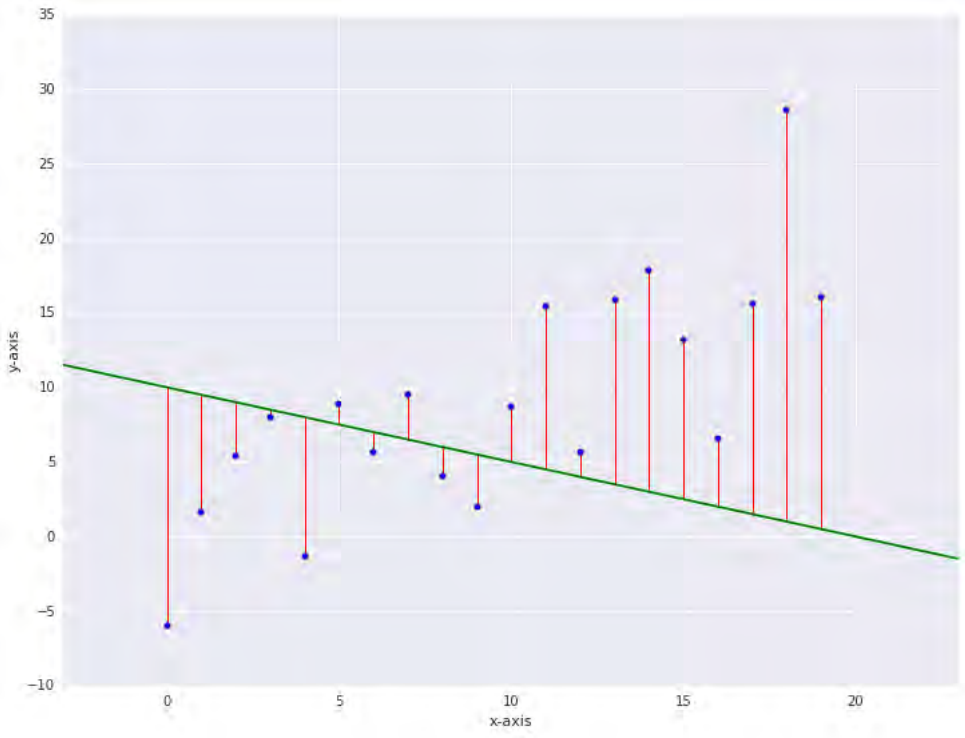
我们想找到一条线（后面图中的绿线）

$$y = ax + b$$

并希望这条绿线和这些蓝点是最贴近的。评判"最近"的方法就是使数据点离之线的距离平方和（图中红色线段的平方的和）

$$\mathrm{RSS}(a, b) = \sum < /em > i = 1^n (y_i - (ax_i + b))^2$$

是最小的。



但是，直接分析这个 RSS 函数比较困难，这时就能用到矩阵的强大表达力了，我们可以把这个公式写为矩阵的形式。设·

$$\mathbf{X} = \begin{bmatrix} 1 & amp; x < em > 1 \, 1 & amp; x_2 \, \vdots & amp; \vdots \, 1 & amp; x < /em > n \end{bmatrix}$$

和

$$\mathbf{y} = \begin{bmatrix} y < em > 1 \\ \vdots \\ y_n \end{bmatrix}$$

以及

$$\beta = [\, a \; b \,],$$

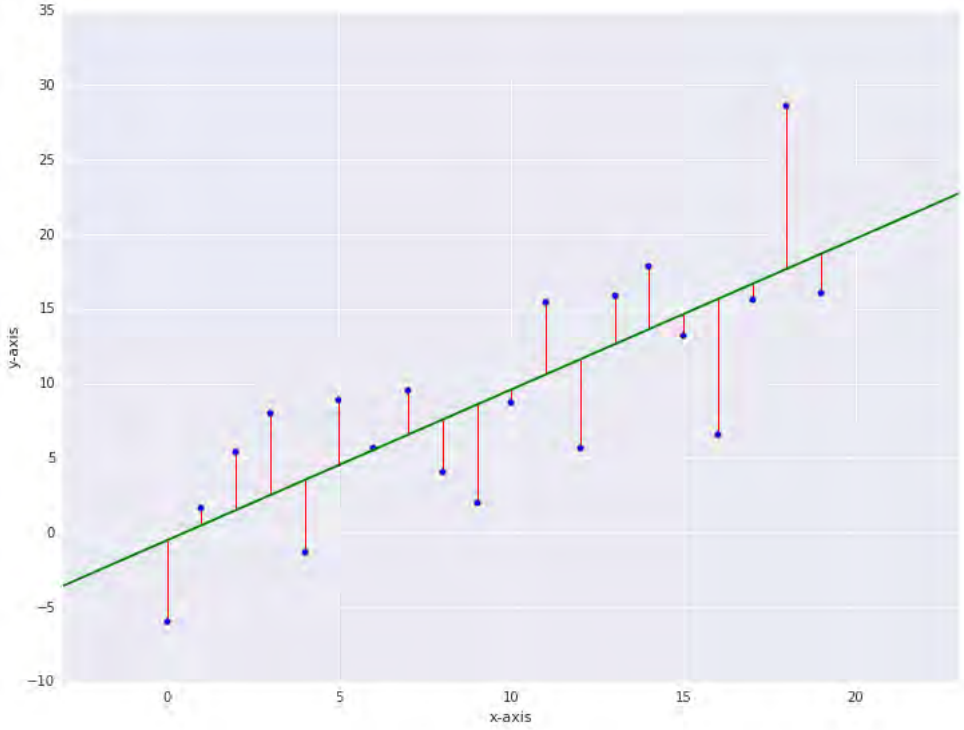
上面的公式可以被转写为矩阵形式

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^{\mathsf{T}}(\mathbf{y} - \mathbf{X}\beta).$$

然后通过矩阵微积分的方法可以推导出，当

$$\beta = (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y}$$

时，RSS 的值是最小的，它给我们下图中的这条绿线。



当然，线性回归的公式 (1) 不仅仅适用于二维空间中，如果我们有更高维的数据点

$$(x < /em > 1, 1, x < em > 1, 2, \ldots, x < /em > 1, k, y < em > 1), (x < /em > 2, 1, x < em > 2, 2, \ldots, x < /em > 2, k, y < em > 2), \ldots, (x$$

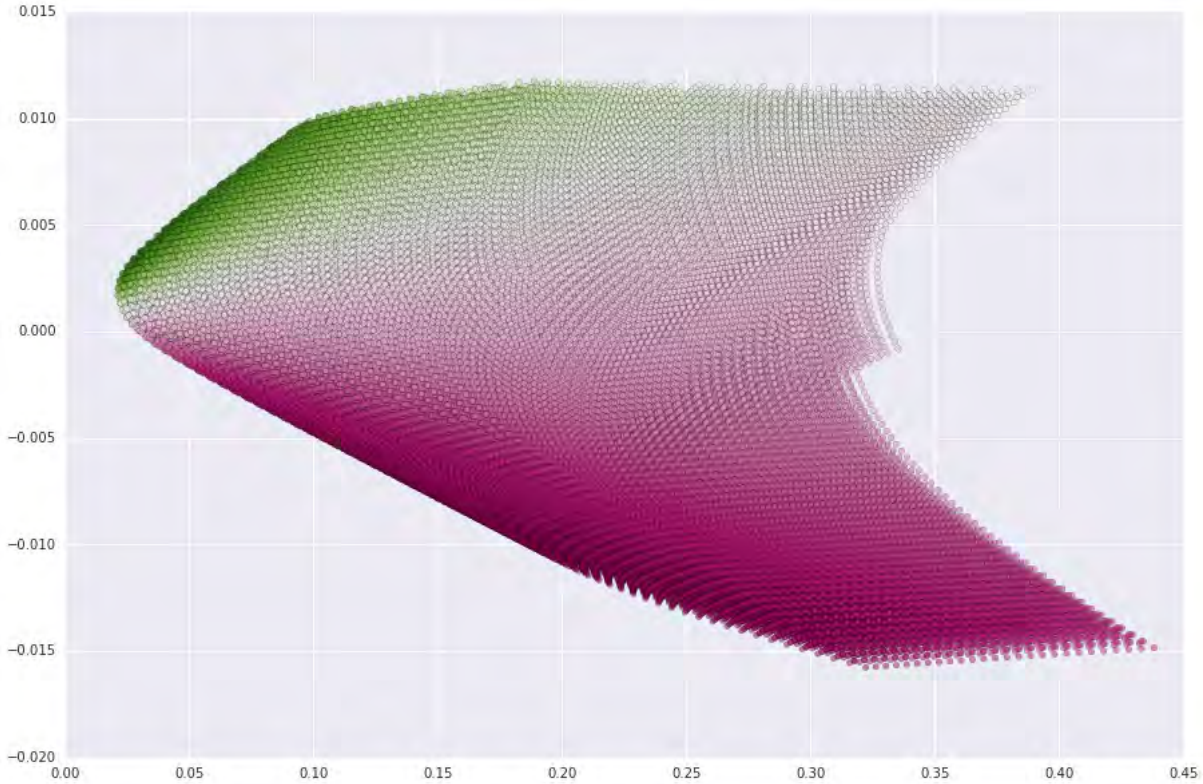
并想找到一条直线
$y = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_k x_k$
来拟合这组高维的数据，等式 (1) 的结论同样适用，只不过矩阵 $\mathbf{X}$ 会更大，有 $k+1$ 列，相应的 $\beta = (a_0, a_1, \ldots, a_k)$ 长度为 $k+1$。值得强调的是，如果没有矩阵的表述方法，公式 (1) 将会是不可想象的复杂，我们有这样简短的表达式全归功于矩阵的简便性。

MPT 资产配置

在现代资产配置理论（MPT）的模型中，假设有风险资产 $i = 1, 2, \ldots, n,$ 对应收益的随机变量 $r < em > i$。对于一个权重向量 $\mathbf{w} = (w_1, \ldots, w_n)$，这个向量对应的组合有收益率和方差

Undefined control sequence \<

所有这些资产组合的期望和标准差放到坐标图上构成一个子弹状的图像



所有这些点在最左端勾勒出的曲线叫做有效前沿，是给定一个收益率能找到的风险最小的组合，这条曲线对于资产配置理论有着至关重要的意义，而计算出这条曲线就需要矩阵的帮助。定义协方差矩阵

$$\mathbf{\Sigma} = \begin{bmatrix} \mathrm{Cov}(r_1, r_1) & amp; \mathrm{Cov}(r_1, r_2) & amp; \dots & amp; \mathrm{Cov}(r_1, r_n) \\ \mathrm{Cov}(r_2, r_1) & amp; \mathrm{Cov}(r_2, r_2) & amp; \dots & amp; \mathrm{Cov}(r_2, r_n) \\ \vdots & amp; \vdots & amp; & amp; \end{bmatrix}$$

以及

$$\mathbf{r} = \begin{bmatrix} \mathbb{E}[r_1] \, \mathbb{E}[r_2] \, \vdots \, \mathbb{E}[r_n] \end{bmatrix}, \ \mathbf{1}_n = \begin{bmatrix} 1 \, 1 \, \vdots \, 1 \end{bmatrix}.$$

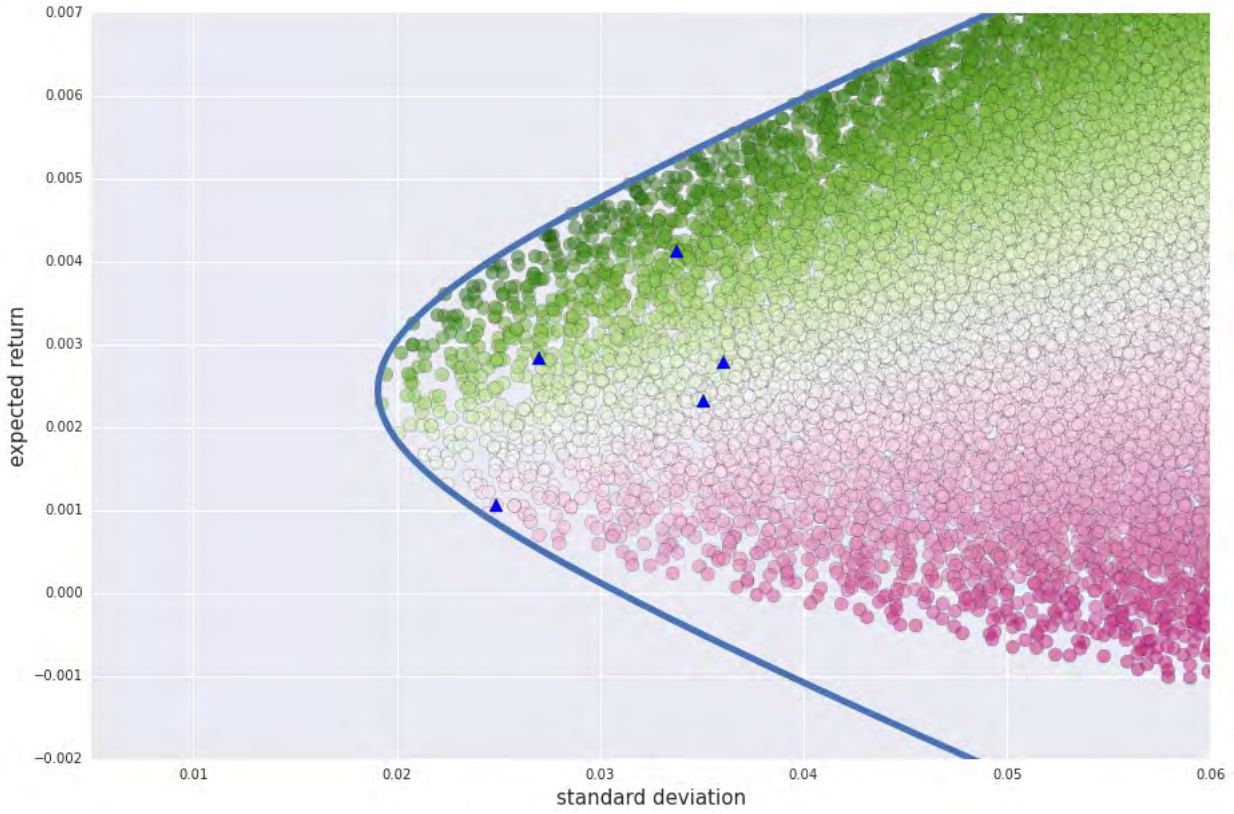
通过拉格朗日乘子，可以算出，设

$$a = \mathbf{r}^\top \mathbf{\Sigma}^{-1} \mathbf{r}, \ b = \mathbf{1}_n^\top \mathbf{\Sigma}^{-1} \mathbf{r}, \ c = \mathbf{1}_n^\top \mathbf{\Sigma}^{-1} \mathbf{1}_n,$$

可以算出有效前沿的曲线

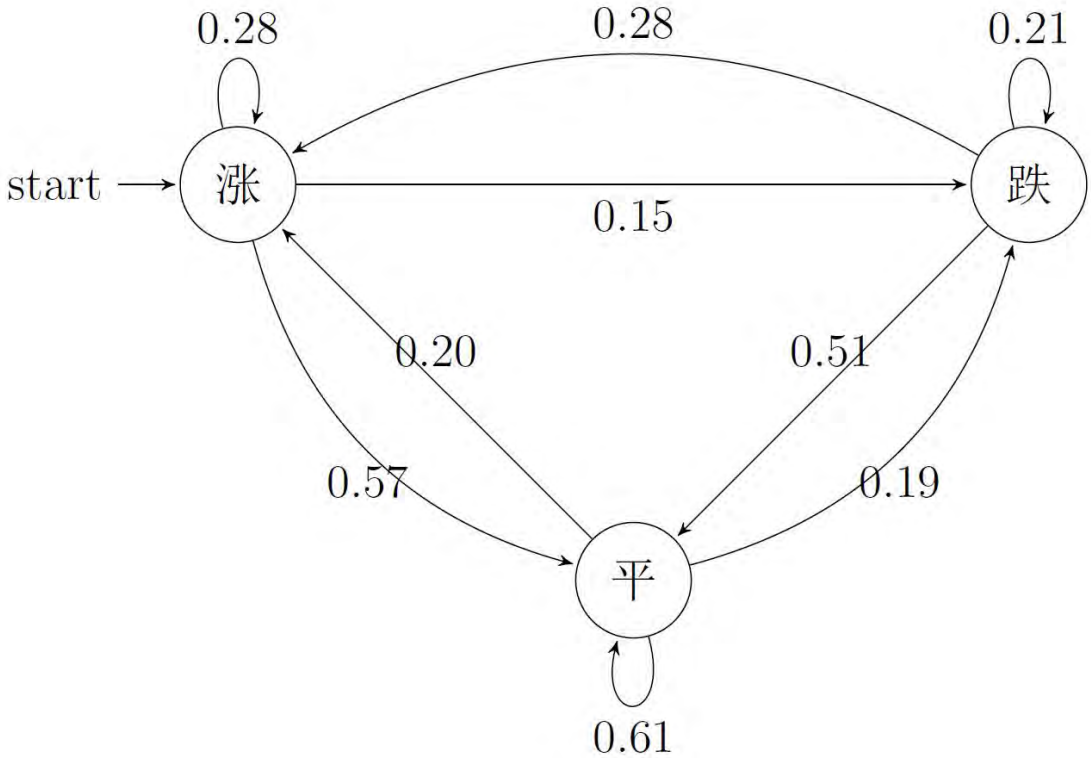
$$\sigma(\mu) = \sqrt{\frac{1}{ac - b^2}(c\mu^2 - 2b\mu + a)},$$

根据这个公式可以画出有效前沿的曲线



Markov 链

Markov 链是随机过程的一种，在这个链中每个时间点出现某种状态的概率取决于上一个时间的状态。比如，我们假设股市的涨跌符合 Markov 链的模型，将每天涨跌的状态分为“涨”、“平”、“跌”三种。“涨”代表当天涨幅 1% 以上，“跌”代表当天跌幅 1% 以上，当涨跌在正负 1% 之内时视为“平”。通过统计，我们得出下述概率关系：



在这个表中，从一个状态到另一个状态的数字代表从今天状态到明天状态的概率，比如说，如果今天涨，那么明天继续涨的概率是 28%，如果今天跌，那么明天平的概率是 51%。考虑这么一个问题，如果当前的状态是“涨”，那么 2 天后状态的概率分布是什么？

解决这个问题需要使用转移矩阵，这个矩阵的第 i 列的第 j 个数就是从状态 i 到状态 j 的概率，如下

$$T = \begin{matrix} \text{涨} & \text{平} & \text{跌} \end{matrix} \\ \begin{bmatrix} 0.28 & 0.20 & 0.28 \\ 0.57 & 0.61 & 0.51 \\ 0.15 & 0.19 & 0.21 \end{bmatrix} \begin{matrix} \text{涨} \\ \text{平} \\ \text{跌} \end{matrix}$$

想计算 2 天后的状态概率，我们取转移矩阵的 2 次方

$$T_2 = T^2 = [\,0.2344 \quad amp;0.2312 \quad amp;0.2392\ 0.5838 \quad amp;0.583 \quad amp;0.5778\ 0.1818 \quad amp;0.1858 \quad amp;0.183\,],$$

从第一列（也就是对应着“涨”的那里一列）读出来，如果今天是“涨”，那么两天后有 23.44% 的概率是涨，58.38% 会平，另有 18.18% 会跌。

那如果是 250 天之后呢？我们取转移矩阵的 250 次方。这个运算虽然对于人脑来说有点大，但用计算机来算还是很快的，

$$T_{250} = T^{250} \approx [\,0.2334 \quad amp;0.2334 \quad amp;0.2334\ 0.5822 \quad amp;0.5822 \quad amp;0.5822\ 0.1843 \quad amp;0.1843 \quad amp;0.1843\,].$$

这个矩阵的三列的的数都是一样的，理解起来也很直观，由于时间隔得太过久远，具体今天是什么状态对 250 天后的状态已经没有什么影响了，这个矩阵的一列代表的就是三种状态整体的分布，也就是说在整体上，每一天都有 23.34% 的概率是涨，58.22% 概率平，18.43% 的概率跌。\

有些时候转移矩阵不一定收敛得这么快（尤其是状态很多矩阵很大的情况下），可能计算出 T^{1000} 发现每列之间的的差别还是比较大，而我们又想知道几种状态无条件的概率分布，那么就需要计算转移矩阵幂的极限

$$T < em > \infty = \lim < /em > k \to \infty T^k.$$

这个看起来可能无从下手，但通过对角化的方法可以解决这个问题，即把 T 分解为

$$T = PDP^{-1},$$

这里 P 是一个可逆矩阵，D 是一个对角矩阵，它在对角线上的值是 T 的特征值，即

$$D = \begin{bmatrix} \lambda < em > 1 \quad amp;0 \quad amp;\dots \quad amp;0\ 0 \quad amp;\lambda_2 \quad amp;\dots \quad amp;0 \vdots \quad amp;\vdots \quad amp;\ddots \quad amp;\vdots 0 \quad amp;0 \quad amp;\dots \quad amp;\lambda_n \end{bmatrix}$$

要注意的是这些特征值 λ 不一定是实数，有可能是复数。在这个表示下，T 的次方的极限可以写为

Undefined control sequence \<

如果 $|\lambda_m| < 1$ 那么 $\lambda_m^k \to 0$，如果 $|\lambda_m| = 1$ 但是 $\lambda_m \neq 1$ 那么 λ_m^k 不会收敛，如果 $\lambda_m = 1$ 那很显然 $\lambda_m^k = 1 \to 1$，而由于 T 的每一列加和都是 1 它的特征值不可能有 Misplaced &。根据这个观测，计算对角矩阵次方的极限 $\lim < /em > k \to \infty D^k$ 就很简单了，相应地 T^k 的极限也就计算出来了。

比如说，通过特征值和特征向量的分析，上面例子中的转移矩阵可以写为

$$T = PDP^{-1}$$

这里

$$P \approx [\,0.3570 \quad amp;0.6667 \quad amp;0.6667\ 0.8905 \quad amp;-0.4167 - 0.3997i \quad amp;-0.4167 + 0.3997i\ 0.2820 \quad amp;0.2500 + 0.3997i \quad amp;($$

$$D \approx [\,1 \quad amp;0 \quad amp;0\ 0 \quad amp;0.05 + 0.0480i \quad amp;0\ 0 \quad amp;0 \quad amp;0.05 - 0.0480i\,],$$

根据上面的结论，

$$\lim_{k\to\infty} T^k = P\,[\,1 \quad amp;0 \quad amp;0\ 0 \quad amp;0 \quad amp;0\ 0 \quad amp;0 \quad amp;0\,]\,P^{-1} \approx [\,0.2334 \quad amp;0.2334 \quad amp;0.2334\ 0.5822 \quad amp;0.5822$$

这和之前计算的 T^{250} 是吻合的。

多元正态分布

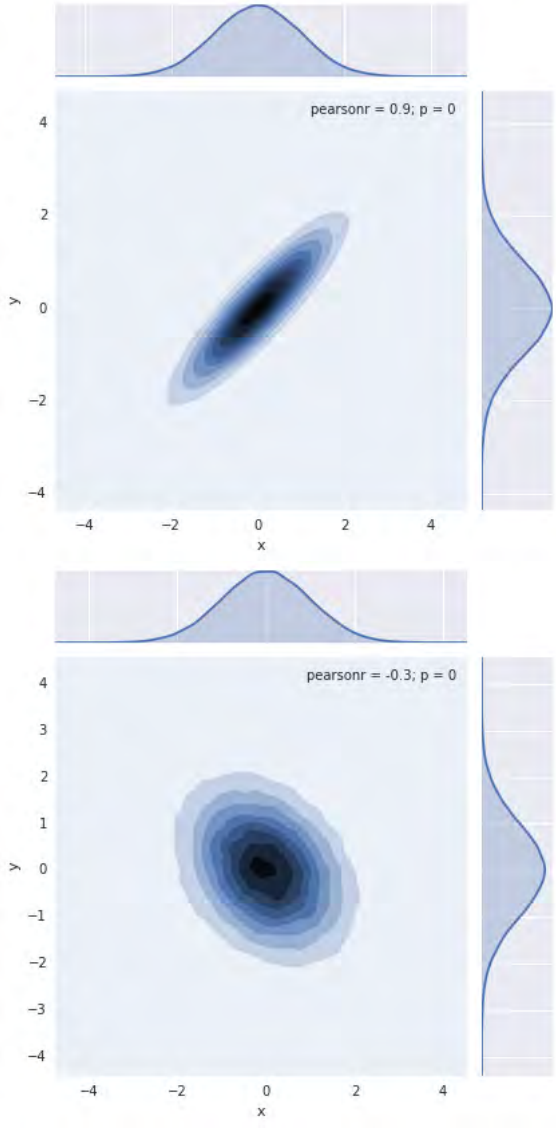
正态分布是概率论和统计学中最最最重要的概率分布。一个均值为 μ 方差为 σ^2 的正态分布函数是

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right).$$

但是现实中很多数据不是一维的，而是多元的，这就要用到多元的正态分布。对于一列长度为 d 的均值向量 $\boldsymbol{\mu} \in \mathbb{R}^d$ 和协方差矩阵 $\boldsymbol{\Sigma}$，相应的多元正态分布函数是

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})\right)$$

举例来说，下面展示的就是两个二元正态分布。



这两个二元正态分布的每一个变量都是 $\mathcal{N}(0,1)$ 的正态分布，但是放到一起的二元分布的热力图却截然不同。这两个二元分布的差别就在于它们的协方差矩阵 $\boldsymbol{\Sigma}$，通过研究这个矩阵可以分析出多元正态分布形状。一个 $n \times n$ 协方差矩阵一个重要的特性就是 $\boldsymbol{\Sigma}$ 是正定矩阵，它有一个正交的特征向量基 $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$，每个 u_i 对应特征值 λ_i，也就是说 $\boldsymbol{\Sigma}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ 并且 $\boldsymbol{u}_i : i = 1, \ldots, n$是 $\mathbb{R}^n$ 的一个基，并且当 $i \neq j$ 时有 $\mathbf{u}_i^{\mathsf{T}} \mathbf{u}_j = 0$。

对于一个分布 $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$，和某个 $0 < y < 1$，它的同概率轮廓为

$$C_y = \mathbf{x} \in \mathbb{R}^n : \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = y.$$

这个轮廓是一个椭圆体，椭圆体的中心是 $\boldsymbol{\mu}$，它的 n 个主轴的方向是

$$\boldsymbol{\mu} + t\mathbf{u}_i : t \in \mathbb{R}$$

分别对应 $i = 1, 2, \ldots, n$，并且主轴的长度分别是 $t\lambda_i$ 对于某个 Misplaced &。

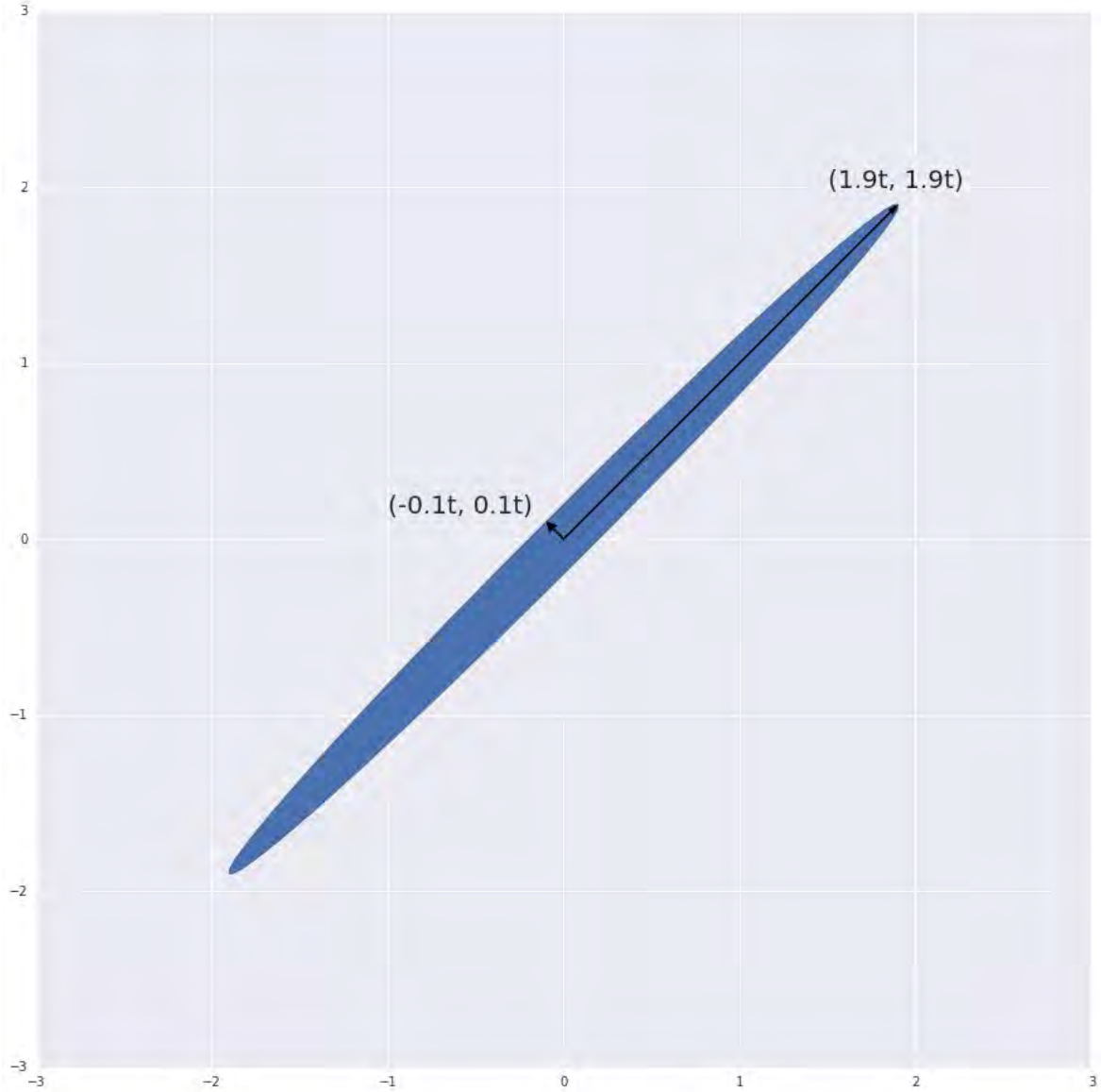
比如说，上面第一个图中的分布的协方差矩阵是

$$\boldsymbol{\Sigma} = [1 \quad amp; 0.9\ 0.9 \quad amp; 1],$$

这个矩阵的特征值是 1.9 和 -0.1，分别对应特征向量

$$[1\,1] \text{ 和 } [1\ -1].$$

那么这个分布的同概率椭球的主轴分别在 $(1,1)$ 和 $(-1,1)$ 的方向，长度分别是 $1.9t$ 和 $0.1t$，如下图所示。



通过对正定矩阵的分析，我们可以对联合正态分布的同概率椭球有清晰的认识，这对这些分布的分析起到很大作用。

结语

不学好线性代数是无法在这个世界生存下去的。

文章迭代记录
v1.0，2017-09-09 文章上线

【量化课堂】凯利公式，你用对了吗？

导语：众所周知，凯利公式揭露了信息论与赌博之间的本质联系，简约而不简单。 但是，重点来了！只有正确运用凯利公式，才能让你的净值增长如虎添翼。打开方式不对，就看不到成效。本文旨在给出凯利公式的正确使用方法，让我们了解在有限的信息里，如何下注能使得资本增值的速度最大化。
*本文由*JoinQuant*量化课堂推出 。难度标签为进阶上，理解深度标签：*level-0
作者：陈小米，宏观经济算命师
编辑：宏观经济算命师

某日，社区用户陈小米找到了本小编，说她在网上看到一篇凯利公式的介绍，但整个方法看着有点不严谨。后来经过我们的讨论，其假设确实有点牵强。所以后来我们一合计，凯利公式被人提到很多，网上的文章也很多，但总有点隔靴搔痒的感觉，同时社区内的有些朋友对于凯利公式不甚清楚。于是决定一起写篇东西吧，介绍介绍。

如果不去证明，只看凯利公式本身，它有着非常简洁的形式。在小编的量化体系中，凯利公式是基础中的基础。虽然基础，但是却是很重要的一环，可以说是小编整个量化体系的根基之一。

但须知差之毫厘，谬之千里。有如天龙八部中的易筋经，打开方式不对，你就看不到武功秘籍。而正确的使用凯利公式，有一个正确的方法论，才能让你的净值增长如虎添翼。
那怎么做呢？请继续往下看。

一、凯利公式产生的背景

人物介绍什么的就不多说了，总之，就是有那么一个牛人，找到了连续赌博中的最佳投注比例公式，横扫赌场。后来他又以此构建了一个对冲基金，战绩不俗。
这个人的大头像也不贴了，陈小米的帖子里面有。我们直接开始正文：

二、赌博怎么用凯利公式?

最早的凯利公式是运用在赌博游戏中的，我们先看看赌博情形下凯利公式的特殊形式：

$$
f = \frac{p < em > win * b - p < /em > loss}{b}$$

其中Pwin表示胜率，Ploss=1-Pwin，b表示赌赢了的赔率（扣除本金后的收益/本金）。f表示单次下注占总资金的比例。
就是这个精巧简洁的公式，将信息论与赌博之间的本质联系揭露出来，告诉我们在有限了解的信息下，如何下注能使得资本增值的速度最大化。

这个公式怎么来的，由于是普及向的文章，我们跳过了。各位就默认这个是对的就行：），真想深入，请看wiki链接：
https://en.wikipedia.org/wiki/Kelly_criterion

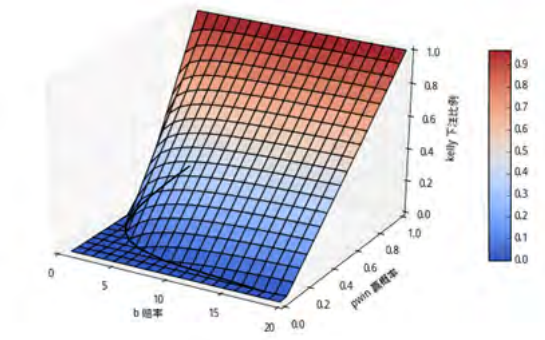
一个简单的例子

假想一个赌博游戏。赢的概率是60%，输的概率40%。入场费随意交。如果赢了获得2倍的入场费金额（b=1），输则输掉入场费。小编有100元做本金，请问小编每次给多少入场费，若干次游戏后几何期望收益能最大?
答：f = （1×0.6-0.4）/1 = 0.2。
也就是说最佳的策略是每次投剩余本金的20%。
这块不难理解，带入公式就能算出来。

注意两点：

1.从概率的角度说，一个期望净收益为负的游戏是不值得参与的，凯利公式也完美的体现了这一点。还是上面的游戏，如果赢的概率40%，输的概率60%，那么，期望净收益就是（1×0.4-0.6）<0；求得的f为-0.2。
负数仓位意味着你有上中两策可以选。中策选择不下注，上策是诱骗别人来跟你下注。

2.赌博版凯利公式只有在稳赢（赢概率=100%）时才会支持押下全部本金，否则都是本金的一定比例。随着本金的减少，下的注也越来越少。如果没有交易费用，下注可无限分割，我们是亏不完的（留得青山在，不怕没柴烧）。
下图能更直观的看到凯利公式对仓位的控制：如果押注的比例限制在0和1之间，对应不同的胜率（Pwin）和赔率（b）时，f会在三维空间上形成一个曲面。
这个曲面与f=0对应的平面相交的那条黑线就是期望为0所形成的曲线。



接下来我们在股票操作中构建一个类似上述的赌博模型，然后引入凯利公式。

三、把股票模型转换为赌博模型

下图来自《财富公式：玩转拉斯维加斯和华尔街的故事》，看看凯利的结果还不错嘛

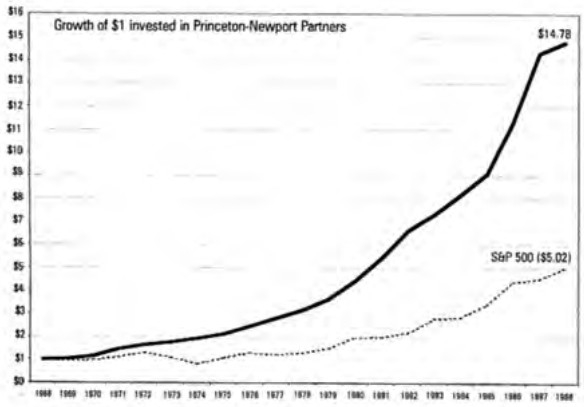
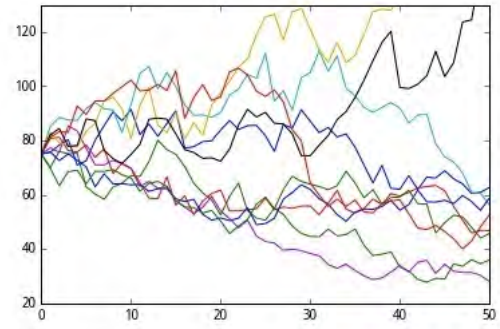
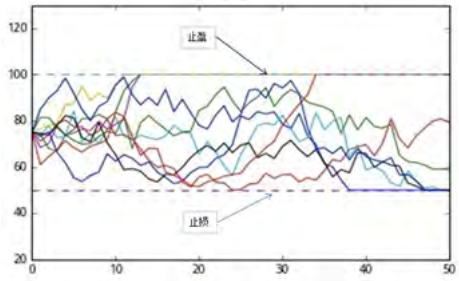


Figure 2. *Performance of the PNP fund. From* Fortune's Formula.

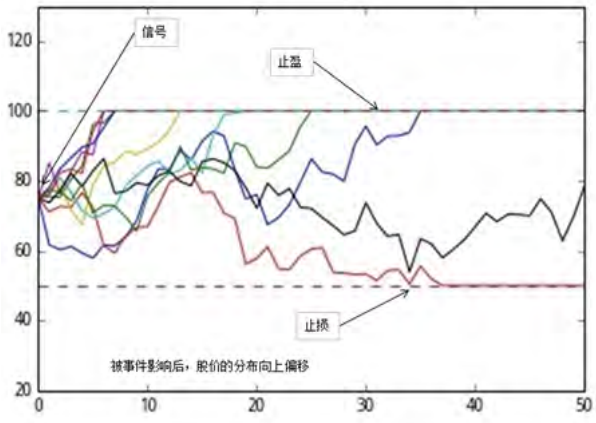
股票的假设和赌博有点不一样。股票是一个连续的过程，未来某一个时刻的收益率不是固定的一个值，而是一个分布。
那么我们怎么做捏？
怎么做可以发挥各种想象力，我们提供一个思路。
首先，在西方那一套理论中，往往用随机游走来描述股价的波动：



怎么把模型转换成与赌博模型类似的呢？方法有很多，这里来一个简单粗暴的。设置一个止盈价格和止损线，碰到了就出局。如果取对数后的股价服从随机游走假设，并且初始点是上沿和下沿的正中间，按照理论，先碰到上沿和先碰到下沿的概率是一样的。（忽略漂移项）。



但是在实际情况中呢，股价不是所有的时候都服从这个随机游走模型。股价先碰到上沿的概率会高于先碰到下沿的概率。比如说，突然出一个利好，财报公布后超预期，降准降息啦，或者单纯的资金面涌入造成短时间多空失衡，等等。
我们管这些事件，或者与这些事件同时发生的一些现象称为信号。比如说，降息的事件本身，就是一个信号。利好之前可能会有人提前知道偷偷买，造成股价跌不下去，这个该跌不跌就是一个信号。资金的涌入造成成交量放大，这个成交量放大也是一个信号。



上图表示事件对股价形成了影响，整体概率分布向上偏移，先碰到止盈的概率大于先碰到止损的概率。

交易所做的事情，就是这么一个寻找信号的过程，找到有效信号，意味着信号背后的事件会使股价的概率分布偏离，赢的期望变大。同时我们设置止盈止损线，这样赔率也就固定了下来。

由此我们就把投资股票的过程转换成一个连续赌博的过程。信号发出就是我们入场点。止盈止损发生的时候，就是我们的出场点。赔率和损失率就是止盈止损与入场价格之差。一次入场和出场就相当于赌博模型中的单次赌博，单次赌博的仓位由凯利公式确定。

逻辑好通顺。

各位童鞋注意，这个信号嘛，是需要同学自己去找的（这就是你们要做的事情啊喂！找到有效的信号就可以赚钱了）。而一旦找到有效信号，由某种方法固化了胜率赔率，凯利公式就华丽登场了。

四、凯利公式登场以后呢？

前文的赌博公式中，赔一次会输掉押注的所有金额。而由于在股市中，我们不会一次性赔光本金，而是赔掉本金的一定比例。所以我们需要使用一般性的凯利公式：

$$f = \frac{p < em > win}{c} - \frac{p < /em > loss}{b}$$

其中：
f：仓位比例
Pwin：赌赢的概率—股市上涨概率
Ploss：赌输的概率—股市下跌概率
b：赢钱率（资产从1增加到1+b）
c：损失率（资产从1减少到1-c）

假设我们找到了一个有效信号。并且根据历史上的统计，过去三年这个有效信号发生了1000次。以信号发出的价格为起点，在20%的正收益时止盈，在20%的负收益时止损。

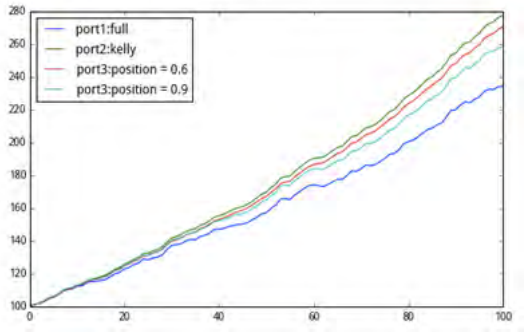
那么在信号发出后，如果先触碰盈的次数570次，先触碰止损的次数430次（这里只是为了举例而做简化，实际中我们需要做更多的工作）。于是，我们就成功的把问题转换成了一个连续赌博的问题：有这么一种赌博，赢一次的赔率为20%，输一次的损失率为20%，赢率为57%

对应公式，有Pwin=0.57，Ploss=0.43，b=0.20，c=0.20
此时f=Pwin/c – Ploss/b = 0.57/0.20 – 0.43/0.20= 70%
也就是说，不管你现在剩余多少钱，应该买入剩余部分的70%的仓位。

接下来，我们用蒙特卡洛模拟的方法做一组测试，看看凯利公式是怎么发挥作用的。假设股票投机产生了T次信号。我们相应的按照上述参数随机生成胜率和赔率，做了T次投机。把这T次投机算成一组完整的投资过程，这样就会得到一个净值的序列。对于任意的T，我们将这个投资过程重复1000次，求净值的几何平数。
我们看看在不同的投机次数T下的效果：

T=100：

```
kelly position is 0.7

不同组合的几何期望收益
full position 234.313167024
kelly position 277.595478022
position = 0.6 270.53080924
position = 0.9 258.494437316
```

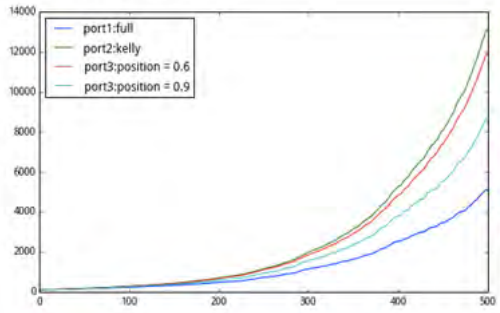


其中蓝线表示每次投机都是满仓；
绿线表示每次投机都是凯利公式给出的仓位（0.7）；
红线是接近凯利公式的仓位（0.6）；
还选了一个每次0.9的仓位，谁知道那个线的颜色是什么名字……

当T=100时，凯利公式仓位下净值增长最快，但是与其他仓位相比，优势不明显。

T=500：

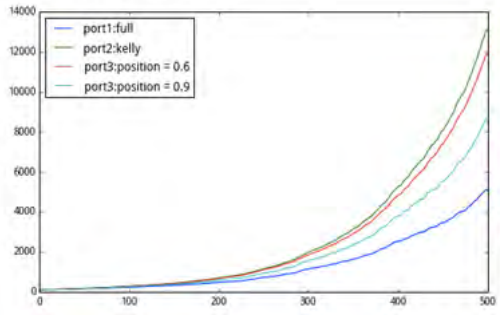
```
kelly position is 0.7

不同组合的几何期望收益
full position 5042.56649829
kelly position 13041.9405994
position = 0.6 11859.0140025
position = 0.9 8529.06397135
```



此时，最接近凯利公式仓位的红线，尚能与凯利公式一战。其他仓位的净值已经与凯利公式拉开了距离。

T=1000：

```
kelly position is 0.7

不同组合的几何期望收益
full position 5042.56649829
kelly position 13041.9405994
position = 0.6 11859.0140025
position = 0.9 8529.06397135
```

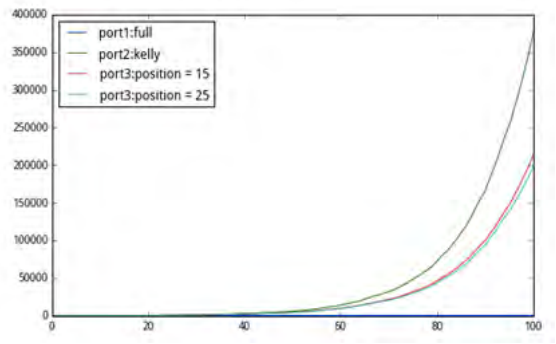


各个仓位明显看到差距。凯利公式完胜！
说明如果想有效的应用凯利公式，你找到的信号不能太稀疏。否则的话随机的因素可能占据上风。但是随着你的交易次数增加，你的净值能否增长就是一个字：

大概率事件。
99.99%曰大概率。

接下来，还有些微小的工作：
加杠杆：
我们看看另外一种情况：现在你找到了一个信号，继续延续上文的方法，但止盈止损线都是2%。历史统计涨过2%的概率是0.7，跌破2%的概率是0.3。那么带入凯利公式，得到的f值是多少呢？
2000%，也就是20倍的押注。
这是什么鬼？！背后的实际意义是什么？
凯利公式在告诉你这个因子太好了，值得你作20倍的杠杆来操作么？如果是那样，我再贪心一点好不好，看下图，加到25倍显然就挫了。。。

```
kelly position is 20.0

不同组合的几何期望收益
full position 218.213273612
kelly position 376131.661933
position = 15 213953.841453
position = 25 198551.819053
```



所以要理解，凯利公式判断押注比例的时候，是综合赢钱期望和输赢之间的差距（类似于方差的概念）进行的。如果说：

$$f = {}^{P_{win}}/_{c} - {}^{P_{loss}}/_{b}$$

中，c=0，计算出来的f是无穷大。在现实中就意味着找到了一个项目，要么赚钱，要么持平，怎么都不会亏。理论上，我们能借多少就要借多少啊！

但理论是理论，实际是实际。前文中隐含的前提是借钱没有摩擦，没有费用，也没有利息的。而在现实中，借钱是有成本的。一般人也借不到这么高的杠杆。
即使凯利公式告诉你要使用高杠杆，我们也不建议加过高的杠杆。除了上述的财务摩擦之外，还有如下原因：

黑天鹅

由于篇幅有限，以后有机会写文章介绍一下长尾效应和黑天鹅。简而言之，我们股票上所有的概率赔率，都是基于历史数据的。而现实中，小概率事件的概率往往会被低估。

所以说，如果凯利公式告诉你，要压大仓位，可千万要三思。你要是不加思考傻乎乎的信了，出了一件在模型历史统计之外的事情，爆仓分分钟教你做人。这么说有点抽象，打个比方，假设股灾前，你找了一个很好的信号，比如跌5%时大概率会反弹的这种。然后如上文所述，凯利公式给你的仓位比例是10倍。你压了两倍杠杆，还嫌少，又配资加到了5倍。

接下来发生了股灾，小概率事件一个接一个。第一天直接击破止损，还没等你反应过来，就跌停了卖不出去。第二天开盘跌停了，卖不出去；第三天开盘跌停了，卖不出去……

就问你怎么办。

永远记住：这个世界充满了因果性和蝴蝶效应。模型只是一个近似的替代。

真正的本金是什么?
有人说，凯利公式计算的仓位总是偏大。其原因嘛，除了上文中提到的，实际股票收益率的分布具有长尾因素，导致极端情况发生的次数比预计要多以外，还有一个原因是很多人没明白凯利公式的投注比例所对应的本金到底是什么。

人们往往会认为凯利公式所针对投注比例是全资产，但其实上并不是，凯利公式所针对的投注比例是你可承受损失的资产。比如说，有一个私募，投入了1000w，但是有0.8的清仓线，其实只能承受200w的损失。那么在凯利公式里，作针对的本金其实只有200w，也就是说，如果f=0.7，你一次压上去的钱只有200w*0.7=140w。

明白了没有？如果现在这个1000w亏的只剩 910w了，那么本金只有110w，一次下注的钱只能是110w0.7=77w
*而如果这个*1000w*赚到了*1150w，*那么本金就变成了（*1150w-800w）0.7=350w，需要压的数量就是350w*0.7=245w。

所以说，如果一个人虽然有100%的仓位，但是实际心理能承受的损失就是20%，那么此时就应该用这个20%作为本钱来带入凯利公式。如果用100%的仓位带入，结果会让你很烦躁。

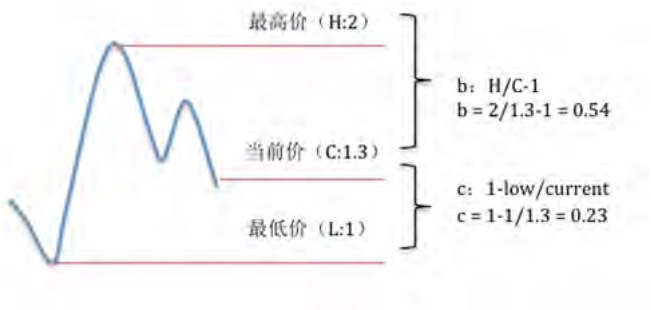
差点忘了，那个错误的例子：
开头提到了，我们在网上找到了一个错误的例子。
注：后面的回测为错误例子
剖析一下：

在股票市场中，kelly formula的形式和赌博中稍有不同，因为失败之后有一个expected loss，我们将公式修改为 f = (bp - cq) / (bc)，来做一个简单的回测。
假定赚钱(p)赔钱概率(q = 1-p)一样，均为0.5，我们认为净赢率(b)为最高价/当前价-1，净损失率(c)为1-最低价/当前价。

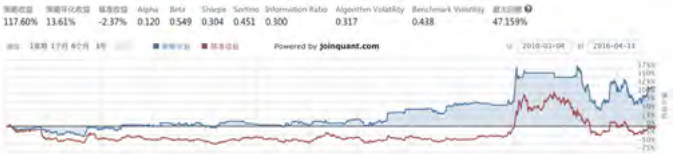
如果当前价大于一年内最高价，满仓，如果当前价小于一年内最低价，空仓，其他情况使用之前得到的f计算出持仓比例(0-1之间)，每周调仓。

我们挑选了单只股票作回测，回测的参照物是该股票的买入持有策略(即为价格曲线)，看看使用Kelly 公式调节仓位有什么样的效果。

我们画了一张图来理解本文的假设：



上面那段话的暗含的条件是，如果一年内最高价H=2，一年内最低价L=1，不论当前你的价格在那里，未来价格先碰到H的概率和先碰到L的概率都相等。
即Pwin = Ploss = 0.5的情况下,如果当前价为1.3， 此时该保持的仓位 f = (0.5×0.54-0.5×0.23)/（0.54×0.23） = 124.80%。
按照上文逻辑，挑选中信证券（600030.SH）进行回测(如果仓位大于100%，则调整成100%)：
1.中信证券



哇塞，比一直持有更赚钱。
到这里，你以为你真的就可以一招吃遍天下，用凯利就可以躺着赚钱了吗？且慢，换一个标的试试。

2.嘉实沪深300ETF

纳尼？大师的一世英名岂不是要栽在这张回测图上了。说好的收益最大化呢？？是不是哪里出错了？？

问题出在哪？
问题出在假设中，文中假设任意的股票任意位置，先触碰最高价和先触碰最低价的概率都是0.5，这显然是不合理的强假设。因此可能一个股票刚好回测效果比较好，另外一个股票回测效果就比较差。

如果假设改进一下，比如先判断一下目前是不是一个震荡中枢里。如果在震荡中枢里，我们假设先碰到震荡下沿和上沿的概率相等，都是0.5。如果突破上沿，说明新的趋势形成，可以满仓追进去。如果跌破下沿，说明反向趋势形成，暂时避开。我觉得可能更合理一些。（这只是一个猜想，大家有兴趣可以验证）

哇。。。
好啦，少年。真正赚钱的是找有效的因子or信号，使得Pwin尽可能大，b尽可能大。而这个因子or信号的挖掘，就是作为矿工孜孜不倦追求的终极目标了。当你找到这样一个神奇的因子，你需要做的就是，用一种方法将其近似成连续的赌博模型。接下来，凯利公式就会给你指引一条正确的道路。

行动起来吧。加入宽客大军。

彩蛋：
评论区有朋友问，如果长期即碰不到上沿也碰不到下沿怎么办。其实这个问题就出在将股票模型转换为连续赌博模型的方法上，这就是该方法的缺陷所在。

如果各位仔细看，可以看出本篇文章中转换为赌博模型的方法，其实是加强了对赔率的控制，舍弃了对时间的控制。因为我们不知道何时才会触碰上下沿。

如果点赞转发数高，我再写一个固定时间，模糊赔率的股票投资至赌博模型转换方法。

文章更迭记录：
v1.1，2016-07-04，添加“导语”
v1.0，2016-05-09，文章上线

【量化课堂】SVM原理入门

导语：把”机器学习“应用到量化投资领域，不同于以往的量化策略。机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法。这篇文章中，主要介绍了SVM（支持向量机）的基本原理，希望大家都能够喜欢!后续会持续发表“机器学习”量化文章，敬请关注。
*本文由*JoinQuant*量化课堂推出 。难度标签为进阶上，理解深度标签：*level-0
作者：yongpeng.r
编辑：宏观经济算命师

文中举的例子是为了理解SVM，所以很简单，准确性不保证！要想应用到实战中，还需要做更多的工作。

文章更迭记录：
v1.1，2016-07-04，添加“导语”
v1.0，2016-06-04，文章上线

0.相关概念¶

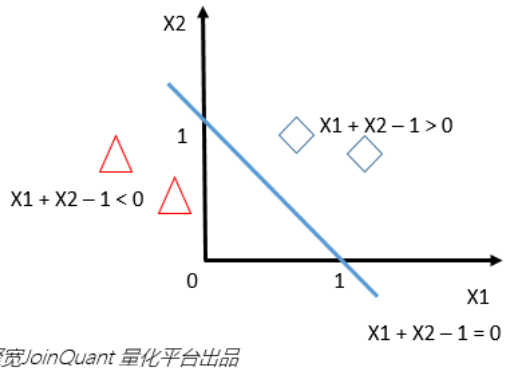
分类器：分类器就是给定一个样本的数据，判定这个样本属于哪个类别的算法。例如在股票涨跌预测中，我们认为前一天的交易量和收盘价对于第二天的涨跌是有影响的，那么分类器就是通过样本的交易量和收盘价预测第二天的涨跌情况的算法。

特征：在分类问题中，输入到分类器中的数据叫做特征。以上面的股票涨跌预测问题为例，特征就是前一天的交易量和收盘价。

线性分类器：线性分类器是分类器中的一种，就是判定分类结果的根据是通过特征的线性组合得到的，不能通过特征的非线性运算结果作为判定根据。还以上面的股票涨跌预测问题为例，判断的依据只能是前一天的交易量和收盘价的线性组合，不能将交易量和收盘价进行开方，平方等运算。

1.线性分类器起源¶

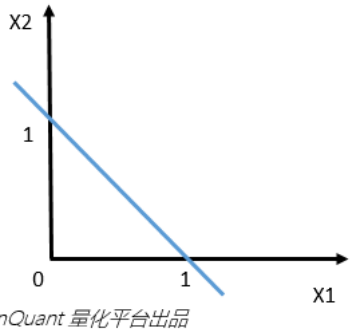
在实际应用中，我们往往遇到这样的问题：给定一些数据点，它们分别属于两个不同的类，现在要找到一个线性分类器把这些数据分成两类。



聚宽JoinQuant 量化平台出品

怎么分呢？把整个空间劈成两半呗（让我想到了盘古）。用二维空间举个例子，如上图所示，我们用一条直线把空间切割开来，直线左边的点属于类别-1（用三角表示），直线右边的点属于类别1（用方块表示）。

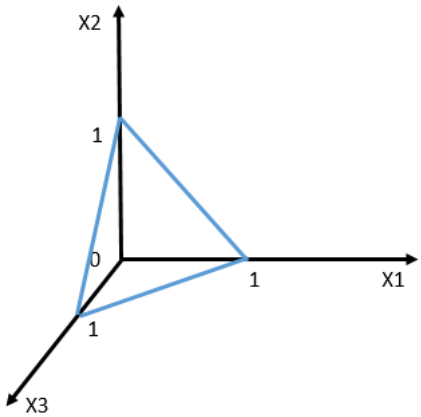
如果用数学语言呢，就是这样的：空间是由X_1和X_2组成的二维空间，直线的方程是$X_1 + X_2 = 1$，用向量符号表示即为 $[1, 1]^T[X_1, X_2] - 1 = 0$。点x在直线左边的意思是指，当把x放入方程左边，计算结果小于0。同理，在右边就是把x放入方程左边，计算出的结果大于0。都是高中数学知识。



聚宽JoinQuant 量化平台出品

在二维空间中，用一条直线就把空间分割开了：

在三维空间中呢，需要用一个平面把空间切成两半，对应的方程是$X_1 + X_2 + X_3 = 1$，也就是 $[1, 1, 1]^T[X_1, X_2, X_3] - 1 = 0$。



聚宽JoinQuant 量化平台出品

在高维（n>3）空间呢？就需要用到n-1维的超平面将空间切割开了。那么抽象的归纳下：如果用x表示数据点，用y表示类别（y取1或者-1，代表两个不同的类），一个线性分类器的学习目标便是要在n维的数据空间中找到一个超平面（hyper plane），把空间切割开，这个超平面的方程可以表示为（W^T中的T代表转置）：

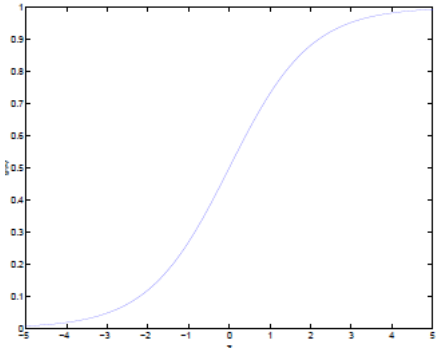
$$W^T X + b = 0$$

感知器模型和逻辑回归：

常见的线性分类器有感知器模型和逻辑回归。上一节举出的例子是感知器模型，直接给你分好类。有时候，我们除了要知道分类器对于新数据的分类结果，还希望知道分类器对于这次分类的成功概率。逻辑回归就可以做这件事情。

逻辑回归（虽然称作回归，但是不是一个回归方法，却是一个分类算法。很蛋疼的说）将线性分类器的超平面方程计算结果通过logistic函数从正负无穷映射到0到1。这样，映射的结果就可以认为是分类器将x判定为类别1的概率，从而指导后面的学习过程。

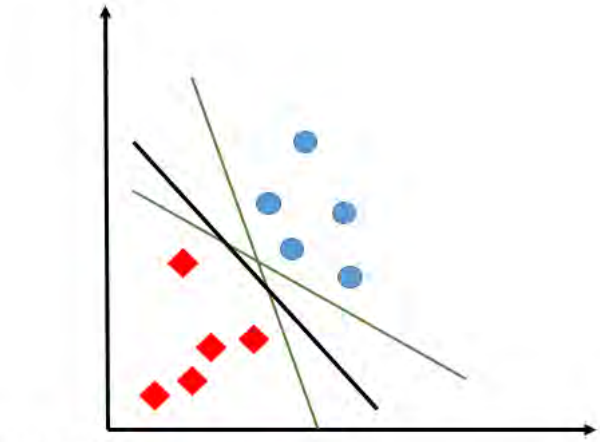
举个例子，看天气预报，用感知器的天气预报只会告诉你明天要下雨（y=1），或者明天不下雨（y=-1）；而用了逻辑回归的天气预报就能告诉你明天有90%的概率要下雨，10%的概率不下雨。

逻辑回归的公式是$\frac{g(z)=1}{1+e^{-z}}$，图像大概长这个样子：  怎么用呢？比如感知器模型中，将特征代入判别方程中，如果得到的值是-3，我们可以判定类别是-1（因为-3<0）。而逻辑回归中呢，将-3代入g(z)，我们就知道，该数据属于类别1的概率是0.05（近似数值，谢谢），那么属于类别-1的概率就是1 – 0.05 = 0.95。也就是用概率的观点描述这个事情。

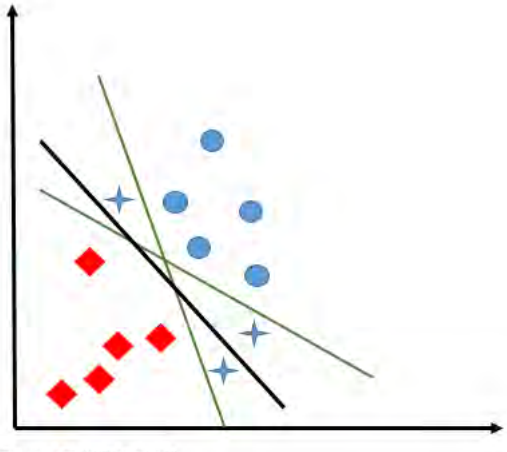
本文深度为"了解"，如果想知道更多的感知器模型和逻辑回归细节，可以参照《统计学习方法》等机器学习的相关书籍。或者持续关注我们的量化课堂，未来会深度探讨。

2.支持向量机 VS 感知器和逻辑回归¶

根据上面的讨论，我们知道了在多维空间下，用一个超平面就把数据分为了两类。这个超平面我们叫它为分离超平面。但是这个分离超平面可以有很多个，



那么用哪个呢？上图中，对于目前的训练数据，绿色和黑色的直线（二维特征空间，分离超平面就是直线啦）都可以很可以很好的进行分类。但是，通过已知数据建立分离超平面的目的，是为了对于未知数据进行分类的。在下图中，蓝



色的星星图案就是新加入的真实数据。这时候我们就可以看出不同的分离超平面的选择对于分类效果的影响了。有的绿线会将三个点都划归蓝色圆圈，有的绿线会将三个点都划归红色正方形。

那么绿线和黑线留下谁？我们认为，已有的训练数据中，每个元素距离分离超平面都有一个距离。在添加超平面的时候，尽可能的使最靠近分离超平面的那个元素与超平面的距离变大。这样，加入新的数据的时候，分的准的概率会最大化。感知器模型和逻辑回归都不能很好的完成这个工作，该我们的支持向量机（support vector machine，SVM）出场了。

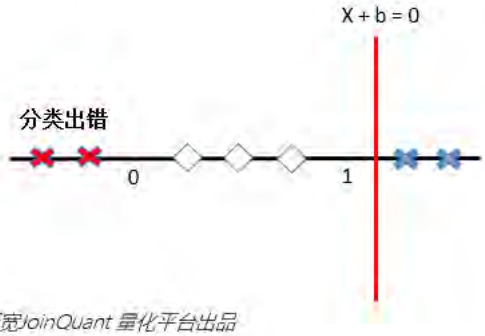
首先，SVM将函数间隔（$|W^T X + b|$，将特征值代入分离超平面的方程中，得到的绝对值）归一化，归一化的目的是除掉取值尺度的影响；其次，对所有元素求到超平面的距离，（这个距离是$\frac{|W^T X+b|}{|W|}$，也就是几何间隔）。给定一个超平面P，所有样本距离超平面P的距离可以记为d_{ij}，这其中最小的距离记为D_P，SVM的作用就是找到D_P最大的超平面。

可以看出，大部分数据对于分离超平面都没有作用，能决定分离超平面的，只是已知的训练数据中很小的一部分。这与逻辑回归有非常大的区别。上图中，决定黑色的这条最优分离超平面的数据只有下方的两个红色的数据点和上方的一个蓝色的数据点。这些对于分离超平面有着非常强大影响的数据点也被称为支持向量（看没看到，这就是传说中的支持向量啦，原来如此）。

3.引入黑科技-核函数¶

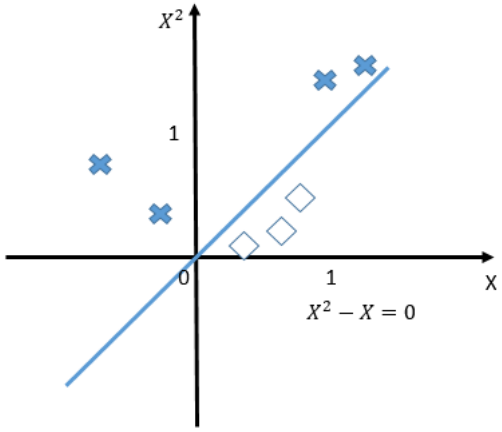
上面说的都是在原始特征的维度上，能直接找到一条分离超平面将数据完美的分成两类的情况。但如果找不到呢？

比如，原始的输入向量是一维的，$0 < X < 1$的类别是1，其他情况记做-1。这样的情况是不可能在1维空间中找到分离超平面的（一维空间中的分离超平面

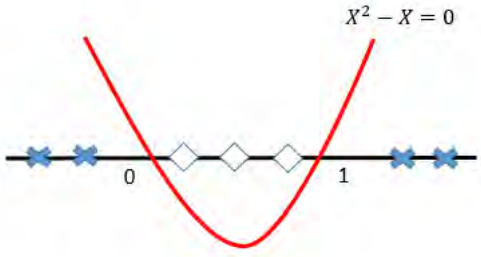


是一个点，$X + b = 0$）。你用一个点切一下试试？ 这就要说到SVM的黑科技—核函数技巧。核函数可以将原始特征映射到另一个高维特征空间中，解决原始空间的线性不可分问题。

继续刚才那个数轴。 如果我们将原始的一维特征空间映射到二维特征空间X^2和X，那么就可以找到分离超平面$X^2 - X = 0$。当$X^2 - X < 0$的时候，就可以判别为类别1，当$X^2 - X > 0$的时候，就可以判别为类别0。如下图：



再将$X^2 - X = 0$映射回原始的特征空间，就可以知道在0和1之间的实例类别是1，剩下



空间上（小于0和大于1）的实例类别都是0啦。 利用特征映射，就可以将低维空间中的线性不可分问题解决了。是不是很神奇，这就是特征映射的牛逼之处了。核函数除了能够完成特征映射，而且还能把特征映射之后的内积结果直接返回，大幅度降低了简化了工作，这就是为啥采用核函数的原因。

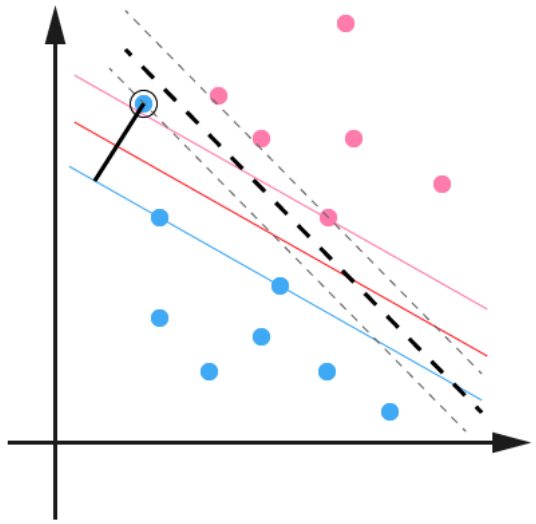
（为啥要返回数据间的内积结果涉及到比较高深的内容，在此先略过，可以在《统计学习方法》等更加专业的资料中自行阅读，未来量化课堂会产生这部分的内容。）

4.异常值的处理-松弛变量的引入¶

你以为就结束了吗？并没有。

在原始空间线性不可分时，可以映射到高维空间之后，转换为线性可分的问题。但是万一映射之后还是不能线性可分，该如何处理呢？

再比如正常的数据中混入了异常数据，很有可能会使应该的最佳分离超平面移位，或者直接使数据变得线性不可分。又怎么办捏?



上图中用黑色的圆圈圈起来的就是一个异常值，这个异常值的存在，使得分离超平面发生了移位。这时候就该引入松弛变量了。松弛变量可以允许某些数据点在不满足分离超平面两边的类别要求，从而使得某些严格线性不可分的数据集也可以使用SVM进行分类了。

由于篇幅有限，这部分内容只是为了完整性，不做深入讨论。欢迎关注量化课堂。

5.SVM的具体使用-sklearn ¶

SVM的基本原理基本上已经说的差不多了，下面咱们就来看看SVM在实际应用该如何使用了。幸运的是，在python下面，sklearn提供了一个非常好用的机器学习算法，我们调用相关的包就好啦。

在下面的这个例子中，特征是通过收盘价数据计算的SMA，WMA，MOM指标，训练样本的特征是从2007-1-4到2016-6-2中每一天的之前的交易日的收盘价计算的SMA，WMA，MOM指标，训练样本的标签就是2007-1-4日到2016-6-2中每一天的涨跌情况，涨了就是True，跌了就是False，测试样本是2016-6-3日的三个指标以及涨跌情况。我们可以判定之后判断结果是正确还是错误，如果通过SVM判断的结果和当天的涨跌情况相符，则输出True，如果判断结果和当天的涨跌情况不符，则输出False。

我这次的预测结果是输出了True哦。

In [1]:

```
import talib
from jqdata import *

test_stock = '399300.XSHE'
start_date = datetime.date(2007, 1, 4)
end_date = datetime.date(2016, 6, 3)

trading_days = get_all_trade_days()
start_date_index = trading_days.index(start_date)
end_date_index = trading_days.index(end_date)

x_all = []
y_all = []

for index in range(start_date_index, end_date_index):
    # 得到计算指标的所有数据
    start_day = trading_days[index - 30]
    end_day = trading_days[index]
    stock_data = get_price(test_stock, start_date=start_day, end_date=end_day, frequency='daily', fields=['close'])
    close_prices = stock_data['close'].values

    #通过数据计算指标
    # -2是保证获取的数据是昨天的，-1就是通过今天的数据计算出来的指标
    sma_data = talib.SMA(close_prices)[-2]
    wma_data = talib.WMA(close_prices)[-2]
    mom_data = talib.MOM(close_prices)[-2]

    features = []
    features.append(sma_data)
    features.append(wma_data)
    features.append(mom_data)

    label = False
    if close_prices[-1] > close_prices[-2]:
        label = True
    x_all.append(features)
    y_all.append(label)


# 准备算法需要用到的数据
```

```
x_train = x_all[: -1]
y_train = y_all[: -1]
x_test = x_all[-1]
y_test = y_all[-1]
print('data done')
```

```
data done
```

In [ ]:

```
from sklearn import svm

#开始利用机器学习算法计算
clf = svm.SVC()
#训练的代码
clf.fit(x_train, y_train)
#得到测试结果的代码
prediction = clf.predict(x_test)

# 看看预测对了没
print(prediction == y_test)
print('all done')
```

【量化课堂】随机森林入门

导语：把"机器学习"应用到量化投资领域，不同于以往的量化策略。机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法。其中，随机森林算法是一种基于统计学习理论的组合分类器。它可以将用户自选的各个因子，以机器训练的方式，自动分析其影响力度，从而给用户投资建议。本文属于入门级别，若有兴趣，后续文章将深入分析，敬请关注。

*本文由*JoinQuant*量化课堂推出 。难度标签为进阶上，理解深度标签：*level-0

作者：yongpeng.r

编辑：宏观经济算命师

一、相关概念

分类器：分类器就是给定一个样本的数据，判定这个样本属于哪个类别的算法。例如在股票涨跌预测中，我们认为前一天的交易量和收盘价对于第二天的涨跌是有影响的，那么分类器就是通过样本的交易量和收盘价预测第二天的涨跌情况的算法。

分裂：在决策树的训练过程中，需要一次次的将训练数据集分裂成两个子数据集，这个过程就叫做分裂。

特征：在分类问题中，输入到分类器中的数据叫做特征。以上面的股票涨跌预测问题为例，特征就是前一天的交易量和收盘价。

待选特征：在决策树的构建过程中，需要按照一定的次序从全部的特征中选取特征。待选特征就是在目前的步骤之前还没有被选择的特征的集合。例如，全部的特征是 ABCDE，第一步的时候，待选特征就是ABCDE，第一步选择了C，那么第二步的时候，待选特征就是ABDE。

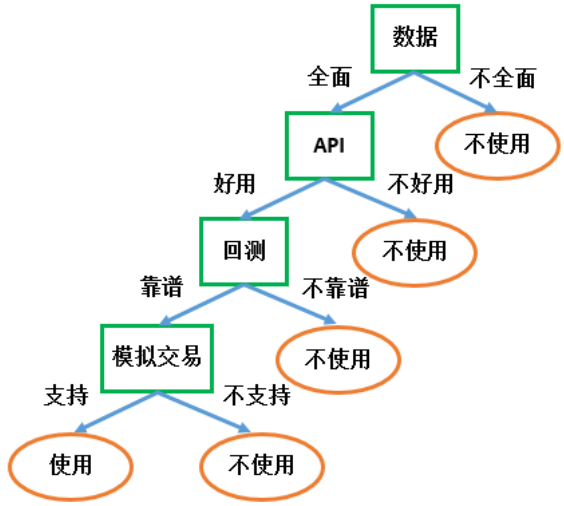
分裂特征：接待选特征的定义，每一次选取的特征就是分裂特征，例如，在上面的例子中，第一步的分裂特征就是C。因为选出的这些特征将数据集分成了一个个不相交的部分，所以叫它们分裂特征。

二、决策树的构建过程

要说随机森林，必须先讲决策树。决策树是一种基本的分类器，一般是将特征分为两类（决策树也可以用来回归，不过本文中暂且不表）。构建好的决策树呈树形结构，可以认为是if-then规则的集合，主要优点是模型具有可读性，分类速度快。

我们用选择量化工具的过程形象的展示一下决策树的构建。假设现在要选择一个优秀的量化工具来帮助我们更好的炒股，怎么选呢？
第一步：看看工具提供的数据是不是非常全面，数据不全面就不用。
第二步：看看工具提供的API是不是好用，API不好用就不用。
第三步：看看工具的回测过程是不是靠谱，不靠谱的回测出来的策略也不敢用啊。
第四步：看看工具支不支持模拟交易，光回测只是能让你判断策略在历史上有用没有，正式运行前起码需要一个模拟盘吧。

这样，通过将"数据是否全面"，"API是否易用"，"回测是否靠谱"，"是否支持模拟交易"将市场上的量化工具贴上两个标签，"使用"和"不使用"。
上面就是一个决策树的构建，逻辑可以用下图表示：

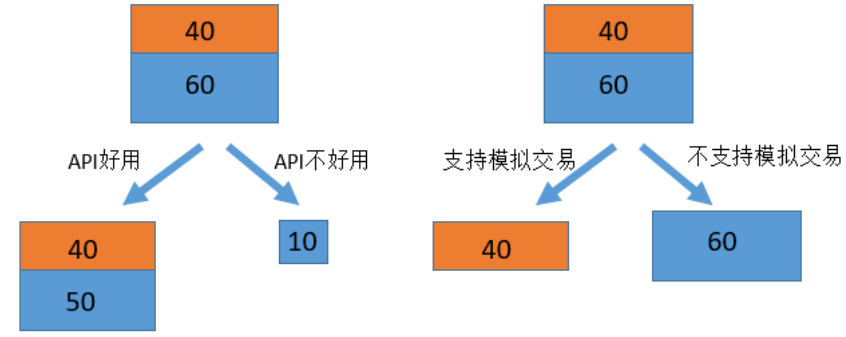


JoinQuant量化交易平台出品

在上图中，绿颜色框中的“数据”“API”“回测”“模拟交易”就是这个决策树中的特征。如果特征的顺序不同，同样的数据集构建出的决策树也可能不同。特征的顺序分别是“数据”“API”“回测”“模拟交易”。如果我们选取特征的顺序分别是“数据”“模拟交易”“API”“回测”，那么构建的决策树就完全不同了。

可以看到，决策树的主要工作，就是选取特征对数据集进行划分，最后把数据贴上两类不同的标签。如何选取最好的特征呢？还用上面选择量化工具的例子：假设现在市场上有100个量化工具作为训练数据集，这些量化工具已经被贴上了“可用”和“不可用”的标签。

我们首先尝试通过“API是否易用”将数据集分为两类；发现有90个量化工具的API是好用的，10个量化工具的API是不好用的。而这90个量化工具中，被贴上“可以使用”标签的占了40个，“不可以使用”标签的占了50个，那么，通过“API是否易用”对于数据的分类效果并不是特别好。因为，给你一个新的量化工具，即使它的API是易用的，你还是不能很好贴上“使用”的标签。



JoinQuant量化交易平台出品

再假设，同样的100个量化工具，通过“是否支持模拟交易”可以将数据集分为两类，其中一类有40个量化工具数据，这40个量化工具都支持模拟交易，都最终被贴上了“使用”的标签，另一类有60个量化工具，都不支持模拟交易，也都最终被贴上了“不使用”的标签。如果一个新的量化工具支持模拟交易，你就能判断这个量化工具是可以使用。我们认为，通过“是否支持模拟交易”对于数据的分类效果就很好。

在现实应用中，数据集往往不能达到上述“是否支持模拟交易”的分类效果。所以我们用不同的准则衡量特征的贡献程度。主流准则的列举3个：ID3算法（J. Ross Quinlan于1986年提出），采用信息增益最大的特征；C4.5算法（J. Ross Quinlan于1993年提出）采用信息增益比选择特征；CART算法（Breiman等人于1984年提出）利用基尼指数最小化准则进行特征选择。

（如果想进行更深一步的学习，可以参考《统计学习方法》或者相关博文进行更一步的学习。未来的量化课堂也会涉及这方面的内容。）

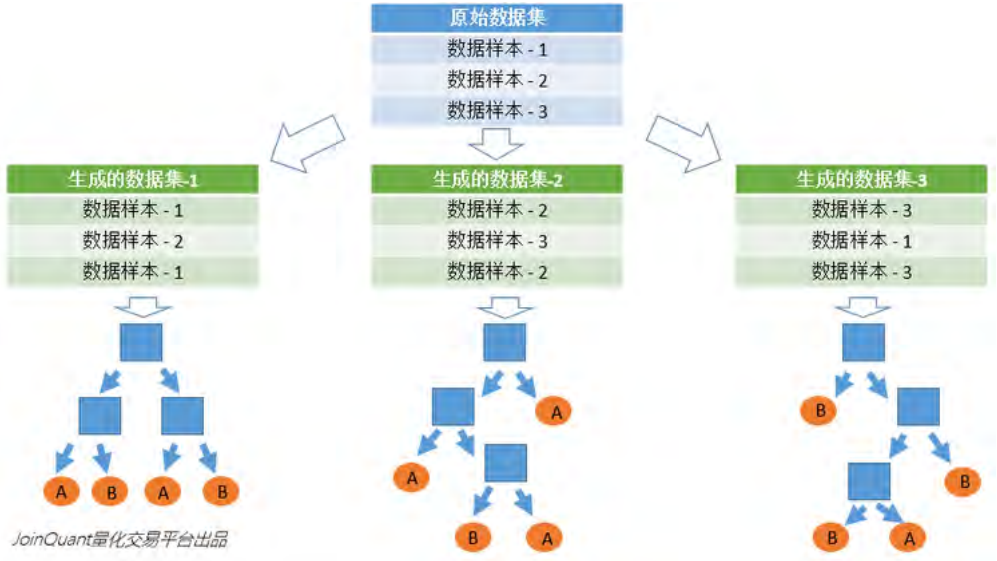
三、随机森林的构建过程

决策树相当于一个大师，通过自己在数据集中学到的知识对于新的数据进行分类。但是俗话说得好，一个诸葛亮，玩不过三个臭皮匠。随机森林就是希望构建多个臭皮匠，希望最终的分类效果能够超过单个大师的一种算法。

那随机森林具体如何构建呢？有两个方面：数据的随机性选取，以及待选特征的随机选取。

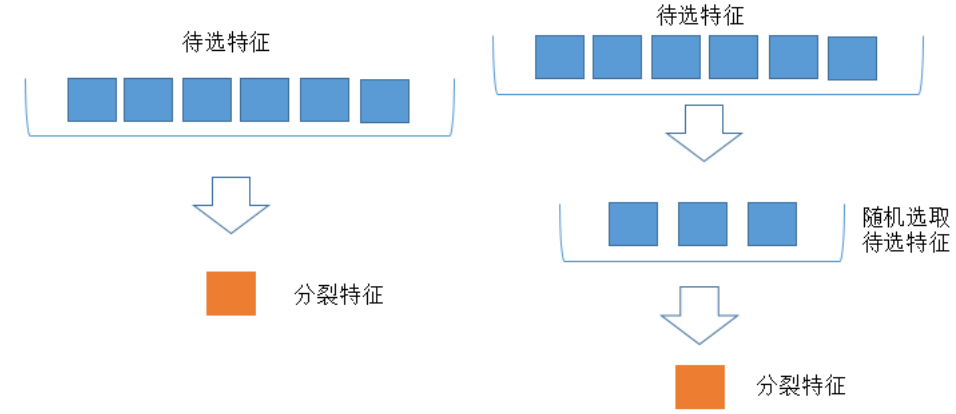
数据的随机选取：
首先，从原始的数据集中采取有放回的抽样，构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。第二，利用子数据集来构建子决策树，将这个数据放到每个子决策树中，每个子决策树输出一个结果。最后，如果有了新的数据需要通过随机森林得到分类结果，就可以通过对子决策树的判断结果的投票，得到随机森林的输出结果了。如下图，假设随机森林中有3棵子决策树，2棵子树的分

类结果是A类，1棵子树的分类结果是B类，那么随机森林的分类结果就是A类。



待选特征的随机选取：
与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。这样能够使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

下图中，蓝色的方块代表所有可以被选择的特征，也就是目前的待选特征。黄色的方块是分裂特征。左边是一棵决策树的特征选取过程，通过在待选特征中选取最优的分裂特征（别忘了前文提到的ID3算法，C4.5算法，CART算法等等），完成分裂。右边是一个随机森林中的子树的特征选取过程。



决策树选取分裂特征过程　　随机森林子树选取分裂特征过程

JoinQuant量化交易平台出品

四、Random Forest 的具体使用-sklearn

以上介绍了随机森林的工作原理，那么在python环境下，我们可以利用python环境下的sklearn包来帮助我们完成任务。举个小例子：
特征是通过收盘价数据计算的SMA，WMA，MOM指标，训练样本的特征是从2007-1-4到2016-6-2中截止前一天的SMA，WMA，MOM指标，训练样本的标类别是2007-1-4日到2016-6-2中每一天的涨跌情况，涨了就是True，跌了就是False，测试样本是2016-6-3日的三个指标以及涨跌情况。我们可以判定之后判断结果是正确还是错误，如果通过Random Forest判断的结果和当天的涨跌情况相符，则输出True，如果判断结果和当天的涨跌情况不符，则输出False。
（和SVM那一篇中例子的作用是一样滴，只是为了展示如何使用，不对预测的准确性做担保啊）

具体代码见下面的研究代码啦

文章更迭记录：
v1.1，2016-07-04，添加“导语”
v1.0，2016-06-15，文章上线

【量化课堂】信息增益入门

导语：我们知道，资产价格的波动受到非常多因素的影响，比如上一个交易日的价格，交易量，国家经济走势，交易者的心理预期，各种技术指标，财务指标，甚至还可以是交易日的天气情况等等。每个人都可以总结出非常多影响资产价格的特征（也可以叫因子，变量）。但是如何在这些特征中，选取出真正有效的特征呢？这时候就可以用到本文中要介绍的信息增益了。

*本文由JoinQuant量化课堂推出。难度标签为进阶上，理解深度标签：*level-0

作者：yongpeng.r
编辑：宏观经济算命师

一、熵

即：下一个交易日的涨跌的不确定性。

由于影响资产价格的因素很多，因此人们往往用随机性描述资产的价格波动，把它看成一个随机变量。但是，随机变量的不确定性不是一成不变的。比如扔硬币，如果 90% 的概率都是正面向上，我们认为它的不确定性比 50% 正面向上的硬币要低。那么如何衡量一个离散变量的不确定性呢？在信息论和概率统计中，人们使用一个变量的'熵'来表示这个变量的不确定性。为了简明起见，我们都用离散变量进行举例。

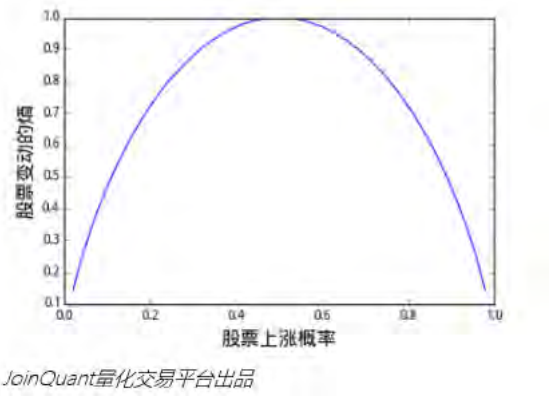
如果一个离散变量 Y 的概率分布是：

$$P(Y = y_i) = p_i,\ i = 1, 2, \ldots, n$$

那么变量 Y 的熵定义为：

$$H(Y) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

由上述公式看出，当变量 Y 的分布只有一个值的时候，它没有任何不确定性，计算出的熵是 0。一个变量的熵越大，它的不确定性也越大，也越难预测。举个例子：如果我们用变量X表示第二天的资产涨跌，X 只能取上涨或者下跌。当 X 取上涨的概率从 0% 变化到 100% 时，其熵的变化是先变大再变小的。如下图所示，资产涨跌概率都是 50% 的时候，熵最大，表示这时候的状态最不稳定，最无序。



JoinQuant量化交易平台出品

二、条件熵

即：知道了今天的涨跌，下一个交易日的涨跌的不确定性。

上文中，我们介绍了离散变量的熵及其计算公式。但是为了更好的预测未来的涨跌情况，我们需要更多的信息。例如，在股票涨跌的预测中，如果不知道其他信息，可以假设涨跌的概率都是 50%，股票变动的熵就是 1；但在已知今天的涨跌的情况下，下一个交易日涨跌的概率就变成 $P(X < em > t \mid X < /em > t-1 = \text{涨})$或者是 $P(X < em > t \mid X < /em > t-1 = \text{跌})$此时怎么进行熵的衡量呢？

假设我们要研究的变量为 Y，已观测到的变量为 X；用 $H(Y \mid X = x_i)$ 表示在 X 取值为 x_i 的情况下，变量 Y 的熵，那么条件熵 $H(Y \mid X)$ 就表示在已知信息 X 的情况下，变量 Y 的不确定性。条件熵的计算公式是：

$$< br > H(Y \mid X) = \sum < em > i = 1^n p < /em > x_i H(Y \mid X = x_i) < br >$$

在这里，$p < em > x_i = P(X = x_i),\ i = 1, 2, \ldots, m$。注意这个公式中的 $H(Y \mid X = x_i)$ 和 $H(Y \mid X)$ 的符号比较像，但 $H(Y \mid X = x_i)$ 的意思是将 $(Y \mid X = x_i)$整体看成一个变量 Z，利用 $H(Z) = -\sum < /em > i = 1^n p_i \log_2 p_i$进行计算，不要被迷惑了。

三、信息增益

即：特征中包含的信息度量。

熵是变量不确定性的度量，条件熵就是在已知某些特征信息的情况下，对变量的不确定性的度量。那么这些已知的特征信息做了多少贡献呢？这就得引入信息增益的概念了。特征 X 对于变量 Y 的信息增益 $g(Y, X)$ 的计算公式如下：

$$< br > g(Y, X) = H(Y) - H(Y \mid X) < br >$$

信息增益也被称为"互信息"，表示知道特征 X 的信息时，变量 Y 的信息不确定性减少程度。信息增益越大，表示特征提供的信息越多，这个特征也越重要。

下面通过一个小例子，具体展示下信息增益的计算过程：

X（今日的涨跌情况）	Y（下一个交易日的涨跌情况）
涨	涨
跌	跌
涨	涨
跌	跌
跌	涨
跌	跌

JoinQuant量化交易平台出品

在上表中，一共有六个 Y 的数据，其中三个是涨，三个是跌，涨和跌的概率分布都是 50%。记 $p_1 = p(Y = \text{涨}) = 1/2$，$p_2 = p(Y = \text{跌}) = 1/2$。那么 Y 的熵是：

Unknown environment 'align'

在 X 为涨的情况下，一共有两个 Y 的数据，都是涨。记 $p_1 = P(Y = \text{涨} \mid X = \text{涨}) = 1$，$p_2 = P(Y = \text{跌} \mid X = \text{涨}) = 0$ 则有：

Unknown environment 'align'

这里再重复一下，需要把 $(Y \mid X = \text{涨})$ 整体看做一个变量，才能更好的理解公式。

在 X 是跌的情况下，一共有四个 Y 的数据，三个是跌，一个是涨，记 $p_1 = P(Y = \text{跌} \mid X = \text{跌}) = 3/4$ $p_2 = P(Y = \text{涨} \mid X = \text{跌}) = 1/4$ 则有：

Unknown environment 'align'

我们考虑 X 本身的涨跌概率，记 $p < em > x_1 = P(X = \text{涨}) = 1/3$ $p < /em > x_2 = P(X = \text{跌}) = 2/3$ 则最终的条件熵为：

Unknown environment 'align'

信息增益为：

Unknown environment 'align'

小结：

上面给大家介绍了信息论中的信息增益的计算方法。通俗而言，利用信息增益可以衡量在引入一个变量之后，原有变量不确定性减少的程度。信息增益越高，表示新引入的变量效果越好。信息增益可以帮助我们了解各个因子是否有效，也可以用来衡量机器学习中的各个特征的重要性。还有更多的用法，大家可以自己去探索。

文章更迭记录：
v1.0，2016-07-02，文章上线

【量化课堂】协整的直观认识

导语：本文旨在直观地为大家介绍协整的概念，帮助大家理解其基本含义，这个概念提出的动机，以及简单的应用场景。

本图由 JoinQuant 量化课堂出品

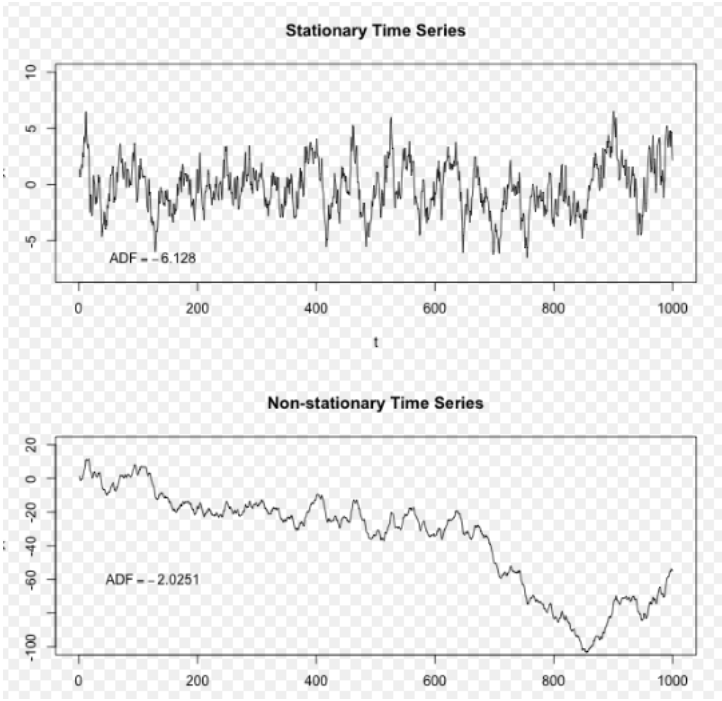
*本文由JoinQuant量化课堂推出 。难度标签为进阶下，理解深度标签：*level-0

作者：石佛，肖睿
编辑：宏观经济算命师
阅读前需要了解：
基本的统计概念，理解深度level-0
平稳的概念，理解深度level-0

这里只是想给大家指出协整的直观定义，并没有涉及严格的数学符号的定义及严密的公式推导。如果大家感兴趣，可以参考：https://en.wikipedia.org/wiki/Cointegration。量化课堂未来也会对其深度内容探讨。

一、平稳/协整

提到协整，就不得不提平稳性。
简单地说，平稳性（stationarity ）是一个序列在时间推移中保持稳定不变的性质，它是我们在进行数据的分析预测时非常喜欢的一个性质。如果一组时间序列数据是平稳的，那就意味着它的均值和方差保持不变，这样我们可以方便地在序列上使用一些统计技术。我们先看一个例子，了解平稳和非平稳序列直观上长什么样。



图片来源：维基百科
上图中，靠上的序列是一个平稳的序列，我们能看到它始终是围绕着一个长期均值在波动，靠下的序列是一个非平稳序列，我们能看到它的长期均值是变动的。
举一个应用的例子，如果某个资产的价格序列（或者两个序列的价差）是平稳的，那么当它在偏离了其均值后，人们可以期待价格会在未来的某一个时间回归这个均值。我们可以借助这个性质进行投资从而获利。假设一只股票的长期均值是9元，而现在的价值是8元。如果经过检验，我们认为这个股票的历史序列具有平稳的性质，并且假设这个平稳性是能保持的，就可以买入这只股票，等待未来它的价格回归9元，从而获得1元的利润。

这就是一个具有平稳性质的股票价格序列：

平稳性是很好用，但在现实中，绝大多数的股票都是非平稳的，那么我们是否还能够利用平稳性质进行获利呢？答案是肯定的，这时协整关系（cointegration）就出场了！如果两组序列是非平稳的，但它们的线性组合可以得到一个平稳序列，那么我们就说这两组时间序列数据具有协整的性质，我们同样可以把统计性质用到这个组合的序列上来。但是需要指出的一点，协整关系并不是相关关系（correlation）。
举个例子，两组时间序列数据的差是平稳的，则我们可以根据这个差的平稳性进行投资获利：当两只股票的价差过大，根据平稳性我们预期价差会收敛，因此买入低价的股票，卖空高价的股票，等待价格回归的时候进行反向操作从而获利。
这就是配对交易（pairs trading）的由来。是不是很清晰。

二、平稳性和检验方法

严格地说，平稳性可以分为严平稳（strictly stationary）和弱平稳（或叫协方差平稳，covariance stationary等）两种。严平稳是指一个序列始终具有不变的分布函数，而弱平稳则是指具序列有不变的常量的描述性统计量。严平稳和弱平稳性质互不包含；但如果一个严平稳序列的方差是有限的，那么它是弱平稳的。我们一般所说的平稳都是指弱平稳。在时间序列分析中，我们通常通过单位根检验（unit root test）来判断一个过程是否是弱平稳的。
一个常见的单位根检验方法是Dickey-Fuller test，大致思路如下：假设被检测的时间序列$Y < em > t$满足自回归模型$Y_t = \alpha Y < /em > t-1 + \varepsilon_t$，其中$\alpha$为回归系数，$\varepsilon_t$为噪声的随机变量。若经过检验，发现Misplaced &，则可以肯定序列是平稳的。在Dickey Fuller Test的基础上，还有衍生出来的augmented Dickey-Fuller test，是考虑了序列的滞后性，在此不做过多陈述，在未来量化课堂的数学专栏里将会介绍。

三、举个应用的例子

我们人为地构造两组数据，由此直观地看一下协整关系。

```
import numpy as np
import pandas as pd
import seaborn
import statsmodels
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import coint
```

构造数据
首先，我们构造两组数据，每组数据长度为100。第一组数据为100加一个向下趋势项再加一个标准正态分布。第二组数据在第一组数据的基础上加30，再加一个额外的标准正态分布。有：

$$< br > X_t = 100 + \gamma_t + \varepsilon_t < br >$$

$$< br > Y_t = X_t + 30 + \mu_t < br >$$

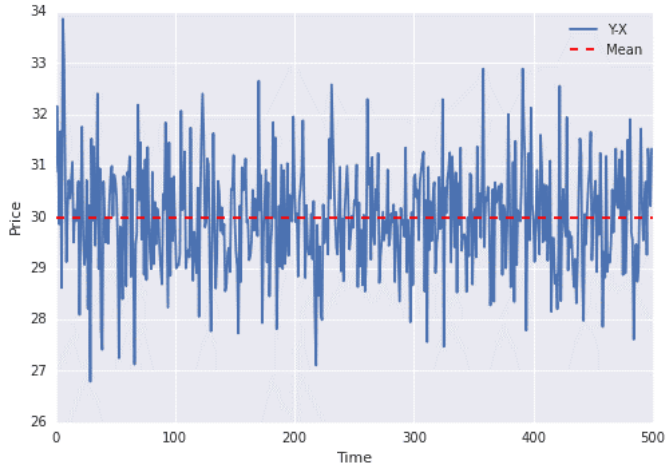
其中γ_t为趋势项，ε_t和μ_t为无相关性的正态随机变量。
代码如下：

```
np.random.seed(100)
x = np.random.normal(0, 1, 500)
y = np.random.normal(0, 1, 500)
X = pd.Series(cumsum(x)) + 100
Y = X + y + 30
for i in range(500):
    X[i] = X[i] - i/10
    Y[i] = Y[i] - i/10
plot(X); plot(Y);
plt.xlabel("Time"); plt.ylabel("Price");
plt.legend(["X", "Y"]);
```

本图由 JoinQuant 量化课堂出品

显然，这两组数据都是非平稳的，因为均值随着时间的变化而变化。但这两组数据是具有协整关系的，因为他们的差序列$Y_t - X_t$是平稳的：

```
plot(Y-X);
plt.axhline((Y-X).mean(), color="red", linestyle="--");
plt.xlabel("Time"); plt.ylabel("Price");
plt.legend(["Y-X", "Mean"]);
```



本图由 JoinQuant 量化课堂出品

上图中，可以看出蓝线$Y_t - X_t$一直围绕均值波动。而均值不随时间变化（其实方差也不随时间变化）。

小结：如果完全从数学的角度讲清楚协整，会比较复杂，日后的量化课堂会有涉及。我们只是在了解（level-0）的层面上做了一个简单介绍，目的还是让大家更好的将协整与实际应用结合起来。

注：文末有可运行的代码块，大家可以在joinquant克隆研究，运行程序。

更多深入研究，请参见后面的应用类文章。

文章更迭记录：

v1.1，2016-10-13，修正概念性错误，感谢 zhangyi 指出

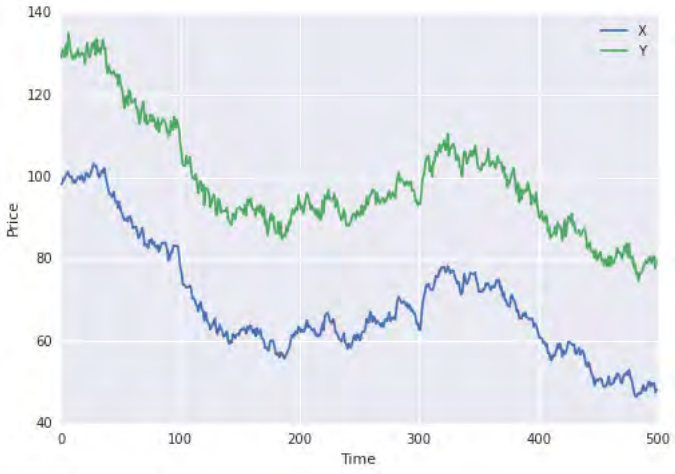
v1.0，2016-07-05，文章上线

In [1]:

```
import numpy as np
import pandas as pd
import seaborn
import statsmodels
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import coint
```

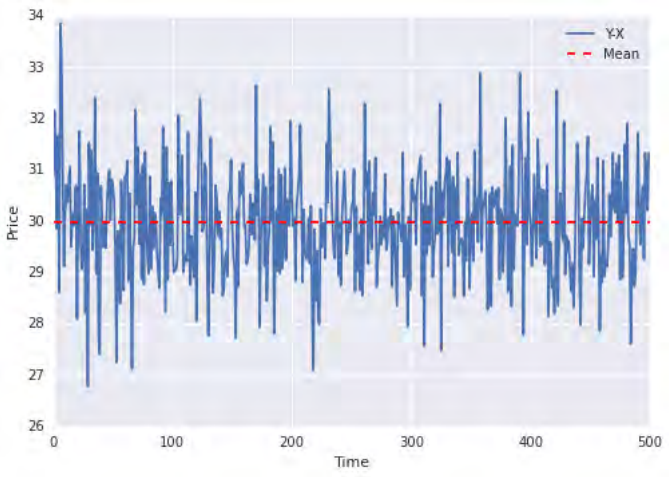
In [2]:

```
np.random.seed(100)
x = np.random.normal(0, 1, 500)
y = np.random.normal(0, 1, 500)
X = pd.Series(cumsum(x)) + 100
Y = X + y + 30
for i in range(500):
    X[i] = X[i] - i/10
    Y[i] = Y[i] - i/10
plot(X); plot(Y);
plt.xlabel("Time"); plt.ylabel("Price");
plt.legend(["X", "Y"]);
```



In [3]:

```
plot(Y-X);
plt.axhline((Y-X).mean(), color="red", linestyle="--");
plt.xlabel("Time"); plt.ylabel("Price");
plt.legend(["Y-X", "Mean"]);
```



In [ ]:

```
```

【量化课堂】朴素贝叶斯入门

导语：在现实生活中，我们经常要利用观测现象（特征数据）推测现象背后的原因。例如我们看到草地湿了，需要判断是不是下雨导致的；今天的交易量大涨，需要判断是有新资金入场、还是存量资金雄起了一把；去医院体检，检查结果为阳性，是因为真的得病了，还是因为医院的误诊。朴素贝叶斯算法可以利用历史数据的分布，给你一个最有可能的结果，使你犯错误的概率最小化。

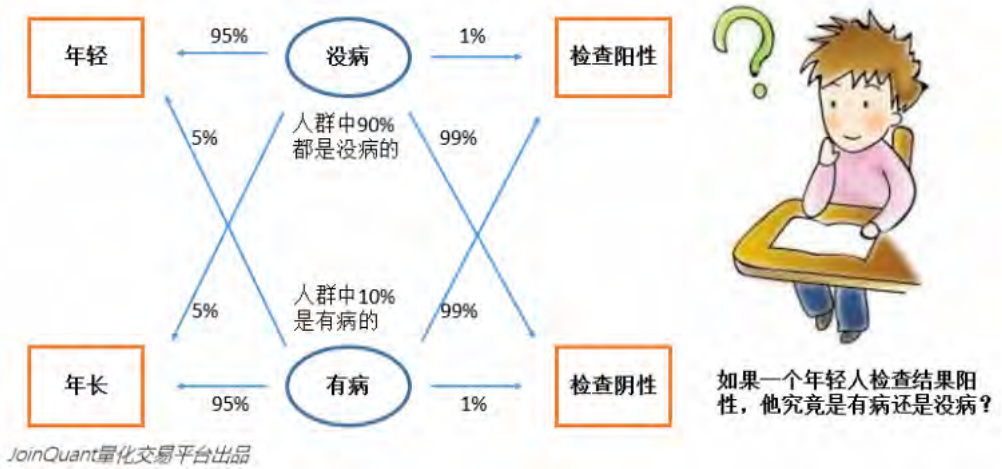
本文由JoinQuant量化课堂推出 。难度标签为进阶上，理解深度标签：level-0

作者：yongpeng.r

编辑：宏观经济算命师

1. 一个例子说清楚的事情绝不用定义：

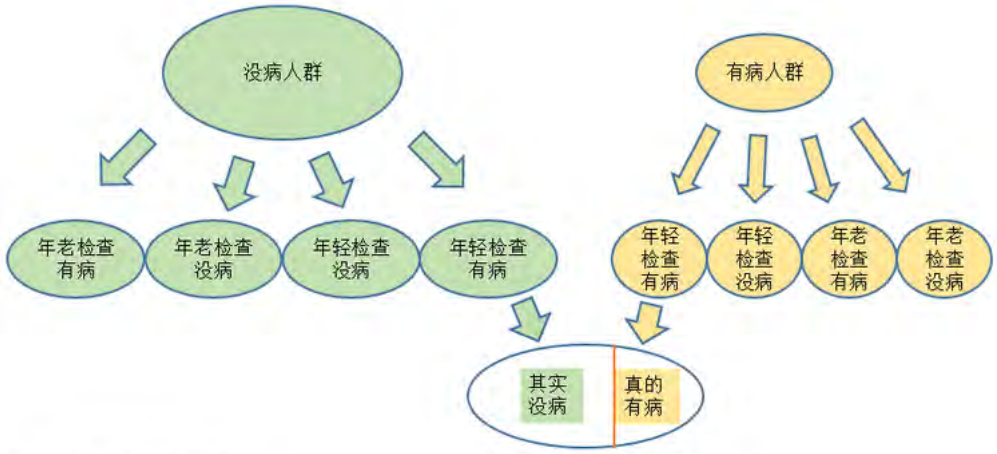
先说一句特别复杂的朴素贝叶斯介绍：朴素贝叶斯法是基于贝叶斯定理，特征条件独立假设和后验概率最大化的分类方法。不知道你们看完了什么感受，反正我是一脸懵逼。下面我用一个小例子让大家明白这到底是怎么回事。
举个例子：某一种病，年轻人得病的概率远远小于年长的。如果一个年轻人检查为阳性，那么他就能直接被确诊吗？要知道，检查为阳性，可能会是误诊的哦。这个时候，朴素贝叶斯就登场了。我们不妨把各种可能性列出来，画一个图：



上图中，箭头附近的数字表示各种情况的概率。例如，没病然而检查为阳性（说明误诊了）的概率是1%，没病而且检查为阴性的概率是99%。如果一个年轻人去医院体检，体检结果是有病，那么这个人到底是有病还是没病呢？或者说这个人真实得病的概率有多大呢？

有的人可能会说，既然有病的人会有99%的被确诊，至少得病的概率比没病的概率要高吧。其实，一个年轻人检查出有病，真正得病的概率比没病的概率还要低！

现在我们来分析下：假设人群中有20000人，按照第一个图中的患病的概率，18000人是没有病的，2000人是有病的；在18000个没有病的人中，年轻人的概率为95%，检查为阳性的概率为1%，那么“年轻”并且“检查为阳性”并且“没病”的人数一共有$18000 \cdot 95\% \cdot 1\% = 171$人；在2000个有病的人中，年轻人的概率是5%，检查为阳性的概率是99%。那么年轻人得病并且被检查出来的人数为$2000 \cdot 5\% \cdot 99\% = 99$人。如果我们现在只知道这个人是年轻人，而且检查结果是阳性：那么他有可能是本身真的得病并且被检查出来的人，也有可能是误诊了的人。前者的概率就是$99/(99+171) = 36.7\%$。后者为$171/(99+171) = 63.3\%$。很显然，我们有更大的可能性相信他没有得病（所以现实生活中，医生会让你多复诊几次）。



年轻检查有病的人员构成

在上面的整个分析过程中，我们就分别用到了特征条件独立假设，贝叶斯定理，条件概率最大化这几个知识点。下面我们进行详细说明。

2. 特征的条件独立假设

我们的目的是通过“目前已知的数据”判断未知的结果，那么这个“目前已知的数据”就被称为特征。在上面判断有没有得病的例子中，特征就是这个人“是否年轻”以及“检查结果是否为阳性”。

这里我们要做一个重要的假设：上述两个特征之间是独立的。在判断这个人有没有病的时候，我们认为这个人“是否年轻”和“检查是否为阳性”之间没有联系。因此，随机抽取一个检查者，他“年轻”并且“检查结果为阳性”的概率就等于“年轻”的概率乘以“检查结果为阳性”的概率。

上面这个假设就是条件独立假设。如果变量不满足独立性，则不可以将两者的概率相乘，比如天空有云的概率是0.5，下雨的概率是0.33，但下雨和“天空中有云”不是独立的，就不能得到“即有云又下雨”的概率为$0.5 \cdot 0.33$这个结论。

3.贝叶斯定理

贝叶斯定理主要描述在给定特征数据的情况下，判定属于某个类别的概率。在下面的公式中，样本的数据用$X = x$表示，样本的类别属于某个类别用$Y = c_k$表示。

$$P(Y = c_k|X = x) = (P(X = x|Y = c_k)P(Y = c_k))/(\sum_k P(X = x|Y = c_k)P(Y = c_k))$$

在章节1给出的例子中，注意看我们计算真正得病的公式：

$$\text{年轻人检查为阳性而真的得病的概率} = A/B = 99/(99 + 171)$$

其中，A为真实得病的人中“检查为阳性”并且“年轻”的数目，B为人群中所有“检查为阳性”并且“年轻”的人的数目，其实这个公式就是贝叶斯定理的公式。现在感觉贝叶斯定理是不是简单了很多。

4. 后验概率最大化

我们知道一个“年轻”人“检查为阳性”，现在需要你告诉他有病还是没病。那么很显然，我们只要用上文公式做计算，看看这个“年轻”并且“检查结果为阳性”的人到底是得病的概率更高，还是没得病的概率更高。这个就是后验概率最大化的直观解释。在上面的例子中，我们就可以告诉他，检查结果为阳性也不意味着你就得病了，但是为了安全起见，需要后续跟进复查。

5. 朴素贝叶斯的具体使用-sklearn

上面通过了一个小例子介绍了一下朴素贝叶斯算法的算法原理，但是如何在实际的代码中使用朴素贝叶斯算法帮助我们完成分类呢？下面介绍下python环境中的朴素贝叶斯算法是如何使用的。

特征是通过收盘价数据计算的SMA，WMA，MOM指标，训练样本的特征是从2007-1-4到2016-6-2中截止前一天的SMA，WMA，MOM指标，训练样本的类别标签是2007-1-4日到2016-6-2中每一天的涨跌情况，涨了就是True，跌了就是False，测试样本是2016-6-3日的三个指标以及涨跌情况。我们可以判定之后判断结果是正确还是错误，如果通过朴素贝叶斯判断的结果和当天的涨跌情况相符，则输出True，如果判断结果和当天的涨跌情况不符，则输出False。（和SVM那一篇中例子的作用是一样滴，只是为了展示如何使用，不对预测的准确性做担保啊）

In [1]:

```
import talib
from jqdata import *

test_stock = '399300.XSHE'
start_date = datetime.date(2007, 1, 4)
end_date = datetime.date(2016, 6, 3)

trading_days = get_all_trade_days()
start_date_index = trading_days.index(start_date)
end_date_index = trading_days.index(end_date)
```

```
x_all = []
y_all = []

for index in range(start_date_index, end_date_index):
    # 得到计算指标的所有数据
    start_day = trading_days[index - 30]
    end_day = trading_days[index]
    stock_data = get_price(test_stock, start_date=start_day, end_date=end_day, frequency='daily', fields=['close'])
    close_prices = stock_data['close'].values

    #通过数据计算指标
    # -2是保证获取的数据是昨天的，-1就是通过今天的数据计算出来的指标
    sma_data = talib.SMA(close_prices)[-2]
    wma_data = talib.WMA(close_prices)[-2]
    mom_data = talib.MOM(close_prices)[-2]

    features = []
    features.append(sma_data)
    features.append(wma_data)
    features.append(mom_data)

    label = False
    if close_prices[-1] > close_prices[-2]:
        label = True
    x_all.append(features)
    y_all.append(label)


# 准备算法需要用到的数据
x_train = x_all[: -1]
y_train = y_all[: -1]
x_test = x_all[-1]
y_test = y_all[-1]
print('data done')
```

```
data done
```

In [4]:

```
from sklearn.naive_bayes import GaussianNB

#开始利用机器学习算法计算
clf = GaussianNB()
#训练的代码
clf.fit(x_train, y_train)
#得到测试结果的代码
prediction = clf.predict(x_test)

# 看看预测对了没
print(prediction == y_test)
print('all done')
```

```
[ True]
all done
```

【量化课堂】机器学习之神经网络入门

导语：想过一个问题没有？人们看到一个猫的图片时，立马就能知道这是一只猫，看到一个狗的图片，立马就能知道这是一只狗。时间非常快而且很少出错，为什么能够达到这么好的效果呢？因为大脑本质上是一个非常高效的处理器。而这个处理器其实是由巨量神经元细胞组成的神经网络。

人们早早（20世纪40年代）就开了脑洞，为什么不尝试通过“模拟神经元细胞网络的行为”这种手段解决实际问题呢？随着机算能力的提高（20世纪80年代以后），神经网络算法慢慢形成了热点。

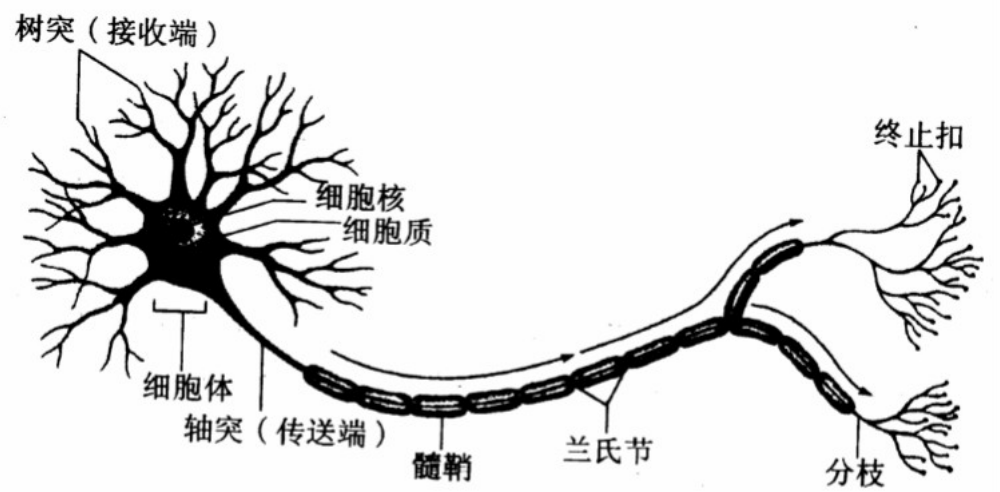
很多常见的问题都可以用神经网络轻松处理好。例如提供手写的字的图片，神经网络算法就可以完成内容的识别；提供股票历史交易日的信息，神经网络可以判断股票明天是涨是跌，甚至可以告诉你最可能的收盘价是多少（准不准另说，反正人脑也未必准）。
*本文由*JoinQuant*量化课堂推出。难度标签为进阶上，理解深度标签：*level-0

作者： yongpeng.r

编辑： 宏观经济算命师

1.神经网络算法的基础-神经元细胞和神经元细胞网络

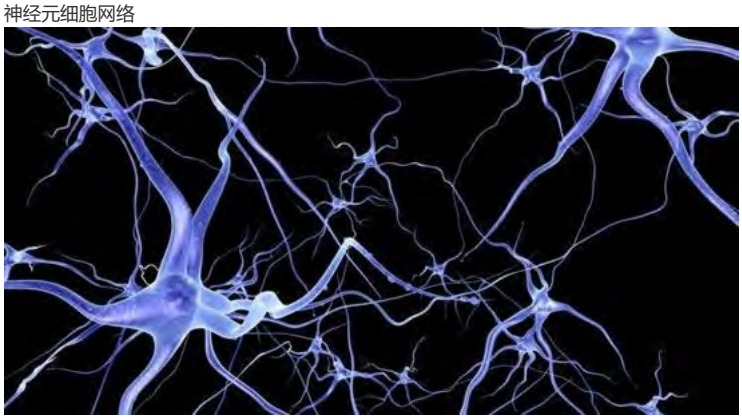
大脑中的神经元细胞
为了大家更好的理解神经网络的工作过程，我先把神经元细胞简单的介绍一下：



上图就是一个标准的神经元细胞结构。神经元通过树突（左上角的那些树状结构）来接收信息，这些信息可以是别的神经元传递过来的信息，也可以是直接物理刺激（例如声音造成的对于细胞的刺激）。通过对这些信息的综合处理，神经元细胞体将处理结果通过轴突（下方的那条长长的传送结构）传递给别的神经元。

大脑的神经细胞只有两种状态：兴奋和不兴奋（即抑制）。神经细胞通过某种方法（人类还没研究透），把所有从树突上进来的信号进行相加，如果信号总和超过某个阀值，就会激发神经细胞进入兴奋（fire）状态。此时神经细胞就会发出一个电信号，并且通过轴突发送给其他神经细胞。如果信号总和没有达到阀值，神经细胞就不会兴奋起来。

神经元细胞网络



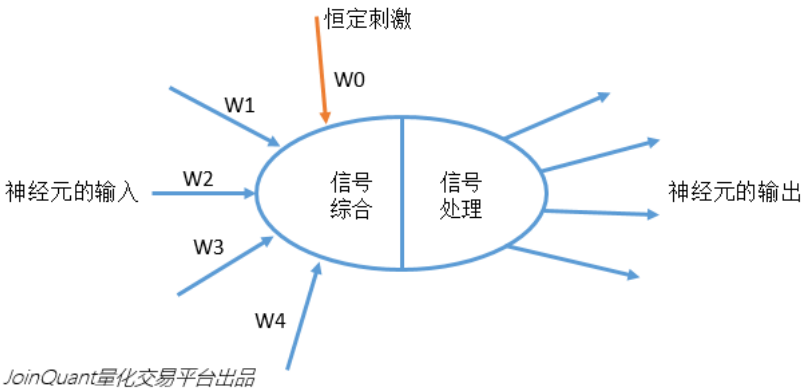
单个神经细胞处理不了任何复杂的问题，必须形成神经网络才行。

神经元细胞网络中、细胞间的联系错综复杂。神经网络综合每个细胞的成果，最终完成了非常复杂的任务。人类目前还没有完全弄清楚神经网络的运作，但并不妨碍开脑洞，自己设计一套模型：）。

2.神经网络算法-神经元和神经网络

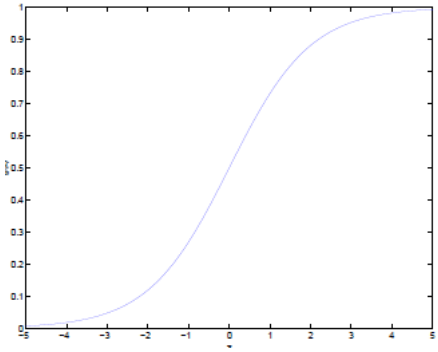
神经元-抄袭神经元细胞行为的人工神经元

人工神经元被用来模拟神经元细胞的行为，其结构如下：



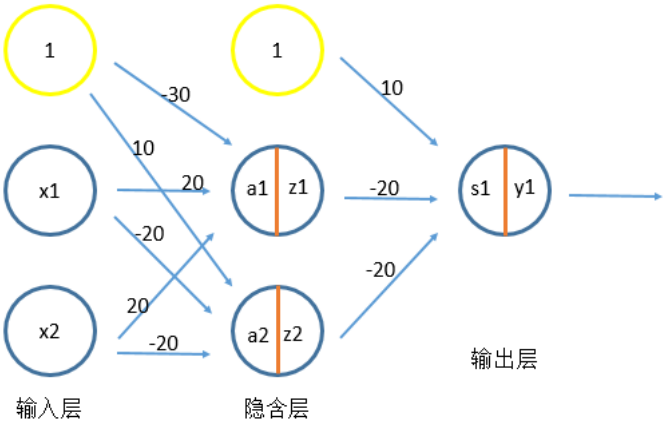
左边的箭头模拟神经元的输入信号，右边的箭头模拟神经元的输出信号。神经元主要做了两部分的工作：首先对于输入信号进行综合，其次通过综合后的信号进行处理，得到输出信号。信号的综合，常用的方法是对输入信号加权求和。信号的处理，常用的方法是通过sigmoid函数处理。

sigmoid函数的表达式为$g(z) = 1/(1 + e^{-z})$，图像是：



可以看出，sigmoid函数的处理结果和真实神经元的处理结果比较类似（sigmoid函数值集中于1和0，对应于神经元细胞的兴奋和不兴奋状态），而且sigmoid函数可以根据函数值很容易的得到导数值。
这样，一个人工神经元就搭建起来了。

神经网络-神经元细胞网络简化版
人工神经网络其实是真实神经网络的简化。一个典型的人工神经网络如下图所示：



JoinQuant量化交易平台出品

神经网络由三层组成，输入层，隐含层，输出层。需要注意的是，输入层和输出层都只有一层结构，隐含层可以是一层，也可以包含多层结构（本图中只有一层）。图中x1和x2是输入数据，y1是输出数据。黄色的圈表示对于神经元的恒定刺激，a1和a2是隐含层输入信号的信号综合结果，z1和z2是隐含层的信号处理结果，s1是输出层输入信号的信号综合结果。

这么说有点抽象，我们实际推演一下：上面这个神经网络可以用来计算异或。异或是一种运算：如果输入的a、b两个值不相同，则异或结果为1。如果输入的a、b两个值相同，异或结果为0。那用神经网络如何做到异或呢?
当输入的x1=1，x2=1的时候：

$$a1 = 1 \cdot (-30) + x1 \cdot 20 + x2 \cdot 20 = 1 \cdot (-30) + 1 \cdot 20 + 1 \cdot 20 = 10$$

$$z1 = sigmoid(a1) \approx 1$$

$$a2 = 1 \cdot 10 + x1 \cdot (-20) + x2 \cdot (-20) = 1 \cdot (10) + 1 \cdot (-20) + 1 \cdot (-20) = -30$$

$$z2 = sigmoid(a2) \approx 0$$

$$s1 = 1 \cdot (10) + z1 \cdot (-20) + z2 \cdot (-20) = 1 \cdot (10) + 1 \cdot (-20) + 0 \cdot (-20) = -10$$

$$y1 = sigmoid(s1) \approx 0$$

同理，当x1=1，x2=0的时候，y1=1；
当x1=0，x2=1的时候，y1=1；
当x1=0，x2=0的时候，y1=0。
本文主要目的是让大家对于神经网络有一个直观的认知。关于神经网络的非线性分类能力、神经网络的训练等过程等比较复杂。请大家继续关注量化课堂，我们会在将来推出相关内容。

3.神经网络的具体使用-pybrain 包

在python环境中，我们可以通过pybrain包来帮助我们构建神经网络，训练神经网络并且利用神经网络来对新的数据进行分类。下面是一个小例子帮助我们了解如何使用pybrain包。（一个小提示：在python2环境下使用pybrain包比较好）

特征是通过收盘价数据计算的SMA，WMA，MOM指标，训练样本的特征是从2007-1-4到2016-6-2中截止前一天的SMA，WMA，MOM指标，训练样本的分类类别是2007-1-4日到2016-6-2中每一天的涨跌情况，涨了就是True，跌了就是False，测试样本为2016-6-3日的三个指标以及涨跌情况。程序的结尾输出的是测试样本中的判断误差，由于只有一个测试样本，如果神经网络的判断结果和真实涨跌情况相同，输出0%；如果神经网络的判断结果和真实涨跌情况不相同，输出是100%。

In [1]:

```
import talib
from jqdata import *
from pybrain.datasets import ClassificationDataSet

# 测试的股票代码，这里用沪深300做测试
test_stock = '000300.XSHG'
# 测试开始日期和终止日期
start_date = datetime.date(2007, 1, 4)
end_date = datetime.date(2016, 6, 3)
# 所有的交易日期
trading_days = get_all_trade_days()
start_date_index = trading_days.index(start_date)
end_date_index = trading_days.index(end_date)
# 训练数据，6月3日之前所有的日期的数据都用来训练
# 这是一个分类数据集，参数分别是输入的特征的数目，输出的结果的数目，类别的数目
# 因为咱们用到了三个特征，输出的结果只有1一个结果，其中的类别是涨或者跌，所以参数是3，1，2
trndata = ClassificationDataSet(3, 1, nb_classes=3)
# 测试数据，6月3日的数据用来测试
tstdata = ClassificationDataSet(3, 1, nb_classes=3)

# 先生成训练数据和测试数据
for index in range(start_date_index, end_date_index):
    # 得到计算指标的所有数据
    start_day = trading_days[index - 30]
    end_day = trading_days[index]
    stock_data = get_price(test_stock, start_date=start_day, end_date=end_day, frequency='daily', fields=['close'])
    close_prices = stock_data['close'].values

    #通过数据计算指标
    # -2是保证获取的数据是昨天的，-1就是通过今天的数据计算出来的指标
    sma_data = talib.SMA(close_prices)[-2]
    wma_data = talib.WMA(close_prices)[-2]
    mom_data = talib.MOM(close_prices)[-2]

    # 训练数据或者测试数据的输入特征
    features = []
    features.append(sma_data)
    features.append(wma_data)
    features.append(mom_data)
    # 训练数据或者测试的标签数据，就是涨或者跌，涨用1表示，平或者跌用0表示
    label = 0
    if close_prices[-1] > close_prices[-2]:
        label = 1
    elif close_prices[-1] < close_prices[-2]:
        label = -1

    # 6月3号之前的数据，都用作训练数据
    if index < end_date_index - 1:
        trndata.addSample(features, [label])
    else:
        tstdata.addSample(features, [label])

trndata._convertToOneOfMany( )
tstdata._convertToOneOfMany( )

print('data done')
```

```
data done
```

In [2]:

```
from pybrain.tools.shortcuts import buildNetwork
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure.modules   import SoftmaxLayer
from pybrain.utilities           import percentError

fnn = buildNetwork( trndata.indim, 5, trndata.outdim, outclass=SoftmaxLayer)
trainer = BackpropTrainer( fnn, dataset=trndata, momentum=0.1, verbose=True, weightdecay=0.01)

# 训练一次，可以通过括号里面的数字调节训练的次数
trainer.trainEpochs( 1 )

print(trainer.testOnClassData(dataset=tstdata )[0])

# 计算预测误差，这个是误差，不是预测准确率，0表示预测对了，100%表示预测错了

tstresult = percentError( trainer.testOnClassData(
           dataset=tstdata ), tstdata['class'] )
print(tstresult)
```

```
Total error:  0.0864583471925
1
0.0
```

【量化课堂】一只兔子帮你理解 kNN

导语：商业哲学家 Jim Rohn 说过一句话，“你，就是你最常接触的五个人的平均。”那么，在分析一个人时，我们不妨观察和他最亲密的几个人。同理的，在判定一个未知事物时，可以观察离它最近的几个样本，这就是 kNN（k最近邻）的方法。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，本文的难度属于进阶（上），深度为 level-1

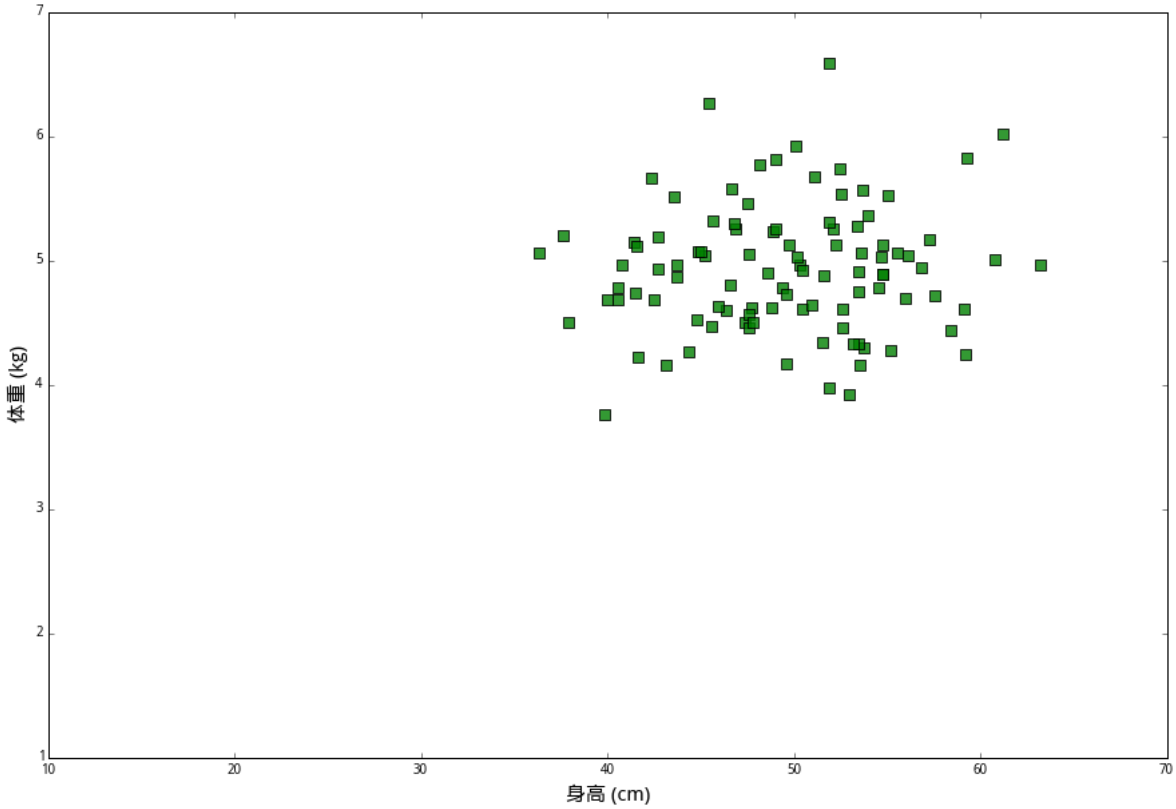
简介

kNN（k-Nearest Neighbours）是机器学习中最简单易懂的算法，它的适用面很广，并且在样本量足够大的情况下准确度很高，多年来得到了很多的关注和研究。kNN 可以用来进行分类或者回归，大致方法基本相同，本篇文章将主要介绍使用 kNN 进行分类。

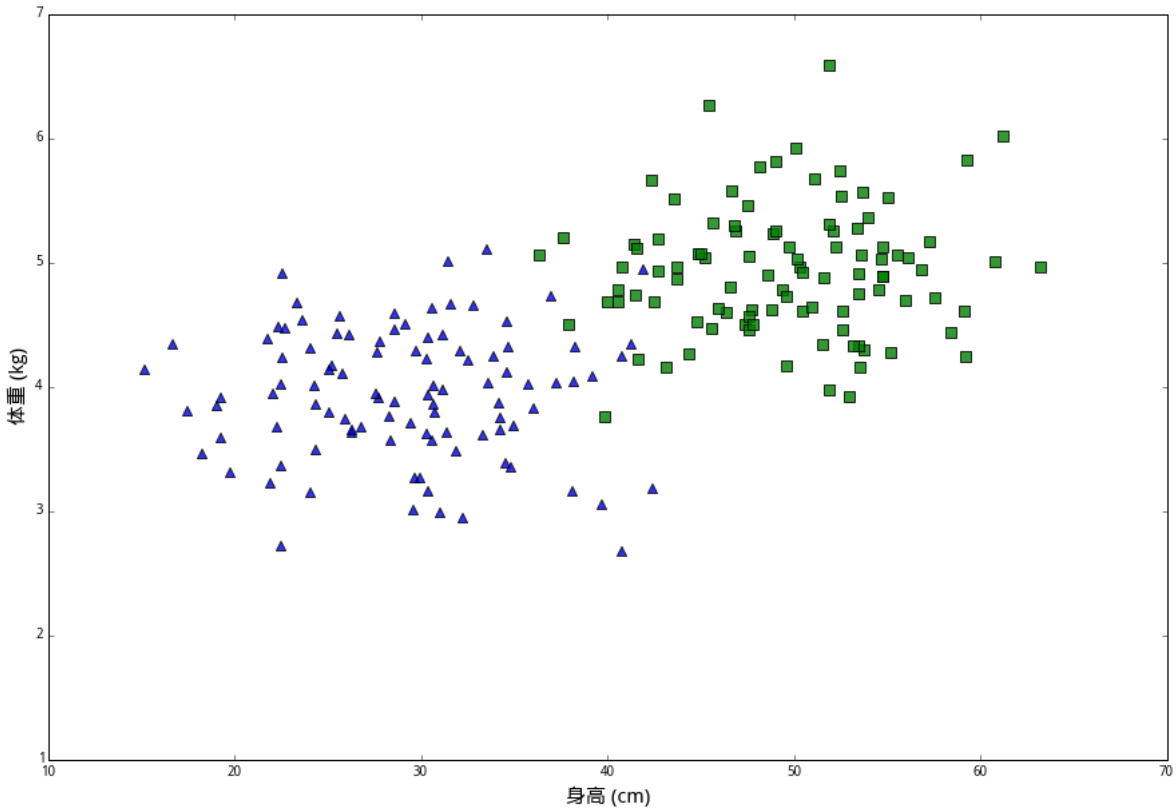
举个例子跟你讲

kNN还真是直接讲例子最好懂。大家都喜欢兔子，所以就来说一说兔子的事情吧。

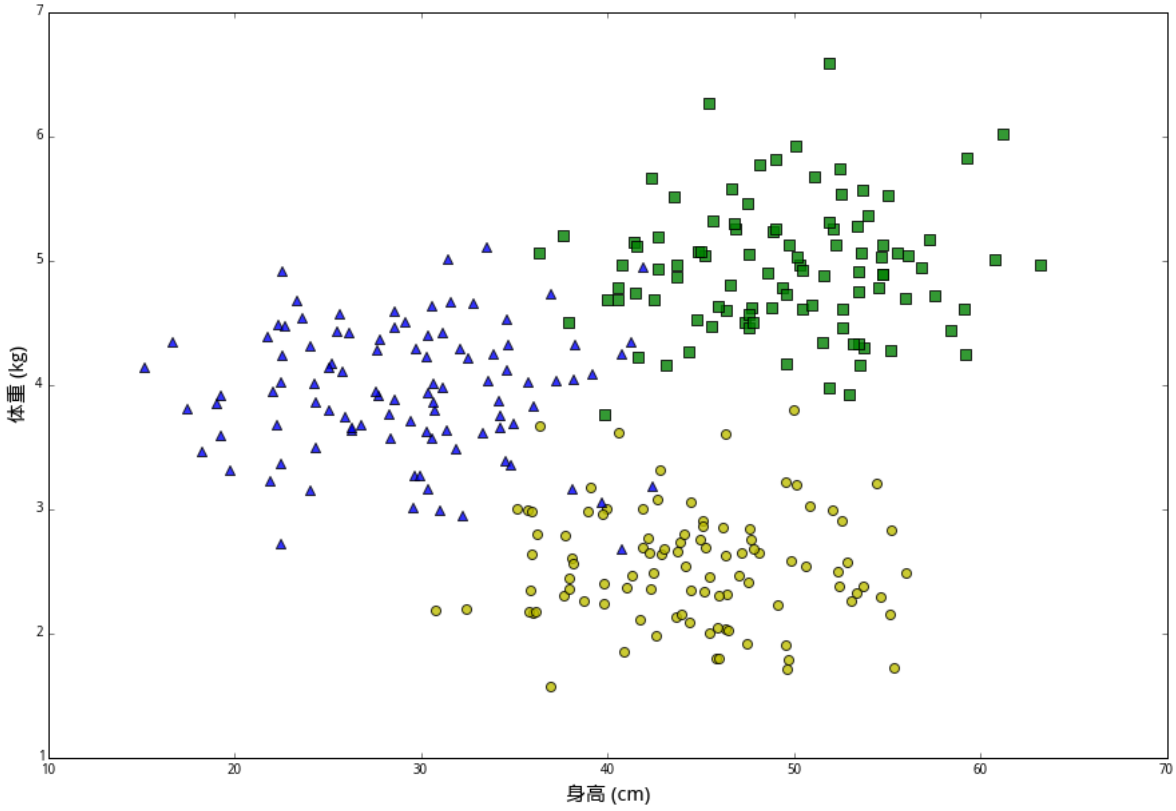
有一种兔子叫作***悲伤（***Grief***）***，它们的平均身高是 50 厘米，平均体重 5 公斤。我们拿来一百个悲伤，分别测量它们的身高和体重，画在坐标图上，用绿色方块表示。



还有一种兔子呢，叫作***痛苦（***Agony***）***。它们体型比较小，平均身高是 30 厘米，平均体重是 4 公斤。我们将一百个痛苦的身高和体重画在同一个坐标图上，用蓝色三角表示。



最后一种兔子叫***绝望***（Despair）。它们的平均身高45厘米，但体重较轻，平均只有2.5公斤。一百只绝望的数据用黄色圆圈表示。

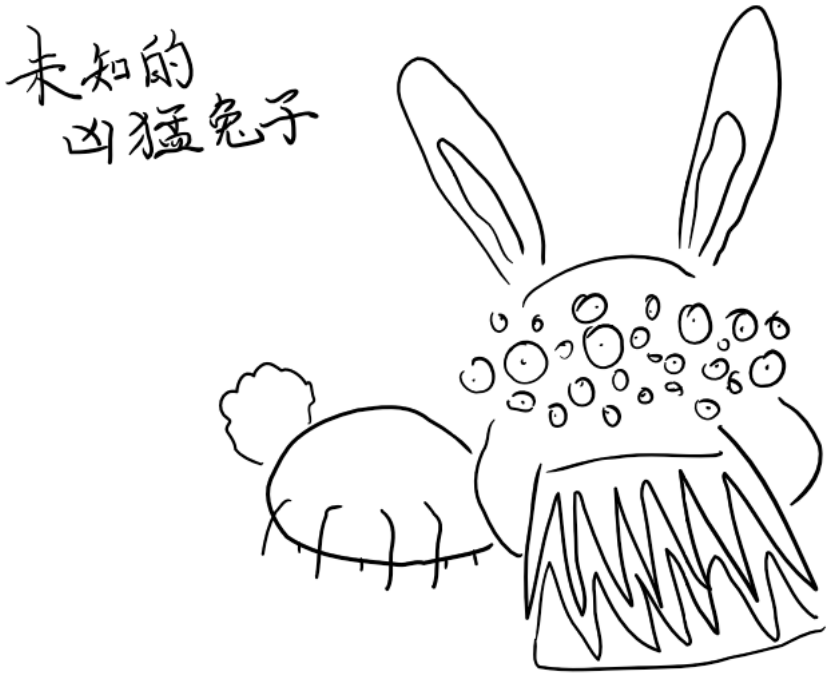


在这些数据中，$(\text{身高}, \text{体重})$ 的二元组叫做*特征*（features），兔子的品种则是分类*标签*（class label）。我们想解决的问题是，给定一个未知分类的新样本的所有特征，通过已知数据来判断它的类别。

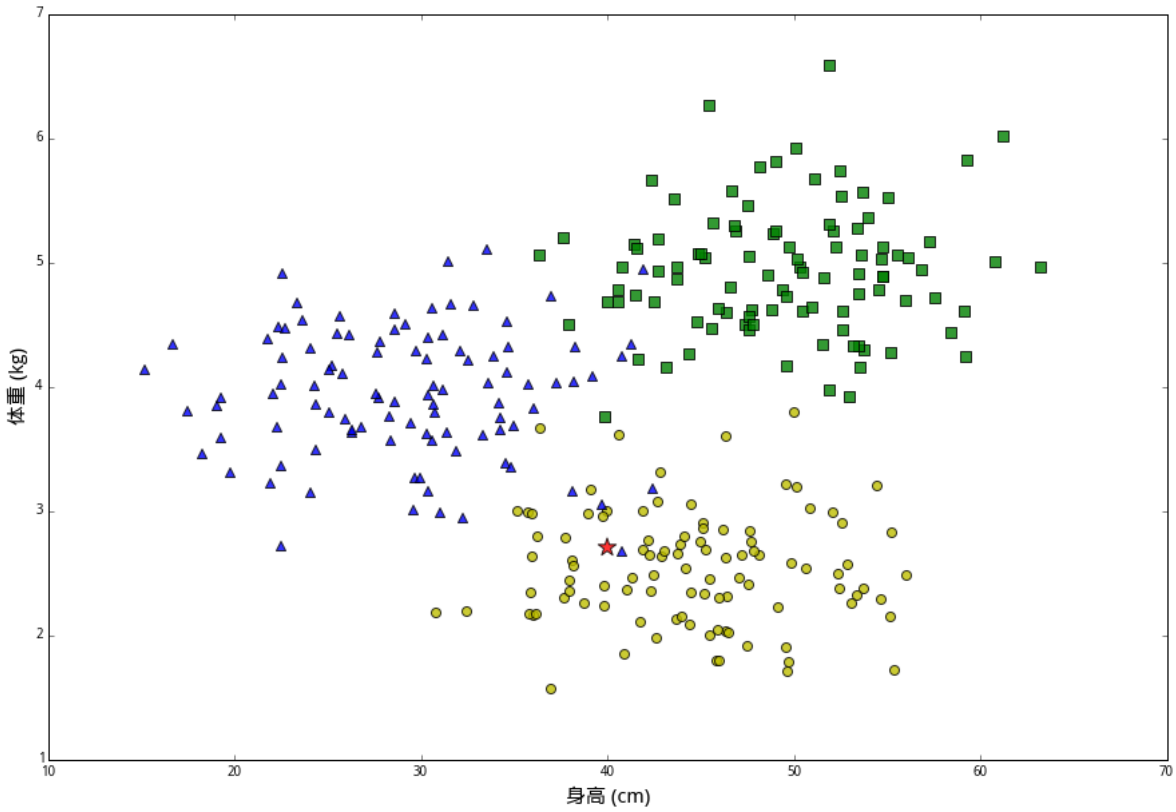
北京十八环外有一个小树林里经常出现这三种兔子。为了了解它们的生态环境，某研究团队想知道三种兔子的数量比例；可是这些兔子又太过危险，不能让人亲自去做，所以要设计一个全自动的机器人，让它自己去树林里识别它遇到的每一个兔子的种类。啊，为了把故事讲圆，还要假设他们经费不足，所以机器只有测量兔子的身高和体重的能力。

那么现在有一迷之兔子，我们想判断它的类别，要怎么做呢？按照最普通的直觉，应该在已知数据里找出几个和我们想探究的兔子最相似的几个点，然后看看那些兔子都是什么个情况；如果它们当中大多数都属于某一类别，那么迷之兔子大概率也就是那个类别了。

于是乎，我们给机器人预设一个整数 k，让它去寻找距离最近的k个数据样本进行分析。好，机器发现了一只兔子，它长着八条腿，三十二只眼睛，毛茸茸的小尾巴，齐刷刷的八十六颗獠牙，面相狰狞，散发着噩梦般的腐臭，发出来自地狱底处的咆哮… 差不多就是这个样子（作者手绘）：

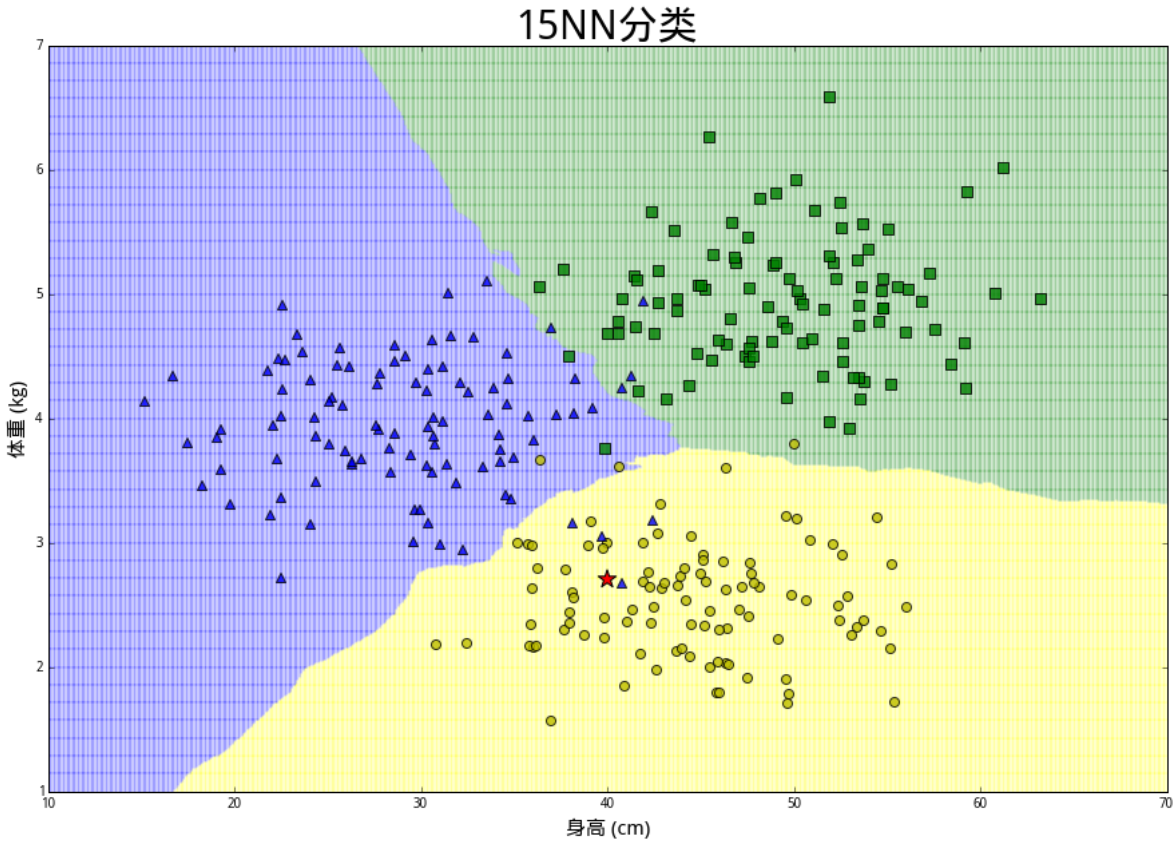


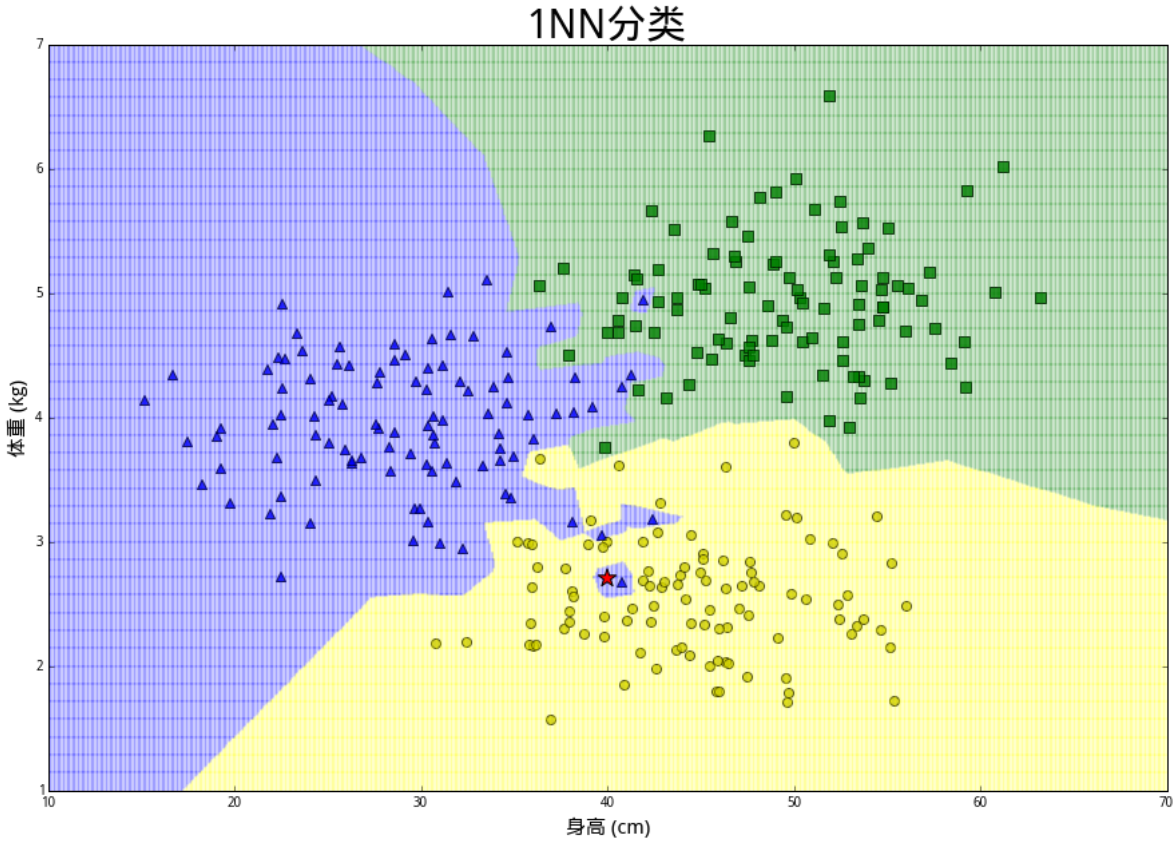
可我们的机器才识别不了那么多，它只测量出这只兔子身长 40 厘米，体重 2.7 公斤，就是下面图中那颗闪闪发亮的红星



kNN 算法如何对这次观测进行分类要取决于k的大小。直觉告诉我们迷之兔像是一只绝望，因为除了最近的蓝色三角外，附近其他都是黄色圆圈。的确，如果设 $k = 15$，算法会判断这只兔子是一只绝望。但是如果设 $k = 1$，那么由于距离最近的是蓝色三角，会判断迷之兔子是一只痛苦。

如果按照15NN和1NN的方法对这个二维空间上的每一个点进行分类，会形成以下的分割





在两组分类中，1NN 的分类边界明显更“崎岖”，但是对历史样本没有误判；而 15NN 的分类边界更平滑，但是对历史样本有发生误判的现象。选择k的大小取决于对偏差和方差之间的权衡，本篇不进行更深探讨，读者在使用 kNN 时凭感觉选一个 k 就好。

距离函数

我们在上面的例子中把一个很重要的概念隐藏了起来，在
选择一个数量k还只是小问题，更重要的是距离的计算方法。毕竟，当我们说“最近的k个点”时，这个“近”是怎么衡量的?

在数学中，一个空间上距离的严格定义如下:
设 M 为一个空间，M 上的一个距离函数是一个函数 $d : M \times M \to \mathbb{R}$，满足

- $d(x, y) \geq 0 \ \forall x, y \in M$
- $d(x, y) = 0 \iff x = y$

• $d(x,y) = d(y,x)\ \forall x, y \in M$
• $d(x,z) \leq d(x,y) + d(y,z)\ \forall x, y, z \in M$
两个点 x, y 之间的距离就是 $d(x,y)$。

我们一般最常用的距离函数是欧氏距离，也称作 $L < em > 2$距离。如果 $x = (x_1, x_2, \ldots, x_n)$和 $y = (y_1, y_2, \ldots, y_n)$是 n 维欧式空间 $\mathbb{R}^n$ 上的两个点，那它们之间的 L_2 距离是

$$d_2(x,y) = \sqrt{\sum < /em > i = 1^n (x < em > i - y_i)^2}.$$

L_2 是更普遍的 L_p 距离在 $p = 2$ 时的特列。L_p 距离的函数 d_p 定义如下：对于 Misplaced &，有

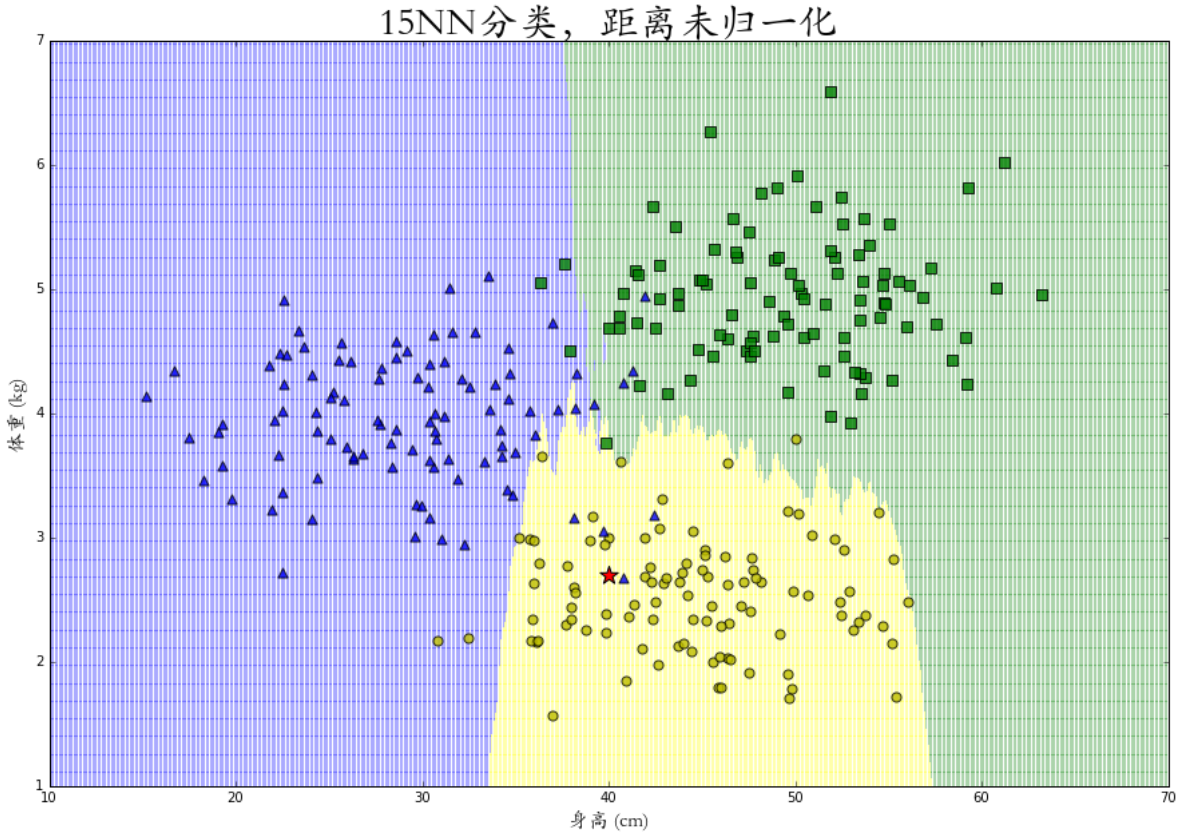
$$d_p(x,y) = \left(\sum < /em > i = 1^n |x < em > i - y_i|^p\right)^{1/p}.$$

还有 $L < /em > \infty$距离

$$d < em > \infty(x,y) = \max < /em > i = 1, \ldots, n |x_i - y_i|.$$

在实际应用中，距离函数的选择应该根据数据的特性和分析的需要而定，本篇就不进行更深入的探讨，一般情况下使用最常用的 L_2 函数即可。

但是！注意！使用 kNN 时需要根据特征数据的取值区间来调整坐标轴的比例，这个做法叫作标准化或者归一化。为什么要这么做呢？拿上面的例子来说，一只兔子的身长（cm）数值平均是它的体重（kg）的 10 倍左右，如果我们在这组数值上直接使用 L_2 距离函数的话就会导致横轴的距离比重明显放大，分类结果也不合理，如下图所示



如果把坐标轴成其他的单位，比如毫米和吨，并用相应的新数值来计算距离，又会得到完全不同的分类标准。甚至，在极端情况下，如果身高用纳米并且体重用吨计量，那么相比之下身高的数值会奇高无比，以至于两点之间的距离是完全由身高决定的，体重则没有任何权重。为了解决这个问题，我们应该在计算距离时把所有坐标轴进行归一化。

在之前的例子中，由于横轴数值大约是竖轴的 10 倍左右，所以我们将横轴（身高）的数值压缩 10 倍，即计算距离时使用

$$d((x_1,x_2),(y_1,y_2)) = \sqrt{\left(\frac{x_1}{10} - \frac{y_1}{10}\right)^2 + (x_2 - y_2)^2}$$

就可以得出合理的 kNN 分类。

一般来说，假设进行 kNN 分类使用的样本的特征是 $(x < em > i1, x < /em > i2, \ldots, x < em > in) < /em > i = 1^m$，取每一轴上的最大值减最小值

$$M < em > j = \max < /em > i = 1, \ldots, mx < em > ij - \min < /em > i = 1, \ldots, mx < em > ij,$$

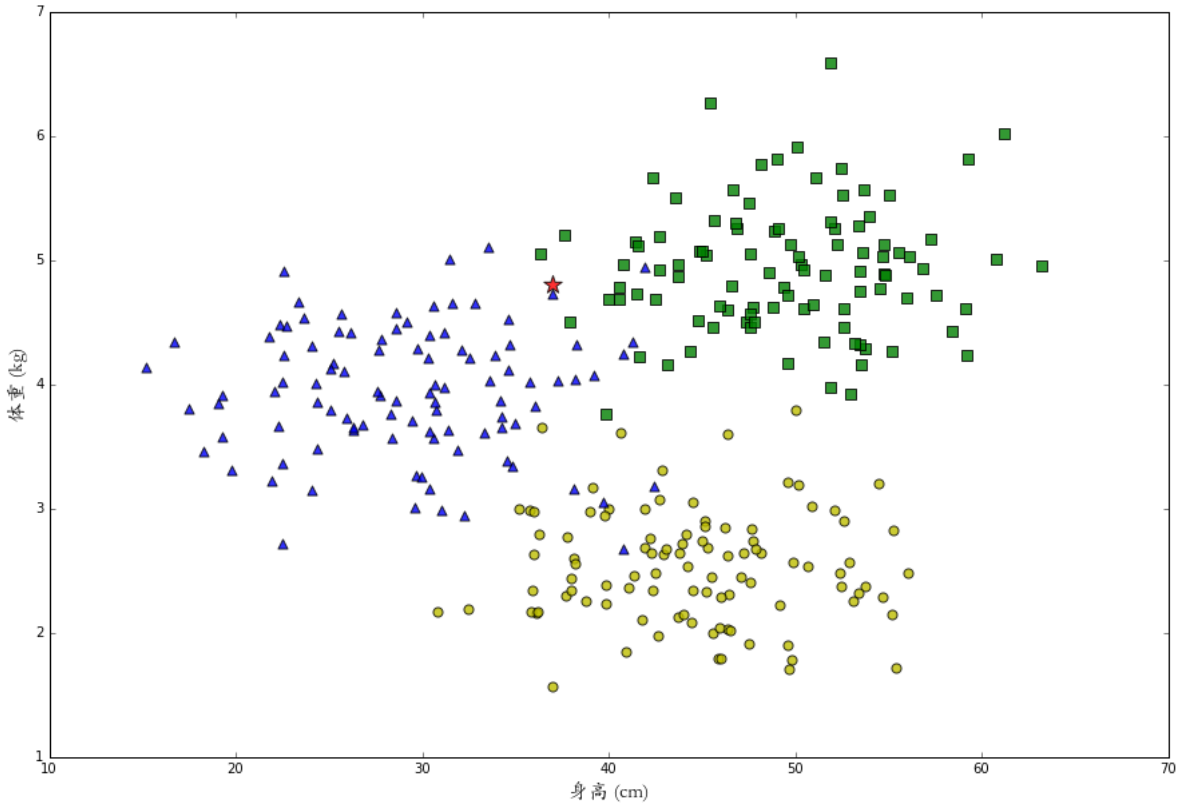
并且在计算距离时将每一个坐标轴除以相应的 M_j 以进行归一化，即

$$d((y_1,\ldots,y_n),(z_1,\ldots,z_n)) = \sqrt{\sum < /em > j = 1^n \left(\frac{y_j}{M_j} - \frac{z_j}{M_j}\right)^2}$$

便可以规避坐标轴比例失衡的问题。

概率 kNN

上面的kNN算法返回的是对一组特征的绝对分类，告诉我们这只兔子被判断为哪一个类别。可有时我们并不想知道一个确切地分类，而想知道它属于某个分类的概率是多大。比如我们发现一只身长 37 体重 4.8 的小兔兔，在下图五角星的位置。

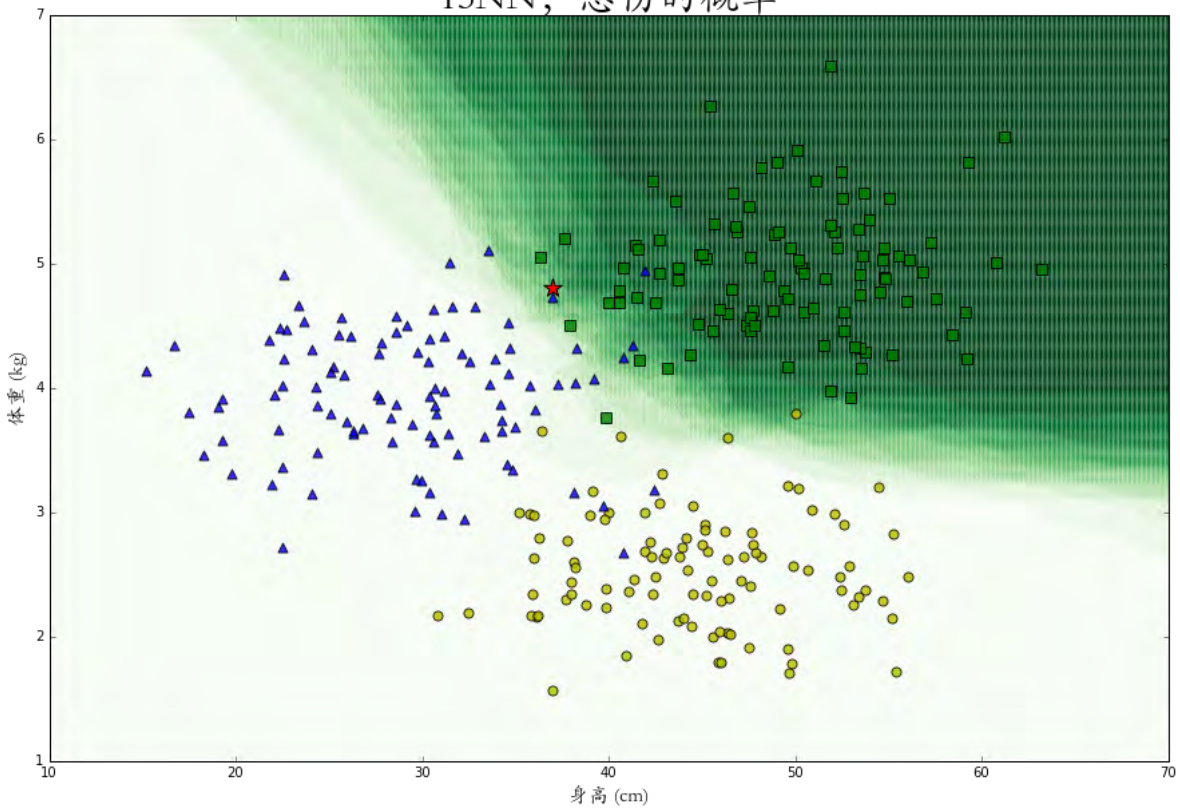


这只兔子的特征数据在悲伤和痛苦的分界处，机器不论判断它属于哪个类别都很有可能是错的。这时，类似“它有一半可能性是痛苦，一半可能性是悲伤”的反馈会更有意义。

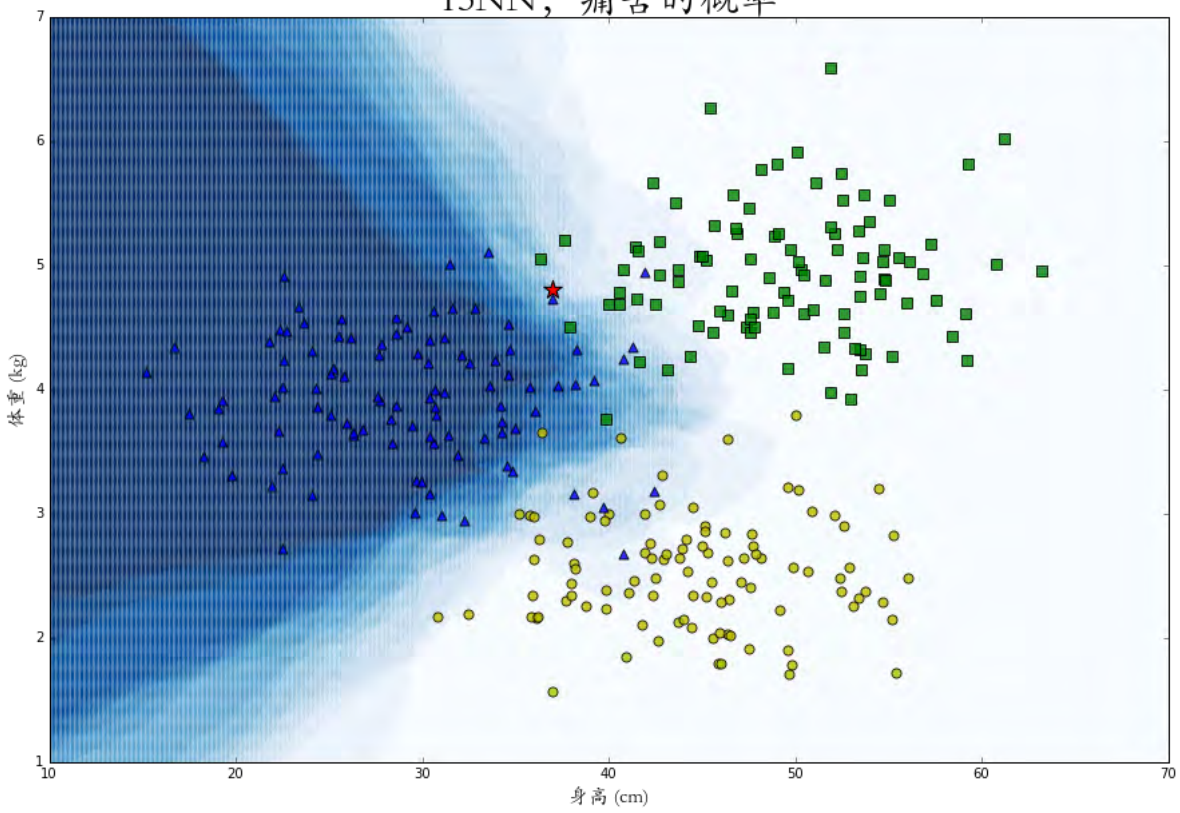
为了这个目的，我们同样找找出距离问题特征最近的 k 个样本，但与其寻找数量最多的分类，我们统计其中每个类别的分别有多少个，再除以 k 得到一个属于每一个类别概率值。比如在上面的图里，距离五角星最近的 15 个样本中，有 8 只悲伤和 7 只痛苦，由此判断：它有 53% 的可能性是悲伤，47% 的可能性是痛苦，0% 的可能性是绝望。

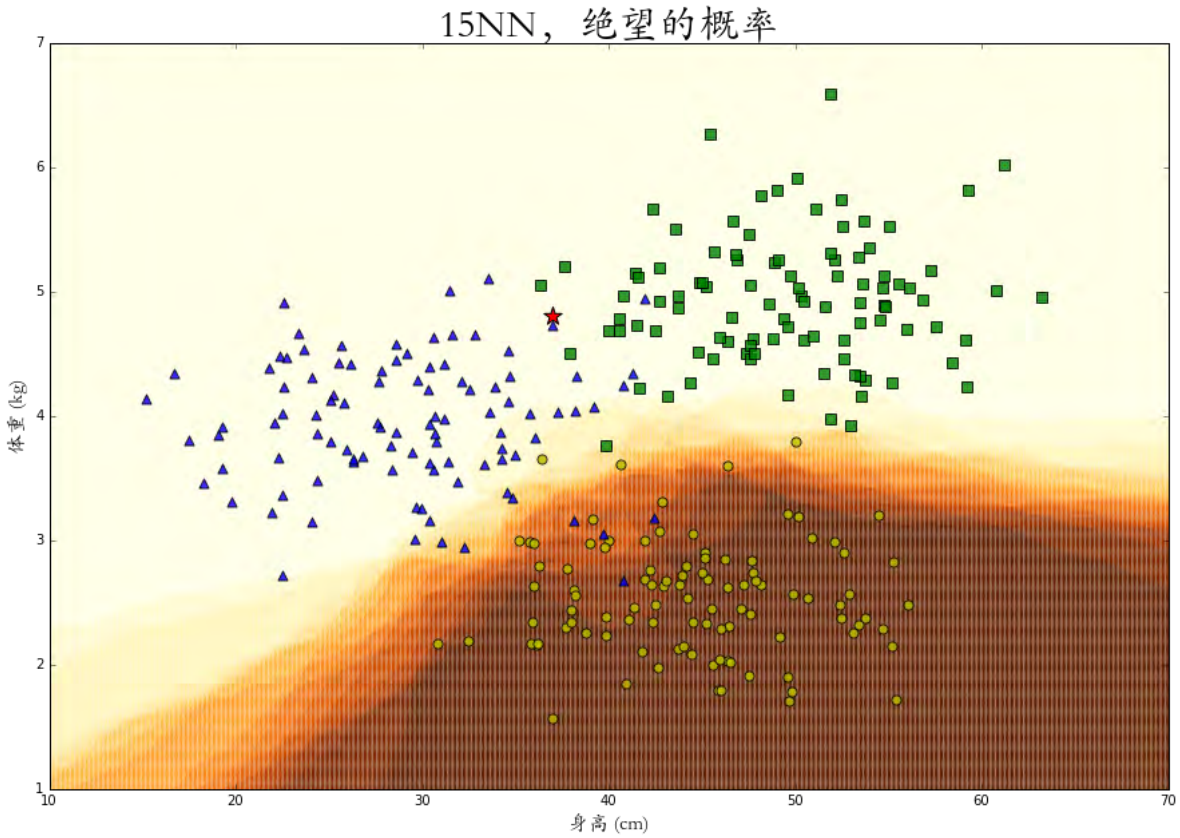
在整个二维空间中的每一个点上进行概率 kNN 算法，可以得到每个特征点是属于某个类别的概率热力图，图中颜色越深代表概率越大。

15NN，悲伤的概率

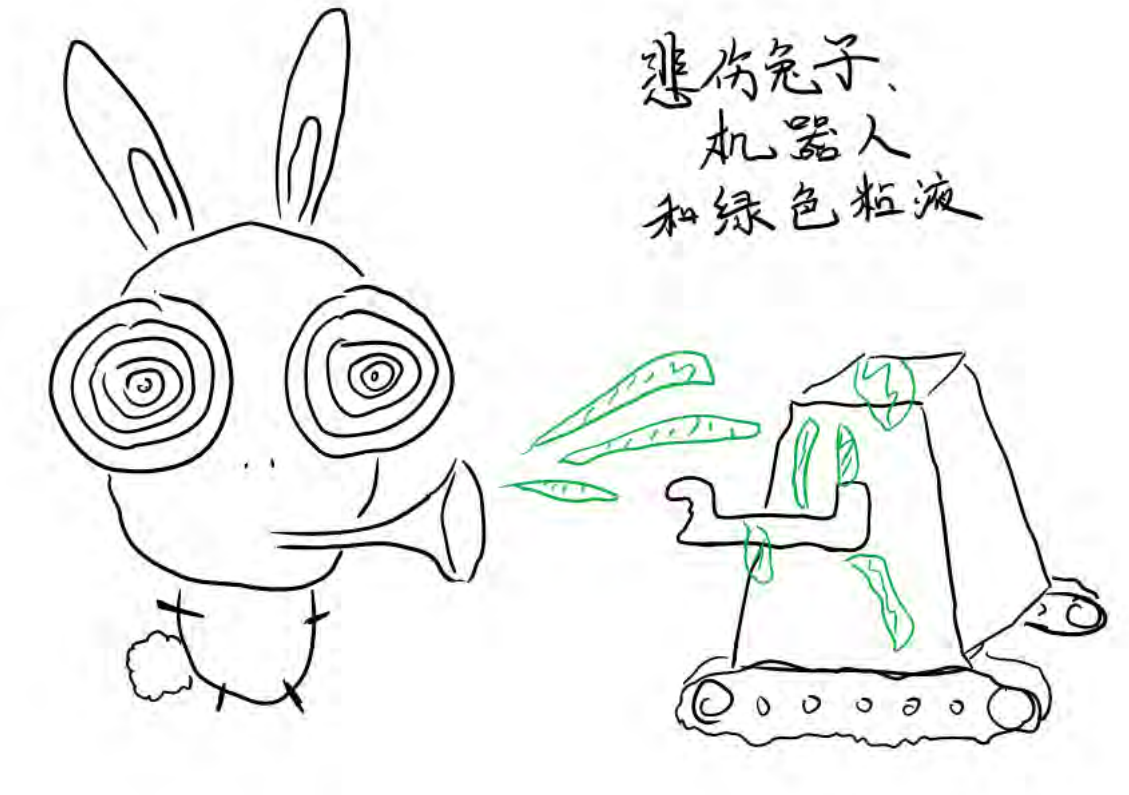


15NN，痛苦的概率





相比于绝对的分类，这些概率的计算会给我们更有效的表述以及更多的应用空间。比如说，我们知道悲伤兔子喜欢向我们的机器人上喷洒奇怪的粘液，毫无作用毫无意义的绿色的粘液，就像这样（作者手绘）：



倒不是因为别的，我们就是觉得这种粘液很恶心，清洗起来也很麻烦，所以我们想让机器人在测量并发现是悲伤之后马上掉头逃跑。但是如果机器发现了一只体型接近痛苦的悲伤，并且普通的 kNN 算法发生误判，没有马上逃跑，那么最后就会被喷了。所以我们使用概率 kNN 的算法并且使用以下风控原则：只要发现兔子有 30% 以上的概率是悲伤，就马上逃跑。从此之后，机器人就再也没被喷过。

结语

不知你有没有发现，我跟你讲了这么多关于兔子的事，却丝毫没有提及如何用代码计算 kNN。这是因为 kNN 虽然思路简单，但实现起来有一个问题，那就是计算量很大；当数据量很多时，拿一组特征来和所有样本依次计算距离并选取最近的 k 个，是非常耗费时间的。所以，在量化课堂接下来的文章中，我们将讲解 kNN 的一个高效算法—kd树。

文章迭代记录
v1.2，2016-09-14，修正公式，感谢 zhouzhizz16 指出
v1.1，2016-08-17，添加研究模块
v1.0，2016-08-16，文章上线

In [1]:

```
import random
import operator
import datetime
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

In [3]:

```
# 100个正态分布的悲伤
grief_heights = np.random.normal(50, 6, 100)
grief_weights = np.random.normal(5, 0.5, 100)

# 100个正态分布的痛苦
agony_heights = np.random.normal(30,6,100)
agony_weights = np.random.normal(4,0.5,100)

# 100个正态分布的绝望
despair_heights = np.random.normal(45,6,100)
despair_weights = np.random.normal(2.5, 0.5, 100)
```

In [5]:

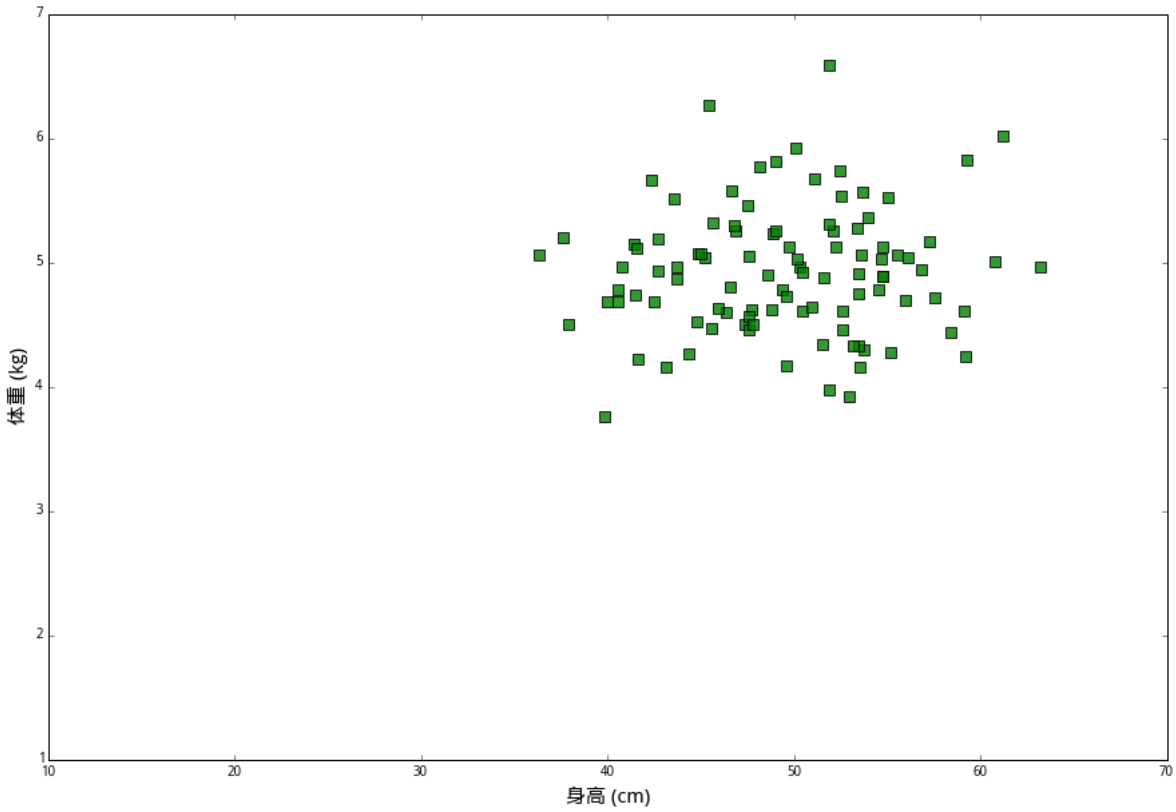
```
# 设置图片大小
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 10
```

In [10]:

```
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8)
plt.axis((10, 70,1,7))
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[10]:

```
<matplotlib.text.Text at 0x7f186a97b1d0>
```

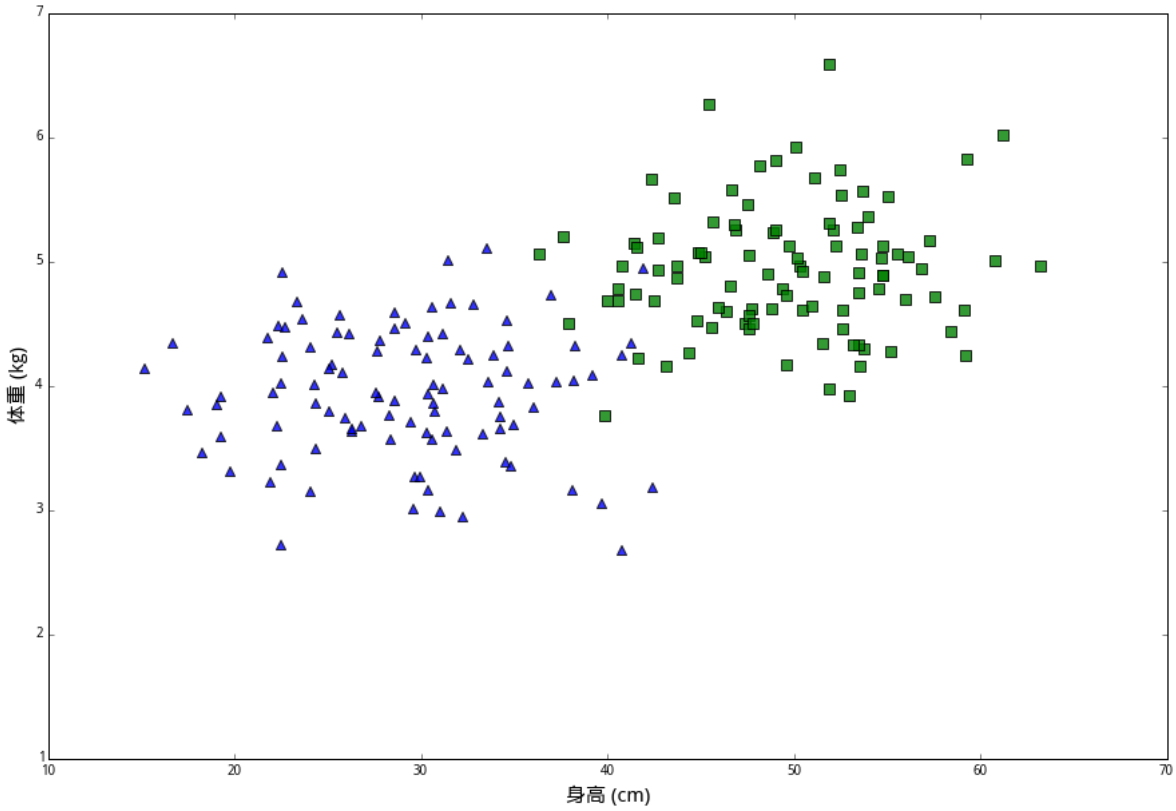


In [12]:

```
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8)
plt.axis((10, 70,1,7))
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[12]:

```
<matplotlib.text.Text at 0x7f186ac57b90>
```



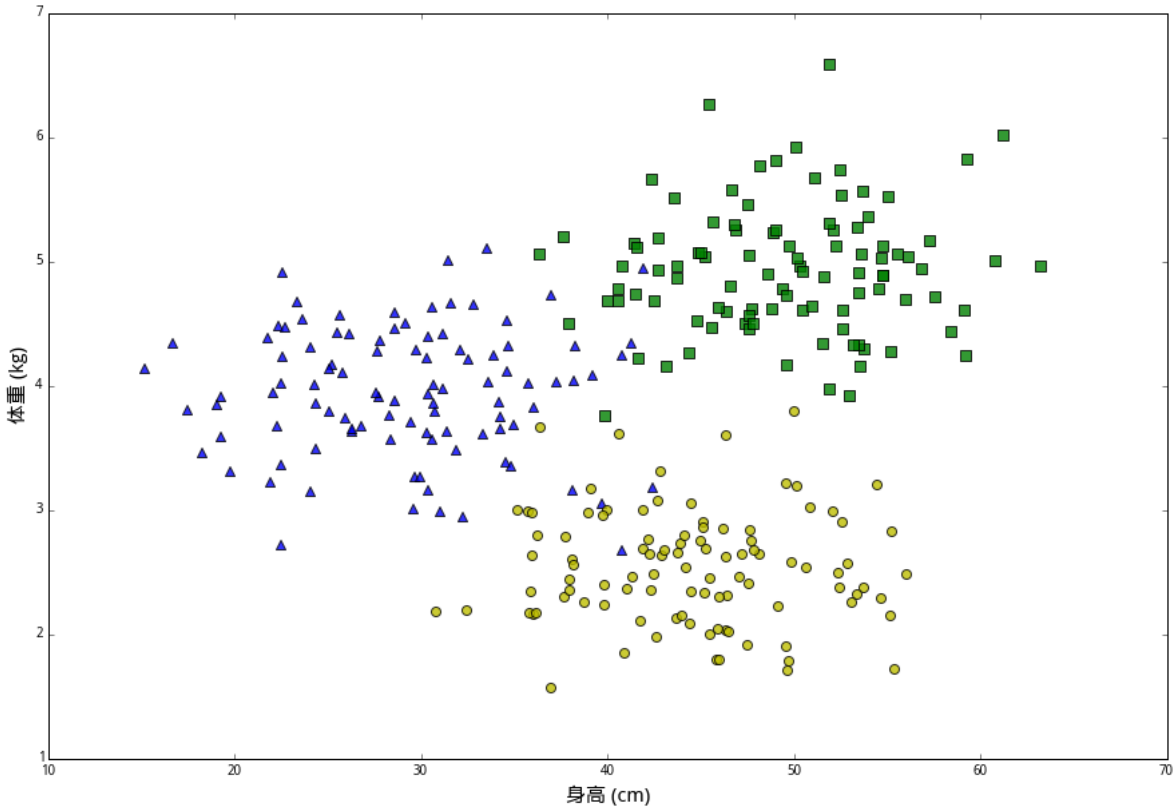
In [13]:

```
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8)
```

```
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[13]:

```
<matplotlib.text.Text at 0x7f186aa584d0>
```

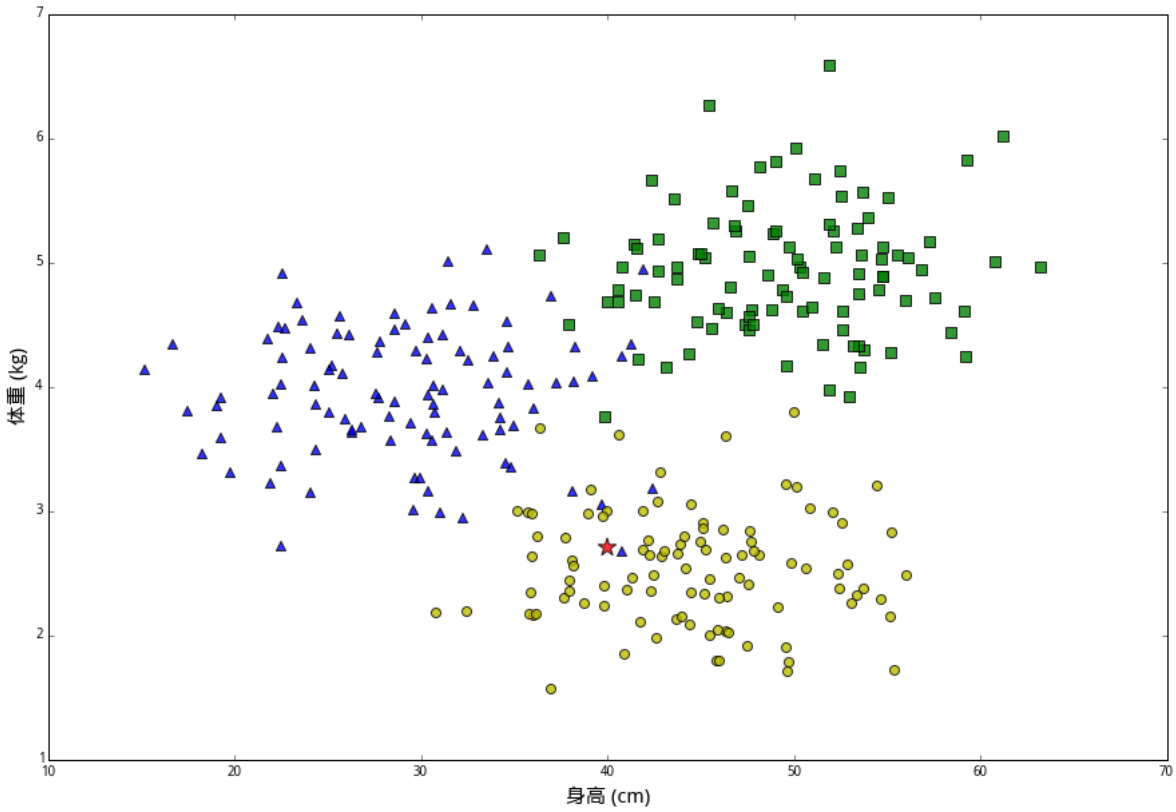


In [14]:

```
plt.scatter(40,2.7, c='r', s=200, marker='*',alpha=0.8)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[14]:

```
<matplotlib.text.Text at 0x7f186a9c5b50>
```



以下是kd树算法，可以直接拿去用。下一篇文章会进行详细的讲解。

In [29]:

```
class kdtree(object):

    # 创建 kdtree 喵
    # point_list 是一个 list 的 pair，pair[0] 是一 tuple 的特征，pair[1] 是类别
    def __init__(self, point_list, depth=0, root=None):

        if len(point_list)>0:

            # 轮换按照树深度选择坐标轴
            k = len(point_list[0][0])
            axis = depth % k

            # 选中位线，切
            point_list.sort(key=lambda x:x[0][axis])
            median = len(point_list) // 2

            self.axis = axis
            self.root = root
            self.size = len(point_list)

            # 造节点
            self.node = point_list[median]
            # 递归造左枝和右枝
            if len(point_list[:median])>0:
                self.left = kdtree(point_list[:median], depth+1, self)
            else:
                self.left = None
            if len(point_list[median+1:])>0:
                self.right = kdtree(point_list[median+1:], depth+1, self)
            else:
                self.right = None
            # 记录是按哪个方向切的还有树根

        else:
            return None

    # 在树上加一点
    def insert(self, point):
        self.size += 1

        # 分析是左还是右，递归加在叶子上
        if point[0][self.axis]<self.node[0][self.axis]:
            if self.left!=None:
                self.left.insert(point)
            else:
```

```
            self.left = kdtree([point], self.axis+1, self)
    else:
        if self.right!=None:
            self.right.insert(point)
        else:
            self.right = kdtree([point], self.axis+1, self)


# 输入一点
# 按切分寻找叶子
def find_leaf(self, point):
    if self.left==None and self.right==None:
        return self
    elif self.left==None:
        return self.right.find_leaf(point)
    elif self.right==None:
        return self.left.find_leaf(point)
    elif point[self.axis]<self.node[0][self.axis]:
        return self.left.find_leaf(point)
    else:
        return self.right.find_leaf(point)


# 查找最近的 k 个点，复杂度 O(DlogN)，D是维度，N是树的大小
# 输入一点、一距离函数、一k。距离函数默认是 L_2
def knearest(self, point, k=1, dist=lambda x,y: sum(map(lambda u,v:(u-v)**2,x,y))):
    # 往下戳到最底叶
    leaf = self.find_leaf(point)
    # 从叶子网上爬
    return leaf.k_down_up(point, k, dist, result=[], stop=self, visited=None)


# 从下往上爬函数，stop是到哪里去，visited是从哪里来
def k_down_up(self, point,k, dist, result=[],stop=None, visited=None):

    # 选最长距离
    if result==[]:
        max_dist = 0
    else:
        max_dist = max([x[1] for x in result])

    other_result=[]

    # 如果离分界线的距离小于现有最大距离，或者数据点不够，就从另一边的树根开始刨
    if (self.left==visited and self.node[0][self.axis]-point[self.axis]<max_dist and self.right!=None)\
        or (len(result)<k and self.left==visited and self.right!=None):
        other_result=self.right.knearest(point,k, dist)

    if (self.right==visited and point[self.axis]-self.node[0][self.axis]<max_dist and self.left!=None)\
        or (len(result)<k and self.right==visited and self.left!=None):
        other_result=self.left.knearest(point, k, dist)

    # 刨出来的点放一起，选前 k 个
    result.append((self.node, dist(point, self.node[0])))
    result = sorted(result+other_result, key=lambda pair: pair[1])[:k]

    # 到停点就返回结果
    if self==stop:
        return result
    # 没有就带着现有结果接着往上爬
    else:
        return self.root.k_down_up(point,k,  dist, result, stop, self)

# 输入 特征、类别、k、距离函数
# 返回这个点属于该类别的概率
def kNN_prob(self, point, label, k, dist=lambda x,y: sum(map(lambda u,v:(u-v)**2,x,y))):
    nearests = self.knearest(point,  k, dist)
    return float(len([pair for pair in nearests if pair[0][1]==label]))/float(len(nearests))


# 输入 特征、k、距离函数
# 返回该点概率最大的类别以及相对应的概率
def kNN(self, point, k, dist=lambda x,y: sum(map(lambda u,v:(u-v)**2,x,y))):
    nearests = self.knearest(point, k , dist)

    statistics = {}
    for data in nearests:
        label = data[0][1]
        if label not in statistics:
            statistics[label] = 1
        else:
            statistics[label] += 1
```

```
        max_label = max(statistics.iteritems(), key=operator.itemgetter(1))[0]
        return max_label, float(statistics[max_label])/float(len(nearests))
```

In [36]:

```
# 设置样本集
grieves = map(lambda x,y:tuple(((x,y),'g')),grief_heights, grief_weights)
agonies = map(lambda u,v:tuple(((u,v),'b')),agony_heights, agony_weights)
despairs = map(lambda a,b:tuple(((a,b),'y')),despair_heights, despair_weights)
# 创建kd树
tree = kdtree(list(concatenate((grieves,agonies,despairs))))
```

In [37]:

```
# 穷举生成空间上的点
all_points = []
for i in range(100,701):
    for j in range(100,701):
        all_points.append((float(i)/10., float(j)/100.))
```

In [38]:

```
# 一共是36万个点
len(all_points)
```

Out[38]:

```
361201
```

In [44]:

```
# 设置归一化距离函数
def normalized_dist(x,y):
    return (x[0]-y[0])**2+(10*x[1]-10*y[1])**2
```

In [45]:

```
# 每个点运算 15NN，并记录计算时间
now = datetime.datetime.now()
fifteen_NN_result = []
for point in all_points:
    fifteen_NN_result.append((point, tree.kNN(point,k=15, dist=normalized_dist)[0]))
print datetime.datetime.now() - now
```

```
0:22:04.439600
```

In [46]:

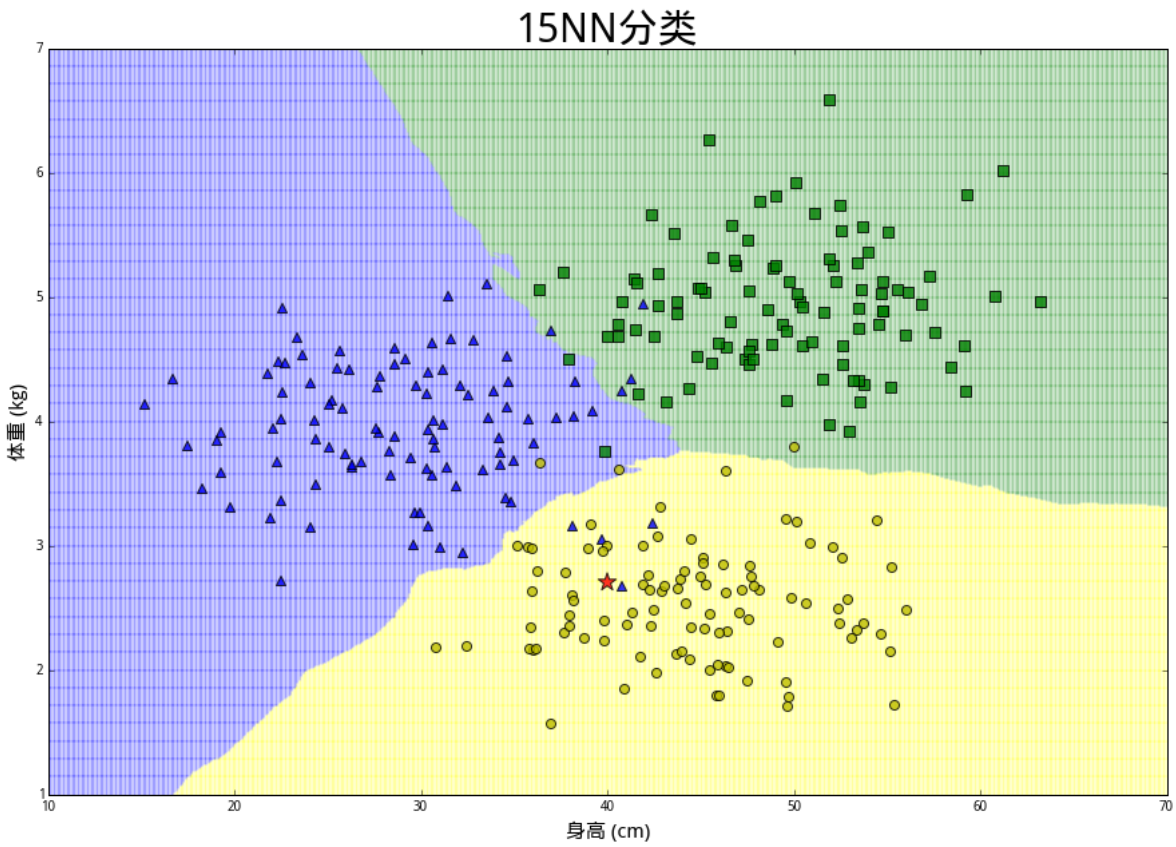
```
# 把每个颜色的数据分开
fifteen_NN_yellow = []
fifteen_NN_green = []
fifteen_NN_blue = []

for pair in fifteen_NN_result:
    if pair[1]=='y':
        fifteen_NN_yellow.append(pair[0])
    if pair[1]=='g':
        fifteen_NN_green.append(pair[0])
    if pair[1]=='b':
        fifteen_NN_blue.append(pair[0])
```

In [60]:

```
plt.scatter(40,2.7, c='r', s=200, marker='*',alpha=0.8, zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8,zorder=10)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8,zorder=10)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8,zorder=10)
plt.scatter([x[0] for x in fifteen_NN_yellow], [x[1] for x in fifteen_NN_yellow], s=1, c='yellow', marker='1', alpha=0.3)
plt.scatter([x[0] for x in fifteen_NN_blue], [x[1] for x in fifteen_NN_blue], s=1, c='blue', marker='2', alpha=0.3)
plt.scatter([x[0] for x in fifteen_NN_green], [x[1] for x in fifteen_NN_green], s=1, c='green', marker='3', alpha=0.3)
plt.axis((10, 70,1,7))
plt.title('15NN分类',size=30)
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[60]:

```
<matplotlib.text.Text at 0x7f185e12ab90>
```



In [39]:

```
# 每个点运算 1NN，并记录计算时间
now = datetime.datetime.now()
one_NN_result = []
for point in all_points:
    one_NN_result.append((point, tree.kNN(point,k=1, dist=lambda x,y: (x[0]-y[0])**2+(10*x[1]-10*y[1])**2)[0]))
print datetime.datetime.now() - now
```

```
0:13:07.874846
```

In [41]:

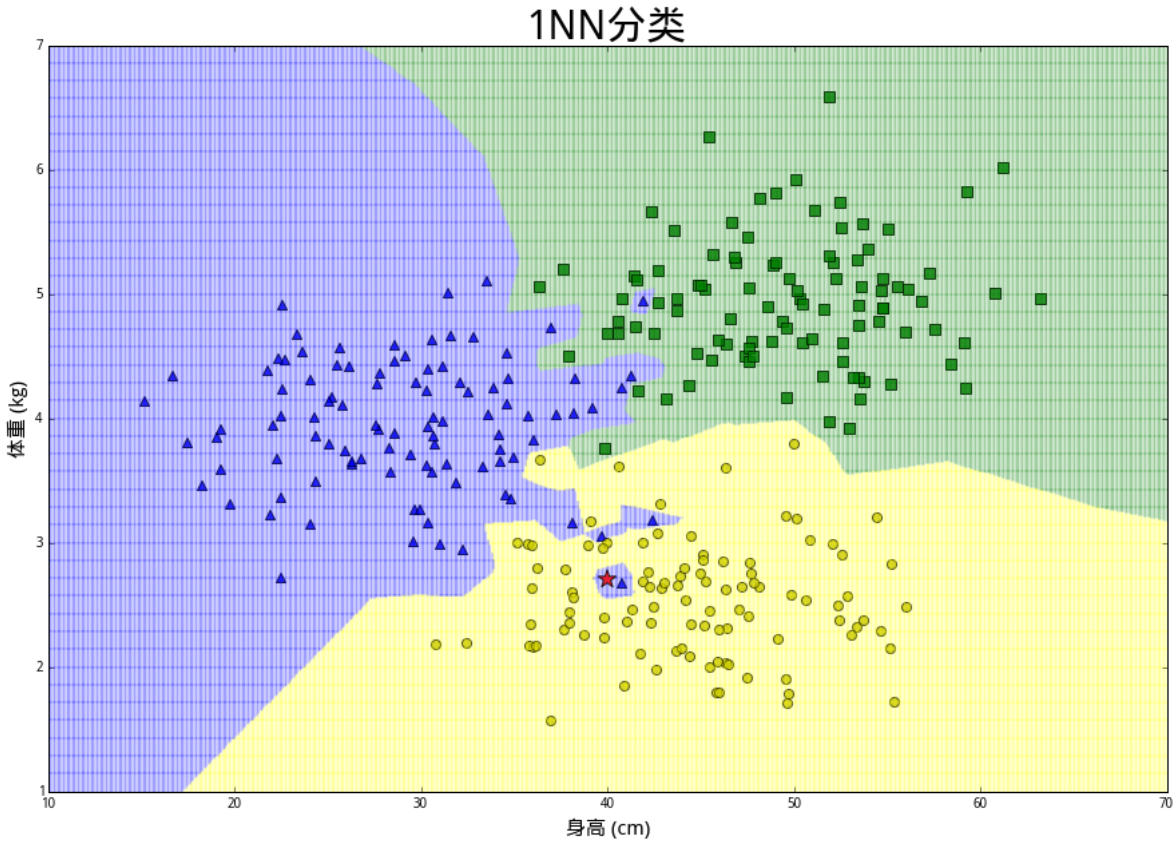
```
# 把每个颜色的数据分开
one_NN_yellow = []
one_NN_green = []
one_NN_blue = []

for pair in one_NN_result:
    if pair[1]=='y':
        one_NN_yellow.append(pair[0])
    if pair[1]=='g':
        one_NN_green.append(pair[0])
    if pair[1]=='b':
        one_NN_blue.append(pair[0])
```

In [43]:

```
plt.scatter(40,2.7, c='r', s=200, marker='*',alpha=0.8, zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8,zorder=10)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8,zorder=10)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8,zorder=10)
plt.scatter([x[0] for x in one_NN_yellow], [x[1] for x in one_NN_yellow], s=1, c='yellow', marker='1', alpha=0.3)
plt.scatter([x[0] for x in one_NN_blue], [x[1] for x in one_NN_blue], s=1, c='blue', marker='2', alpha=0.3)
plt.scatter([x[0] for x in one_NN_green], [x[1] for x in one_NN_green], s=1, c='green', marker='3', alpha=0.3)
plt.axis((10, 70,1,7))
plt.title('1NN分类',size=30)
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[43]:

```
<matplotlib.text.Text at 0x7f18672a3150>
```



In [52]:

```
# 使用未归一化的距离计算 15NN
now = datetime.datetime.now()
bad_NN_result = []
for point in all_points:
    bad_NN_result.append((point, tree.kNN(point,k=15)[0]))
print datetime.datetime.now()-now
```

```
0:44:43.191188
```

In [54]:

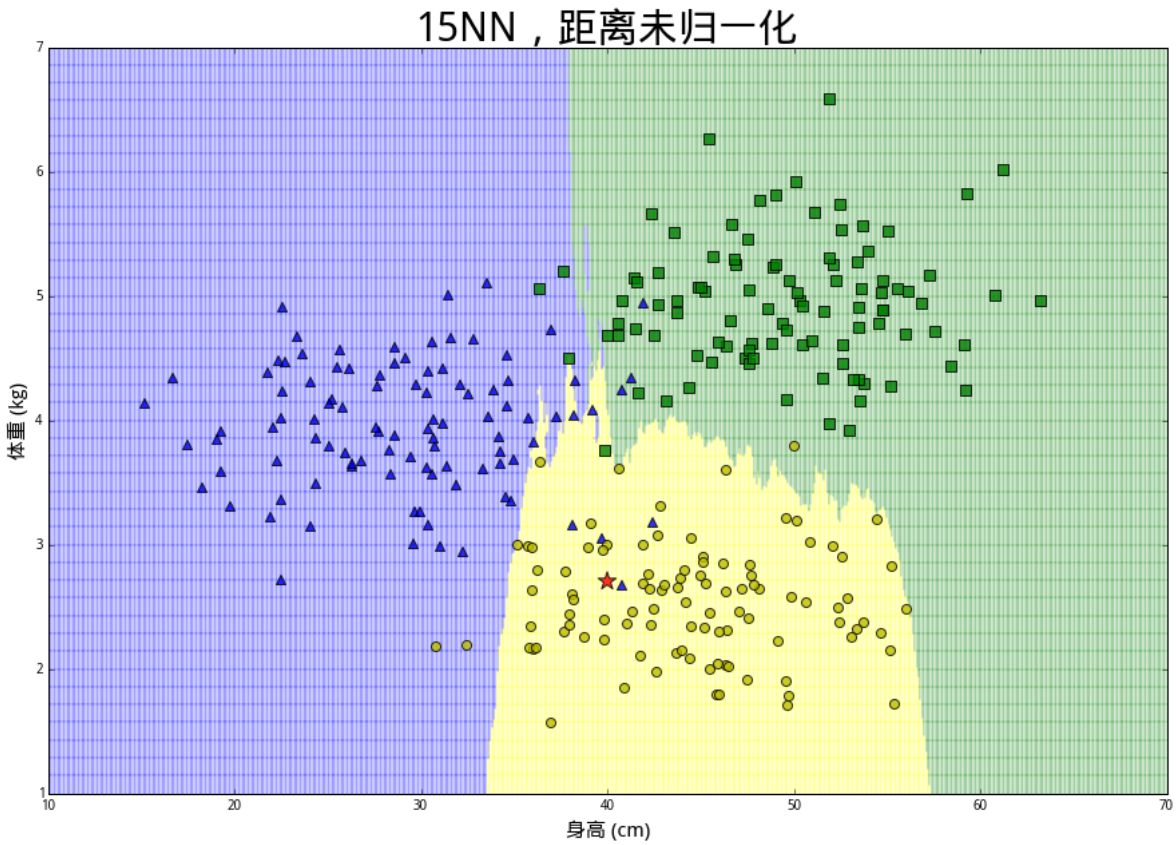
```
# 把每个颜色的数据分开
bad_NN_yellow = []
bad_NN_green = []
bad_NN_blue = []

for pair in bad_NN_result:
    if pair[1]=='y':
        bad_NN_yellow.append(pair[0])
    if pair[1]=='g':
        bad_NN_green.append(pair[0])
    if pair[1]=='b':
        bad_NN_blue.append(pair[0])
```

In [61]:

```
plt.scatter(40,2.7, c='r', s=200, marker='*',alpha=0.8, zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8,zorder=10)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8,zorder=10)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8,zorder=10)
plt.scatter([x[0] for x in bad_NN_yellow], [x[1] for x in bad_NN_yellow], s=1, c='yellow', marker='1', alpha=0.3)
plt.scatter([x[0] for x in bad_NN_blue], [x[1] for x in bad_NN_blue], s=1, c='blue', marker='2', alpha=0.3)
plt.scatter([x[0] for x in bad_NN_green], [x[1] for x in bad_NN_green], s=1, c='green', marker='3', alpha=0.3)
plt.axis((10, 70,1,7))
plt.title('15NN，距离未归一化',size=30)
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[61]:

```
<matplotlib.text.Text at 0x7f186573c650>
```

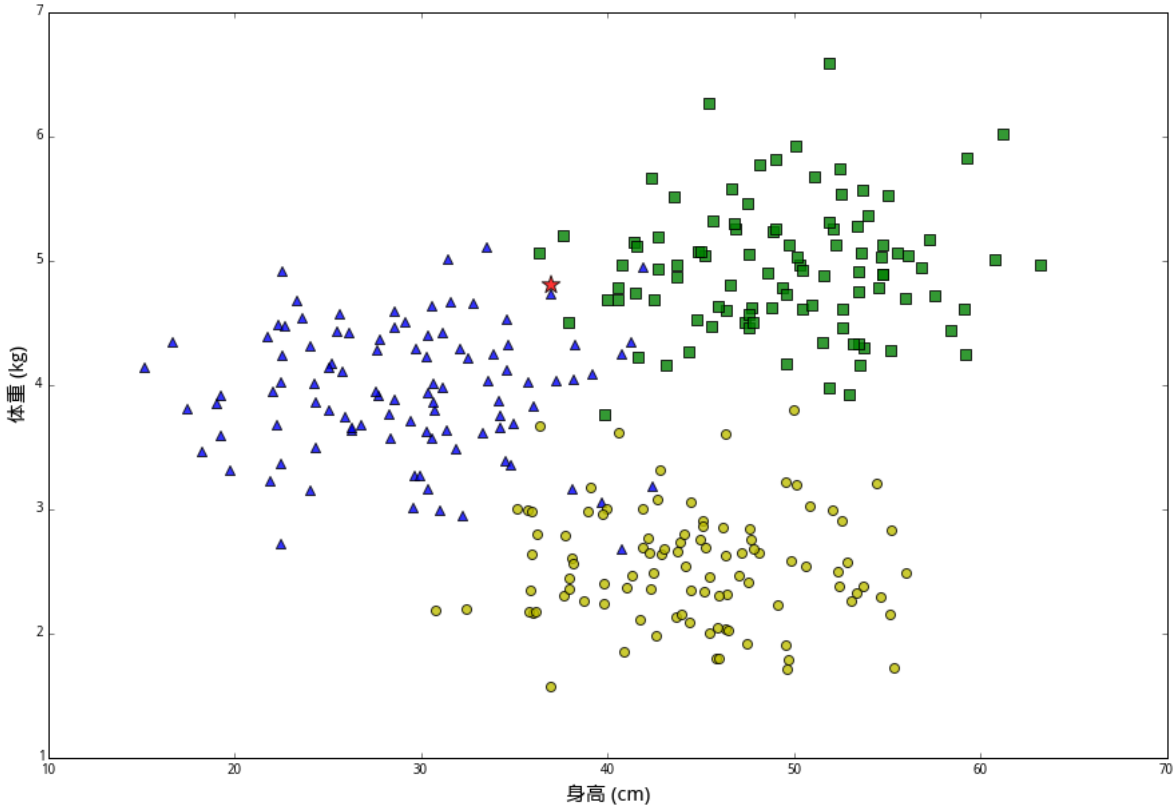


In [56]:

```
plt.scatter(37,4.8, c='r', s=200, marker='*',alpha=0.8,zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8)
plt.axis((10, 70,1,7))
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[56]:

```
<matplotlib.text.Text at 0x7f185fb7f3d0>
```



In [59]:

```
# 15NN 计算是悲伤的概率
now = datetime.datetime.now()
grief_prob = []
for point in all_points:
    grief_prob.append((point, tree.kNN_prob(point, label='g', k=15, dist=normalized_dist)))
print datetime.datetime.now() - now
```

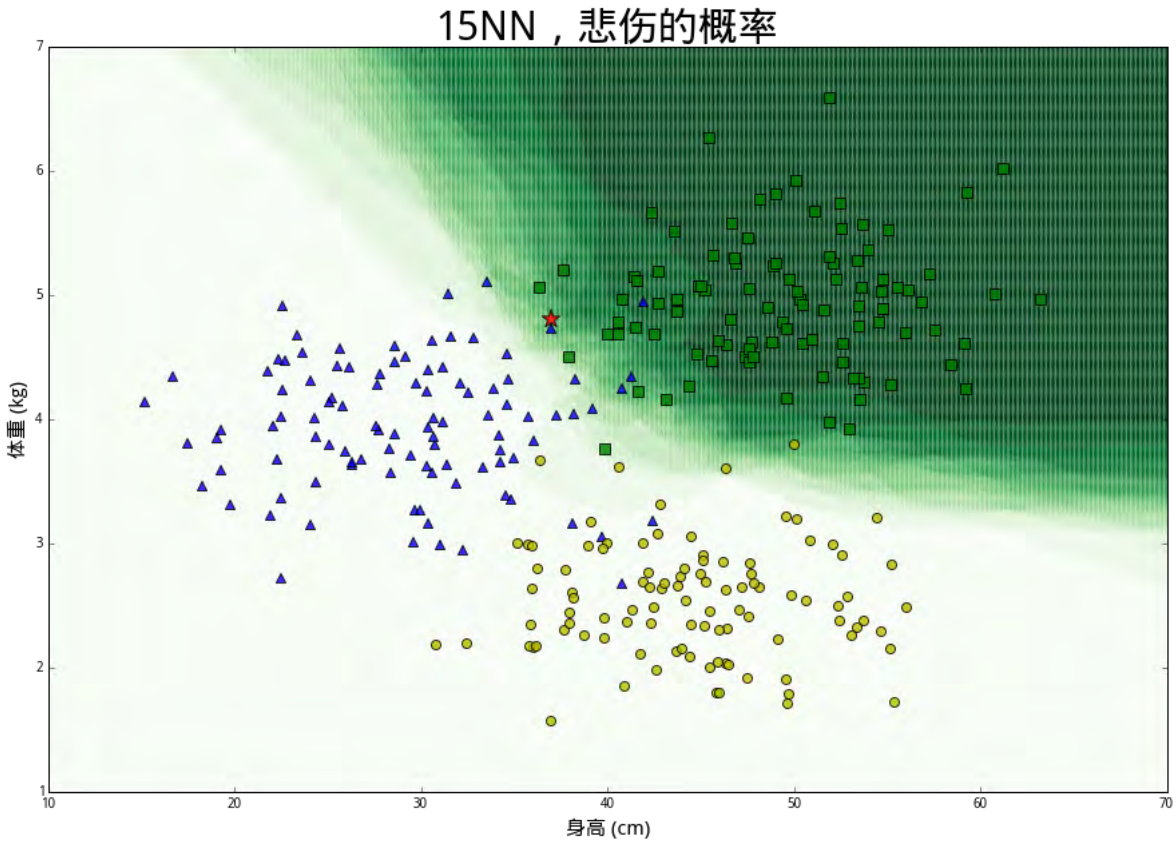
```
0:21:57.029318
```

In [67]:

```
plt.scatter(37,4.8, c='r', s=200, marker='*',alpha=0.8,zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8,zorder=10)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8,zorder=10)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8,zorder=10)
plt.scatter([x[0][0] for x in grief_prob], [x[0][1] for x in grief_prob], s=1, c=[x[1] for x in grief_prob], marker='1',cmap='Greens'
plt.axis((10, 70,1,7))
plt.title('15NN，悲伤的概率',size=30)
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[67]:

```
<matplotlib.text.Text at 0x7f185c7aed50>
```



In [66]:

```
# 15NN 计算是痛苦的概率
now = datetime.datetime.now()
agony_prob = []
for point in all_points:
    agony_prob.append((point, tree.kNN_prob(point, label='b', k=15, dist=normalized_dist)))
print datetime.datetime.now() - now
```

```
0:21:23.089519
```

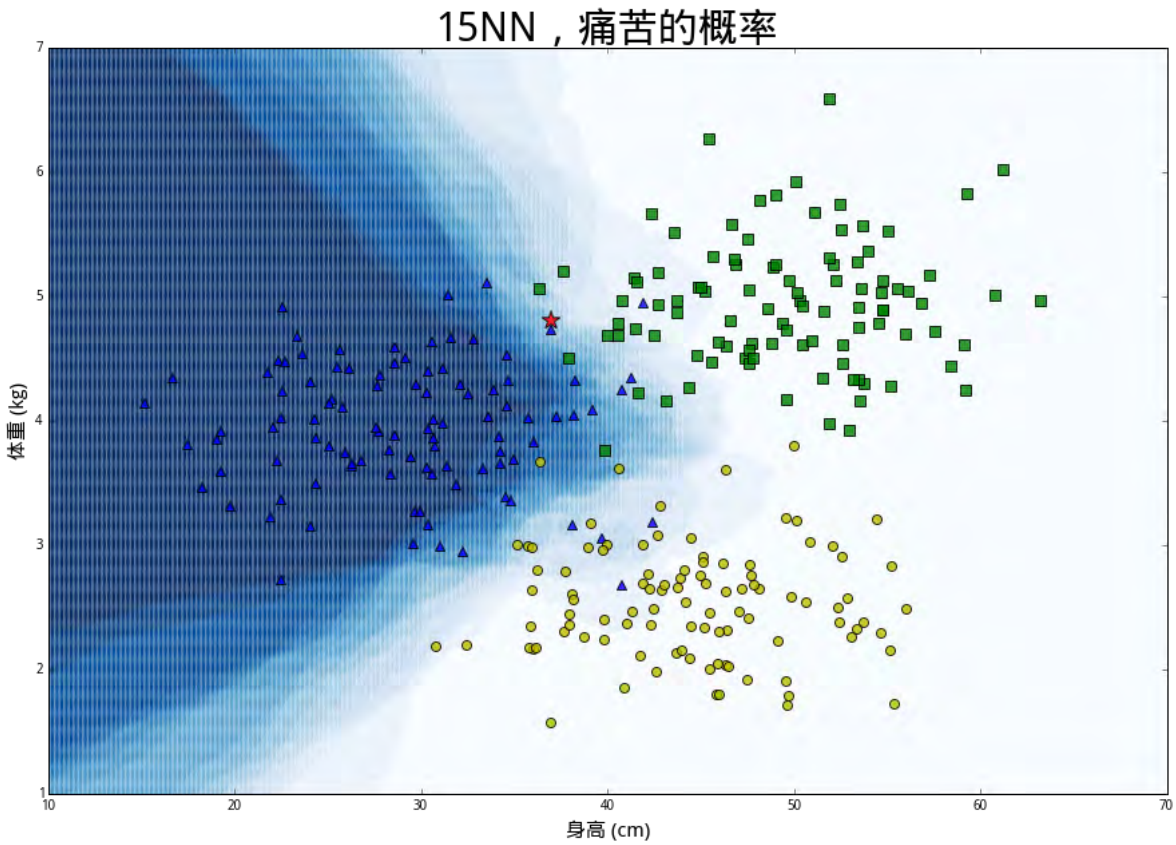
In [73]:

```
plt.scatter(37,4.8, c='r', s=200, marker='*',alpha=0.8,zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8,zorder=10)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8,zorder=10)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8,zorder=10)
plt.scatter([x[0][0] for x in agony_prob], [x[0][1] for x in agony_prob], s=1, c=[x[1] for x in agony_prob], marker='1',cmap='Blues')
plt.axis((10, 70,1,7))
```

```
plt.title('15NN，痛苦的概率',size=30)
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[73]:

```
<matplotlib.text.Text at 0x7f184f545490>
```



In [69]:

```
# 15NN 计算是绝望的概率
now = datetime.datetime.now()
despair_prob = []
for point in all_points:
    despair_prob.append((point, tree.kNN_prob(point, label='y', k=15, dist=normalized_dist)))
print datetime.datetime.now() - now
```

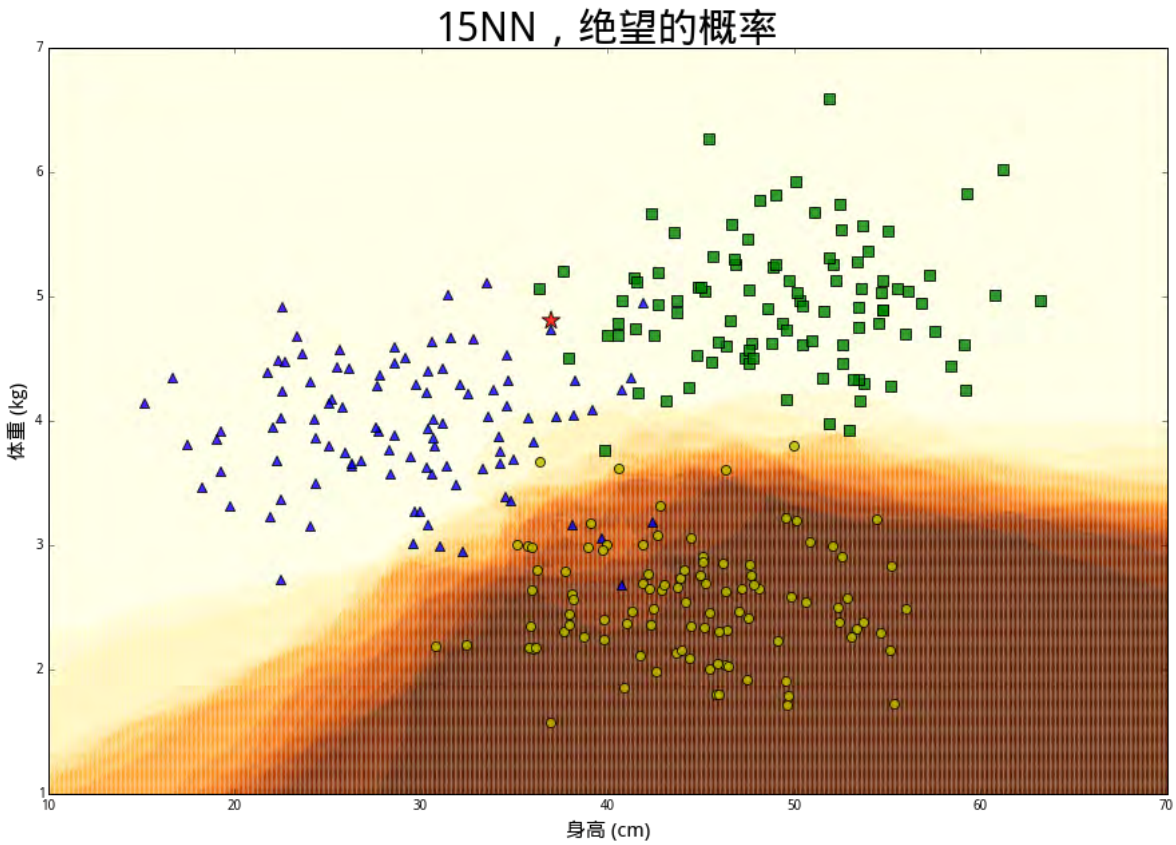
```
0:22:11.185548
```

In [74]:

```
plt.scatter(37,4.8, c='r', s=200, marker='*',alpha=0.8,zorder=10)
plt.scatter(grief_heights,grief_weights,c='g',marker='s',s=50,alpha=0.8,zorder=10)
plt.scatter(agony_heights,agony_weights,c='b',marker='^',s=50,alpha=0.8,zorder=10)
plt.scatter(despair_heights,despair_weights, c='y', s=50, alpha=0.8,zorder=10)
plt.scatter([x[0][0] for x in despair_prob], [x[0][1] for x in despair_prob], s=1, c=[x[1] for x in despair_prob], marker='1',cmap='Y
plt.axis((10, 70,1,7))
plt.title('15NN，绝望的概率',size=30)
plt.xlabel('身高 (cm)',size=15)
plt.ylabel('体重 (kg)', size=15)
```

Out[74]:

```
<matplotlib.text.Text at 0x7f184dbd4750>
```



【量化课堂】kd 树算法之思路篇

导语：kd 树是一种二叉树数据结构，可以用来进行高效的 kNN 计算。kd 树算法偏于复杂，本篇将先介绍以二叉树的形式来记录和索引空间的思路，以便读者更轻松地理解 kd 树。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂退出，本文的难度属于进阶（上），深度为level-1。

阅读本文之前请掌握 kNN（level-1）的知识。

前言

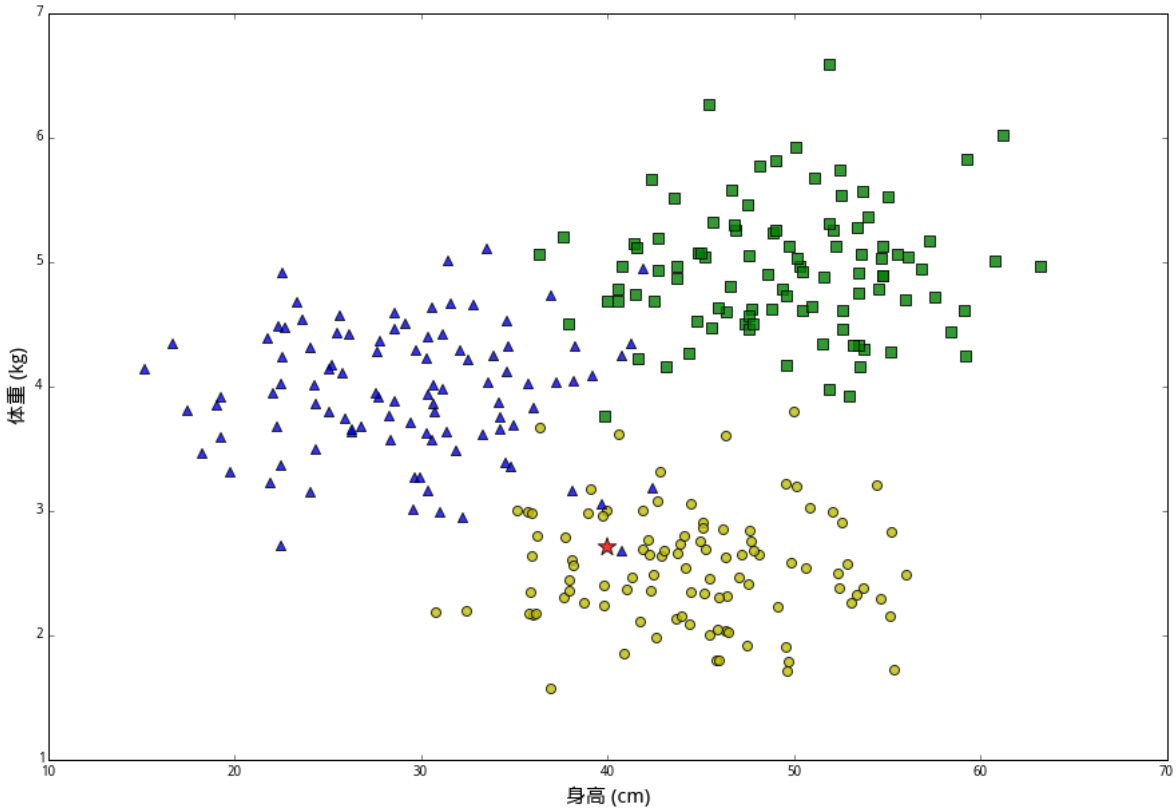
kd 树（k-dimensional tree）是一个包含空间信息的二项树数据结构，它是用来计算 kNN 的一个非常常用的工具。如果特征的维度是 D，样本的数量是 N，那么一般来讲 kd 树算法的复杂度是 $O(D \log N)$，相比于穷算的 $O(DN)$ 省去了非常多的计算量。

因为 kd 树的概念和算法较为复杂，固将本教程分为“思路篇”和“详细篇”。两篇的内容在一定程度上是重叠的，但是本篇注重于讲解 kd 树背后的思想和直觉，告诉读者一颗二项树是如何承载空间概念的，我们又该如何从树中读取这些信息；而之后的详细篇则详细讲解 kd 树的定义，如何构造它并且如何计算 kNN。出于教学起见，本文讲的例子和算法与严格的 kd 树有一些差异。有算法经验或者想尝试挑战的读者可以直接跳过本篇去读详细篇。

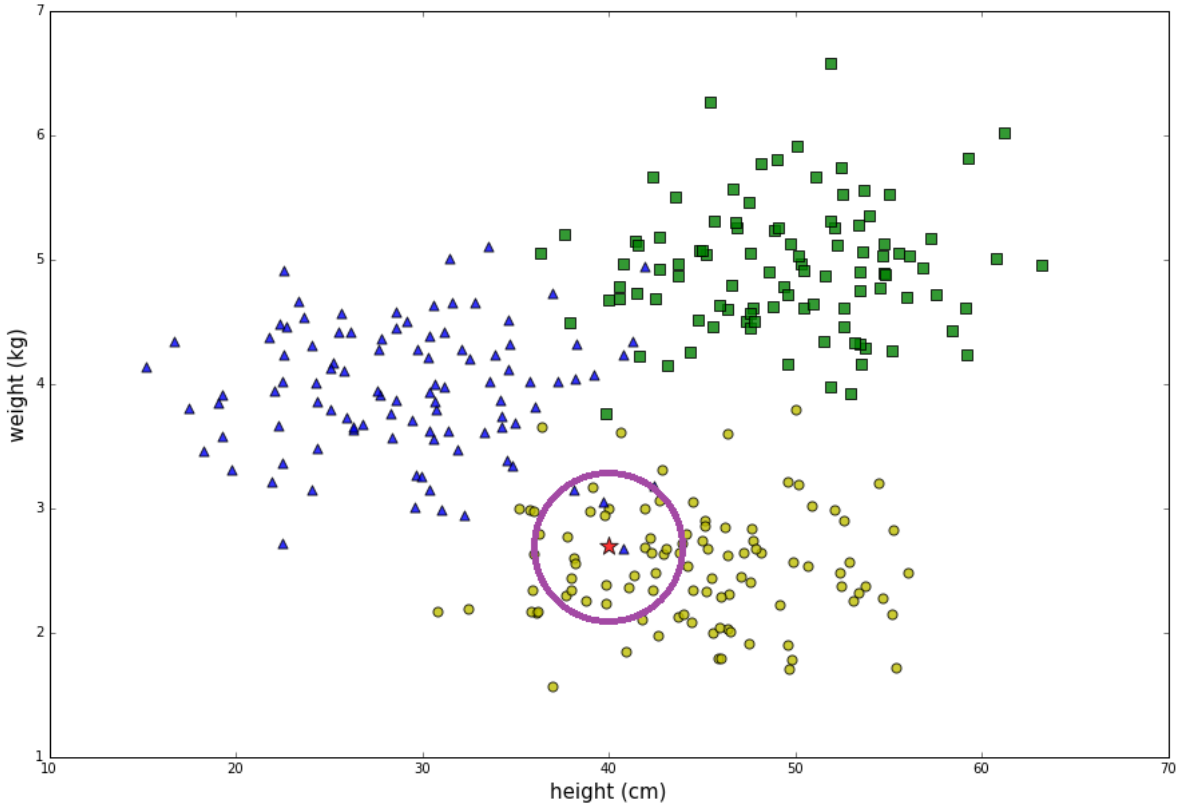
关于在学习编程和算法时有没有必要自己制作轮子的问题，一直存在着很多的争议。作者认为，做不做轮子暂且不论，但是有必要去了解轮子是怎么做出来的。Python 的 scikit-learn 机器学习包提供了蛮算、kd 树和 ball 树三种 kNN 算法，学完本篇的读者若无兴趣自撰算法，可以非常轻松地使用该包，详细可见 scikit-learn 之 kNN 分类。

直觉

给定一堆已有的样本数据，和一个被询问的数据点（红色五角星），我们如何找到离五角星最近的15个点？



先忽略在编程上的实现，想一想一个人如何主观地执行。嗯，他一定会把在五角附近的一些点中，分别计算每一个的距离，然后选最近的15个。这样可能只需要进行二三十次距离计算，而不是300次。



如图，只对紫圈里的点进行计算。

啊哈！问题来了。我们讲到的“附近”已经包含了距离的概念，如果不经过计算我们怎么知道哪个点是在五角星的“附近”？为什么我们一下就认出了“附近”而计算机做不到？那是因为我们在观看这张图片时，得到的输入是已经带有距离概念的影像，然而计算机在进行计算时得到的则是没有距离概念的坐标数据。如果要让一个人人为地从300组坐标里选出最近的15个，而不给他图像，那么他也省不了功夫，必须要把300个全部计算一遍才行。

这样来说，我们要做的就是在干巴巴的坐标数据上进行加工，将空间分割成小块，并以合理地方法将信息进行储存，这样方便我们读取“附近”的点。

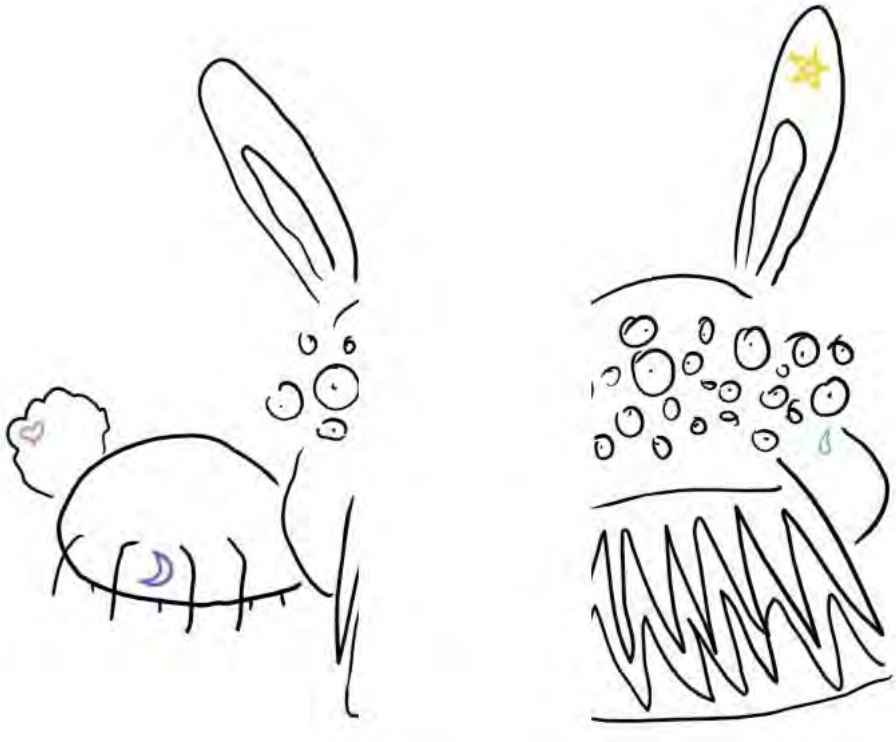
切割

这只危险的兔子，它又回来了！它今天上了四个纹身，爱心、月牙、星星和眼泪，下面是它的照片。

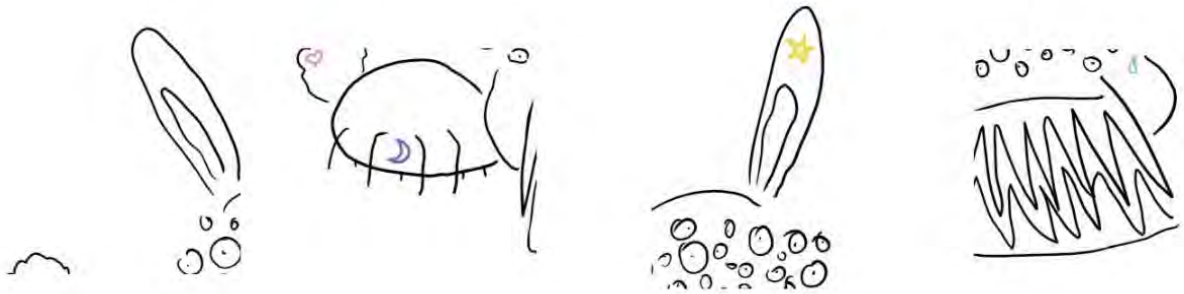
我们来回答一个简单的问题：在这幅照片上，距离爱心最近的纹身是什么？记得上一篇文章中，我们选用的特征是每一只兔子的身高和体重；这次就不一样了，在这个问题中，每个纹身的特征是照片平面上的横轴和竖轴的坐标。

对于这个问题，如果进行蛮算的办法我们需要计算 3 次距离（分别和月亮、眼泪和星星算一次）。下面我们要做的是把整个空间按照左右和上下进行等分，并且把分割后的小空间以二叉树形式进行记录，这样可以很快地读取邻近的点而省去计算量。

好，我们先竖向沿中间把这个兔子切成两半



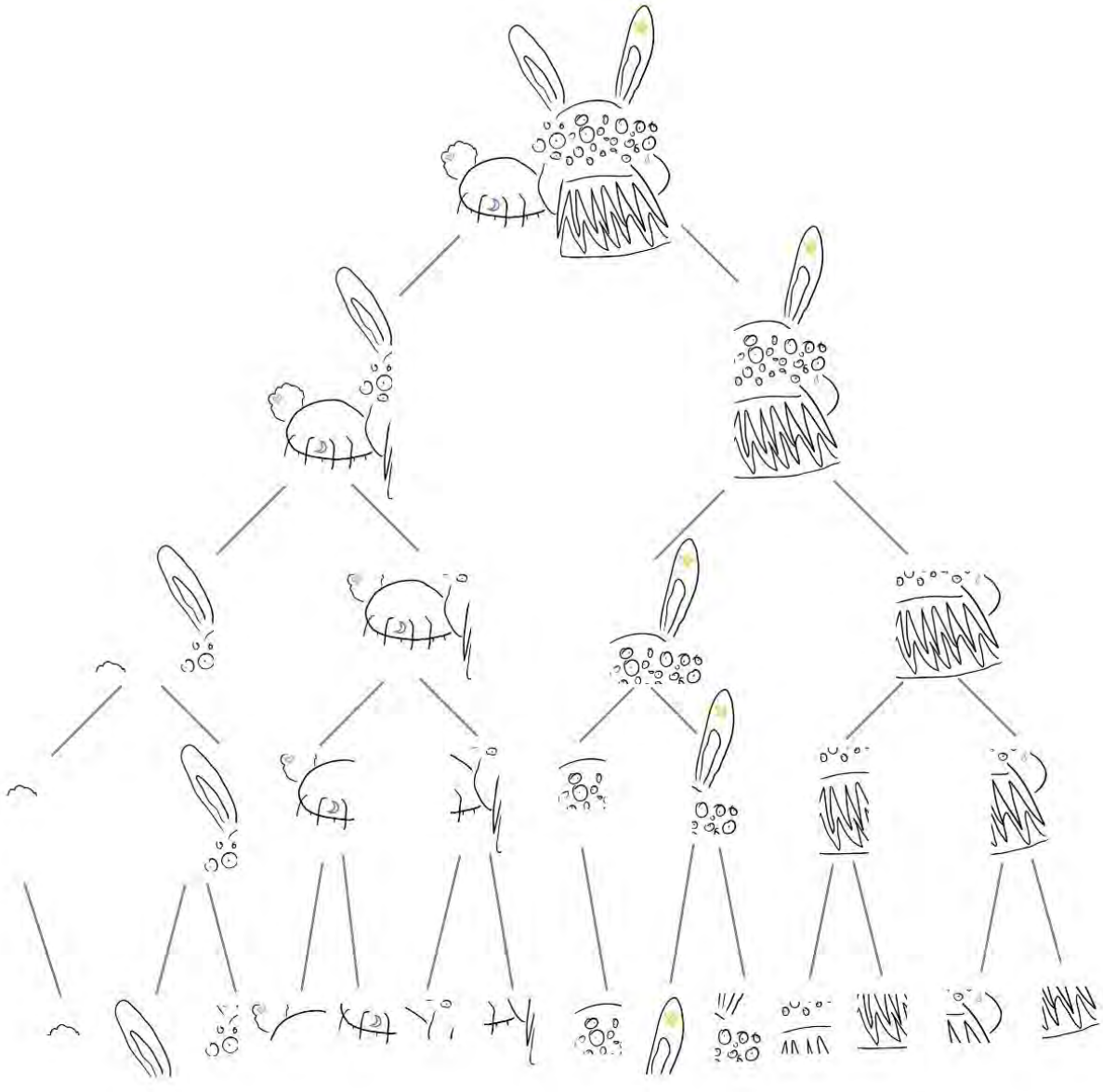
再沿横向从中间切成四份



再沿着竖向平分八份

最后再沿横向切一次。这次有些区域是完全空白的，我们就把它舍弃不要了，得到 14 份：

我们再按照上下左右的关系把切开的图片做成一个二叉树，树的每一个节点是一幅图，它的两个枝是这幅图平分出来的子图。

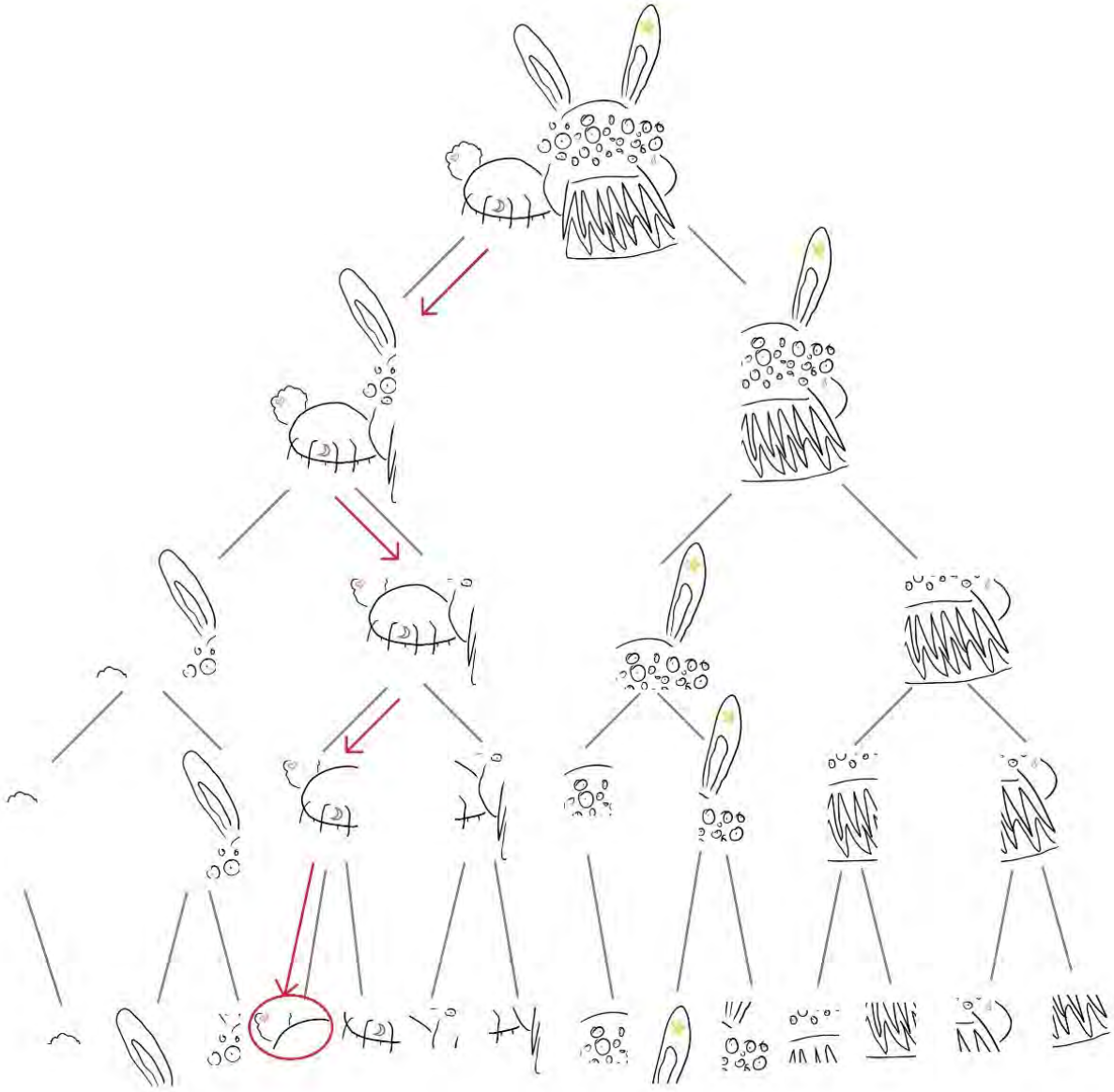


可以看出这个树状结构包含了很多局部性的信息，因为它的每一个节点都是按照上下或者左右进行平分的，因此如果两个点在树中的距离较近，那么它们的实际距离就是比较近的。

搜寻

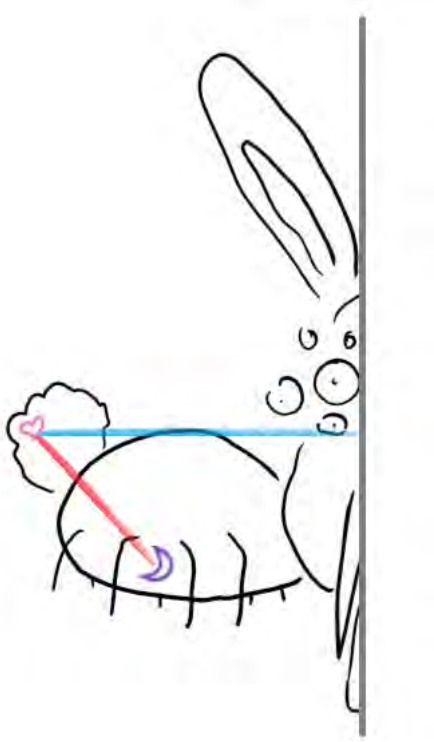
接下来我们要通过这棵二叉树找到离爱心最近的纹身。

首先从树的最顶端开始，向下搜寻找到最底部包含爱心的节点。这个操作非常简单，因为每一次分割要么是沿着某纵线 $x = a$ 要么是沿着横线 $y = a$，因此只需要判断爱心的 x 或 y 轴坐标是大于 a 还是小于 a，便知道是向左还是右边选择树枝。



在找到了爱心之后，我们沿着相同的路径向上攀爬。只爬了一节就发现了屁股上的两个纹身

这里看出，在8平分的情况下，爱心和月亮是在同一个区域的。在某种意义上来讲它们是“近”的，但是我们还不能确定它们是最近的，因此还要继续向上攀爬寻找。再继续向上爬两个节点，都没有出现爱心和月亮以外的纹身。在下面这个节点中



我们发现爱心和月亮之间的距离（红线）要小于爱心和分割线的距离（蓝线），也就是说，不论分割线的右边是什么情况，那边的纹身都不可能离爱心更近。因此可以判断，离爱心最近的图形是月亮。

这样，我们只计算了一次爱心和月亮之间的距离和一次爱心和分割线之间的距离，而不是分别计算爱心和其他三个纹身的距离。并且，要知道，爱心和分割线之间距离的计算非常简单，就是爱心的 x 坐标和分割线的 x 坐标的差（的绝对值），相比于计算两点之间的距离

$$\sqrt{((x_1-y_1)^2+(x_2-y_2)^2)}$$

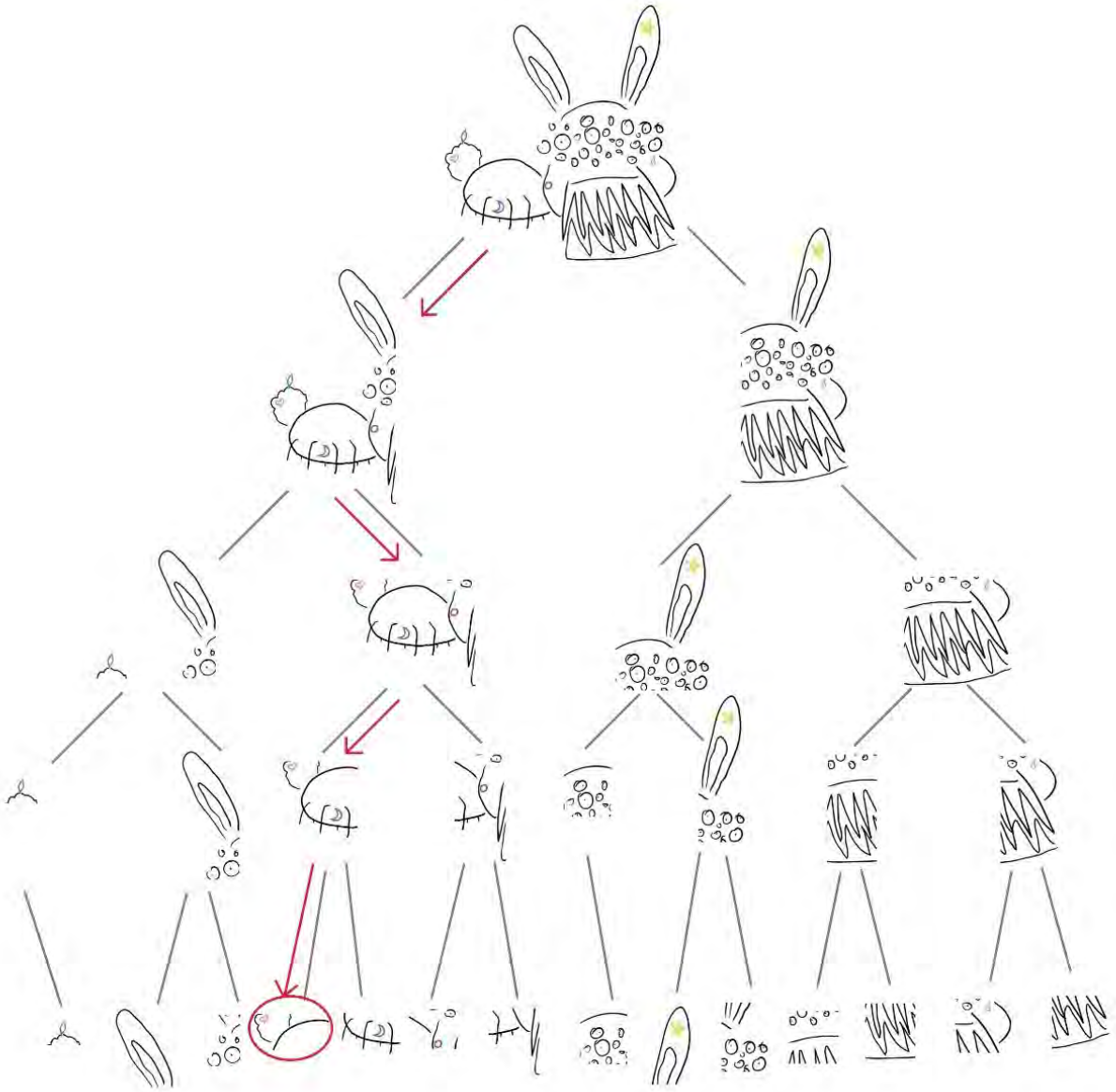
要省下很多计算量。

麻烦

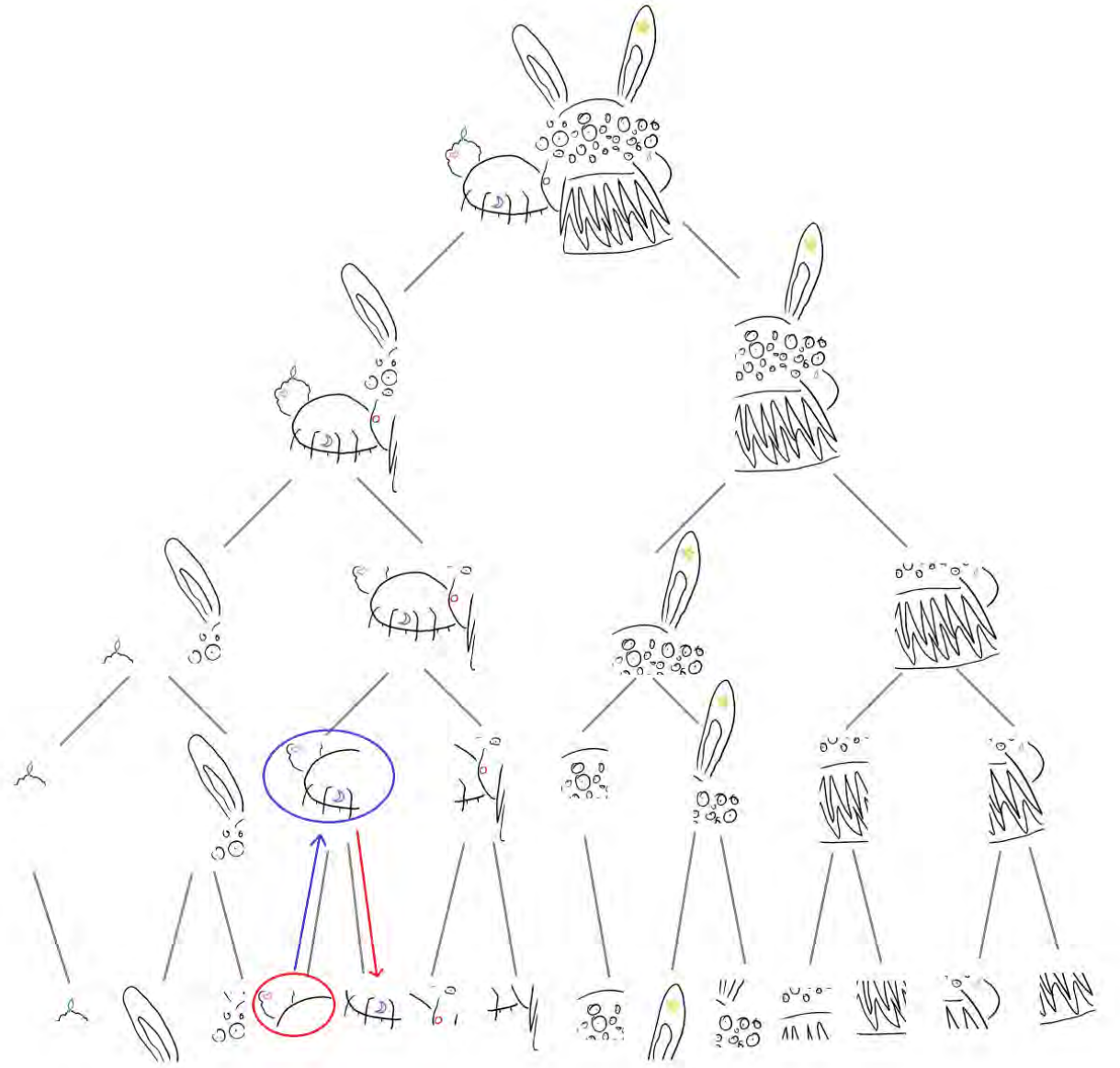
啊，但也有可能这个搜寻最近点的过程没那么顺利。在上面的计算中，在找到了离爱心比较近的月亮之后，我们发现爱心距离分割线的距离比较远，因此确定月亮的确就是最近的。但是，在分割线的另一边有一个更近的纹身，那么情况就稍微复杂了。

就说这个兔子啊，又去加了两个纹身，一片叶子和一个圆圈。

二叉树分割上也相应地多出这两个纹身。我们想找到离爱心最近的纹身，所以依旧向下搜寻先找到爱心。

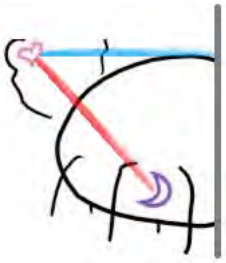


我们找来一张纸，记下在已访问节点中距离爱心最近的纹身和所对应的距离。现在这张纸还是空的。

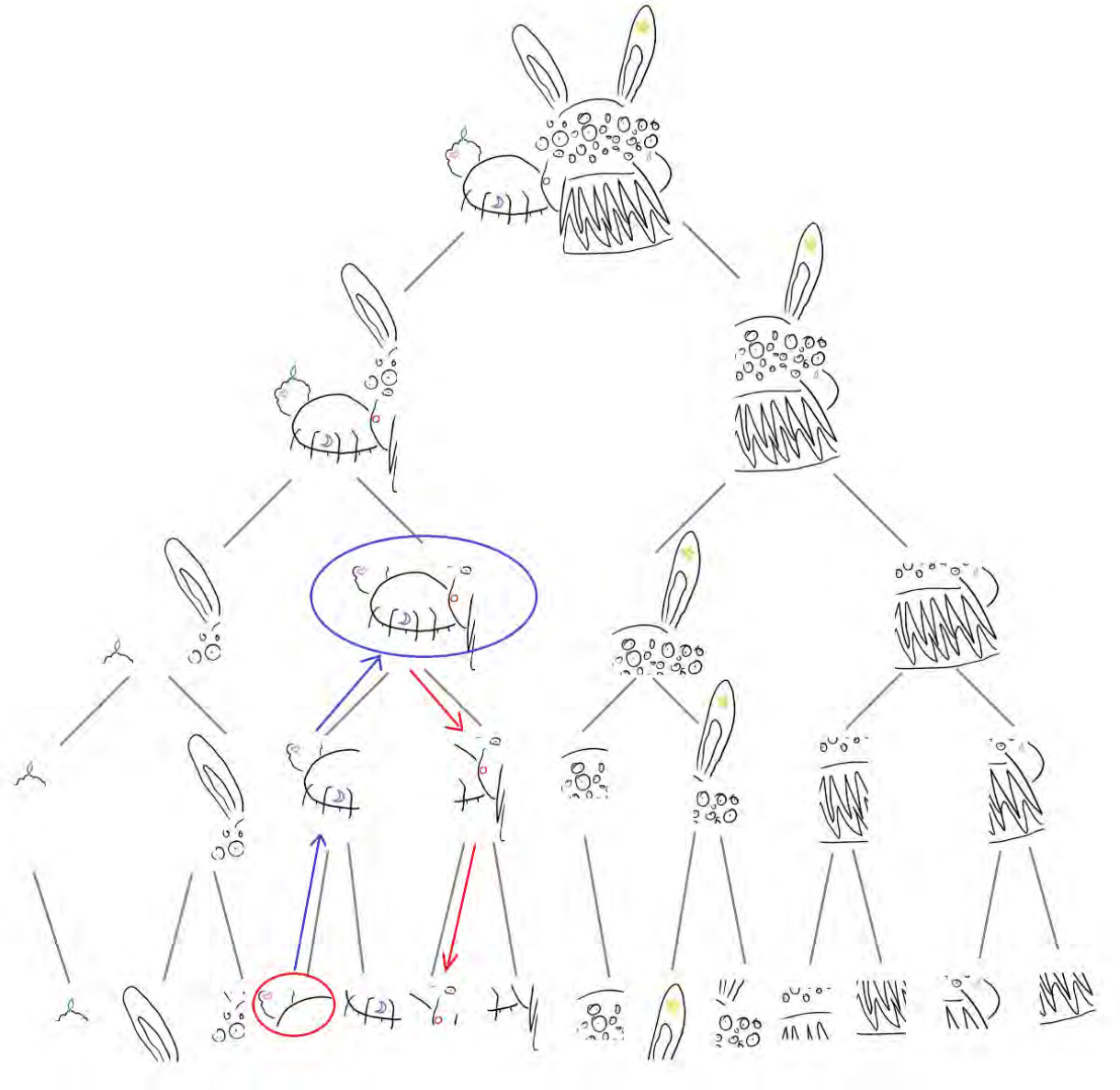


向上爬了一节，发现那一节的另一个枝里有月亮，于是跑下去查看月亮的坐标，计算爱心和月亮的距离，并在纸上记录 $(\text{图形} = \text{月亮}，\text{距离} = d_1)$。

再回到蓝圈的节点向上爬，继续向上爬。我们发现，d_1（红线）大于爱心和分割线的距离（蓝线）。

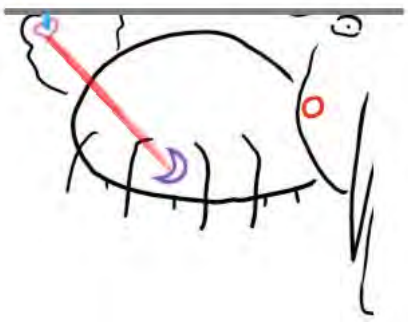


也就是说分割线的另一边可能有更近的点，所以从另一个分枝开始向下搜，找到…

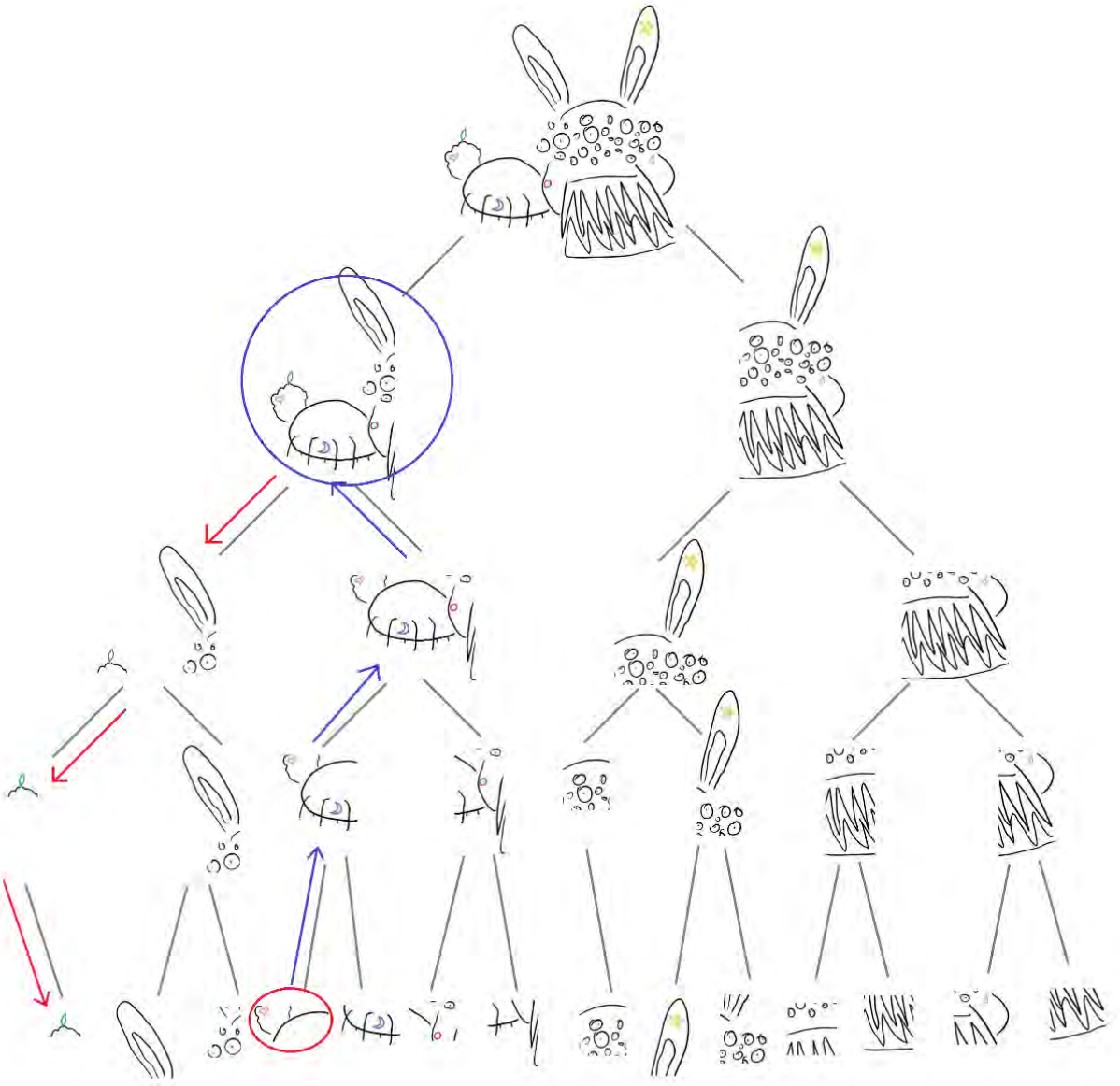


在另一个分枝中我们追溯到圆圈，并计算它与爱心的距离 d_2，发现 Misplaced &，比月亮远，所以丢弃不要。

再向上爬一个节，我们发现 d_1（红线）大于爱心和切分线之间的距离（蓝线）

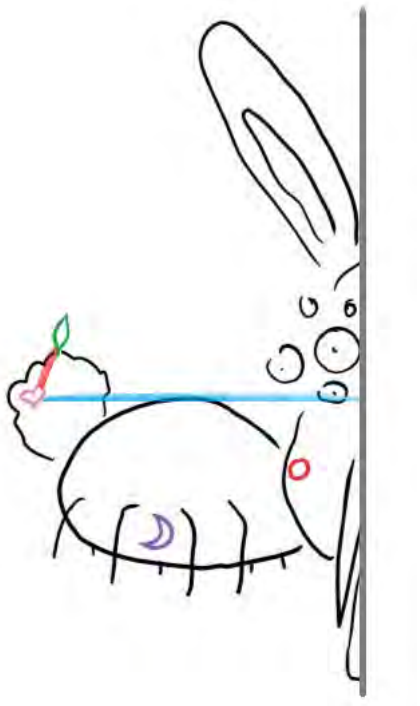


因此，切分线的另一端可能有更近的纹身，因此我们从另一个树枝向下搜索…

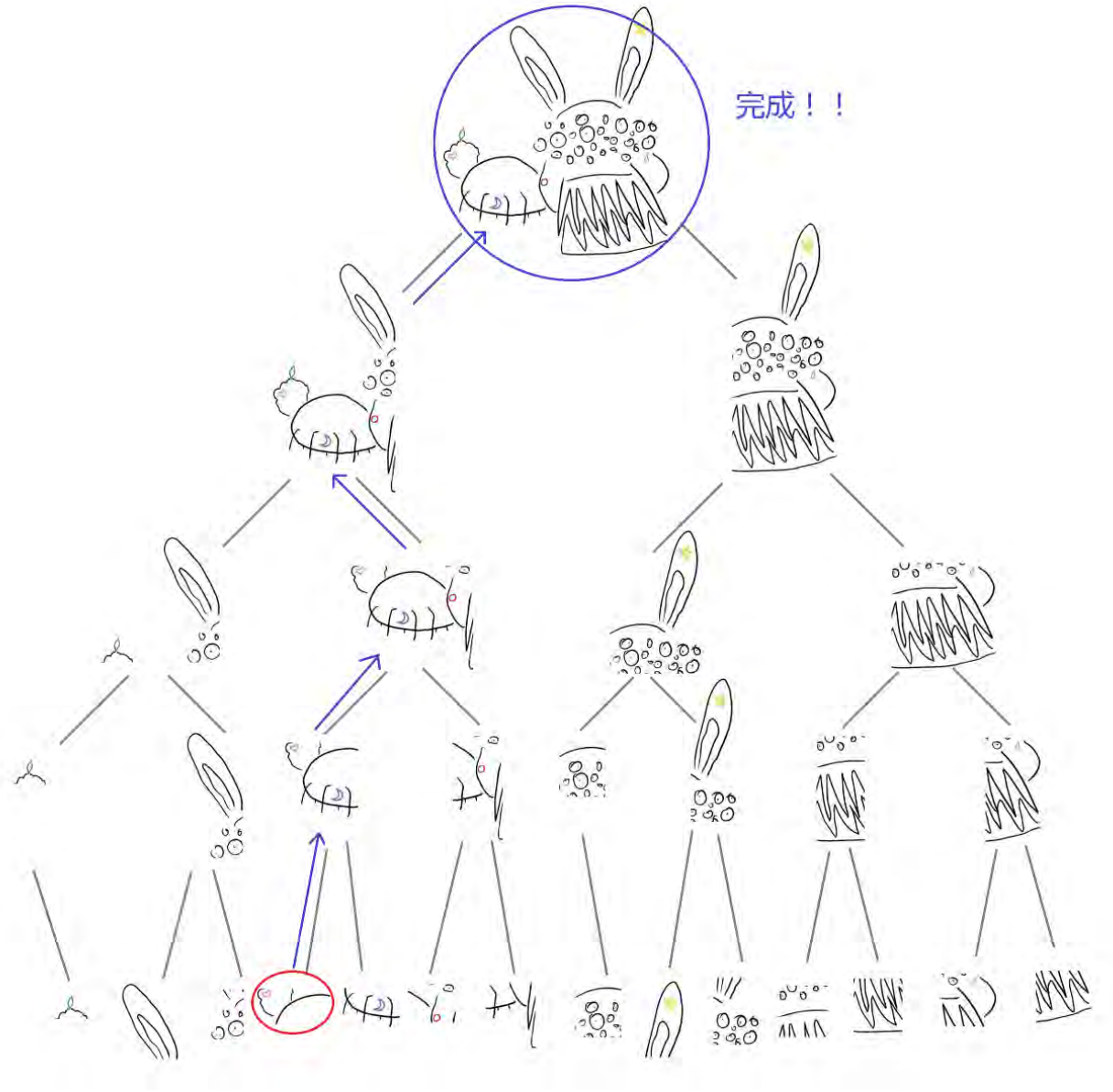


找到了叶子。（所幸在这个分枝里只搜索到了叶子，如果有更多的图形的话，还需要进行多层的递归。具体的过程会在后面的详细篇中讲解。）计算叶子和爱心之间的距离，得 d_3，并发现 $d_3 < d_1$，比月亮更近，于是更新纸上的记录为 $(\text{纹身} = \text{叶子}，\text{距离} = d_3)$。

再向上攀登一节，我们发现 d_3 小于爱心和切分线的距离，因此另一边的数据就不用考虑了。



这次我们已经爬到了树的最顶端，完成了搜索，纸上记载的 $(\text{叶子}，d_3)$ 就是最近的纹身和对应的距离。



结语

在以上的算法中，当我们已经找到了比切分线更近的点时，就不需要继续搜索切分线另一边的点了，因为那些只会更远。于是，通过把整个空间进行分割并以树状结构进行记录，我们只需要在问题点附近的一些区域进行搜寻便可以找到最近的数据点，节省了大量的计算。

到此为止，本篇文章友好地介绍了如何使用二叉树的形式记录距离信息并快速地进行搜索，但文中所讲的还不是 kd 树。下一篇文章，kd 树算法之详细篇，将系统性地介绍 kd 树的定义和在 kd 树上的 kNN 算法。

文章迭代记录
v1.0，2016-09-01，文章上线

【量化课堂】kd 树算法之详细篇

导语：在上一篇《kd 树算法之思路篇》中，我们介绍了如何用二叉树格式记录空间内的距离，并以其为依据进行高效的索引。在本篇文章中，我们将详细介绍 kd 树的构造以及 kd 树上的 kNN 算法。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，本文的难度属于进阶（下），深度为 level-1

阅读本文前请掌握 kNN（level-1）的知识。

kd 树的结构

kd树是一个二叉树结构，它的每一个节点记载了【特征坐标，切分轴，指向左枝的指针，指向右枝的指针】。
其中，特征坐标是线性空间 $\mathbb{R}^n$ 中的一个点 $(x_1, x_2, \ldots, x_n)$。

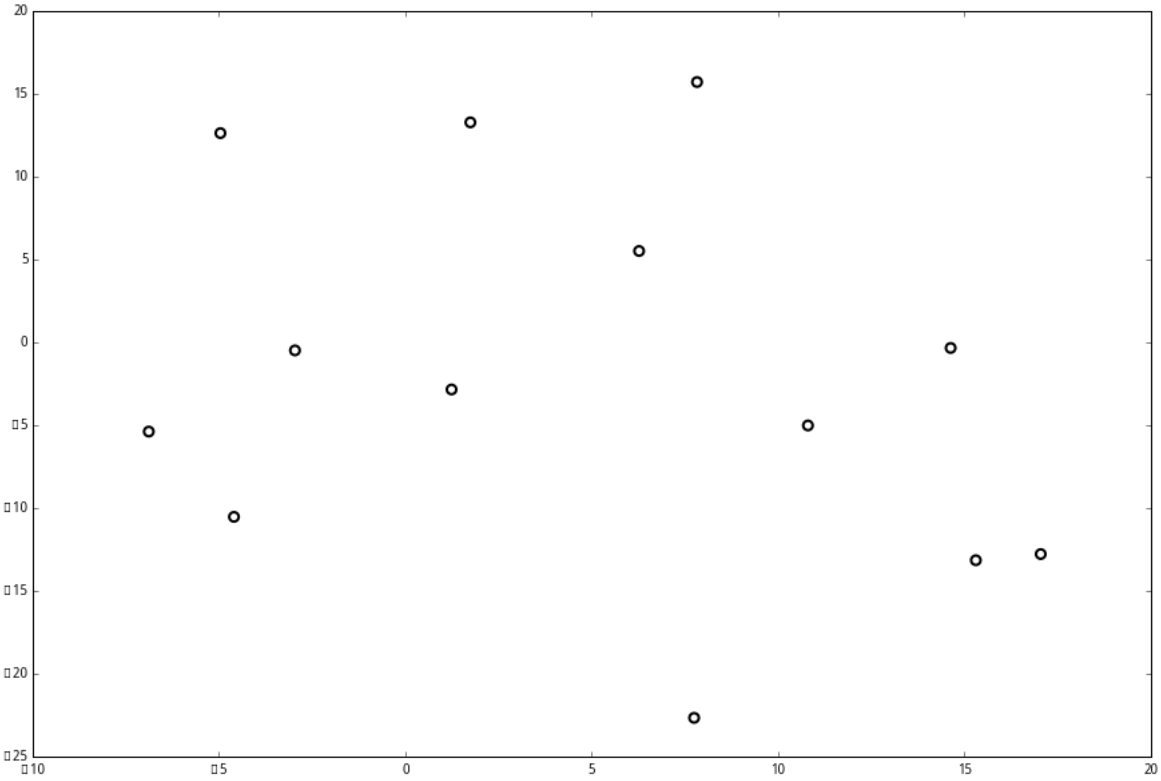
切分轴由一个整数 r 表示，这里 $1 \leq r \leq n$，是我们在 n 维空间中沿第 r 维进行一次分割。
节点的左枝和右枝分别都是 kd 树，并且满足：如果 y 是左枝的一个特征坐标，那么 $y_r \leq x_r$；并且如果 z 是右枝的一个特征坐标，那么 $z_r \geq x_r$。

给定一个数据样本集 $S \subseteq R^n$ 和切分轴 r，以下递归算法将构建一个基于该数据集的 kd 树，每一次循环制作一个节点：
– 如果 $|S| = 1$，记录 S 中唯一的一个点为当前节点的特征数据，并且不设左枝和右枝。（$|S|$ 指集合 S 中元素的数量）
– 如果 Misplaced &：
- 将 S 内所有点按照第 r 个坐标的大小进行排序；
- 选出该排列后的中位元素（如果一共有偶数个元素，则选择中位左边或右边的元素，左或右并无影响），作为当前节点的特征坐 标，并且记录切分轴 r；
- 将 S_L 设为在 S 中所有排列在中位元素之前的元素； S_R 设为在 S 中所有排列在中位元素后的元素；
- 当前节点的左枝设为以 S_L 为数据集并且 r 为切分轴制作出的 kd 树；当前节点的右枝设为以 S_R 为数据集并且 r 为切分轴制作出 的 kd 树。

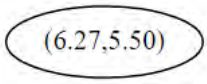
再设 $r \leftarrow (r+1) \mod n$。（这里，我们想轮流沿着每一个维度进行分割； $\mod n$ 是因为一共有 n 个维度，在 沿着最后一个维度进行分割之后再重新回到第一个维度。）

构造 kd 树的例子

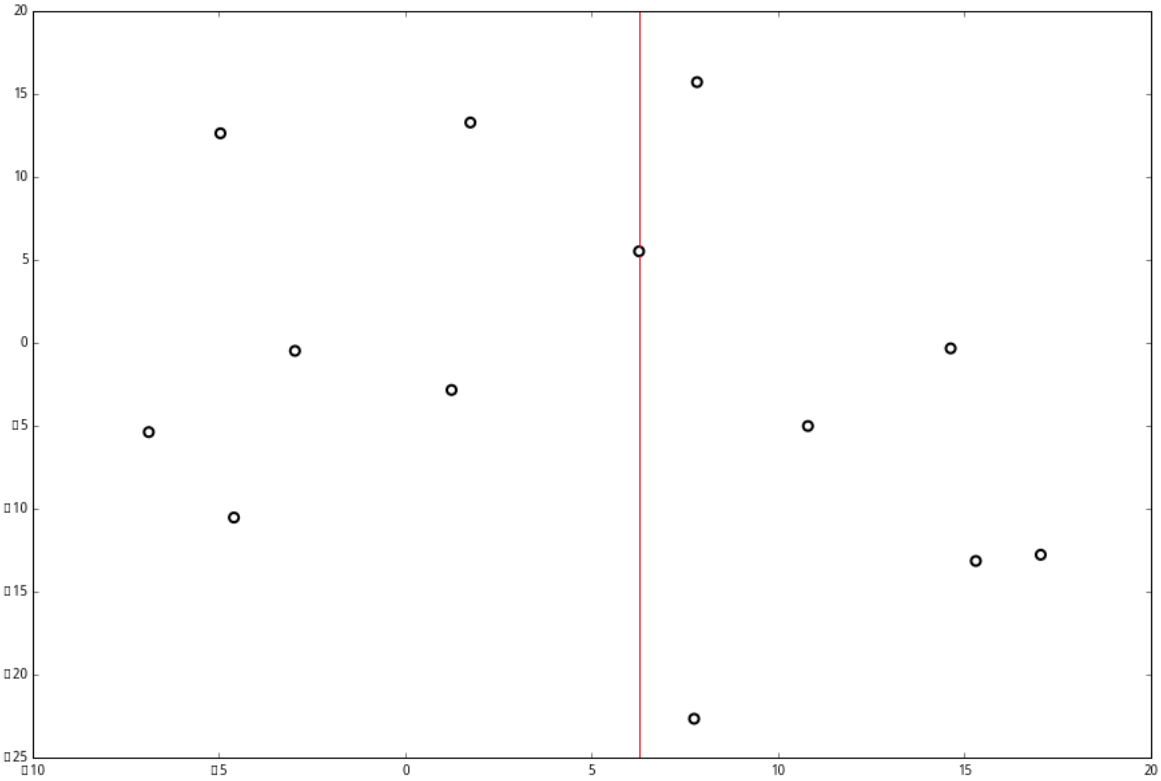
上面抽象的定义和算法确实是很不好理解，举一个例子会清楚很多。首先随机在 $\mathbb{R}^2$ 中随机生成 13 个点作为我们的数据集。起始的切分轴 $r = 0$；这里 $r = 0$ 对应 x 轴，而 $r = 1$ 对应 y 轴。



首先先沿 x 坐标进行切分，我们选出 x 坐标的中位点，获取最根部节点的坐标

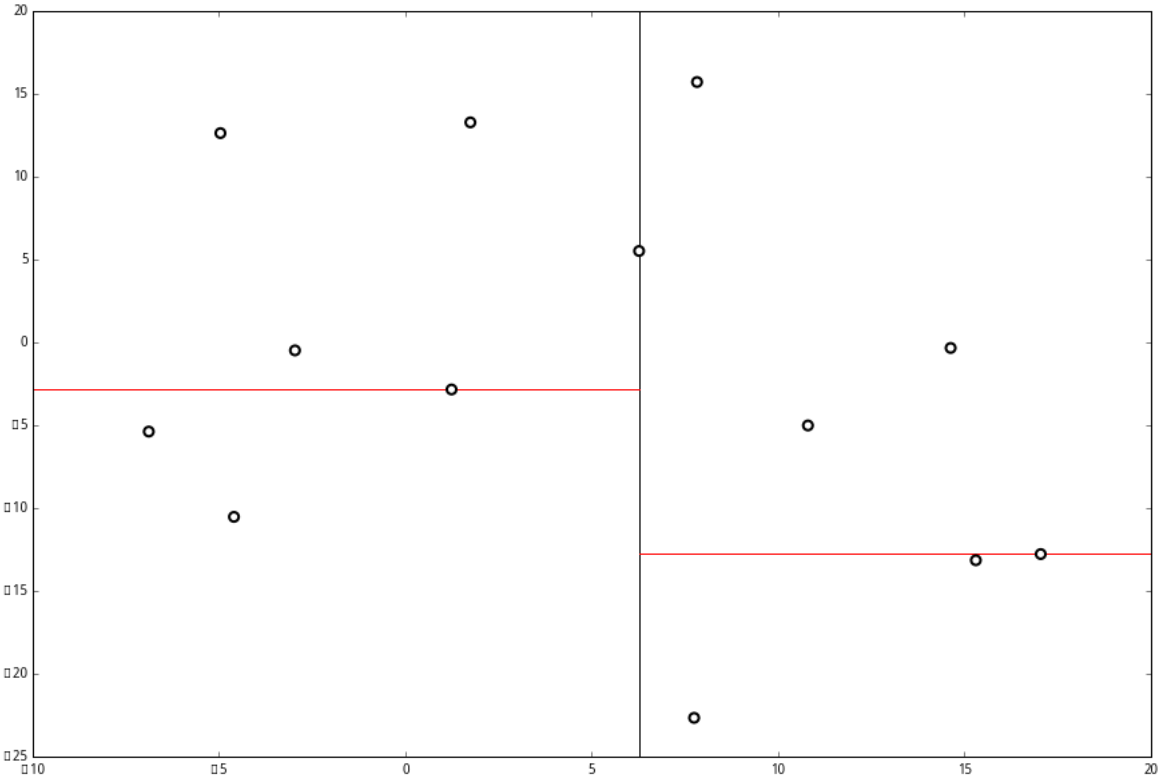


并且按照该点的x坐标将空间进行切分，所有 x 坐标小于 6.27 的数据用于构建左枝，x坐标大于 6.27 的点用于构建右枝。

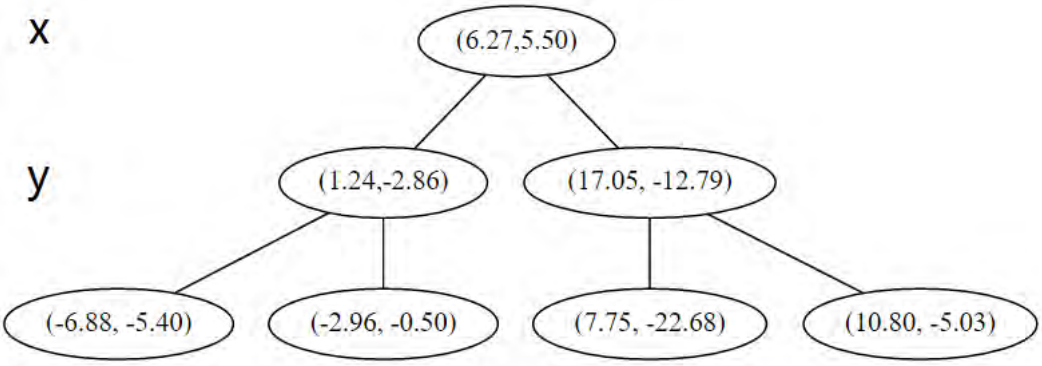


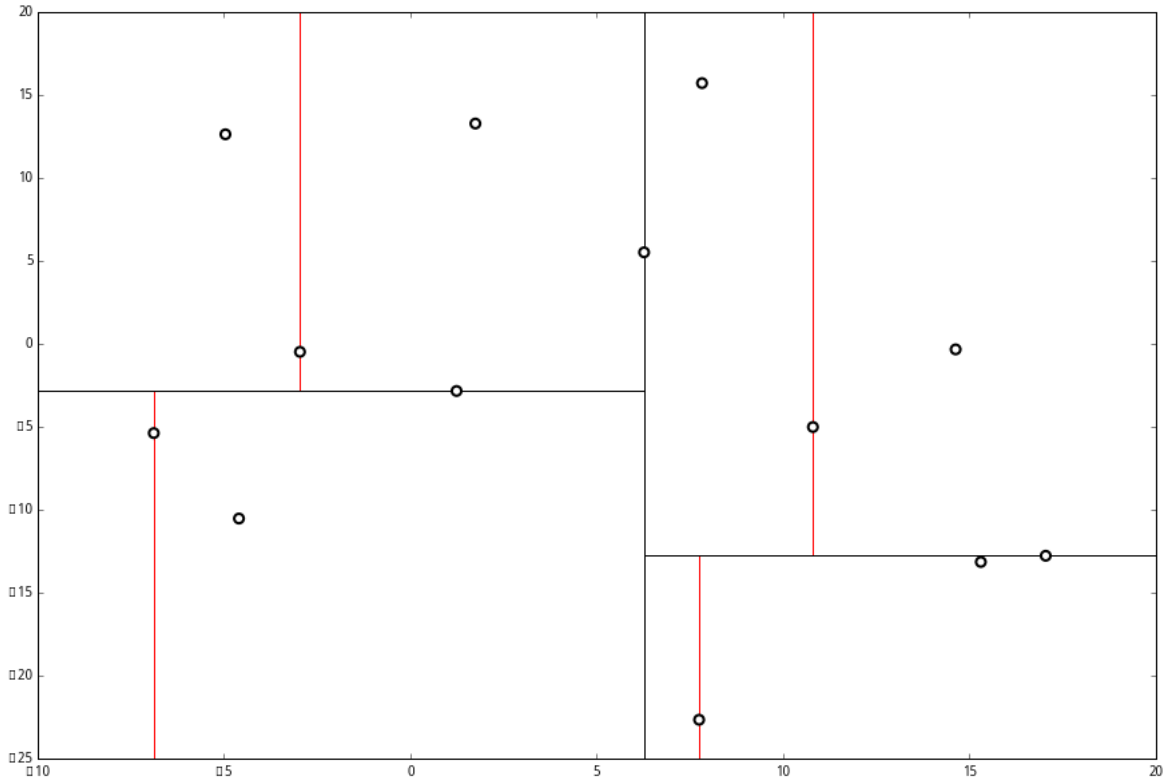
在下一步中 $r = 0 + 1 = 1 \mod 2$对应 y 轴，左右两边再按照 y 轴的排序进行切分，中位点记载于左右枝的节点。得到下面的树，左边的x 是指这该层的节点都是沿 x 轴进行分割的。

空间的切分如下

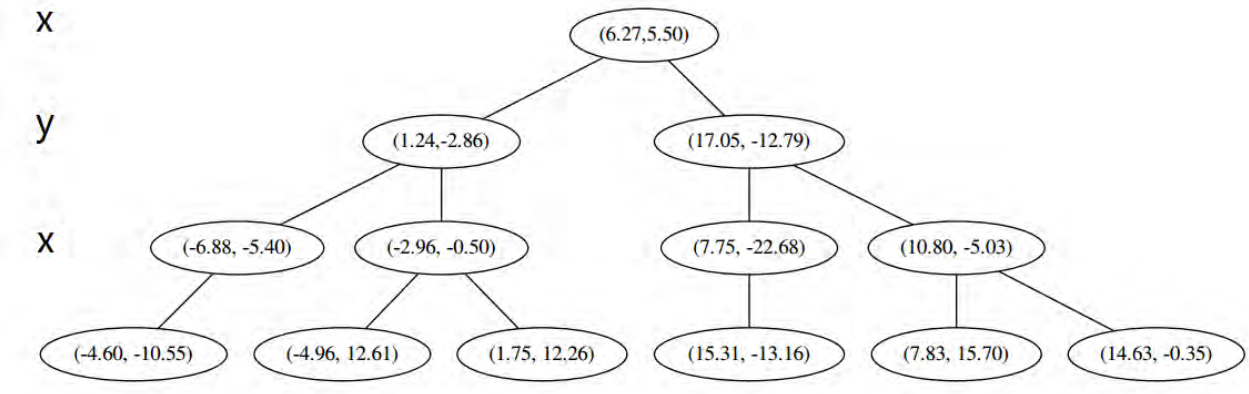


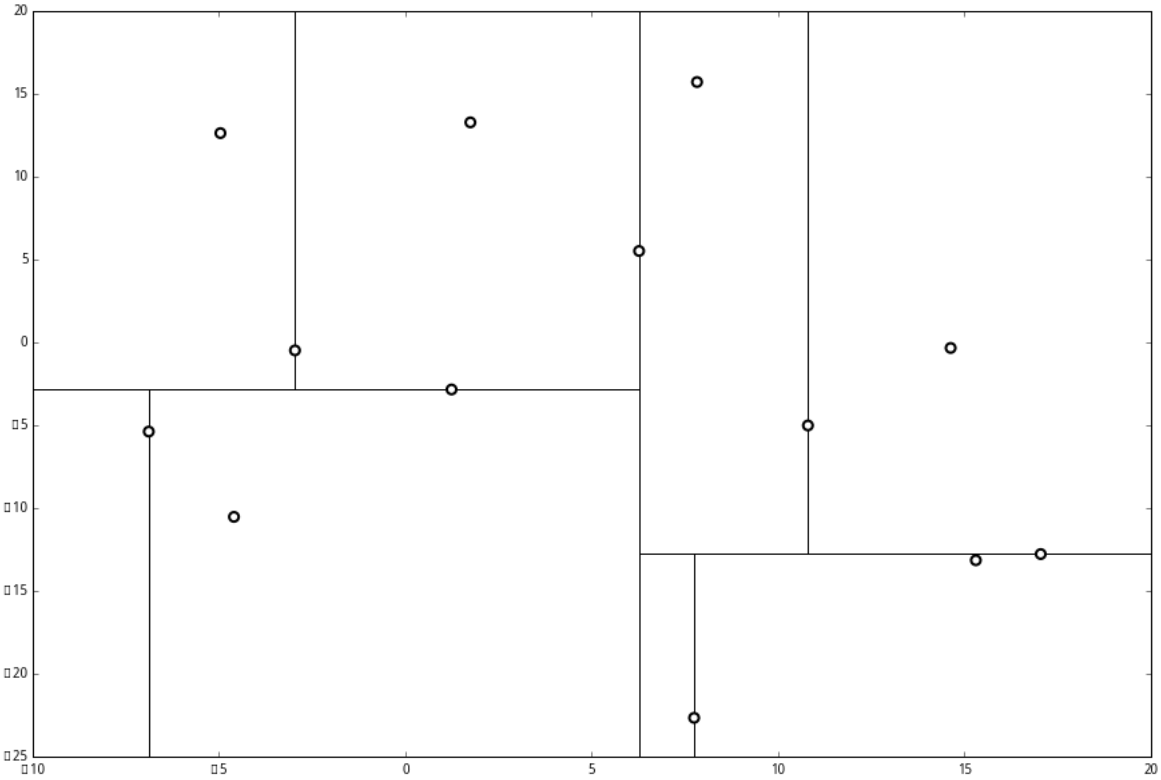
下一步中 $r \equiv 1+1 \equiv 0 \mod 2$，对应 x 轴，所以下面再按照 x 坐标进行排序和切分，有





最后每一部分都只剩一个点，将他们记在最底部的节点中。因为不再有未被记录的点，所以不再进行切分。





就此完成了 kd 树的构造。

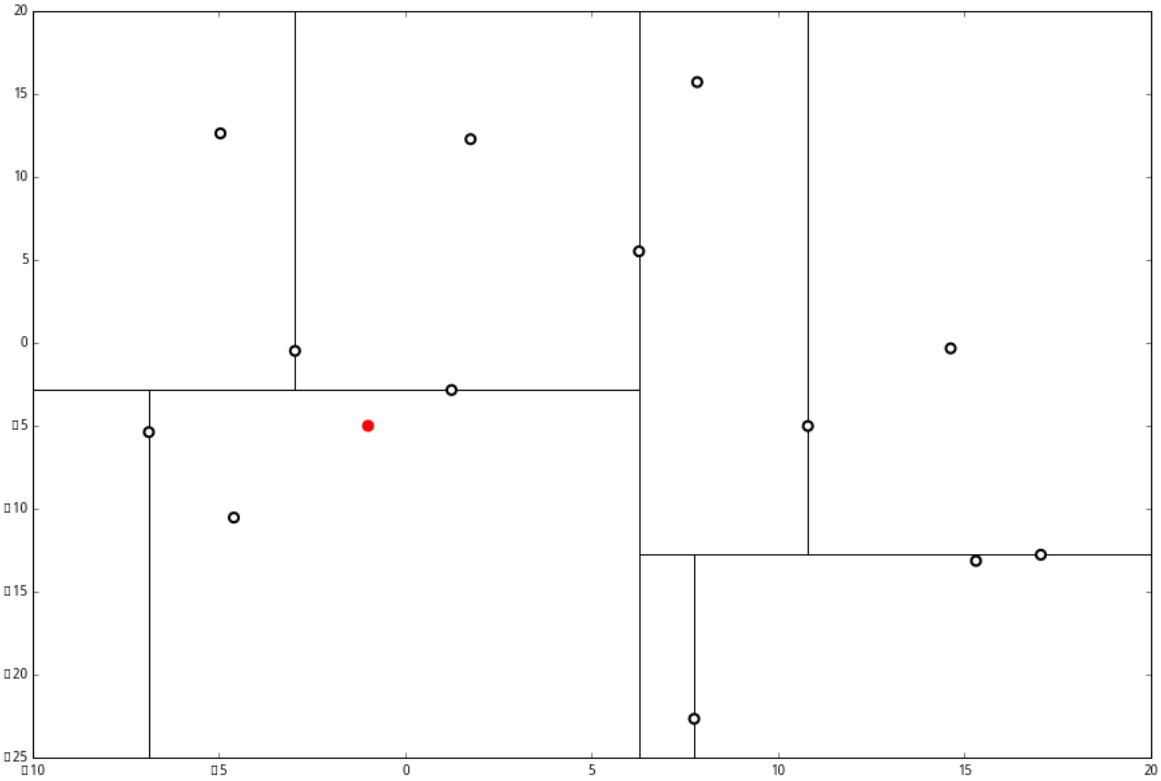
kd 树上的 kNN 算法

给定一个构建于一个样本集的 kd 树，下面的算法可以寻找距离某个点 p 最近的 k 个样本。

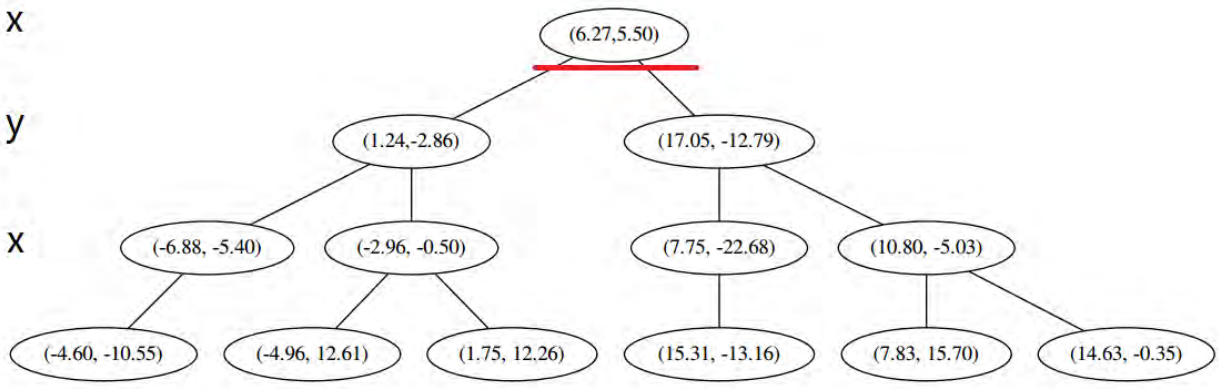
零、设 L 为一个有 k 个空位的列表，用于保存已搜寻到的最近点。
一、根据 p 的坐标值和每个节点的切分向下搜索（也就是说，如果树的节点是按照 $x_r = a$ 进行切分，并且 p 的 r 坐标小于 a，则向左枝 进行搜索；反之则走右枝）。
二、当达到一个底部节点时，将其标记为访问过。如果 L 里不足 k 个点，则将当前节点的特征坐标加入 L；如果 L 不为空并且当前节点 的特征与 p 的距离小于 L 里最长的距离，则用当前特征替换掉 L 中离 p 最远的点。
三、如果当前节点不是整棵树最顶端节点，执行 (a)；反之，输出 L，算法完成。
a. 向上爬一个节点。如果当前（向上爬之后的）节点未曾被访问过，将其标记为被访问过，然后执行 (1) 和 (2)；如果当前节点被访 问过，再次执行 (a)。
1. 如果此时 L 里不足 k 个点，则将节点特征加入 L；如果 L 中已满 k 个点，且当前节点与 p 的距离小于 L 里最长的距离， 则用节点特征替换掉 L 中离最远的点。
2. 计算 p 和当前节点切分线的距离。如果该距离大于等于 L 中距离 p 最远的距离并且 L 中已有 k 个点，则在切分线另一边不会有更近的点，执行 (三)；如果该距离小于 L 中最远的距离或者 L 中不足 k 个点，则切分线另一边可能有更近的点，因此在当前节点的另一个枝从 (一) 开始执行。

啊呃… 被这算法噎住了，赶紧喝一口下面的例子

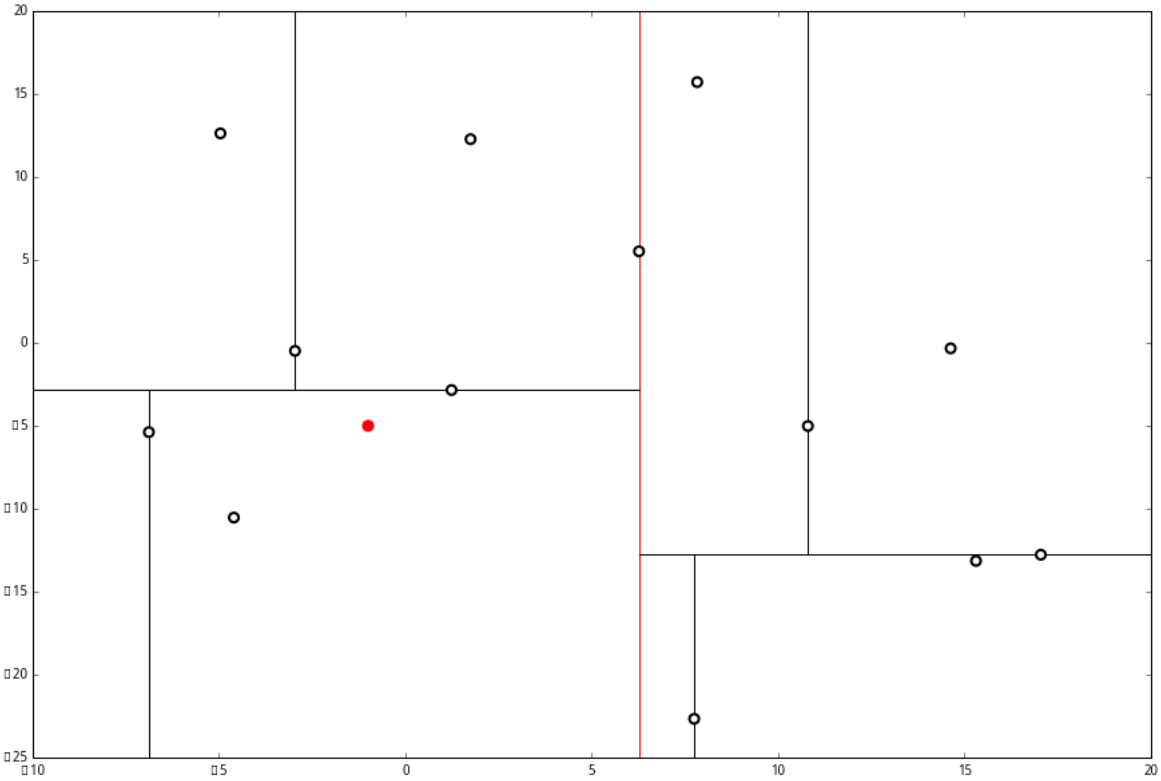
设我们想查询的点为 $p = (-1, -5)$，设距离函数是普通的 L_2 距离，我们想找距离问题点最近的 $k = 3$ 个点。如下：



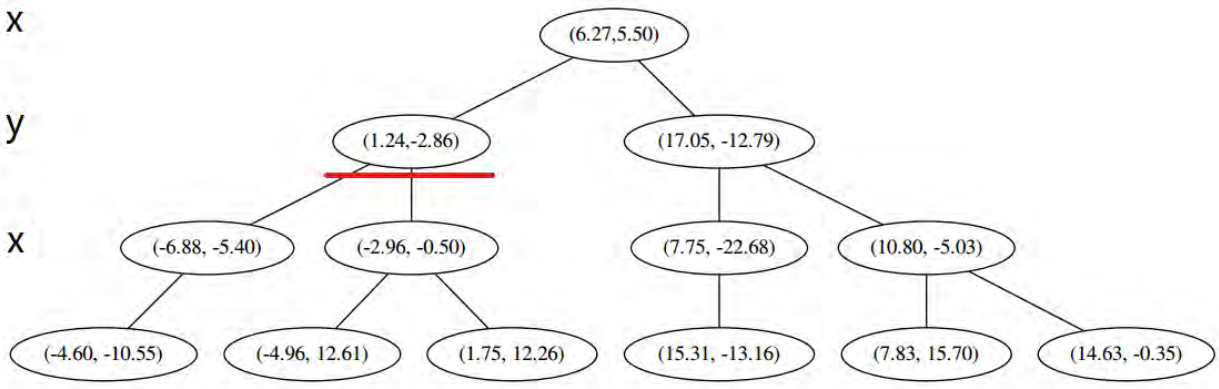
首先执行 (一)，我们按照切分找到最底部节点。首先，我们在顶部开始



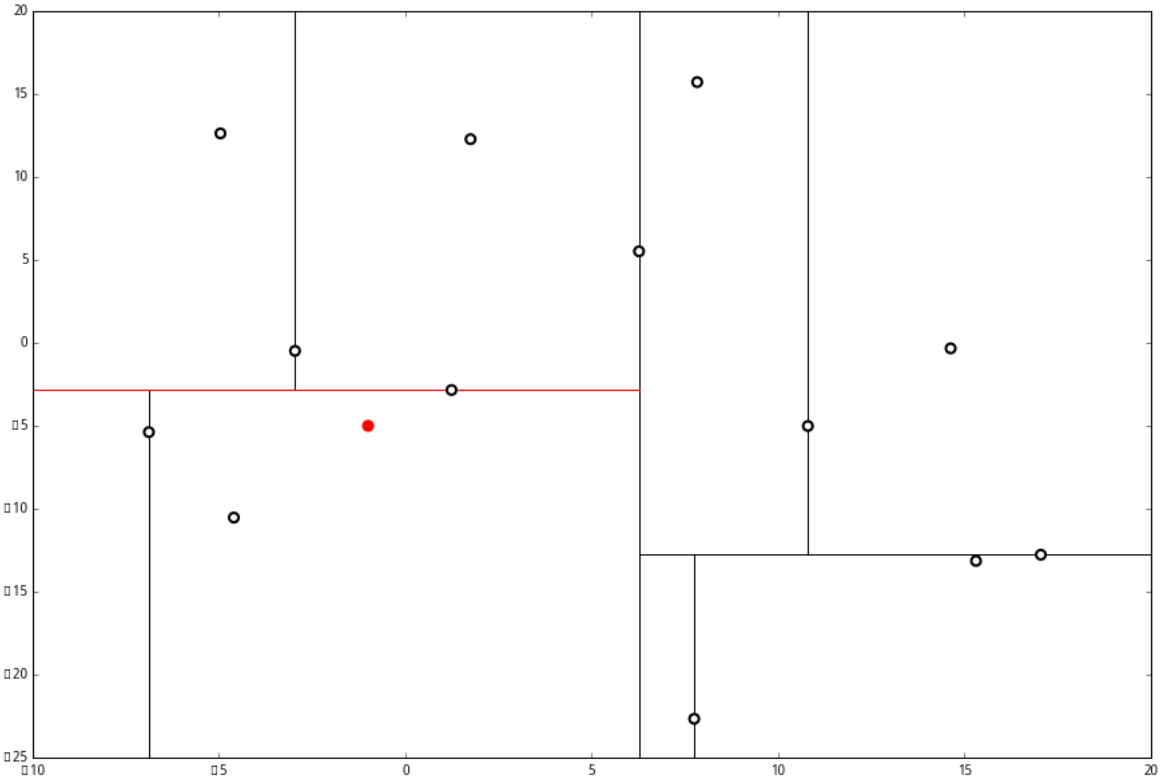
和这个节点的 x 轴比较一下，



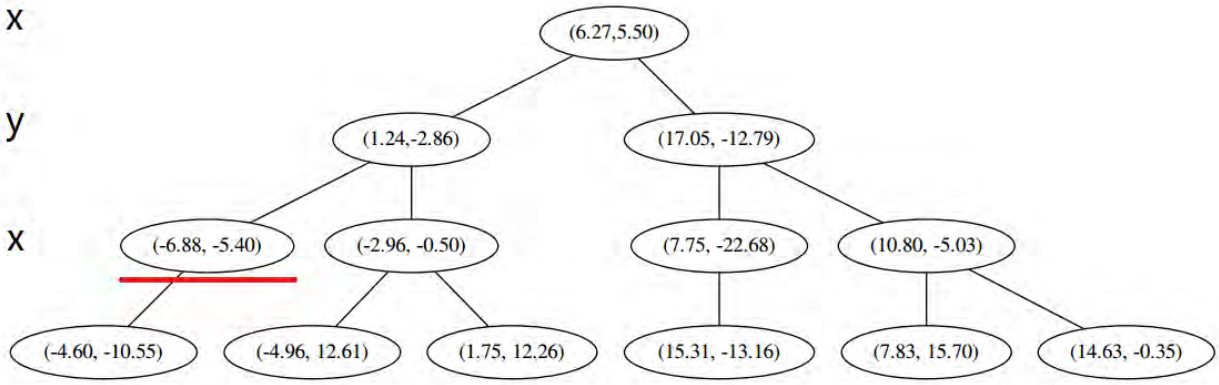
p 的 x 轴更小。因此我们向左枝进行搜索：



这次对比 y 轴，

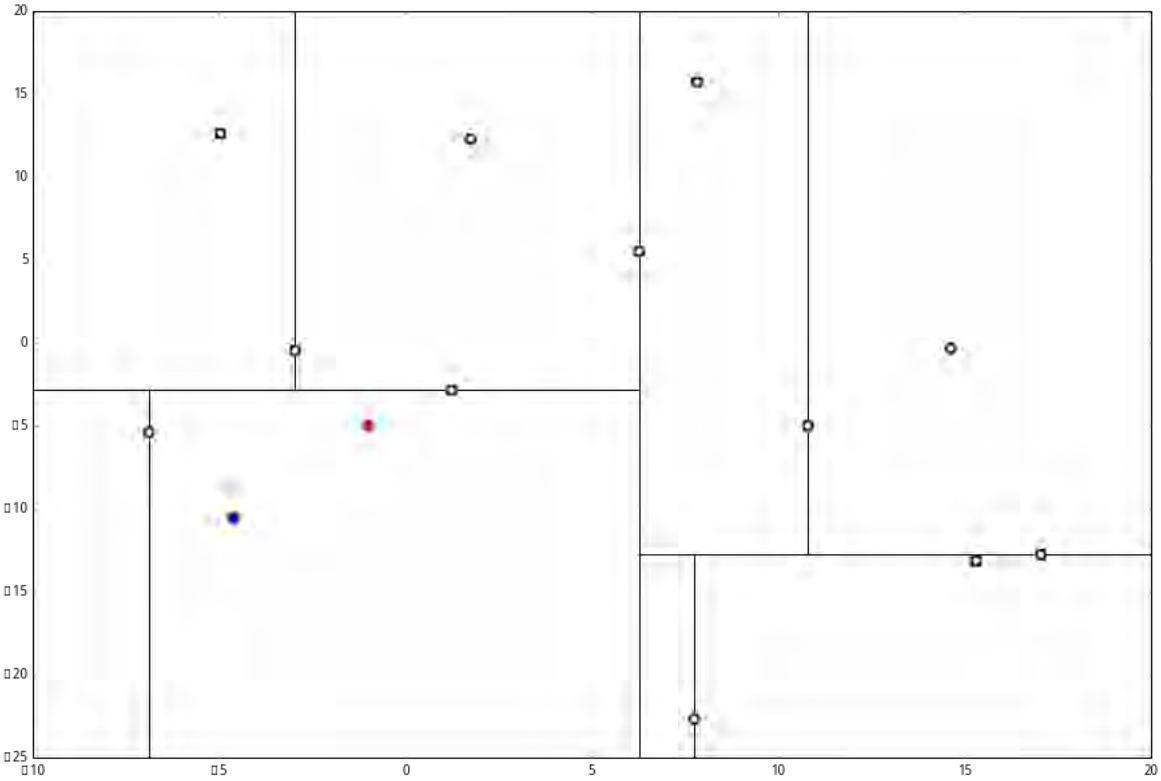


p 的 y 值更小，因此向左枝进行搜索：



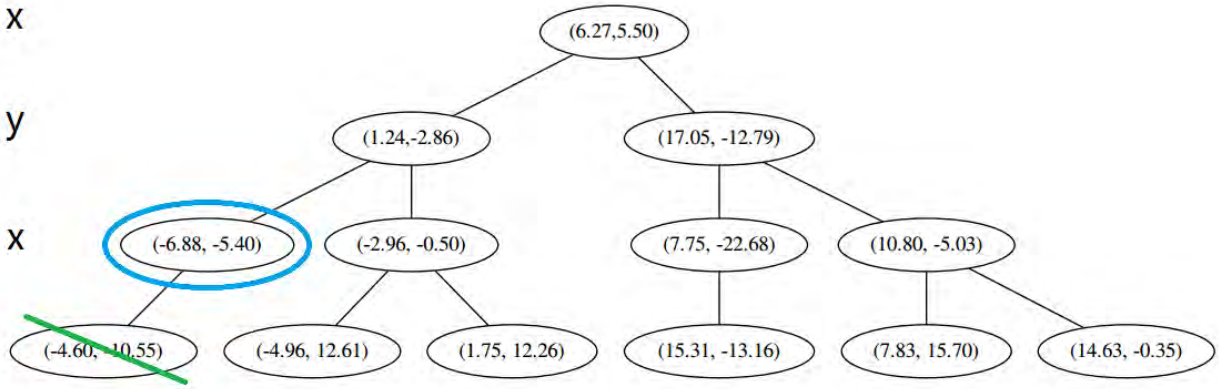
这个节点只有一个子枝，就不需要对比了。由此找到了最底部的节点 $(-4.6, -10.55)$。

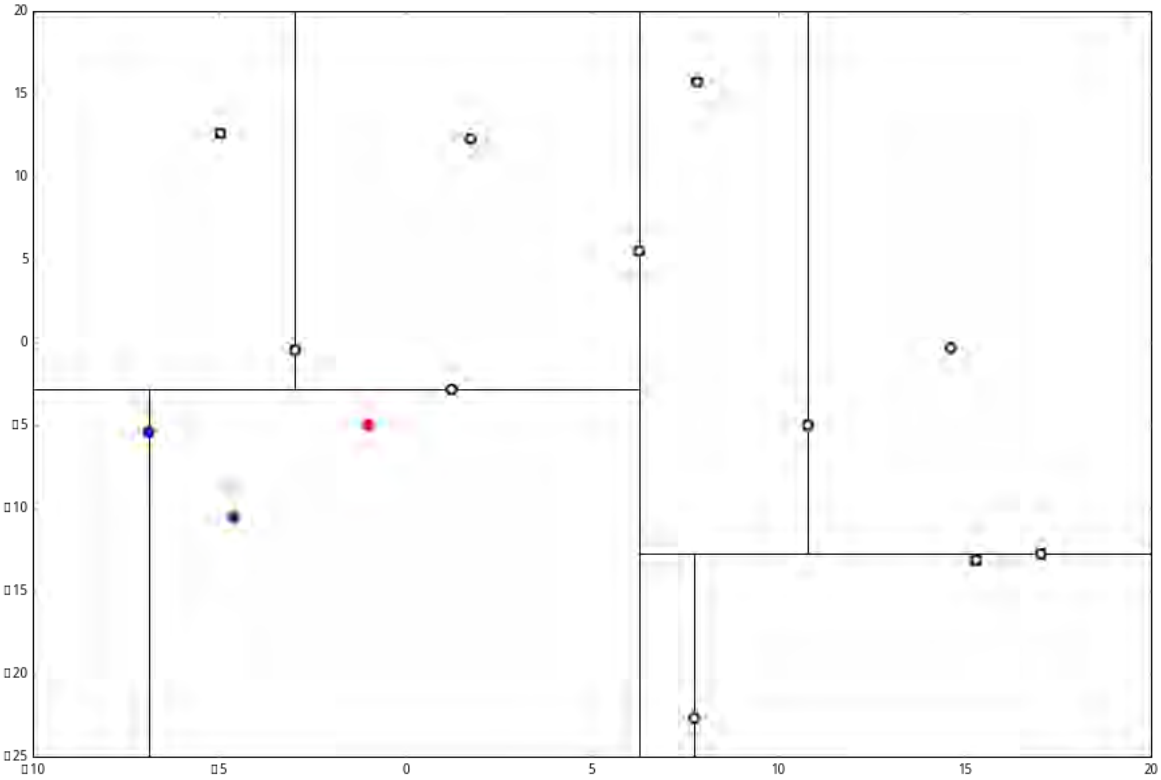
在二维图上是



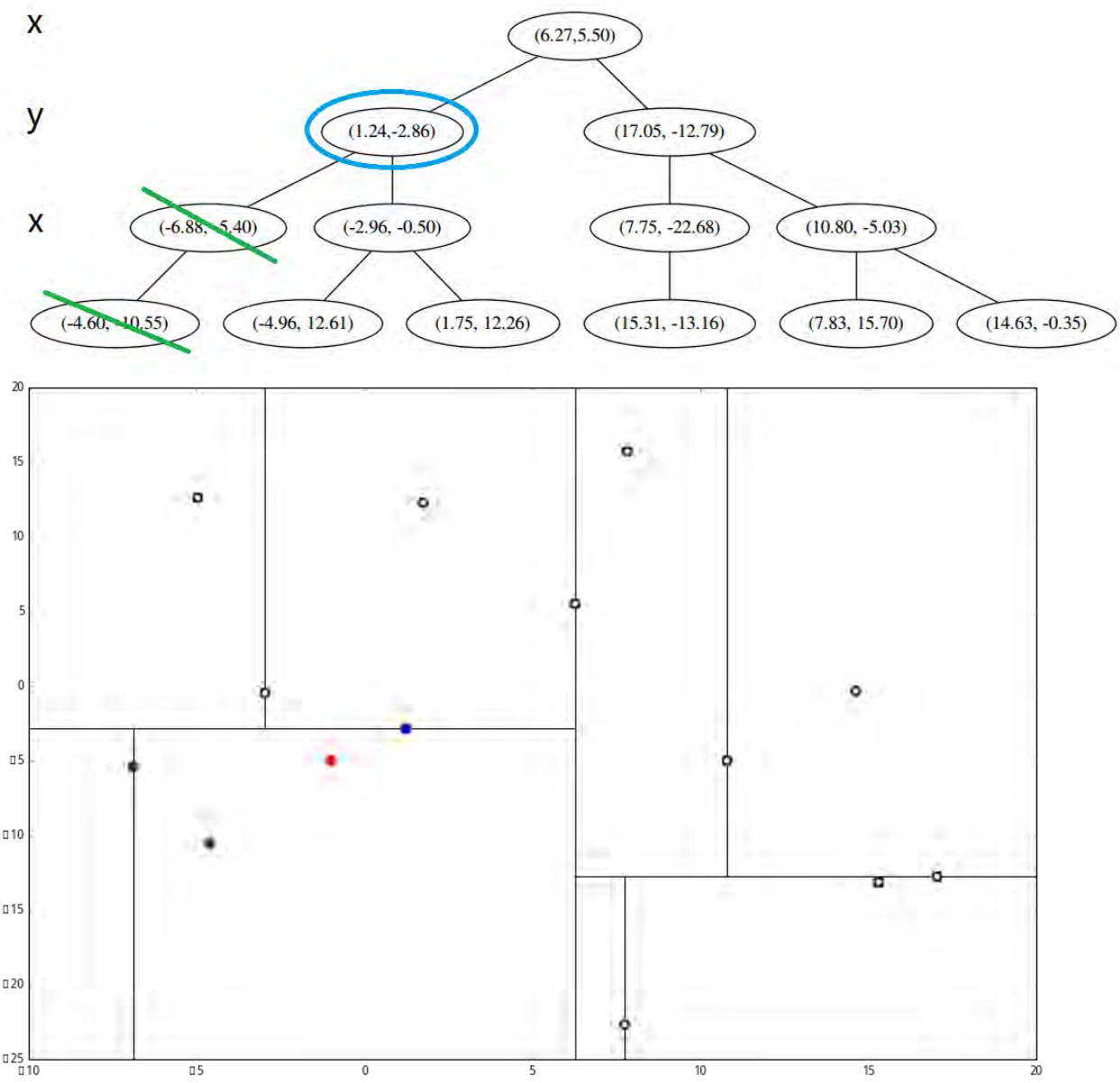
此时我们执行 (二)。将当前结点标记为访问过，并记录下 $L = [(-4.6, -10.55)]$。啊，访问过的节点就在二叉树上显示为被划掉的好了。

然后执行 (三)，嗯，不是最顶端节点。好，执行 (a)，我爬。上面的是 $(-6.88, -5.4)$。



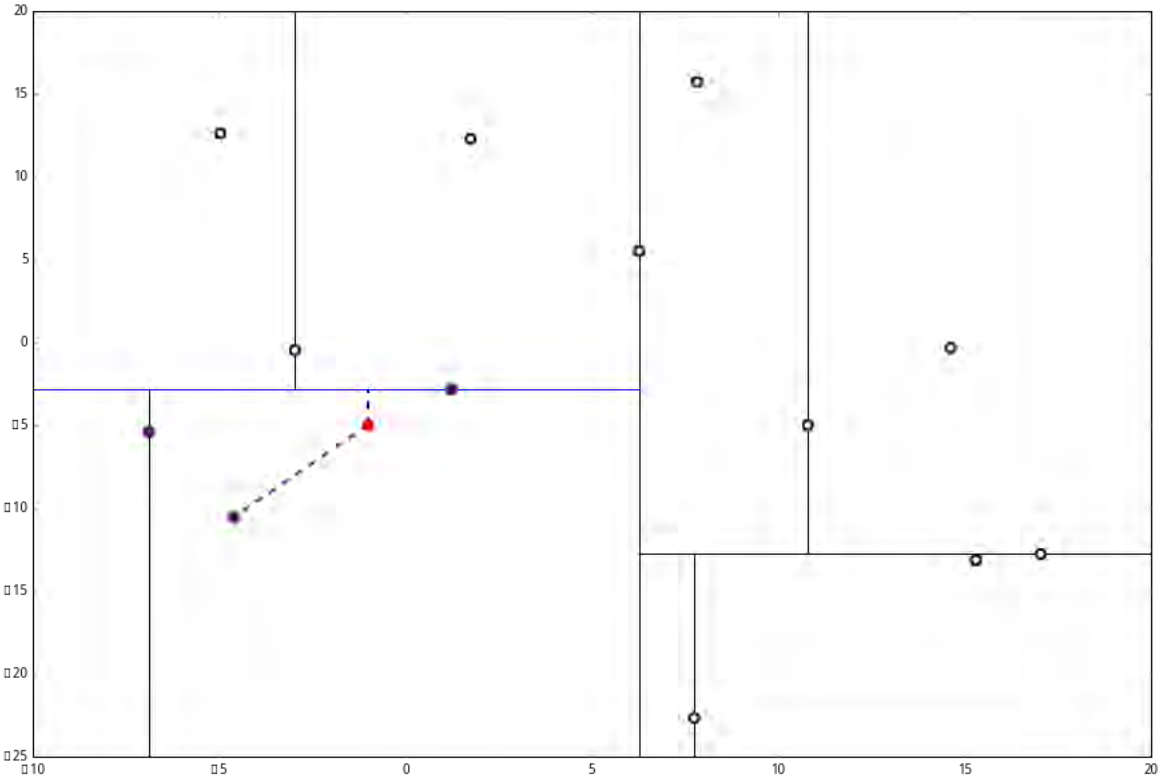


执行 (1)，因为我们记录下的点只有一个，小于 $k=3$，所以也将当前节点记录下，有 $L=[(-4.6,-10.55),(-6.88,-5.4)]$。再执行 (2)，因为当前节点的左枝是空的，所以直接跳过，回到步骤 (三)。(三) 看了一眼，好，不是顶部，交给你了，(a)。于是乎 (a) 又往上爬了一节。

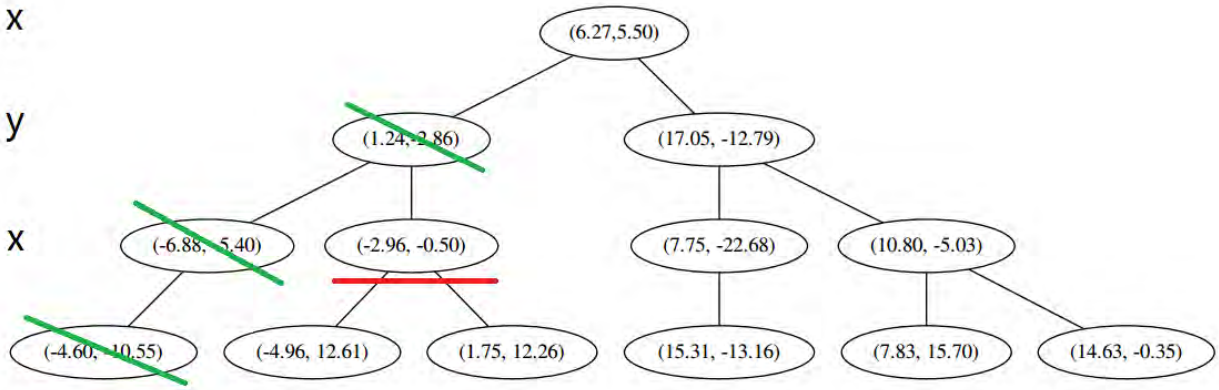


(1) 说，由于还是不够三个点，于是将当前点也记录下，有 $L = [(-4.6, -10.55), (-6.88, -5.4), (1.24, -2.86)]$。当然，当前结点变为被访问过的。

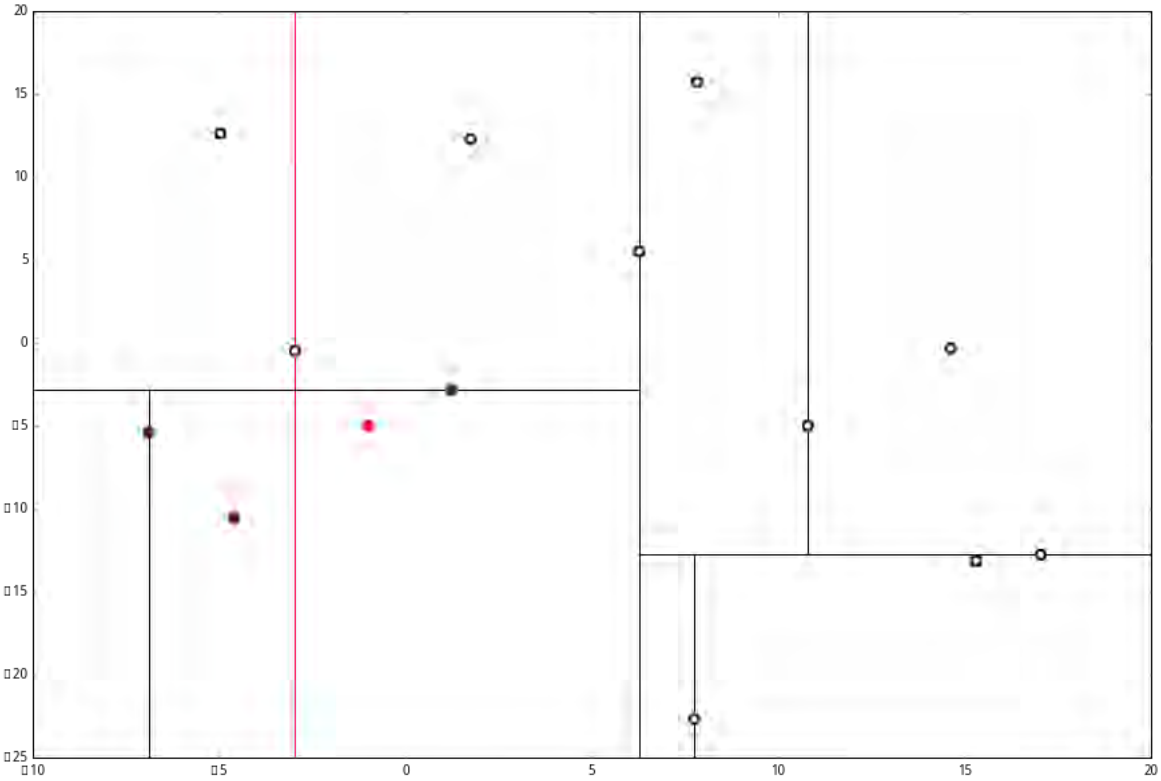
(2) 又发现，当前节点有其他的分枝，并且经计算得出 p 点和 L 中的三个点的距离分别是 $6.62, 5.89, 3.10$，但是 p 和当前节点的分割线的距离只有 2.14，小于与 L 的最大距离：



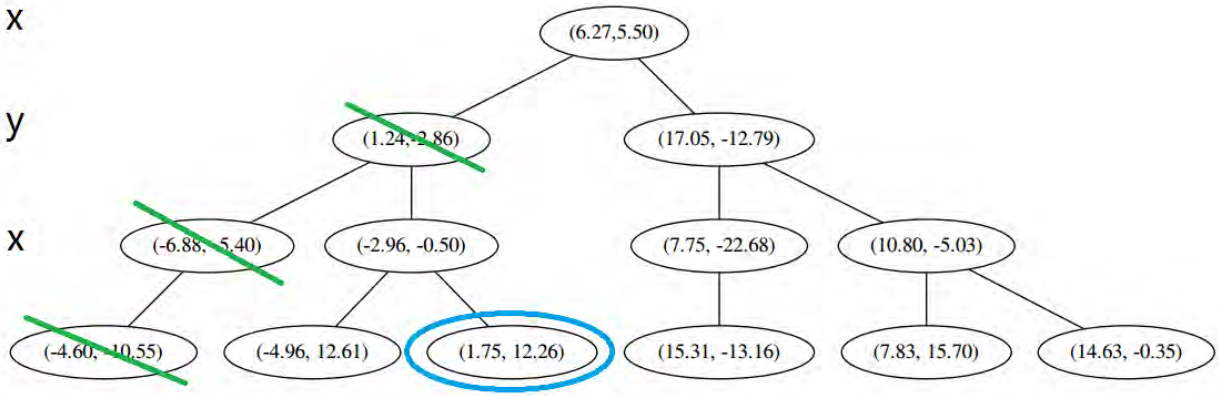
因此，在分割线的另一端可能有更近的点。于是我们在当前结点的另一个分枝从头执行 (一)。好，我们在红线这里：



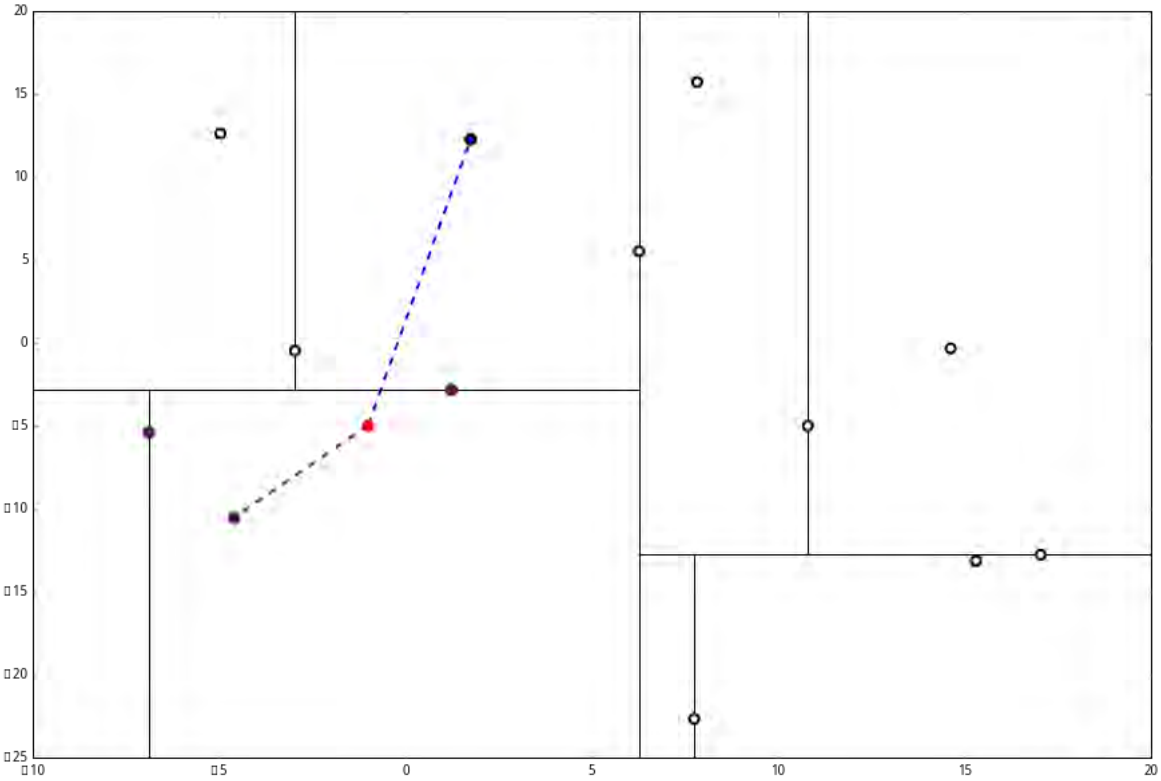
要用 p 和这个节点比较 x 坐标:



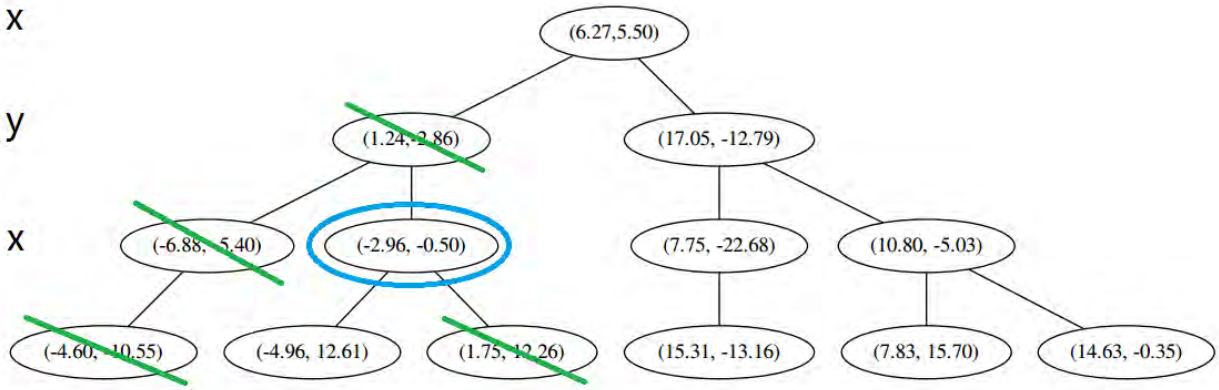
p 的 x 坐标更大，因此探索右枝 $(1.75, 12.26)$，并且发现右枝已经是最底部节点，因此启动 (二)。



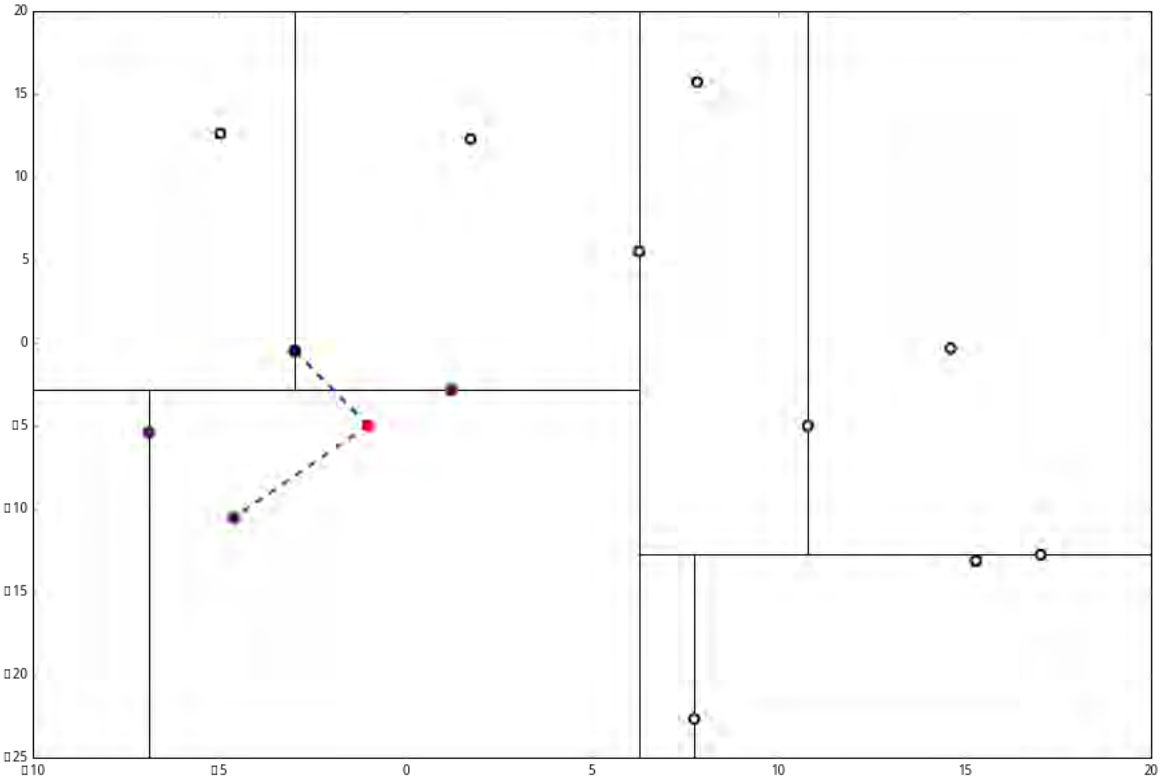
经计算，$(1.75, 12.26)$ 与 p 的距离是 17.48，要大于 p 与 L 的距离，因此我们不将其放入记录中。



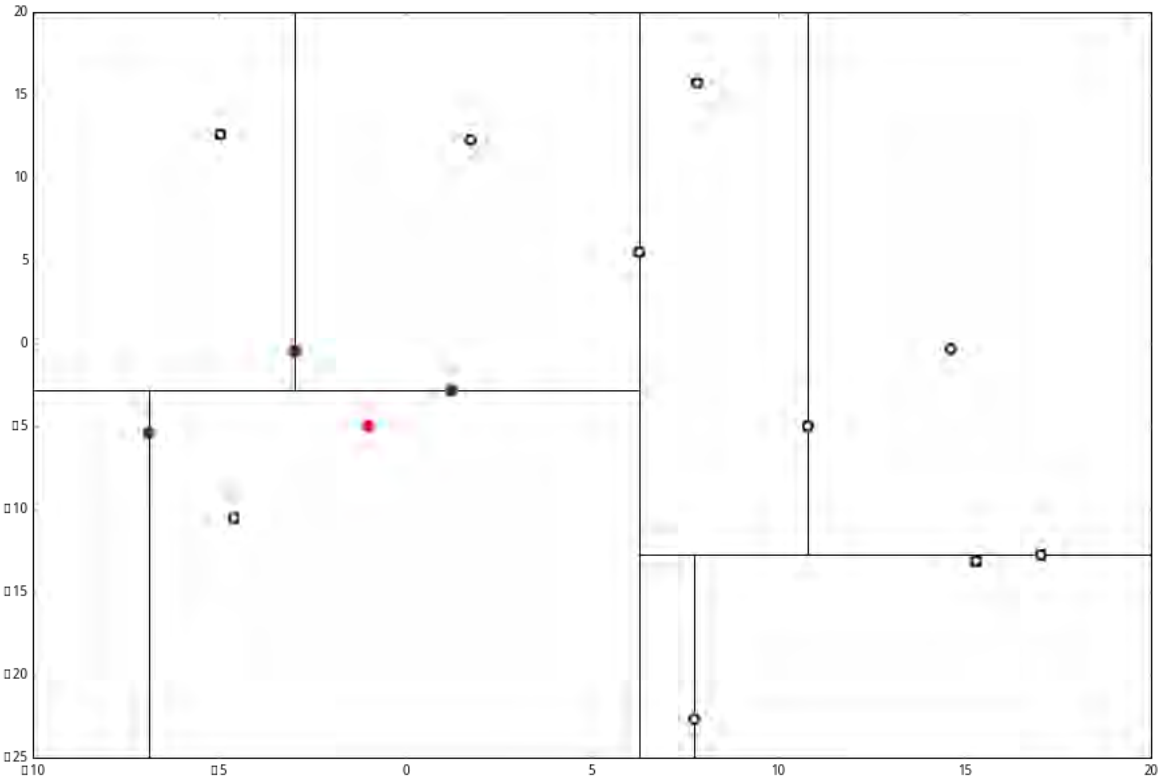
然后 (三) 判断出不是顶端节点，呼出 (a)，爬。



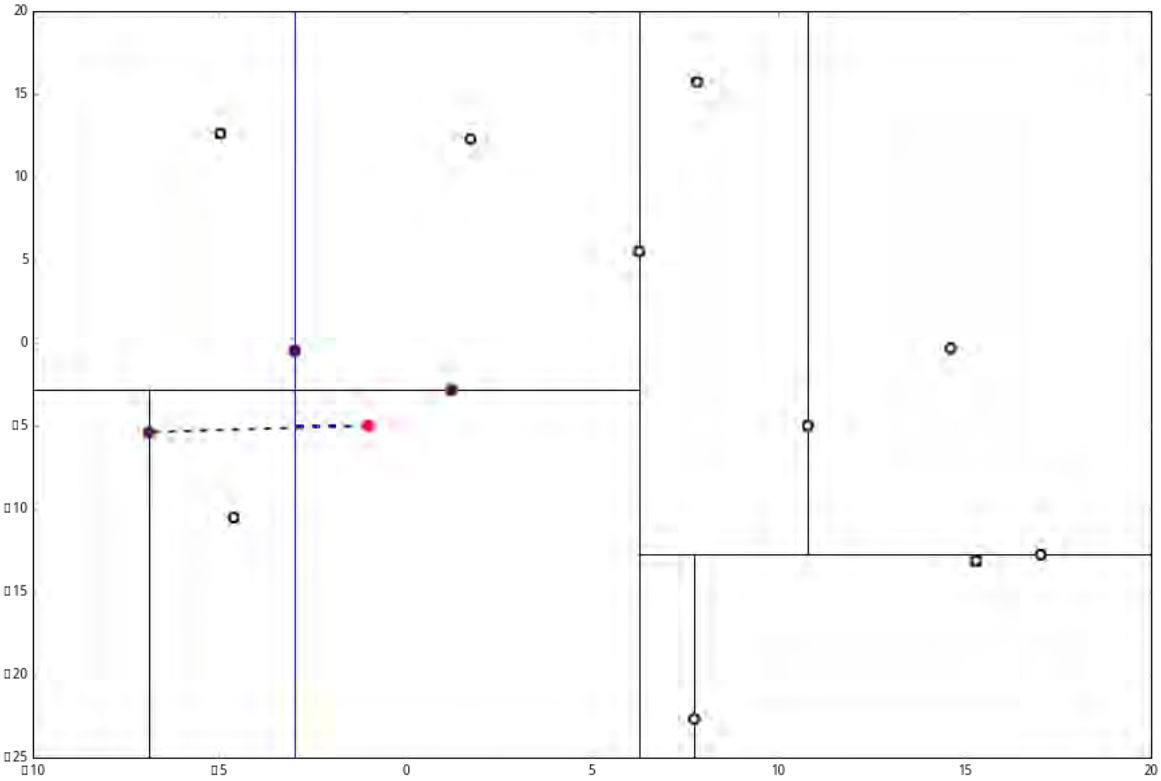
(1) 出来一算，这个节点与 p 的距离是 4.91，要小于 p 与 L 的最大距离 6.62。



因此，我们用这个新的节点替代 L 中离 p 最远的 $(-4.6, -10.55)$。

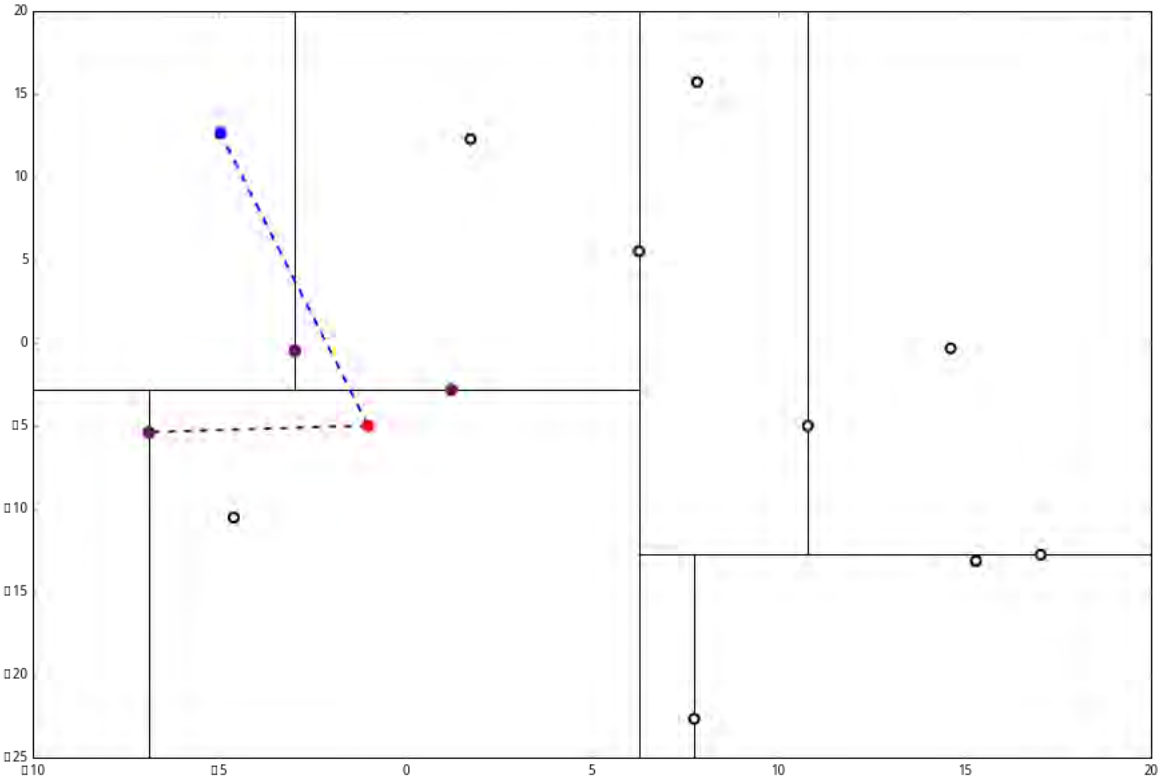


然后 (2) 又来了，我们比对 p 和当前节点的分割线的距离

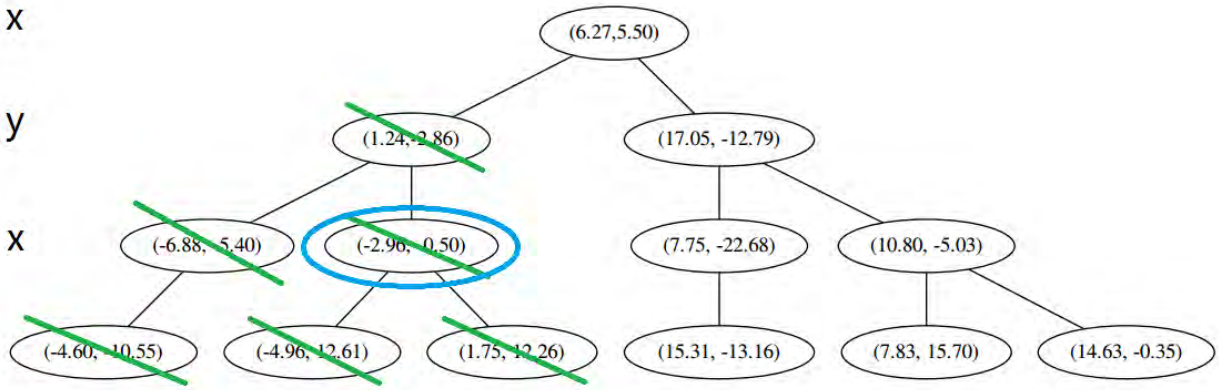


这个距离小于 L 与 p 的最小距离，因此我们要到当前节点的另一个枝执行 (一)。当然，那个枝只有一个点，直接到 (二)。

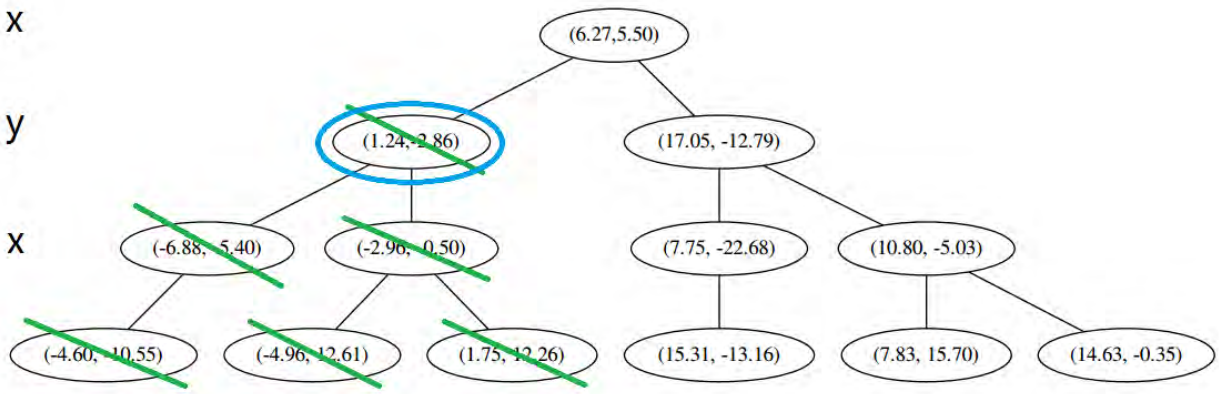
计算距离发现这个点离 p 比 L 更远，因此不进行替代。



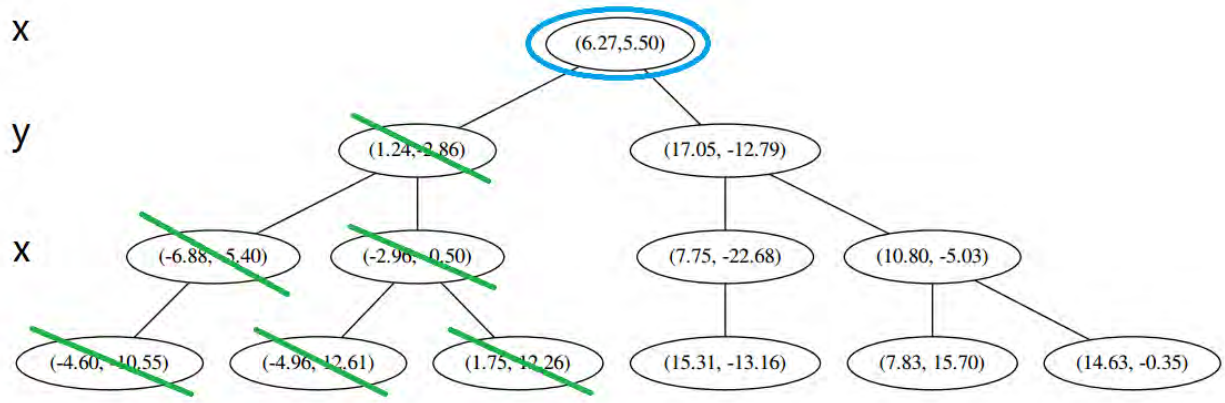
(三) 发现不是顶点，所以呼出 (a)。我们向上爬，



这个是已经访问过的了，所以再来（a），

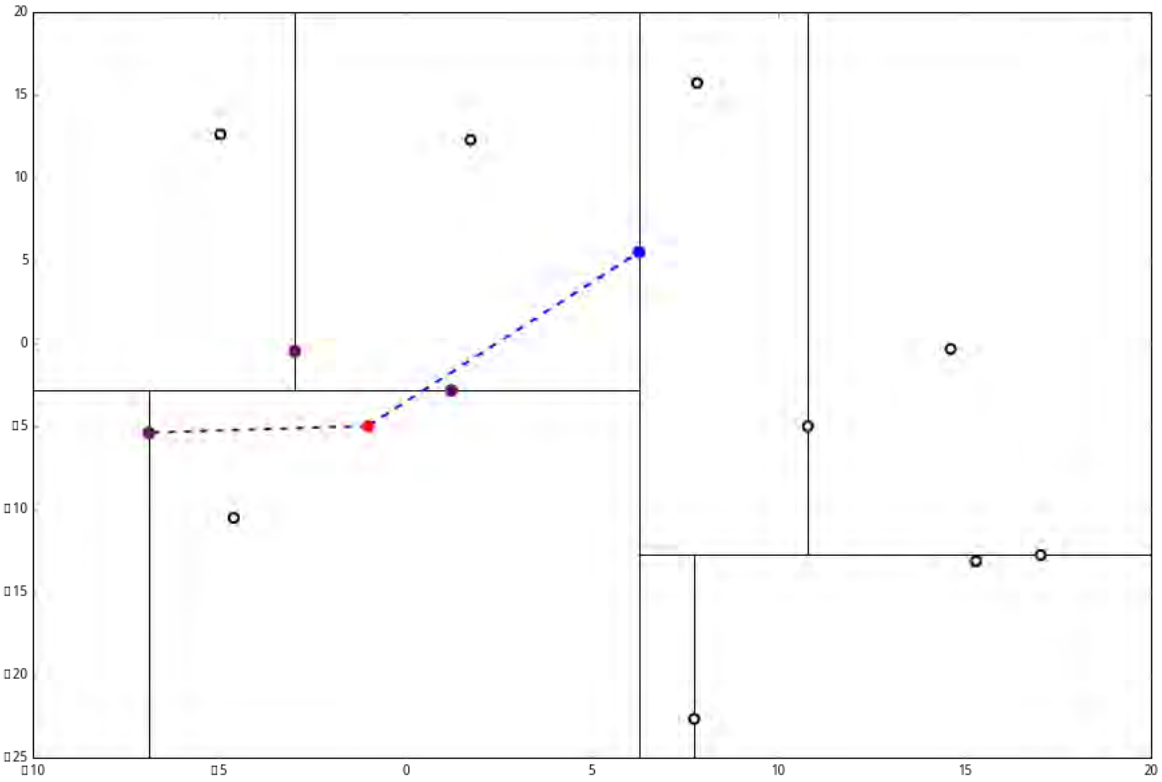


好，（a）再爬，

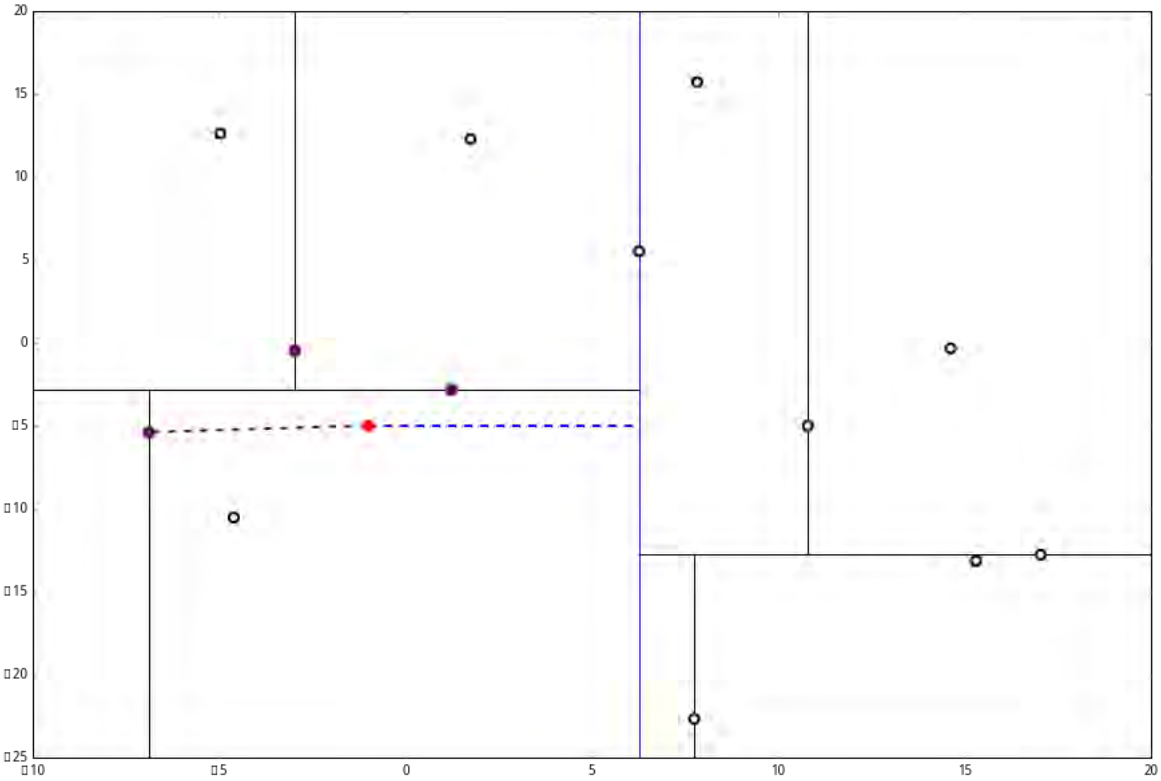


啊！到顶点了。所以完了吗？当然不，还没轮到 (三) 呢。现在是 (1) 的回合。

我们进行计算比对发现顶端节点与p的距离比L还要更远，因此不进行更新。



然后是 (2)，计算 p 和分割线的距离发现也是更远。



因此也不需要检查另一个分枝。

然后执行 (三)，判断当前节点是顶点，因此计算完成！输出距离 p 最近的三个样本是 $L = [(-6.88, -5.4), (1.24, -2.86), (-2.96, -2.5)]$。

结语

kd 树的 kNN 算法节约了很大的计算量（虽然这点在少量数据上很难体现），但在理解上偏于复杂，希望本篇中的实例可以让读者清晰地理解这个算法。喜欢动手的读者可以尝试自己用代码实现 kd 树算法，但也可以用现成的机器学习包 scikit-learn 来进行计算。量化课堂的下一篇文章就将讲解如何用 scikit-learn 进行 kNN 分类。

文章更迭记录：
v1.2，2016-11-01，修正算法，感谢 nemo1982 指出
v1.1，2016-09-14，修正错字，感谢 nico 指出
v1.0，2016-09-12，文章上线

【量化课堂】数学规划简介

导语：在一定的资源和条件限制下，将某一个目标最大化或最小化，这就是数学规划问题。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，本文的难度属于进阶（上），深度为 level-0

前言

数学规划(mathematical programming)，也称*数学优化*(mathematical optimization)，是数学中的一个分支，它主要研究的目标在给定的区域中寻找可以最小化或最大化某一函数的最优解。数学规划在几乎所有的科学领域都有着不容忽视的应用，所以一直都是一门受到着重关注和研究的学科。本文是对数学规划问题的一个浅显的介绍，不涉及任何理论和算法。

数学规划

一个数学规划问题一般具有以下形式

Unknown environment 'align'

在这里面，函数 $f : \mathbb{R}^n \to \mathbb{R}$，$c_i : \mathbb{R}^n \to \mathbb{R}$，$b_i \in \mathbb{R}$。函数 f 是我们想最小化的*目标函数*(objective function)；而函数 c_i 和常量 b_i 则对可行集构成一些限制，不等式 $c_i(x) \leq b_i$ 叫做一个*约束*(constraint)，所有满足这些约束的点的集合

$$\Omega = x \in \mathbb{R}^n : c_i(x) \leq b_i, \ \forall i = 1, \ldots, m$$

被称作这个问题的*可行区域*(fesaible region)。

我们看一下实际应用中的问题是如何以数学规划的方式表达。假设呢，现在我们有两种金融资产可以进行选择，第一个资产平均每年有 10% 的收益，但是在最坏情况下可能损失 20%；第二个资产每年平均有 5% 的收益，但最大损失也不会超过 5%。方便起见，假设这两个资产是可以无限拆分的，也就是说我们可以买 $1/2$ 个或者 $e^{\pi}/(1+\sqrt{5})$ 个。现在假设我们有本金 100，可以投资于这两个资产；我们希望在最坏的情况下损失不超过本金的 10%，也就是 10，目标嘛，当然是要最大化预期收益率。

好哟。首先确定我们的问题，是这两种资产要各配置多少钱。那么，这个问题所在的空间是 $\mathbb{R}^2$，对于里面的一个向量 x，坐标 x_1 和 x_2 分别是两种资产购买的净值。其次，我们想最大化的是预期收益，也就是

$$f(x_1, x_2) = 0.1x_1 + 0.05x_2.$$

有两个要求。第一个是购买的资产总值不可以超过起始资金 100。让 c_1 代表两种资产价值总和的函数，b_1 代表总值产 100，有

$$c_1(x_1, x_2) = x_1 + x_2 \leq 100 = b_1.$$

还有呢，最坏情况下损失不可以超过 10，那么设 c_2 为最大损失的函数，设 b_2 等于 10，有

$$c_2(x_1, x_2) = 0.2x_1 + 0.05x_2 \leq 10 = b_2.$$

好，新鲜的规划问题就出炉了：

Unknown environment 'align'

当然，也可以写成矩阵的形式

Unknown environment 'align'

格式上的细节

你说，啊，教程里都是骗人的。说好了的最小化 $f(x)$，为什么上面的问题是最大化呢?

---不，我才不会骗小孩。

上面的问题是可以转换成最小化问题的，只需要在目标函数前加上一个负号，最大化 $f(x)$ 和最小化 $-f(x)$ 是等同的，如果 $(x_1^{<}em>, x_2^{<}/em>)$ 是 $-f$ 的最小解，那么它也必定是 f 的最大解。

同理的，如果有大于号的限制

$$c_i(x) \geq b_i,$$

加上负号就可以转换为等同的小于号限制

$$-c_i(x) \leq -b_i.$$

另外，如果我们需要等于号的限制

$$c_i(x) = b_i,$$

也可以将它转换为一个大于一个小于两个限制的交集

Unknown environment 'align'

第二个大于有可以转变成小于

Unknown environment 'align'

同样变成了上面的标准格式。

经过这些转换的技巧，标准格式

Unknown environment 'align'

实际上可以表达各种各样的问题。对于单独的问题，我们一般会以最方便的方式写出来，用大于号、等于号或最大化都没有关系；但在整体地研究优化理论时，一般会用统一的问题格式，这样研究和推导的过程会更方便。

规划问题的分类

数学规划问题可以分为很多很多种类，这里我们将介绍四个最常用到的类别。这四类问题并不是并列关系，而是（很大程度上的）包含关系，其中更小的类别会包含更少的问题，但是由于这些问题普遍有着同样的结构和性质，在这个类别中会有一致受用高效算法；而更大的类别会包含更多的优化问题，但是由于缺乏良好的函数结构，解决起来一般会更麻烦。

线性规划

如果一个规划问题的目标函数 f 和约束函数 c_i 都是线性函数，我们说它是一个*线性规划*(linear programming)问题。线性规划都可以写成矩阵的形式：

Unknown environment 'align'

这里，$c \in \mathbb{R}^n$，$A \in \mathbb{R}^{m \times n}$ 并且 $b \in \mathbb{R}^m$。第2节中举的例子就是一个线性规划问题。

线性函数是在实数域上最容易分析的一类函数，它们有很强的结构性，直线嘛，从哪里看都是一模一样的。也正是因为如此，解决线性规划问题也相对简单，所有线性规划问题都可以在多项式时间内找到最优解。

二次规划

满足以下格式的规划问题都被称作*二次规划*(quadratic programming)：

Unknown environment 'align'

这里，$Q \in \mathbb{R}^{n \times n}$ 是一个对称矩阵（也就是说 $Q^{\mathsf{T}} = Q$），$c \in \mathbb{R}^n$，$A \in \mathbb{R}^{m \times n}$ 还有 $b \in \mathbb{R}^m$。二次规划比线性规划多出了一个 $x^{\mathsf{T}} Q x$ 项，也是由矩阵的乘积表示的，所以虽然不是线性函数，但是仍可以使用很多线性代数的技巧进行分析。二次规划问题的难度取决于矩阵 Q：如果 Q 是正定矩阵（所有的特征值都大于 0），那么可以在多项式时间内找到最优解；如果 Q 是不定的（有小于 0 的特征值），寻找最优解一般是 NP-难的。

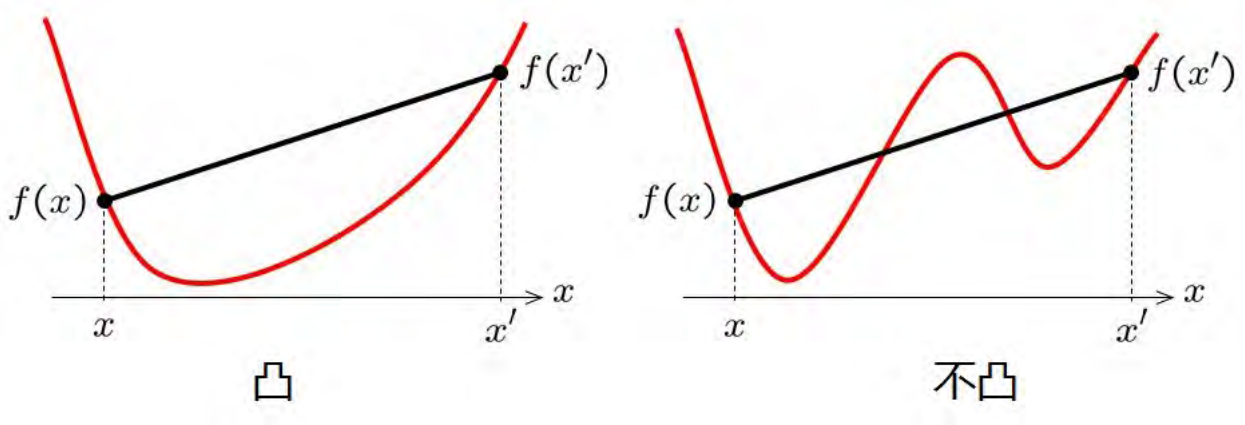
应该指出，任何一个线性规划问题都是二次规划问题 --- 将二次项的 Q 设成零矩阵即可。

凸规划

讲到凸规划，我们应该先了解凸函数什么。如果 $f : \mathbb{R}^n \to \mathbb{R}$ 是一个连续函数，并且对于任何 Misplaced & 以及 $\lambda \in [0, 1]$，满足

Misplaced &

我们说 f 是一个*凸函数*(convex function)。在直观的理解上，就是在这个函数的图上的任意连点之间连一条直线，这个函数不会超过这条直线。



那么，如果在一个规划问题中，

Unknown environment 'align'

函数 f 和 c_i 都是凸的，我们说它是一个*凸规划*(convex programming)问题。凸函数有什么好处呢，我们举一个形象的例子：在上面的两个函数曲线（红色）上，我们在 Misplaced & 的位置放一个圆珠，让它自己滚下去。在第一个凸函数里，圆珠会一路滚下去并停在整个函数的最低点；而在第二个不凸的函数里，圆珠会停在右边的局部低点里，却到不了全局的最低点。从这可以看出，如果 f 是凸函数，寻找它的最低点会更容易。但即便如此，很多凸规划问题的最优解还是 NP-难的。

对于一个二次规划问题，如果二次项的矩阵 Q 是半正定的，那么它也是一个凸规划问题；但是如果 Q 是不定的，那么二次目标函数并不是一个凸函数。

非线性规划

非线性规划是一个非常大的问题类型，一般来说，如果规划问题中

Unknown environment 'align'

函数 f 和 c_i 都是连续的，它就是一个*非线性规划*(non-linear programming)问题。也许这个类别的名字起得有点不恰当，因为线性规划问题其实也属于非线性规划问题，而这个“非”字可以这么理解：线性规划是非线性规划中极其小的一部分，如果我们有一个线性规划问题，那么用线性规划的方法就可以很简便地解决，并不需要非线性规划的理论，所以非线性规划实际研究的是线性规划以外的问题。

非线性规划的涉及面太广，以至于很多未解的数学难题都可以写做为非线性规划的形式，可见普遍的非线性规划问题有多难。因此，在研究非线性规划问题时，我们一般会加上一些额外的条件限制，比如：目标函数 f 是可导的或是平滑的、f 或 f 的导数是 Lipschitz 的、问题的可行区域是凸的，等等。不难看出，上面的所有其他类别的规划问题都属于非线性规划问题。

结语

本文概括性地介绍了数学规划的基本内容和一些问题的分类。在量化课堂未来的文章中，我们将介绍各类规划问题的数学理论、解题算法以及应用方法，敬请期待。

文章迭代记录
v1.1，2016-11-08，修正逻辑问题，感谢 pigque 指出。
v1.0，2016-10-18，文章上线。

【量化课堂】无约束的非线性规划问题 -- 线搜索方法

导语：非线性规划是一种应用场景甚广的优化问题，从神经网络训练的反向传播，到资产组合的配置优化，都离不开寻找非线性函数的极小点的技巧。本文将讲解无约束非线性规划的基本思路和一些简单的算法。

作者：肖睿
编辑：宏观经济算命师

阅读本文需要掌握线性代数和多元微积分的知识。
建议读者对数学规划有基本了解。

无约束非线性规划问题

无约束的非线性规划问题具有以下形式

$$\min_{x\in\mathbb{R}^n} f(x).$$

这里 $f:\mathbb{R}^n\to\mathbb{R}$ 是一个连续函数。但只是单纯的连续函数是很难被优化的，我们一般会要求函数 f 满足 $\mathcal{C}^1$ 或者 $\mathcal{C}^2$ 的性质，这些性质在之后会有定义。

这种问题叫做“无约束”因为它的可取点是任何 $x\in\mathbb{R}^n$；相对应的，有约束的规划问题一般有以下形式

Unknown environment 'align'

这里 $g_1,\dots,g_m$ 是一些约束函数，而 $b_1,\dots,b_m$ 是一些实数。

在解决非线性规划问题时，我们最想做的是计算出一个全局极小点。什么是全局极小点呢?
定义. 一个函数 f 的*全局极小点*（global minimizer）是一个 $x^\in\mathbb{R}^n$，其满足对于任何一个 $x\in\mathbb{R}^n$ 都有 $f(x^)\le f(x)$。
但是找出函数的全局极小点只是我们的理想，这其实是非常困难的，而找出一个局部极小点是我们能做的最好的尝试。
定义. 一个函数 f 的*局部极小点*（local minimizer）是一个 $x^\in\mathbb{R}^n$，其满足，存在 $\mathbb{R}^n$ 中的一个开集 $\mathcal{N}\ni x^$，对于所有 $x\in\mathcal{N}$ 有 $f(x^)\le f(x)$。这个定义有时也称作弱局部极小点（weak local optimizer），因为它允许在局部内有和它取值相同的点。
定义. 一个函数 f 的*强局部极小点*（strong local optimizer）是一个 $x^\in\mathbb{R}^n$，其满足，存在 $\mathbb{R}^n$ 中的一个开集 $\mathcal{N}\ni x^$，对于所有 $x\in\mathcal{N}$ 并且 $x\neq x^$ 有 $f(x^)<f(x)$。
不过我还有一个坏消息，其实局部极小点我们可能也找不到。嗯，不行。但是我们可以做到的，是找到一个足够接近局部极小点的向量，以至于 f 在该位置的取值非常接近实际的局部最小值。这就是本文中介绍的算法要达成的目标。

定义和符号

在一切开始之前，我们有必要讲清楚文中用到的数学符号和它们的定义。首先是关于微分的三个定义：

定义. 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一个连续函数，用 $x_1,\dots,x_n$ 表示 $\mathbb{R}^n$ 的标准基。如果对于所有满足 $l_1+\dots,+l_n=m$的 $l_1,\dots,l_n\ge 0$，m 阶导数 $\frac{\partial^m f}{\partial x_1^{l_1}\dots\partial x_n^{l_n}}$ 存在并且是连续的，我们说 f 是一个 $\mathcal{C}^m$ 函数。

定义. 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一个 $\mathcal{C}^1$ 函数，那么 f 的\emph{梯度(gradiet)}是一个连续函数 $\nabla f:\mathbb{R}^n\to\mathbb{R}^n$，定义为

$$\nabla f(x):=\left[\frac{\partial f}{\partial x_1}(x)\ \frac{\partial f}{\partial x_2}(x)\vdots\frac{\partial f}{\partial x_n}(x)\right],\ \forall x\in\mathbb{R}^n.$$

高维函数的梯度和在一维函数的导数是同等的概念，它告诉我们一个函数在某点向各个方向的斜率是多少，通过这个信息我们可以寻找坡度向下的方向进行优化。
定义. 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一个 $\mathcal{C}^2$ 函数，那么 f 的*海塞矩阵*（Hessian matrix）是一个连续函数 $\nabla^2 f:\mathbb{R}^n\to\mathbb{R}^{n\times n}$（有时也写作 H_f），定义为

Undefined control sequence \<

然后是线性代数中（半）正定矩阵的定义：
定义. 对于一个 $n\times n$ 矩阵 $A\in\mathbb{R}^{n\times n}$，如果 $A=A^{\mathsf{T}}$ 并且对于每一个向量 $x\in\mathbb{R}^n$ 有 $x^{\mathsf{T}}Ax\ge 0$，我们说 A 是一个*半正定矩阵*（positive semi-definite matrix）。
定义. 对于一个 $n\times n$ 矩阵 $A\in\mathbb{R}^{n\times n}$，如果 $A=A^{\mathsf{T}}$ 并且对于每一个向量 $x\in\mathbb{R}^n\backslash 0$ 有 Misplaced &，我们说 A 是一个*正定矩阵*（positive definite matrix）。
下面是两个至关重要的定理，读者也可以在微积分的教材里找到它们。

两个定理

首先是一个在多元微积分中很常见的定理，它告诉我们如何判别一个函数的（强）局部极小点。

定理 1. 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一个 $\mathcal{C}^2$ 函数，并且设 $x^\in\mathbb{R}^n$。如果 $\nabla f(x^)=0$ 并且 $\nabla^2 f(x^)$ 是一个半正定矩阵，那么 $x^$ 是 f 的一个局部极小点；如果 $\nabla f(x^)=0$ 并且 $\nabla^2 f(x^)$ 是一个正定矩阵，那么 x^* 是 f 的一个强局部极小点。

你也许想到，啊哈！我们用这个定理的结论不就可以解决最小化问题了吗？为什么说不能找到极小点？

那是因为，找到一个函数（在这里就是梯度 ∇f）的零集其实也是千古难题，有时我们甚至会用非线性优化问题来估测一个函数等于零的位置。所以呐，还是老老实实地继续往下看吧。

下面的泰勒定理可以帮助我们分析函数在一个局部里的表现，方便我们找到极小点。

定理 2.（泰勒定理） 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一个 $\mathcal{C}^1$ 函数，并设 $p\in\mathbb{R}^n$。那么存在某个 $t\in(0,1)$，有

$$f(x+p)=f(x)+\nabla f(x+tp)^{\mathsf{T}}p.$$

并且，如果 f 是 $\mathcal{C}^2$ 函数，那么存在某个 $t\in(0,1)$，有

$$f(x+p)=f(x)+\nabla f(x)^{\mathsf{T}}p+\frac{1}{2}p^{\mathsf{T}}\nabla^2 f(x+tp)p.$$

这个定理告诉我们，选定 $\mathbb{R}^n$ 上的一个点 x 和一个方向 p，函数 f 从 x 到 $x+p$ 的变化等于 f 在某一点的梯度和向量 p 的点积，而这个点总能在 x 和 $x+p$ 之间找到。

依照这个定理，我们还可以在仅知 $f(x)$ 和 $\nabla f(x)$ 的情况下在局部内线性估测 f 的值。我们已知对于任何 $p\in\mathbb{R}^n$，有

$$f(x+p)=f(x)+\nabla f(x+tp)^{\mathsf{T}}p,$$

可是每一个 p 对应着不同的 t，用这个方法来计算 $f(x+p)$ 并不方便。但是，如果我们允许一定误差的话，可以用 $\nabla f(x)$ 代替 $\nabla f(x+tp)$ 来进行估测，有

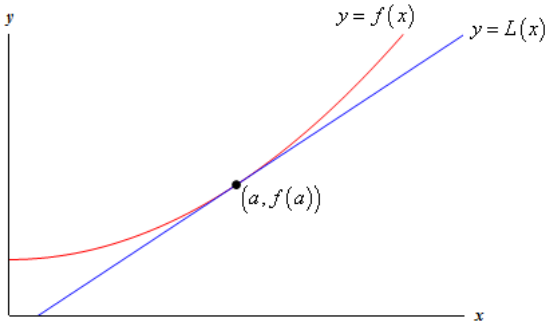
$$f(x+p)\approx f(x)+\nabla f(x)^{\mathsf{T}}p.$$

因为

$$\left[f(x)+\nabla f(x)^{\mathsf{T}}p\right]-\left[f(x)+\nabla f(x+tp)^{\mathsf{T}}p\right]=(\nabla f(x)-\nabla f(x+tp))^{\mathsf{T}}p,$$

所以如果 p 足够小，那么（由于 ∇f 是连续的）$\nabla f(x)-\nabla f(x+tp)$ 的差也很小，以至于等式右侧的误差同样很小。

从基础微积分的知识中，我们知道如果 f 是一个一维函数，那么上面的做法是以 $(x,f(x))$ 为中心画出了一条切线（也就是一阶泰勒展开）



这条切线在一定范围之内和实际函数非常接近。在更高维度中，函数 $\phi(p)=f(x)+\nabla f(x)^{\mathsf{T}}p$ 的图像则构成以 $(x,f(x))$ 为中心的一个超切面。

泰勒定理中的第二个公式同样给了我们估测方法

$$f(x+p)\approx f(x)+\nabla f(x)^{\mathsf{T}}p+\frac{1}{2}p^{\mathsf{T}}\nabla^2 f(x)p.$$

这个用到了二阶项的估值会比之前的估测更准确；使用这个估值的要求更高一点，需要 f 是 $\mathcal{C}^2$ 函数（在实际应用中基本都可以满足）。

泰勒定理为我们提供了一种搜索一个函数极小点的方法。假设 x 不是函数 f 的局部极小点并且我们知道 f 在一个点 x 的取值 $f(x)$ 以及梯度 $\nabla f(x)$，如果选得一个方向 $p\in\mathbb{R}$ 满足 $\nabla f(x)^{\mathsf{T}}p<0$，那么存在某个 $\alpha>0$，使得 $f(x+\alpha p)<f(x)$。如此重复，在第 k 次迭代是从 xk开始，找到合适的 p_k 和 α_k 并更新 $x<k+1=x_k+\alpha_k p_k$，这样，让 $f(x_k)$ 的取值越来越小，以收敛到一个局部极小点。

以上的方法叫做*线搜索*（line search），是解决非线性规划问题的一类主要方法。线搜索的主要问题在于要寻找合适的*方向*（direction）p_k 和*步长*（step length）α_k。

方向

在线搜索的每一次迭代中，我们想要方向 p_k 满足 $\nabla f(x_k)^{\mathsf{T}}p_k<0$。我们知道向量 $\nabla f(x_k)$ 和 p_k 之间的角度 θ_k 满足

$$\cos\theta_k = \frac{\nabla f(x_k)^{\mathsf{T}} p_k}{|\nabla f(x_k)||(p_k)|},$$

因此符合条件的 p_k 也恰恰是和梯度 $\nabla f(x_k)$ 的角度大于 90° 的向量。下面介绍选取方向 p_k 的几个常用的方法。
梯度下降

最符合直觉的方向是选择 f 下降幅度最大的方向。也就是说，固定 p 的长度不变，我们想找到最小的 $\nabla f(x_k)^{\mathsf{T}} p$。于是乎，我们解决一个简单的最小化问题

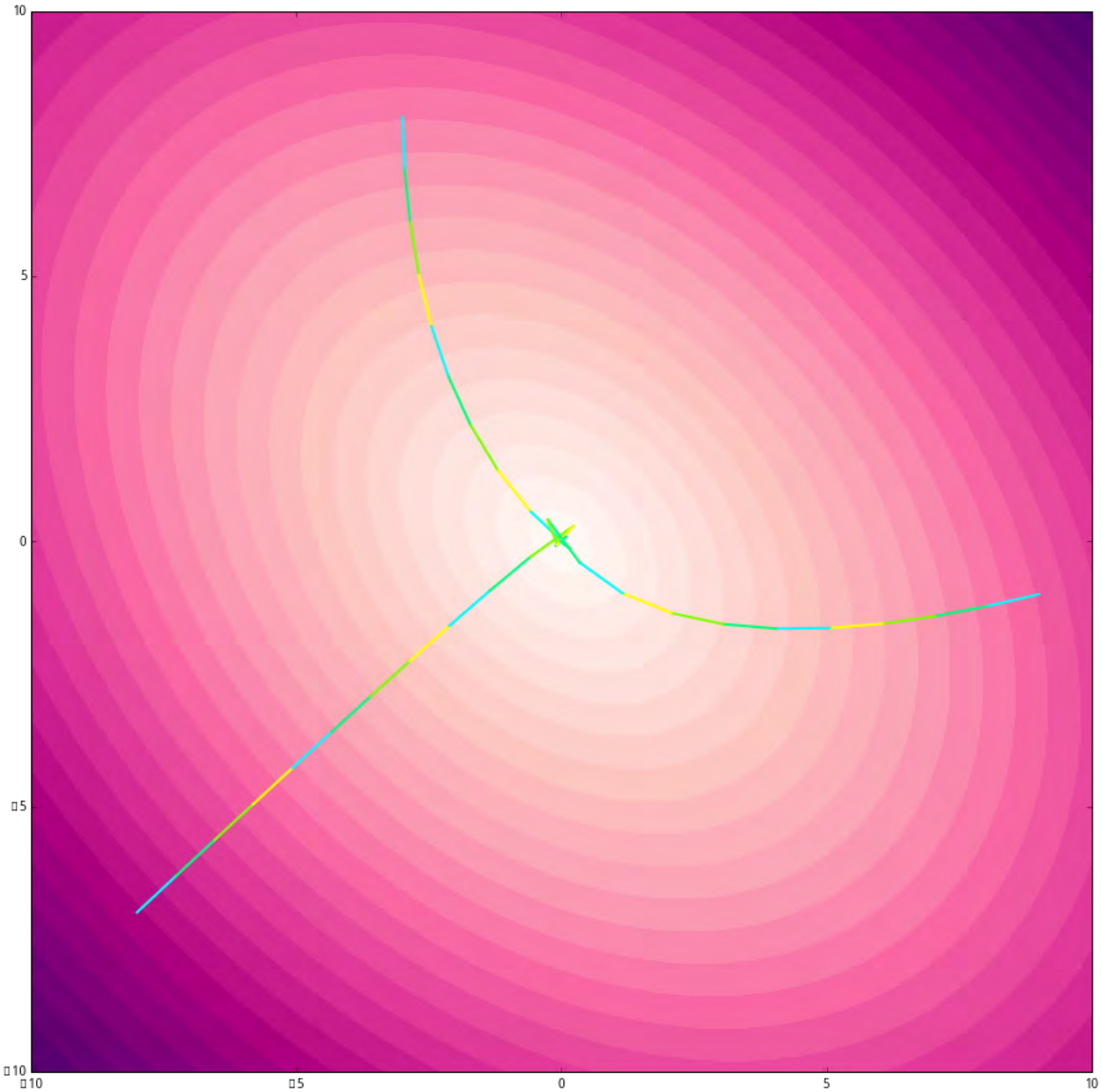
Unknown environment 'align'

用 θ 表示 p 和 $\nabla f(x_k)$ 之间的角度，有

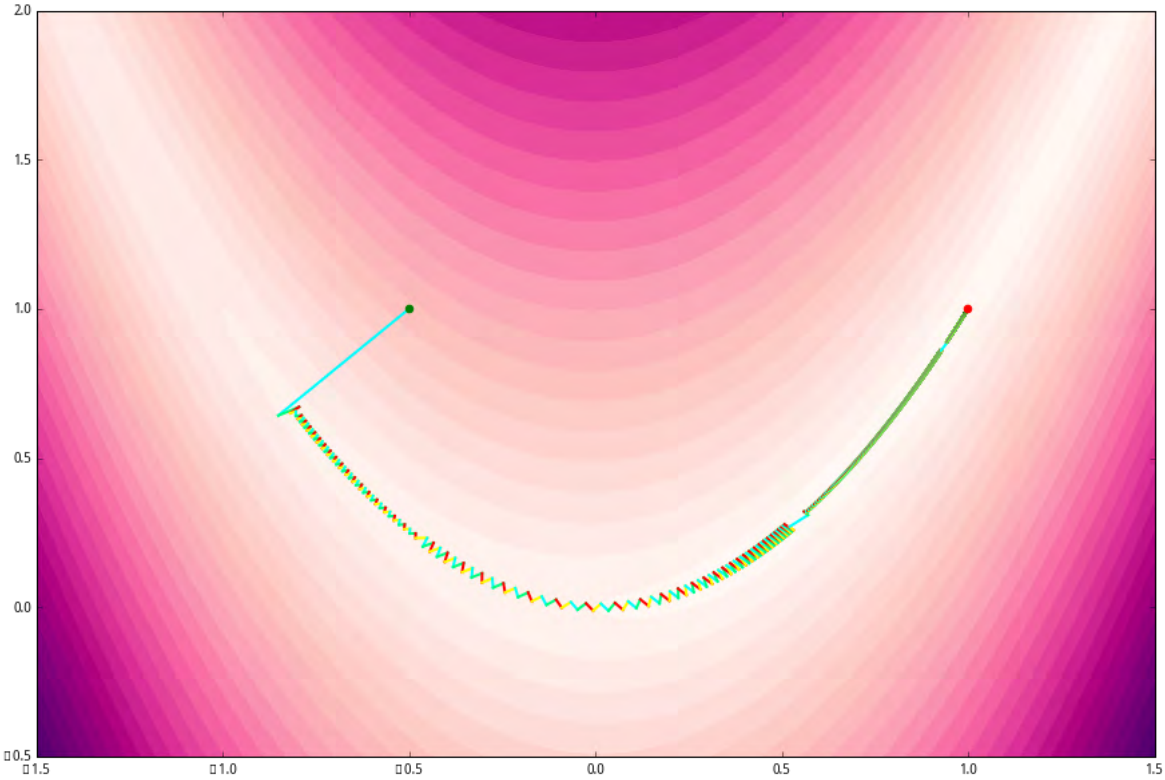
$$\nabla f(x_k)^{\mathsf{T}} p = |\nabla f(x_k)||p| \cos\theta = |\nabla f(x_k)| \cos\theta.$$

很明显，当 $\cos\theta = -1$ 时这个值被最小化，对应向量 $p^\star = -\nabla f(x_k)/|\nabla f(x_k)|$。因此，作为最大下降的方向，可以选择 $p_k = -\nabla f(x_k)$，这个方向一般称为*梯度下降*（gradient descent）或者*最大下降*（steepest descent）。

最大下降方法的优点有两个。一是它保证全局的收敛性：不论我们的起始点 x_0 在哪里，反复的迭代都会收敛向一个局部极小点。第二是它对函数 f 的要求很小，只需要 f 是连续可导的。但同时，它的缺点也很明显，那就是它在很多情况下的收敛速度很慢。最大下降的方向与 f 在 x_k 的水平集 $x \in \mathbb{R}^n : f(x) = f(x_k)$程垂直关系，因此根据起始点的位置，很可能在迭代中画出锯齿形的低效率路径。

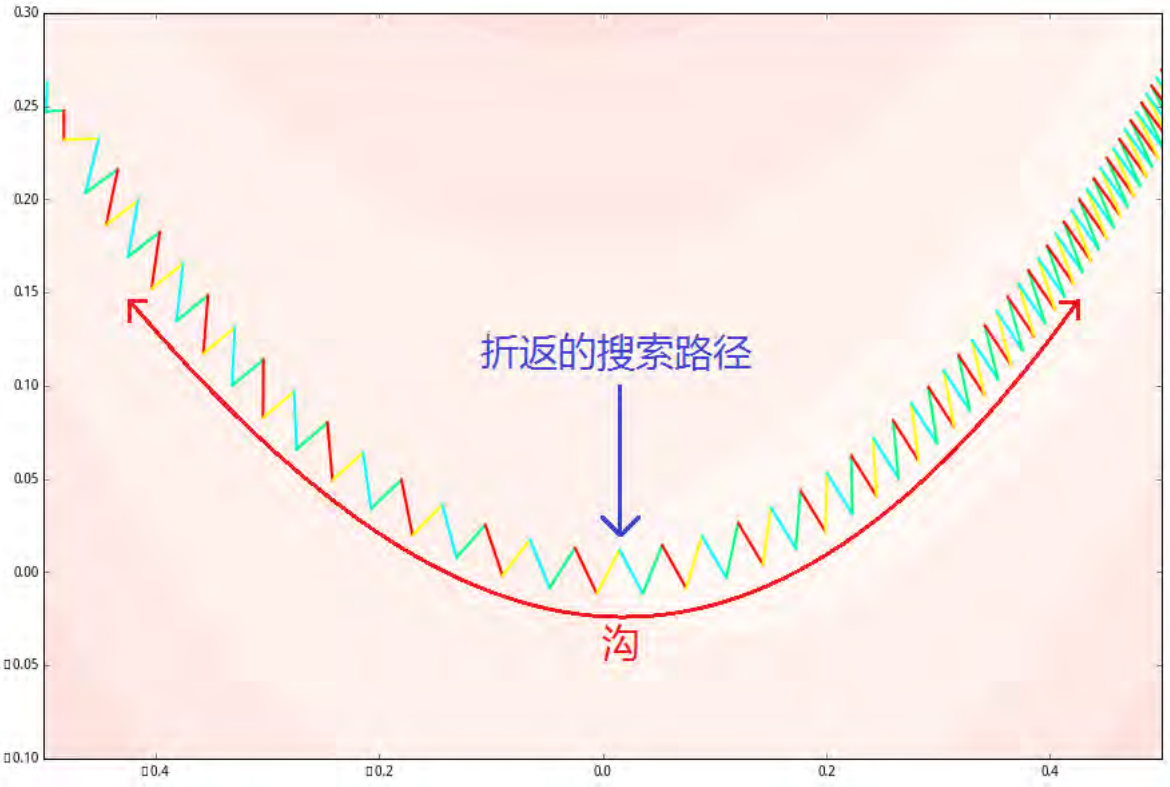


上图是一个二次函数 $Q(x) = 3x_1^2 + 2x_1x_2 + 3x_2^2$ 的热力图，它的最小值在 $(0,0)$ 的位置。图中的三根曲线是从三个不同的点起始的梯度下降方法的搜寻路径，线段每变换一次代表一次迭代，其中每一条曲线都在 11 到 14 次迭代中找到了距离极小点误差小于 0.00001 的位置。当然了，二次函数一般都比较规整，梯度下降算法的表现还算不错，下面的例子就不太好了。



上图中的是 Rosenbrock 函数 $R(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$，它的极小点在 $(1, 1)$（图中红点）。可以在热力图上看出这个函数的表现不是那么友好，它挖了一个又弯又窄的沟，函数的梯度指向的不是沟的下游而是沟的中央，所以梯度下降法像在编小辫一样地反复地折返，用了 5491 次迭代才从 $(-0.5, 1)$（图中绿点）达到与极小点的误差小于 0.00001 的位置。

我们放大算法在沟里搜索的路径，足以看出它多没有效率。



那碰上这种奇葩的函数我们该怎么办呢怎么办呢？往下看。

牛顿方法

另一个重要的衰减方向（可能可以说是最重要的）是*牛顿方向*（Newton's direction）。记得在泰勒定理中，二次可导的函数 f 在 x_k 附近的二次泰勒估值是

$$f(x_k + p) \approx f(x_k) + \nabla f(x_k)^{\mathrm{T}} p + \frac{1}{2} < br > p^{\mathrm{T}} \nabla^2 f(x_k) p =: m_k(p)$$

（这里 $=: m_k(p)$ 是指我们把 $m_k(p)$ 定义为等号左边的表达式）。暂时假设 $\nabla^2 f(x_k)$ 是一个正定矩阵，我们想将二次估值 $m_k(p)$ 最小化，那么求一阶导数得到

$$\nabla m_k(p) = \nabla f(x_k) + \nabla^2 f(x_k)p,$$

将它设为 0，求得

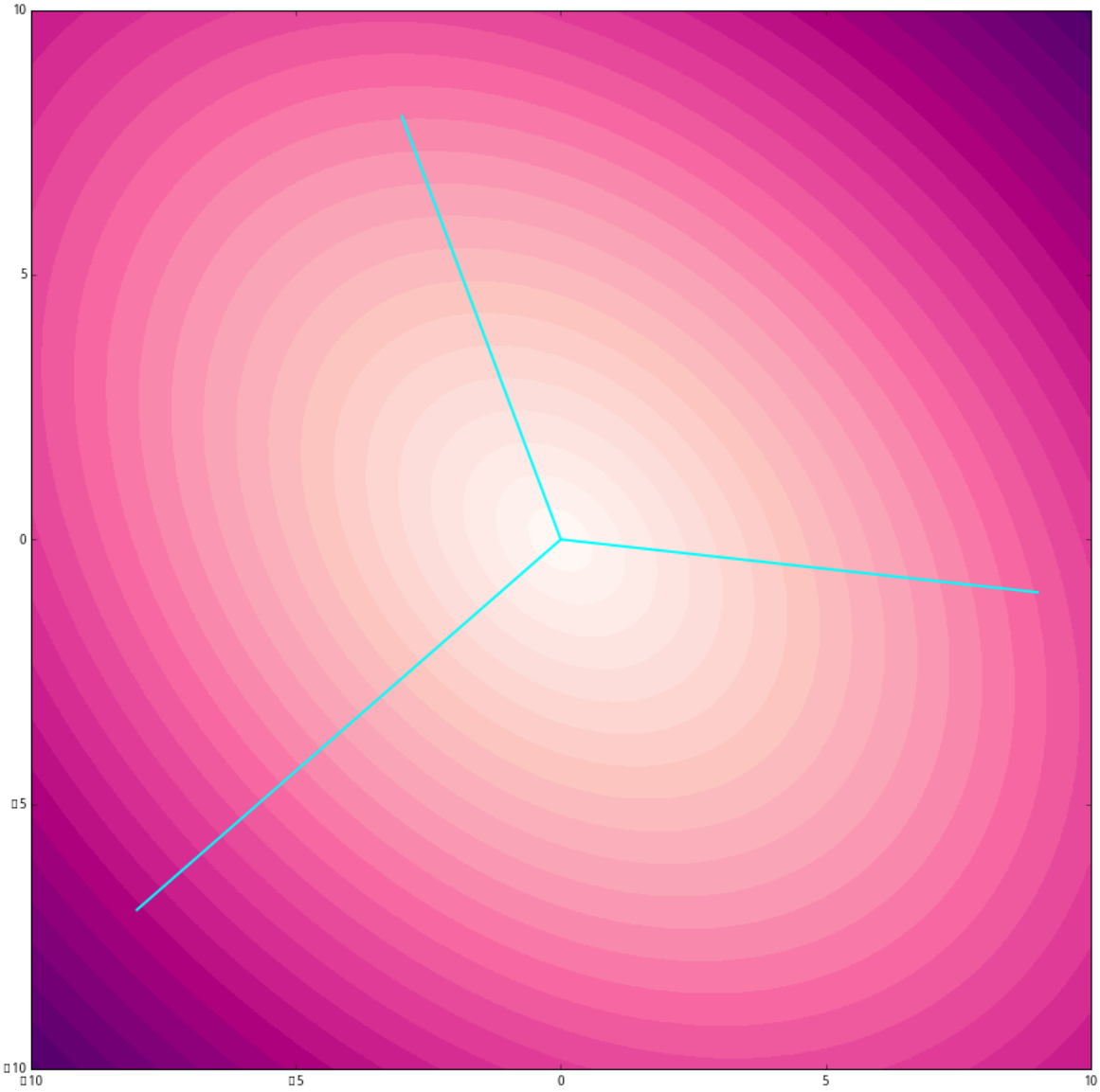
$$p_k = -\nabla^2 f(x_k)^{-1}\nabla f(x_k).$$

因为 m_k 的二次导数是 $\nabla^2 f(x_k)$，在假设中是正定矩阵，根据定理 1.，上面所求 p_k 确定是极小点。

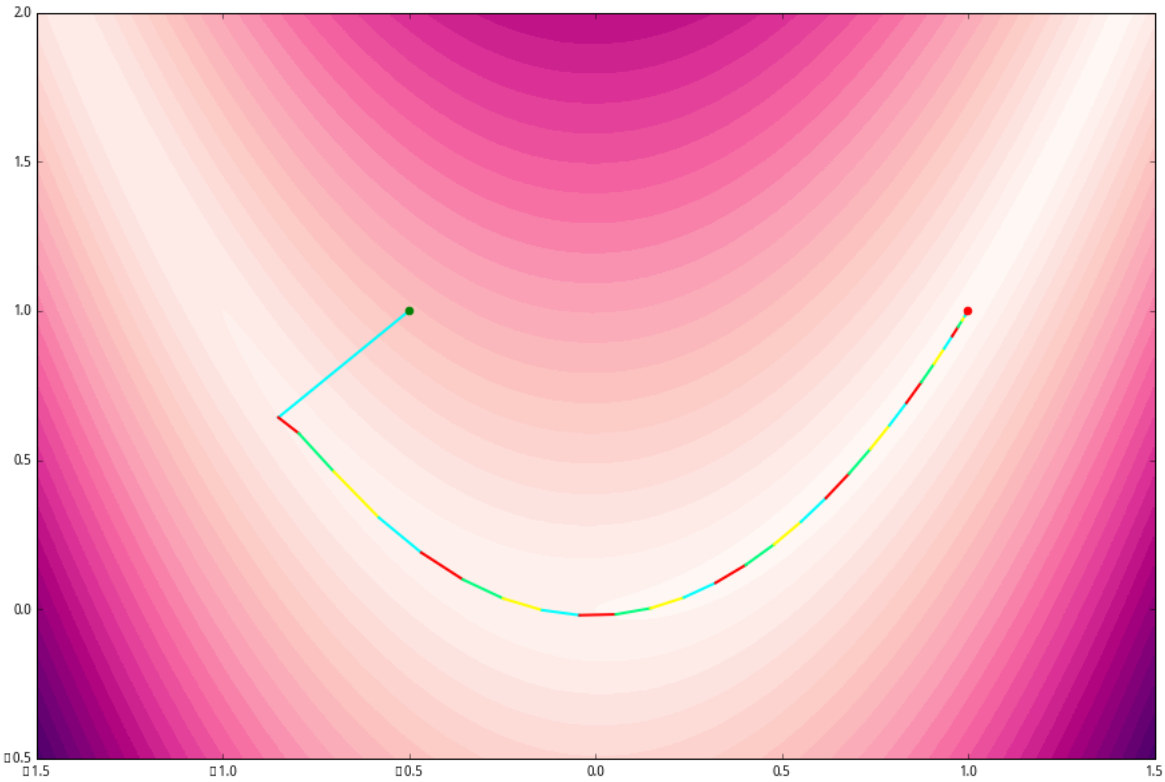
理论上来说，牛顿方法的收敛速度要比梯度下降更快，因为梯度下降使用的是一次展开进行来估测方向，误差一般在 $O(|p_k|^2)$；而牛顿方法使用二阶展开，误差一般在 $O(|p_k|^3)$，所以选择的方向会更准确。可以理解为，牛顿方法选择的方法更有“远见”，因为梯度下降的方向虽然在眼前的下降速度是最快的，但是在跑出一定距离之后就会比牛顿方向的表现差，所以要重新选择方向的次数比牛顿方法更多。

当然，牛顿方法的缺点也比较明显：首先它需要函数 f 是二次可导的，并且需要在 x_k 的海塞矩阵 $\nabla^2 f(x_k)$ 是正定的（这一般需要起始点 x_0 距离极小点足够近）。再者，即使 $\nabla^2 f(x_k)$ 是正定的，方向 p_k 也不一定是降低方向，有可能 Misplaced & 。而且计算 f 的海塞矩阵再求它的逆矩阵有时也需要很大的计算量。牛顿方法的种种缺陷都有相应的解决方法，本文中不进行探索，可以期待未来的文章中的讲解。

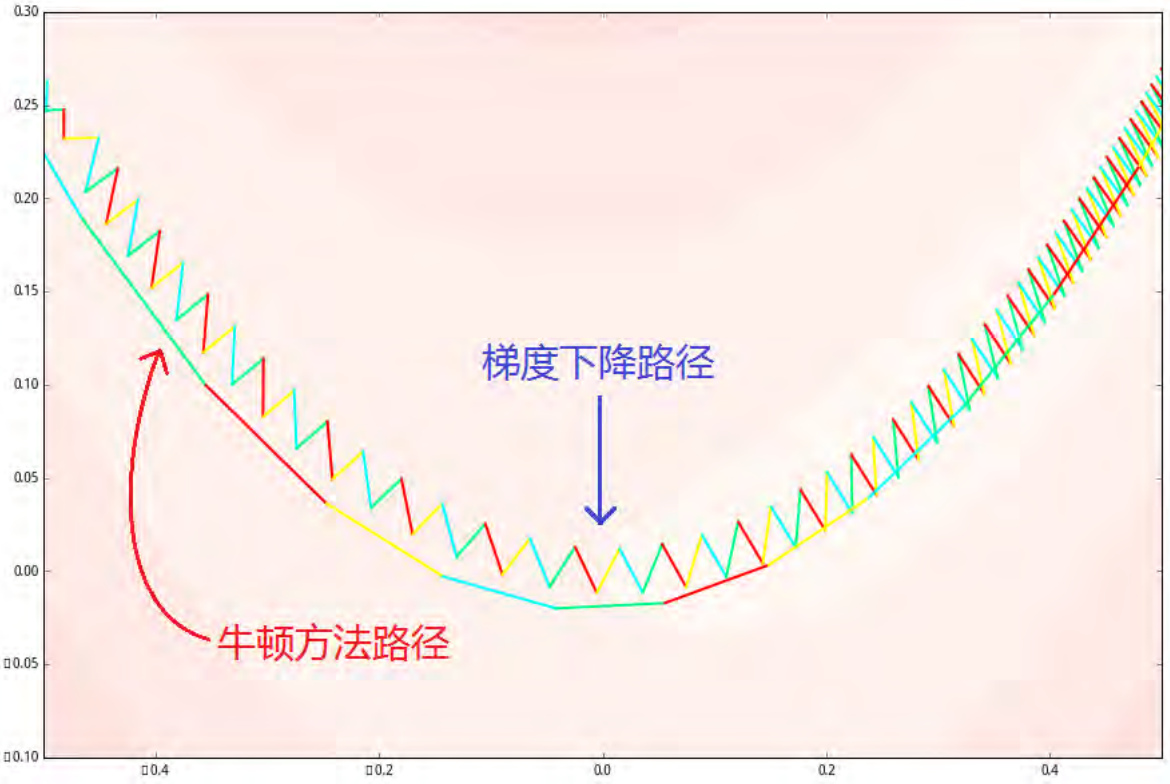
一个比较直接地使用牛顿方向的算法就是在 $\nabla^2 f(x_k)$ 为正定的时候使用牛顿方向，而不是的时候使用梯度下降，这样可以在足够接近局部极小点的时候以高效率收敛。判断正定矩阵的方法也并不困难，因为一个对称矩阵是正定的当且仅当它的所有特征值(eigenvalues)都是大于零的，这个用矩阵计算库（比如 Numpy）都可以计算出来。
我们来看牛顿方法在二次函数 $Q(x)$ 上的计算。



居然一次就找到了最优解！但是稍微仔细想一下，二次函数的二次泰勒展开并不只是在局部估测这个函数，而是完全就等于这个函数。所以当我们最小化了二次泰勒展开时，实际上已经找到问题的极小点。好，我们看一下创造困难的 Rosenbrock 函数。



牛顿方法用了 31 步就找到了误差小于 0.00001 的近似解，相比于梯度下降的 5491 步是一个巨大的进步。不得不说，函数的二次导数 $\nabla^2 R$ 中包含了这条壕沟的很多秘密。梯度下降和牛顿方法的表现差异在放大的图里一目了然。



当然了，可以选择的搜索方向远远不止梯度和牛顿两种，不过它们两个是所有方向算法的始祖。至于一些更复杂的方法，我们以后再慢慢来讲。

步长

上面介绍了每一步从 x_k 出发的方向 p_k，但这还不够，我们还需要选择一个在这个方向上行走的步长 α_k 以达到新的点 $x_k + \alpha_k p_k$。在选择每一次迭代的步长时，我们都面临一个权衡：如果一步过长了，那么可能会走得太远，跨过了极小点；如果一步太短，那么并不能将目标函数 f 的取值降低太多。
最大下降步长

一个最直接最“贪心”的步长选择是在射线 $x < em > k + \alpha p_k : \alpha \geq 0$上 f 取值最小的点。也就是说，解决一个最小化问题

$$\arg\min < /em > \alpha \geq 0 f(x_k + \alpha p_k),$$

并以求出的极小点作为步长 α。这是一个在一维切片上的优化问题，将导数设为 0，

Unknown environment 'align'

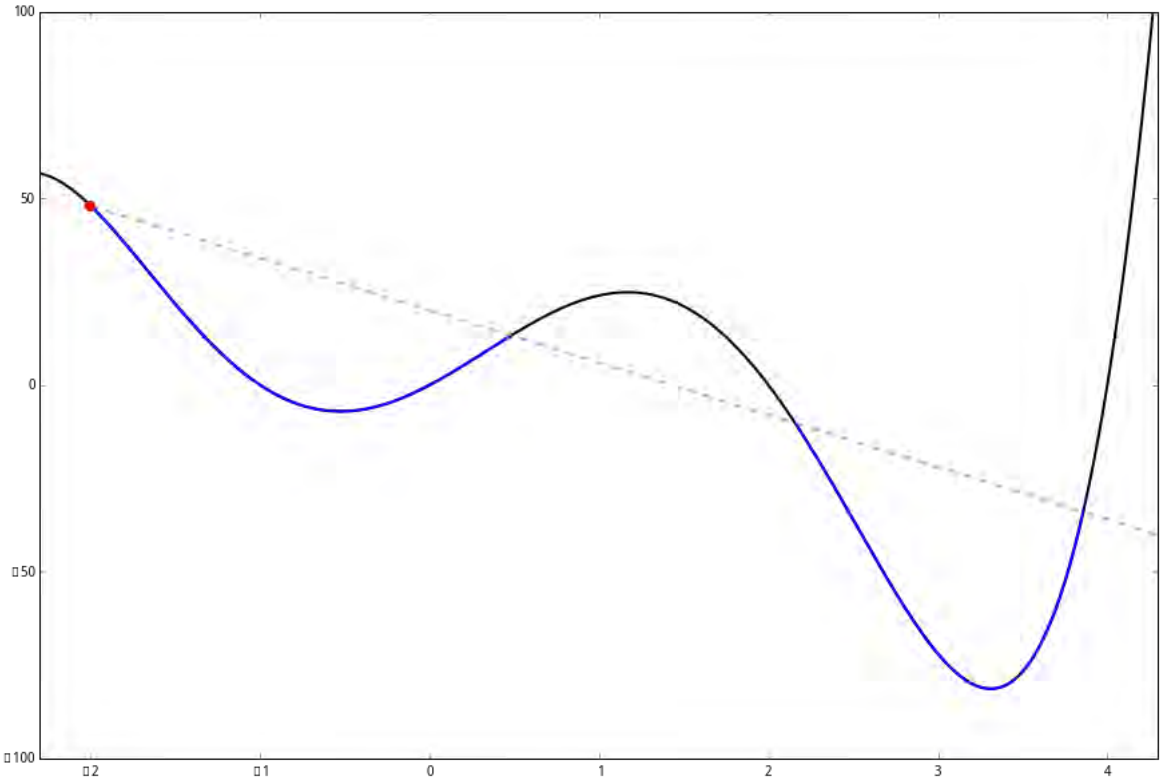
（运算中使用了高维版的链式法 Misplaced & ）可以看出极值出现在 f 的梯度和方向 p_k 垂直的位置。不巧的是，等式右侧函数的零点可能并不好求，一般来说寻找一个“足够好”的步长会比计算上面的“最好”步长更有效率。那我们怎么判断一个步长是不是足够好呢？接下来的 Wolfe 条件是一个常用的辨别方法。

Wolfe 条件

在 Wolfe 条件中有两个不等式，需要首先选取 $c_1 \in (0,1)$ 以及 $c_2 \in (c_1,1)$ 作为参数。条件中的第一个不等式要求步长 α 满足

$$f(x_k+\alpha p_k) \le f(x_k) + c_1\alpha\nabla f(x_k)^{\mathsf{T}} p_k.$$

也就是说，从 x_k 到 $x_k+\alpha p_k$ 的移动对 f 的缩减应该和步长 α 以及方向导数 $\nabla f(x_k)^{\mathsf{T}} p_k$ 达成一定以上的比例。这个条件叫做*足量衰减条件*（sufficient decrease condition）或者 Armijo *条件*（Armijo condition）。

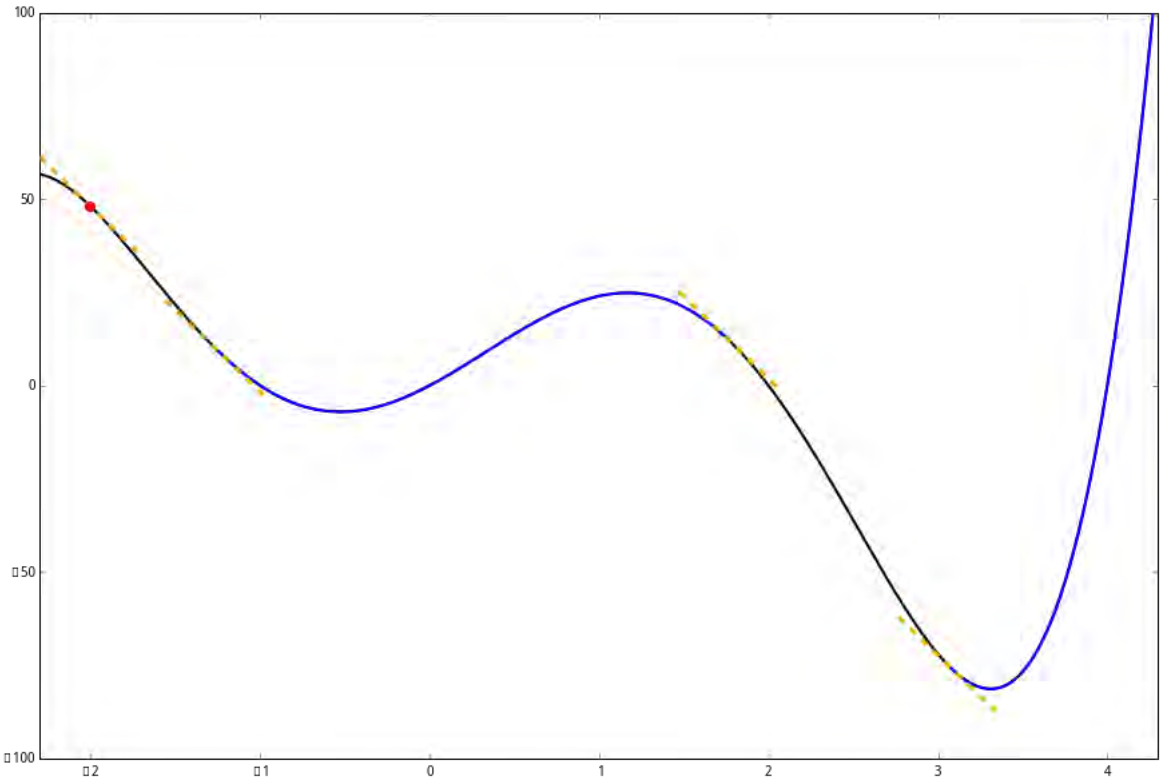


上图的横轴是 $x_k+\alpha p_k : \alpha \in \mathbb{R}$ 的一维直线，实线曲线是目标函数 f 在这个一维截面上的图像。我们从左侧的红点处出发，虚线表示的是直线 $L(x_k+\alpha p_k) = f(x_k) + c_1\alpha\nabla f(x_k)^{\mathsf{T}} p_k$ 的图像，如果 $c_1 = 1$ 的话那么虚线恰恰是曲线在 x_k 的切线。稍经分析可以看出，所有曲线低于虚线的位置都满足第一个不等式。实际应用中的 c_1 一般设得很小，10^{-4} 是一个比较常用的值，不过应随应用场景进行调整。

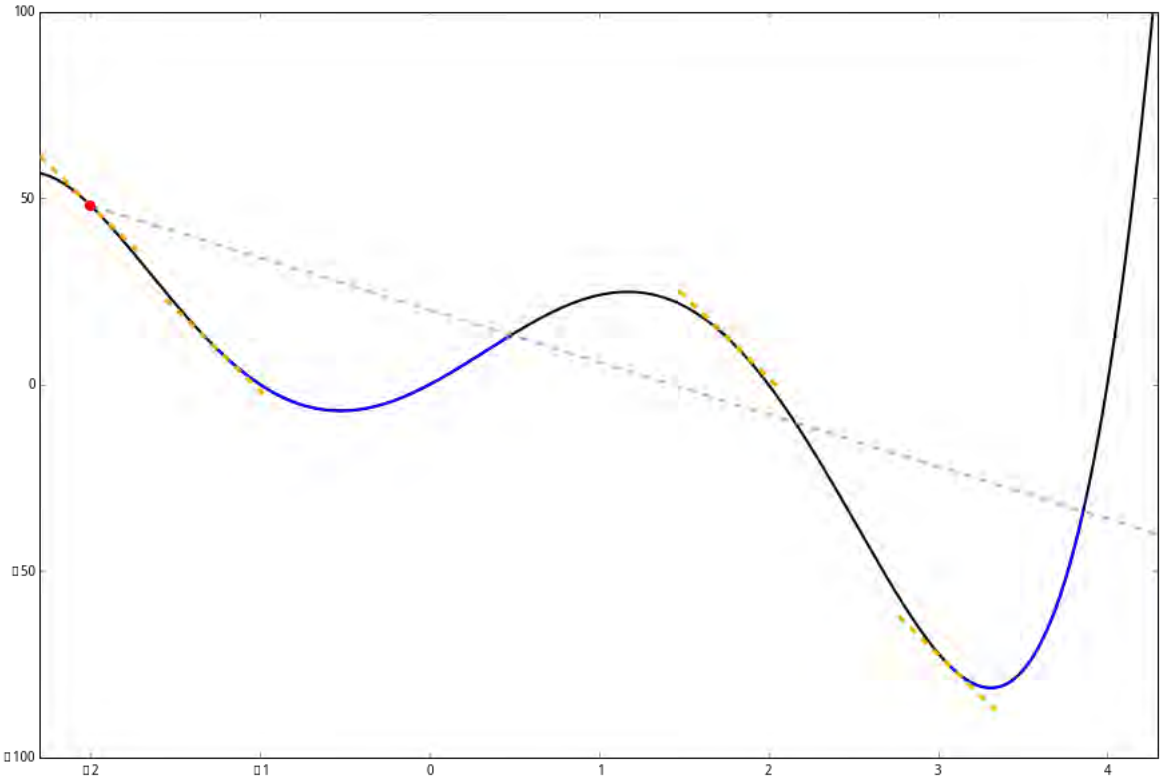
这里可以看出一个问题：起点的附近都小于虚线，没有被不等式排除；但我们并不想选过短的步长，因为这样离极小点依旧很远。因此，Wolfe 条件中有第二个不等式

$$\nabla f(x+\alpha p_k)^{\mathsf{T}} p_k \ge c_2\nabla f(x_k)^{\mathsf{T}} p_k.$$

这个不等式叫做*曲率条件*（curvature condition），它的意思是落脚点的梯度应该大于一定比例的起始点的梯度。我们使用的向量 p_k 使得该方向的梯度 $\nabla f(x_k)^{\mathsf{T}} p_k$ 小于 0，因此满足不等式的点的梯度应该更缓和（更接近 0），或者是向相反方向的（大于 0）。毕竟，在一个局部的极值点上，函数的梯度必定是 0，所以从一个梯度为负的点出发，我们应该要求梯度变得越来越大，才有可能找到极小点。



上图中，所有梯度大于起始点（也就是下降速度比起始点的下降速度慢）的区域都满足第二个不等式。在应用中，c_2 应根据选择 p_k 的方法进行选择。在这个示例中，同时满足 Wolfe 条件中两个不等式的区域如下图所示。



寻找一个满足 Wolfe 条件的步长并不是很困难，但确实是比较麻烦。今天就假设我们又笨又懒，只想找一个简单粗暴有效可以完成任务的方法，往下看。

回溯算法

记得 Wolfe 条件中有两个不等式：足量下降条件和曲率条件，前者保证了落脚点的取值和局部最小值更接近，后者保证落脚点足够平缓。现在，我们要抛弃要求平缓落脚的曲率条件，只要求每步有足够的下降，这个很好满足，因为 p_k 是一个下降方向并且 $c_1 < 1$，所以只要 Misplaced & 足够小，必定有 $f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^{\mathsf{T}} p_k$。所以有以下*回溯算法*（backtrack algorithm）：

输入：函数 f、当前点 x_k 以及方向 p_k;

设定参数 Misplaced & 和 $0 < \rho < 1$;

$\alpha \leftarrow \overline{\alpha}$;

while $f(x_k + \alpha p_k) \geq f(x_k) + c_1 \alpha \nabla f(x_k)^{\mathsf{T}} p_k$:

$\alpha \leftarrow \rho \alpha$.

输出 α.

这个算法输出的步长会有足够的下降，并且它保证了这一步不会太短：如果回溯的输出是 α_k，那么我们知道 α_k/ρ 作为一个步长是太长了，因为它不满足足量下降条件；所以虽然 α_k 可能不够精确，但不会太短。

初始的 $\overline{\alpha}$ 决定了输出步长的上限，用梯度下降或者牛顿方法时都可以将它设为 $\overline{\alpha}=1$。另外一个参数 ρ 的选择面临一个权衡：如果 ρ 接近 0 的话，回溯算法会在更少的循环中完成，但整体的优化搜索收敛得会更慢；如果 ρ 接近 1 则正相反。一般来说，选择的方向越精确就可以把 ρ 设得越大。

回溯算法对于牛顿方向是一个非常好用的步长计算方法，但对于一些文中没有提到方向（比如拟牛顿(quasi-Newton)和共轭梯度(conjugate gradient)）就不太好用了，不过，这些都是以后的话题了。

一个完整的线搜索算法

正如之前所说，线搜索方法是由一个选择方向的算法和一个寻找步长的算法组成的，在有了这两个主要成分之后就可以拼凑起来。给定一个函数 f 和起始点 x_0，我们计算从 x_0 出发的下降方向 p_0，再计算相对应的步长 α_0，并得到一个新的点 $x_1=x_0+\alpha_0 p_0$。再从 x_1 开始，计算 $x_2=x_1+\alpha_1 p_1$。如此重复，得到一个序列 $x_3,x_4,x_5,\ldots$，并且知道（或者不知道但假设）这个序列会收敛于一个局部极小点。那么问题来了，在绝大多数情况下我们不可能得到确切的极小点，那么我们该在什么时候终止迭代的过程呢？

一般用到的终止条件有两个：迭代次数和函数误差。迭代次数很好理解，我们设一个次数 K，那么在计算出 xK 后就终止计算，并且以 x_K 作为极小点的估测；这是一个以运算时间作为标准的终止条件。与之相对的，函数误差是一个计算精度为标准的条件，我们设定一个 Misplaced &，并且假设序列 $x_0,x_1,\ldots$ 收敛于某个点 $x/em><em>\in\mathbb{R}^n$，希望在 $|f(xk)-f(x/em></em>)|<\varepsilon$ 时中止计算。不过这并不现实，虽然 $f(x*)$ 比 $x/em>*$ 更好找，但很多时候我们还是很难把它计算出来；一个可以替代的条件是，当出现 $|f(xk)-f(x/em>)|$ 的情况时终止计算，并输出 x_k。精度和时间的终止条件可以同时使用。

所有部件都已齐全，一个完整的线搜索的优化算法如下：

输入：函数 f 和起始点 $x0$;
设定参数 Misplaced &、Misplaced & 以及其他的选择方向和计算步长算法需要的参数;
$k\leftarrow 0$;
while $k<K$ and $|f(x_k)-f(x/em>)|\geq\varepsilon$
    计算 $pk=$ 方向算法(f,x_k);
    计算 $\alpha_k=$ 步长算法(f,x_k,p_k);
    $x/em>k+1\leftarrow xk+\alpha_k p_k$
    $k\leftarrow k+1$.
输出 x_k.

以牛顿方向和回溯步长算法为例，我们有以下算法：

输入：二次可导函数 f 和起始点 x_0;
设定参数 Misplaced &;
$k\leftarrow 0$;
while $k<K$ and $|f(x_k)-f(x/em>)|\geq\varepsilon$
    计算 $\nabla f(xk)$ 和 $\nabla^2 f(x_k)$;
    if $\nabla^2 f(x_k)$ 是正定矩阵:
        $p_k\leftarrow -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$,
    else:
        $p_k\leftarrow -\nabla f(x_k)$.
    $\alpha\leftarrow\overline{\alpha}$;
    while $f(x_k+\alpha p_k)\geq f(x_k)+c\alpha\nabla(x_k)^{\mathsf{T}}p_k$:
        $\alpha\leftarrow\rho\alpha$.
    $x/em>k+1\leftarrow xk+\alpha p_k$;
    $k\leftarrow k+1$.
输出 x_k.

Python 代码在后面的研究模块中可以找到。

结语

无约束的非线性规划问题有很多在实际场景中的应用，但更多的应用还是在于有约束的规划问题。比如在我们都非常关心的资产配置问题中，如果用 xi 代表在金融资产 i 上的分配权重，我们会要求资产配置的总和是 1，也就是 $\sum/em>i=1^n x_i=1$；我们可能要求不能做空，那么有 $x_i\geq 0$；也许又要求任何一种资产的权重都不超过总资产的三分之一，于是又有 $x_i\leq 1/3$。这些要求让优化问题变得更困难但也更有趣，有时我们需要为一组特定的约束条件专门研究一套算法来解决。一切才刚刚开始，请读者期尽情待量化课堂未来的文章。

文章更迭记录：
v1.1，2016-11-14，修正公式
v1.0，2016-10-22，文章上线

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
```

In [32]:

```
# 线搜索算法框架
# 输入 (目标函数，梯度函数，起始点，方向算法，步长算法，方向算法的参数，步长算法的参数，
#       终止条件的 epsilon，终止条件的迭代次数)
# 注意：如果 epsilon 和 iterations 都不填的话，你就安静地坐下来，等待时间的洪流带你去往彼方吧。
# 输出一个优化的搜寻路径
@jit
def line_search(f, grad, init, direction_algo, steplength_algo, \
                direction_params=None, steplength_params=None, epsilon=0, iterations=float('inf')):
    sequence = [init]
    x = init
    prev_val = float('inf')
    counter = 0
    while abs(f(x)-prev_val)>epsilon and counter<iterations:
        p = direction_algo(f,grad,x,direction_params)
        a = steplength_algo(f,grad,x,p,steplength_params)
        prev_val = f(x)
        counter += 1
        x = x+a*p
        sequence.append(x)
    return(sequence)
```

In [4]:

```
# 梯度下降方向算法
# 输入 (目标函数，梯度函数，x)
# 输出在 x 的负梯度向量
def grad_descent(f, grad, x, params=None):
    return -grad(x)
```

In [42]:

```
# 牛顿方向算法
# 输入 (目标函数，梯度函数，x，(海塞矩阵函数,))
# 输出（如果海塞矩阵是正定的）牛顿方向，或者负梯度
def newton_method(f, grad, x, params):
    hess = params[0](x)
    if is_pos_def(hess):
        return -(numpy.linalg.inv(hess)).dot(grad(x))
    else:
        return -grad(x)
```

In [47]:

```
# 判断一个矩阵是否正定
# 输入矩阵
# 输出 Boolean
def is_pos_def(mat):
    return(np.all(np.linalg.eigvals(mat) > 0) and np.all(mat==mat.T))
```

In [7]:

```
# 回溯步长算法
# 输入 (目标函数，梯度函数，起始点，方向，参数)
# 参数是 (初始 alpha，每次迭代的缩减倍数，Armijo 条件中的 c1)
# 返回一个合适的步长
@jit
def backtrack(f,grad,x,p, params):
    (alpha, rho, c) = params
    a = alpha
    while f(x+a*p)>f(x)+c*a*p.dot(grad(x)):
        a = a*rho
    return a
```

In [ ]:

```

```

In [8]:

```
def plot_colored_path(sequence):
    for i in range(len(sequence))[:-1]:
        if i%4==0:
            color = 'aqua'
        elif i%4==1:
            color = 'red'
        elif i%4==2:
            color = 'springgreen'
        else:
```

```
            color = 'yellow'
        plt.plot([sequence[i][0], sequence[i+1][0]], [sequence[i][1], sequence[i+1][1]], c=color, linewidth=2)
```

In [ ]:

```

```

In [9]:

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 15
```

In [10]:

```
xx,yy = np.meshgrid(np.arange(-10,10,0.01), np.arange(-10,10,0.01))
vecs = np.column_stack((xx.flatten(),yy.flatten()))
```

In [11]:

```
A = np.array([[3,1],[1,3]])
```

In [12]:

```
@jit
def calculate(vecs, A):
    values = array([0]*len(vecs))
    for i in range(len(vecs)):
        values[i] = vecs[i].dot(A).dot(vecs[i])
    return values
```

In [13]:

```
values_quad = calculate(vecs, A)
```

In [14]:

```
@jit
def sqrt_all(values):
    new = array([0]*len(values))
    for i in range(len(values)):
        new[i] = values[i]**(0.5)
    return new
```

In [15]:

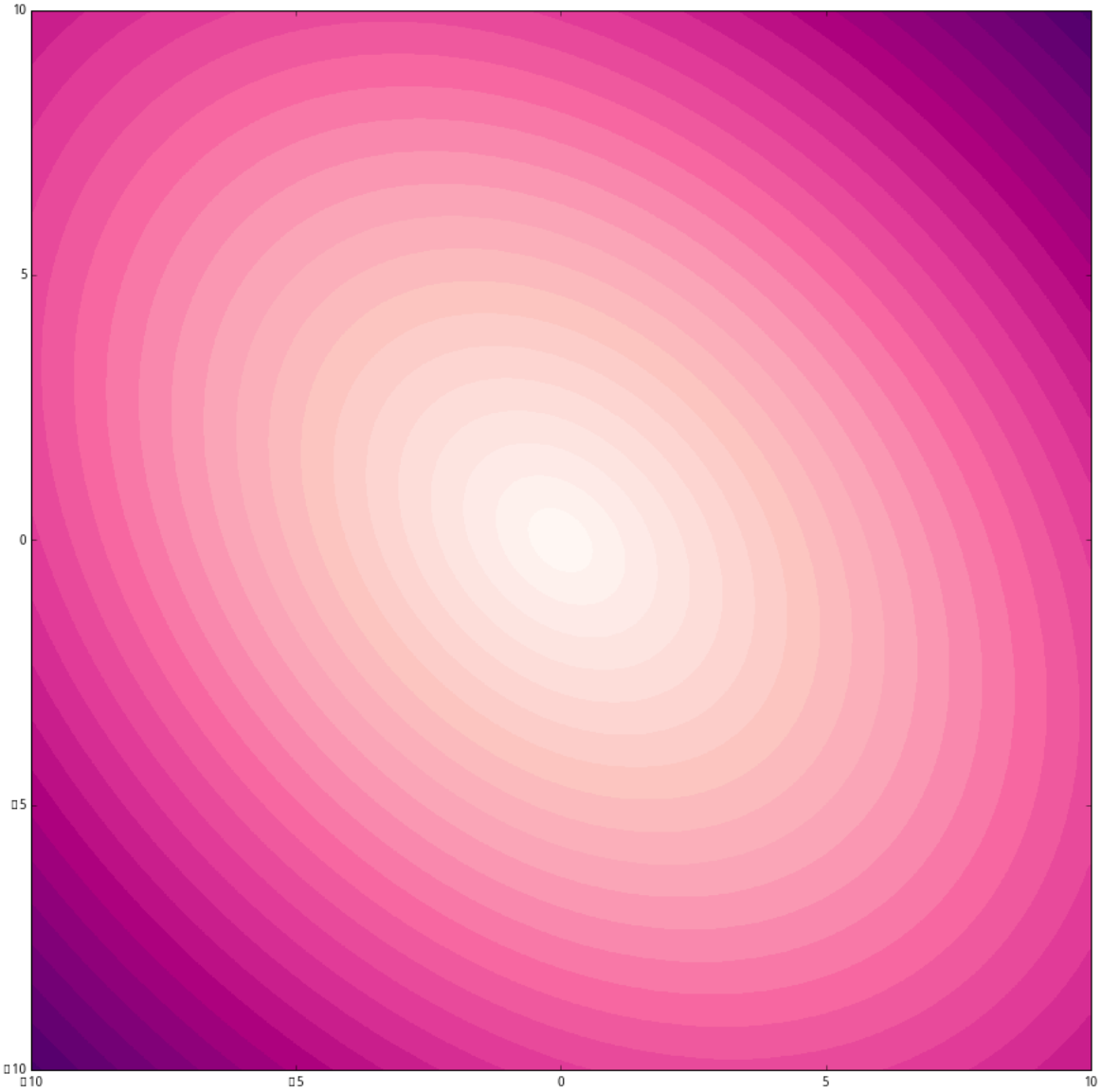
```
sqrted_quad = sqrt_all(values_quad)
```

In [16]:

```
plt.pcolormesh(xx, yy, sqrted_quad.reshape(xx.shape), cmap='RdPu')
```

Out[16]:

```
<matplotlib.collections.QuadMesh at 0x7ff3b3283f90>
```



In [17]:

```
def f1(x):
    return x.dot(A.dot(x))
```

In [18]:

```
def grad1(x):
    return 2*A.dot(x)
```

In [19]:

```
def hess1(x):
    return A
```

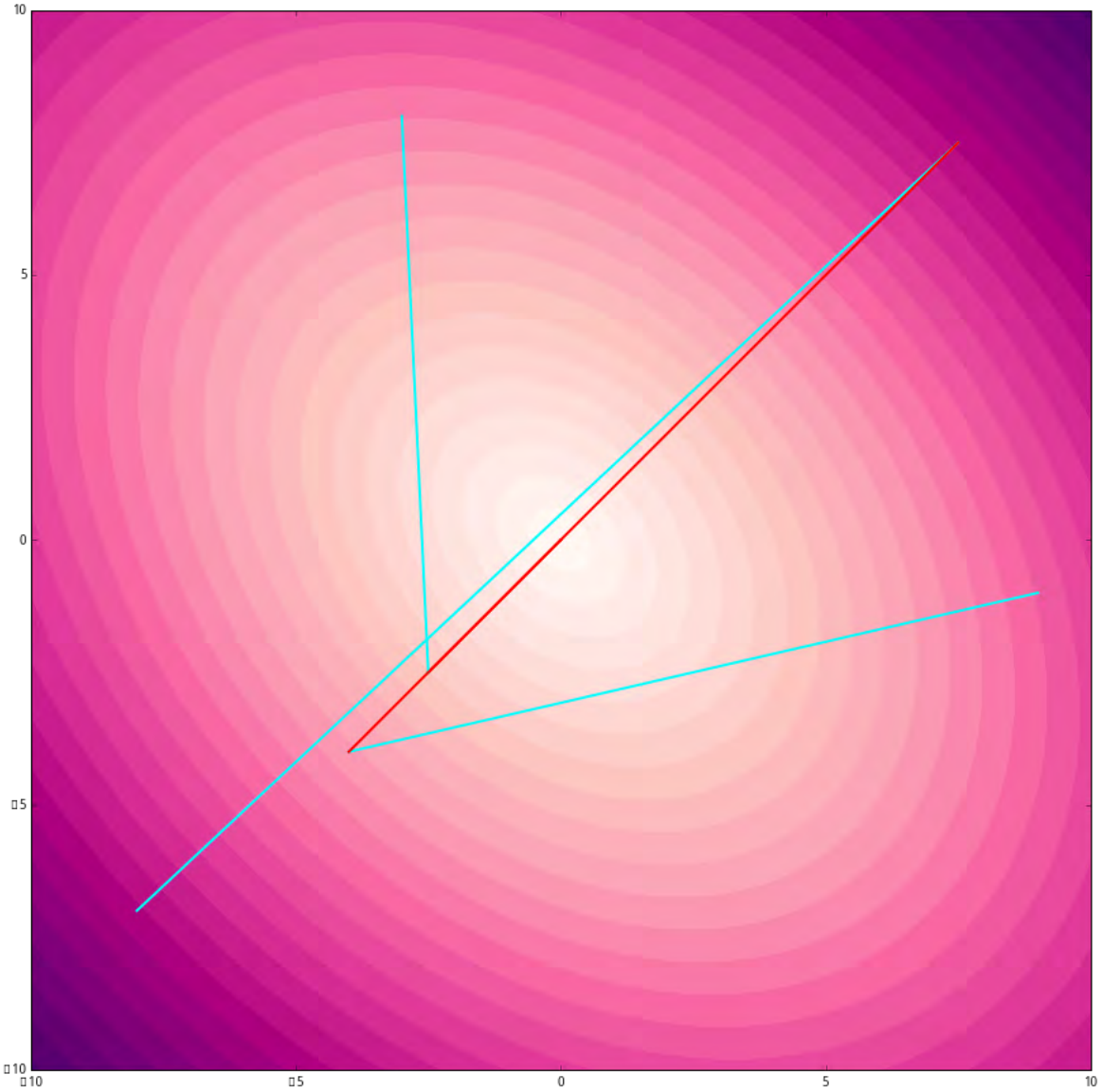
In [51]:

```
grad_result1 = line_search(f1, grad1, array([-3.,8.]), grad_descent, backtrack,None,\
                           (1.,0.5,0.0001),epsilon=0.00001)
grad_result2 = line_search(f1, grad1, array([-8.,-7.]), grad_descent, backtrack,None,\
                           (1.,0.5,0.0001),epsilon=0.00001)
grad_result3 = line_search(f1, grad1, array([9.,-1.]), grad_descent, backtrack,None,\
                           (1.,0.5,0.0001),epsilon=0.00001)
```

下图梯度下降和文中不同，文中方向选择是归一化的梯度，而这里没有归一化。

In [52]:

```
plt.pcolormesh(xx, yy, sqrted_quad.reshape(xx.shape), cmap='RdPu')
plot_colored_path(grad_result3)
plot_colored_path(grad_result1)
plot_colored_path(grad_result2)
```

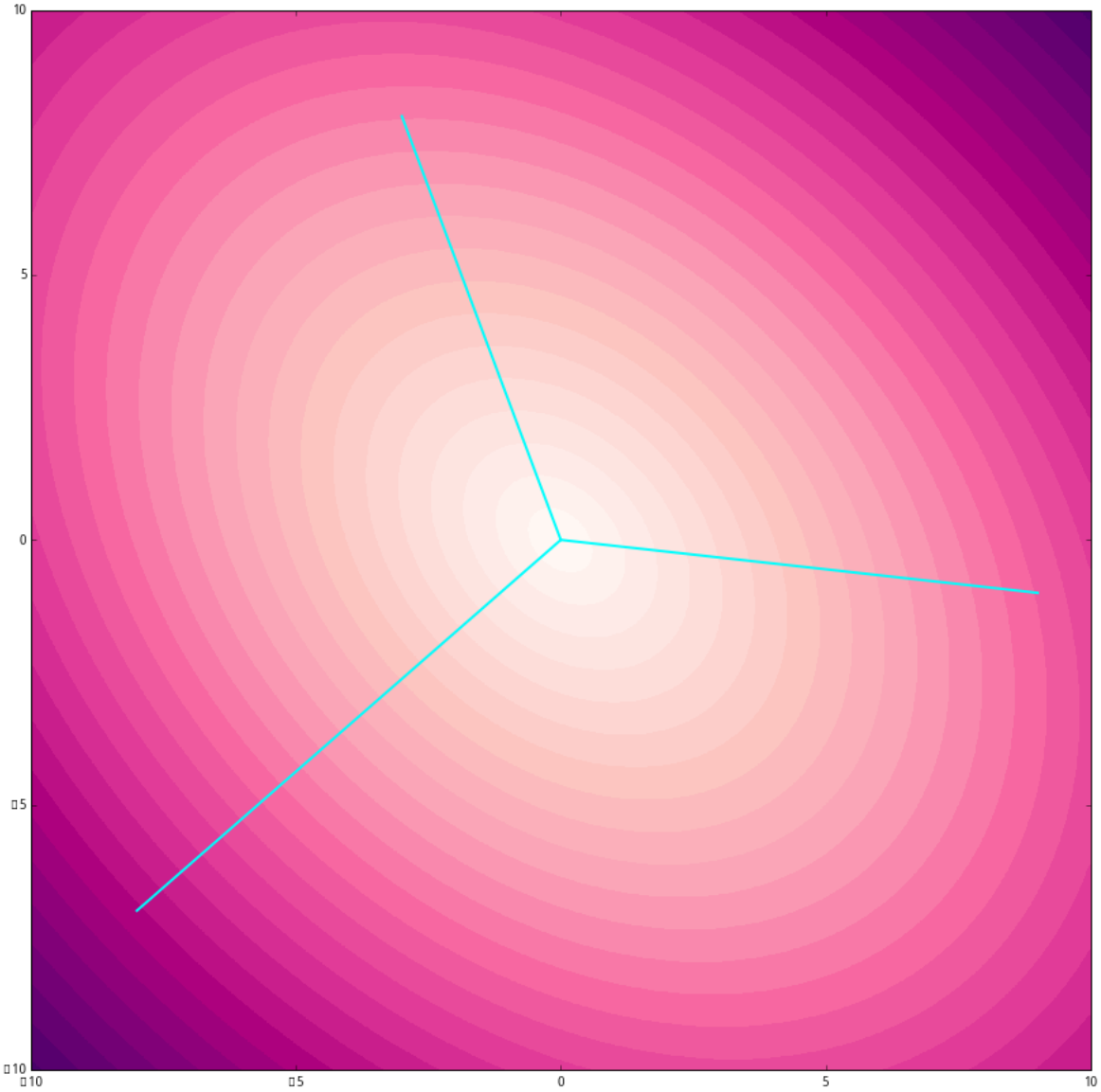


In [53]:

```
grad_result4 = line_search(f1, grad1, array([-3.,8.]), newton_method, backtrack,(hess1,),\
                           (1.,0.5,0.0001),epsilon=0.00001)
grad_result5 = line_search(f1, grad1, array([-8.,-7.]), newton_method, backtrack,(hess1,),\
                           (1.,0.5,0.0001),epsilon=0.00001)
grad_result6 = line_search(f1, grad1, array([9.,-1.]), newton_method, backtrack,(hess1,),\
                           (1.,0.5,0.0001),epsilon=0.00001)
```

In [54]:

```
plt.pcolormesh(xx, yy, sqrted_quad.reshape(xx.shape), cmap='RdPu')
plot_colored_path(grad_result4)
plot_colored_path(grad_result5)
plot_colored_path(grad_result6)
```



In [ ]:

```
```

In [55]:

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 10
```

In [57]:

```
xx,yy = np.meshgrid(np.arange(-1.5,1.5,0.001), np.arange(-0.5,2,0.001))
vecs = np.column_stack((xx.flatten(),yy.flatten()))
```

In [68]:

```
def f2(x):
    return (1-x[0])**2+100*(x[1]-x[0]**2)**2
```

In [59]:

```
def compute(vecs):
    values = array([0]*len(vecs))
    for i in range(len(values)):
        values[i] = f2(vecs[i])
    return values
```

In [64]:

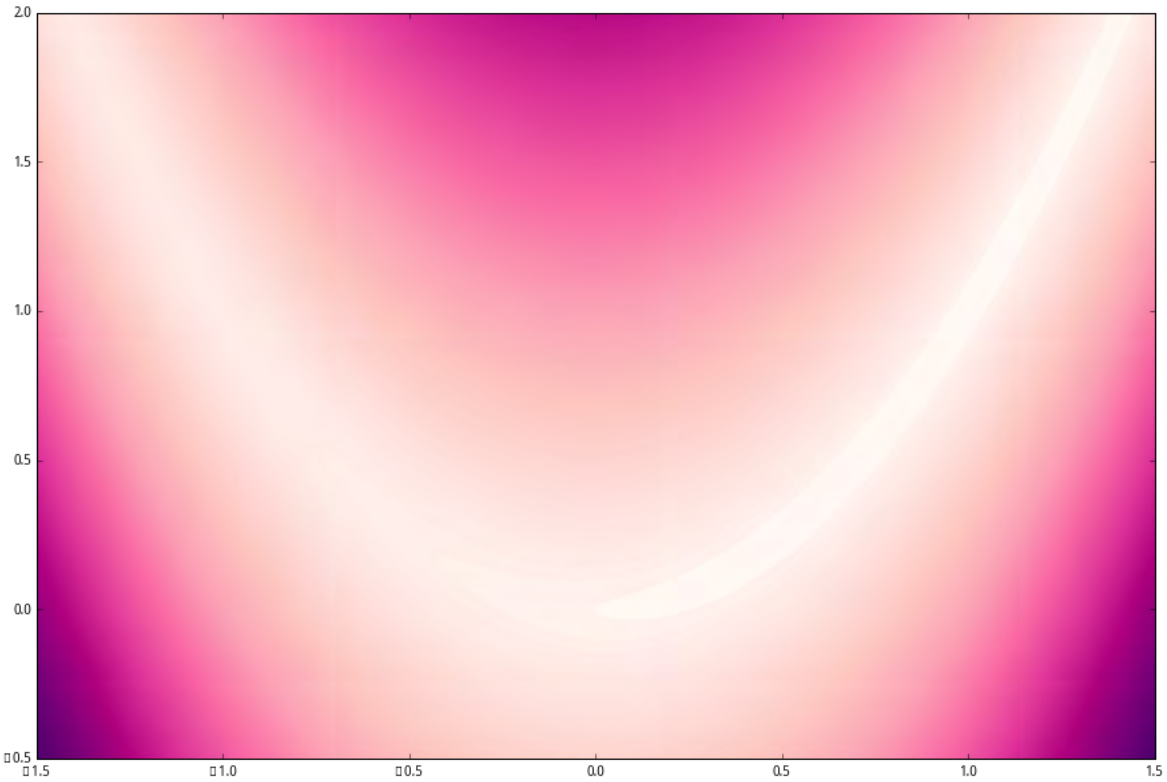
```
values_trench = compute(vecs)
sqrted_trench = sqrt(values_trench)
```

In [65]:

```
plt.pcolormesh(xx, yy, sqrted_trench.reshape(xx.shape), cmap='RdPu')
```

Out[65]:

```
<matplotlib.collections.QuadMesh at 0x7ff3b3585150>
```



In [67]:

```
def grad2(x):
    return array([-2*(1-x[0])-400*(x[1]-x[0]**2)*x[0],  200*(x[1]-x[0]**2)])
```

In [69]:

```
def hess2(x):
    return array([[2-400*x[1]+1200*x[0]**2 , -400*x[0]],[-400*x[0], 200]])
```

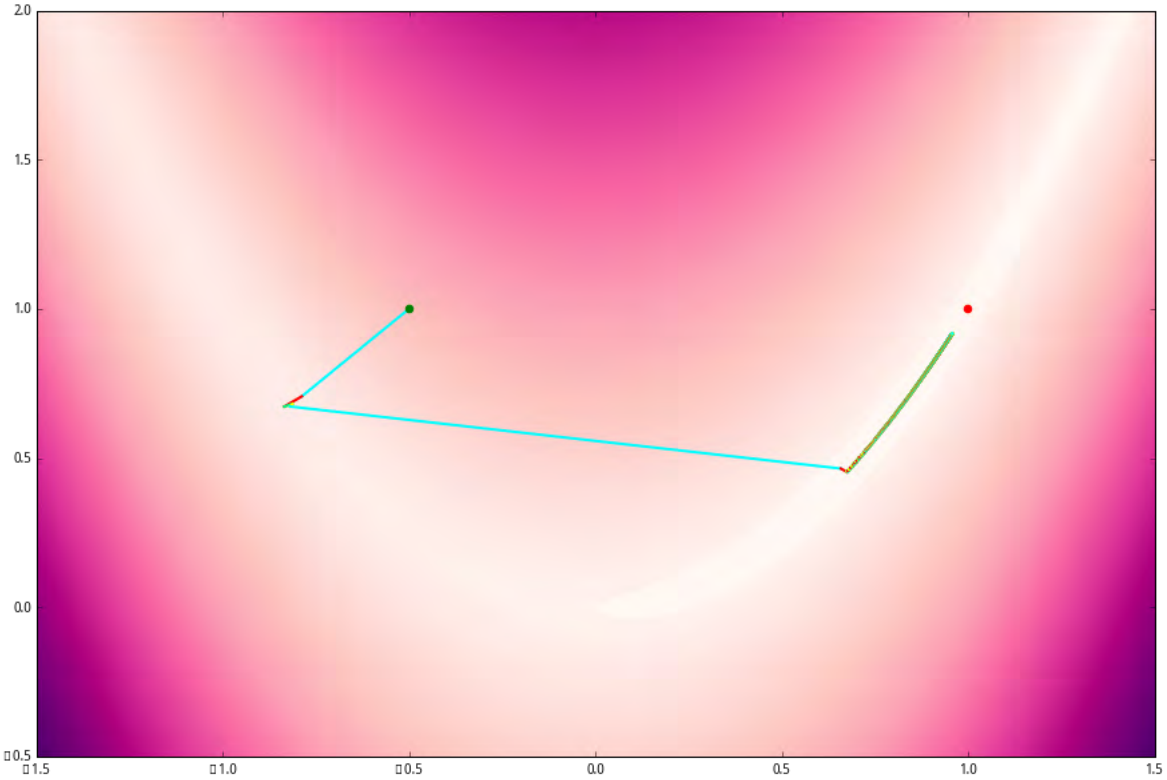
In [75]:

```
trench_result1 = line_search(f2, grad2, array([-0.5,1.]), grad_descent, backtrack,None,\
                             (1.,0.5,0.0001),epsilon=0.0000001)
```

下图梯度下降和文中不同，文中方向选择是归一化的梯度，而这里没有归一化。并且，文中我们假设已经知道极小点的赋值，所以终止条件是和最小值相差 0.00001 时终止；而这里我们假设不知道最小值，在两次迭代的差距小于 0.0000001 时终止。可见实际效果很差。

In [76]:

```
plt.pcolormesh(xx, yy, sqrted_trench.reshape(xx.shape), cmap='RdPu')
plt.scatter([-0.5],[1], s=30, color='g',zorder=10)
plt.scatter([1],[1], s=30, color='r', zorder=10)
plt.axis((-1.5,1.5,-0.5,2))
plot_colored_path(trench_result1)
```

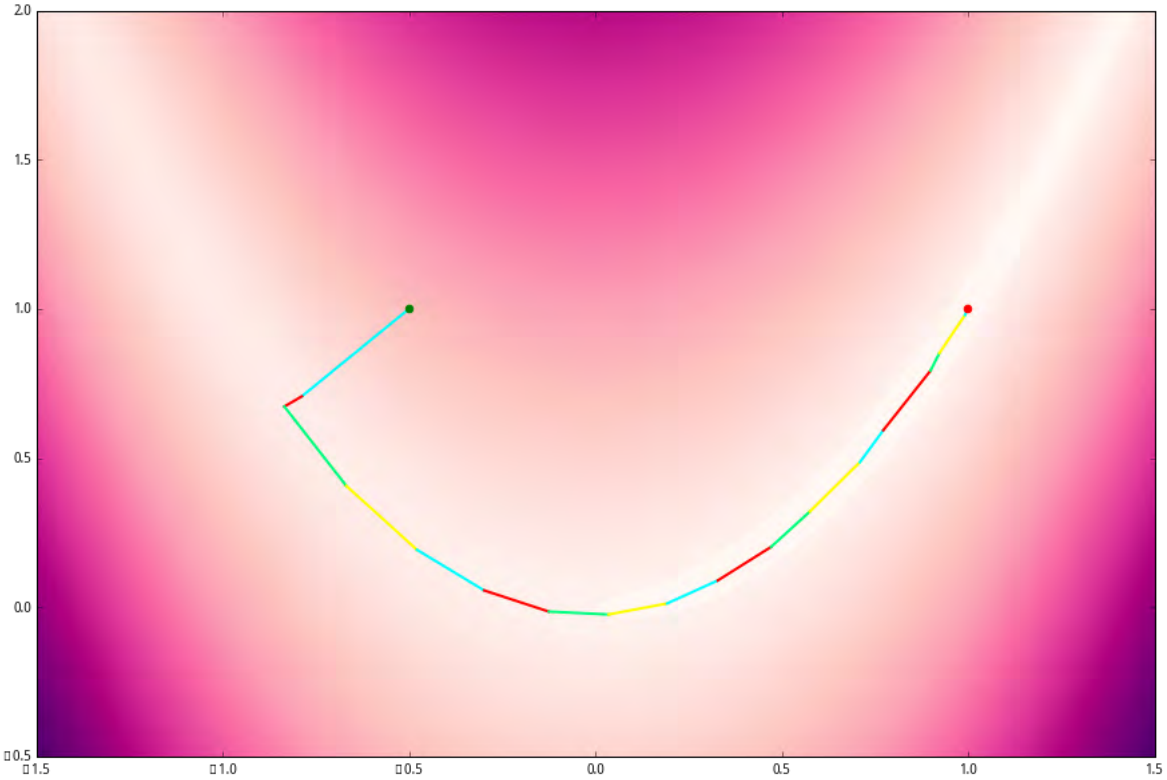


In [73]:

```
trench_result2 = line_search(f2, grad2, array([-0.5,1.]), newton_method, backtrack,(hess2,),\
                             (1.,0.5,0.0001),epsilon=0.00001)
```

In [74]:

```
plt.pcolormesh(xx, yy, sqrted_trench.reshape(xx.shape), cmap='RdPu')
plt.scatter([-0.5],[1], s=30, color='g',zorder=10)
plt.scatter([1],[1], s=30, color='r', zorder=10)
plt.axis((-1.5,1.5,-0.5,2))
plot_colored_path(trench_result2)
```



In [ ]:

```

```

【量化课堂】拉格朗日乘子

导语：拉格朗日乘子法是用于解决一类有约束的非线性规划问题的方法。它并不是一个计算结果的算法，而是把规划问题转化成一个更简便的格式，便于使用其他的算法进行计算。著名的马科维兹均值-方差问题就可以用拉格朗日乘子解决。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，难度为进阶（上），深度为 level-1。

阅读本文前需要掌握线性代数、多元微积分以及非线性规划的基础知识。

要解决的问题

设 f 和 g 是 $\mathcal{C}^1$ 的 $\mathbb{R}^n \to \mathbb{R}$ 函数，并且 $c \in \mathbb{R}$ 是一个实数，我们考虑以下的数学规划问题：

Unknown environment 'align'

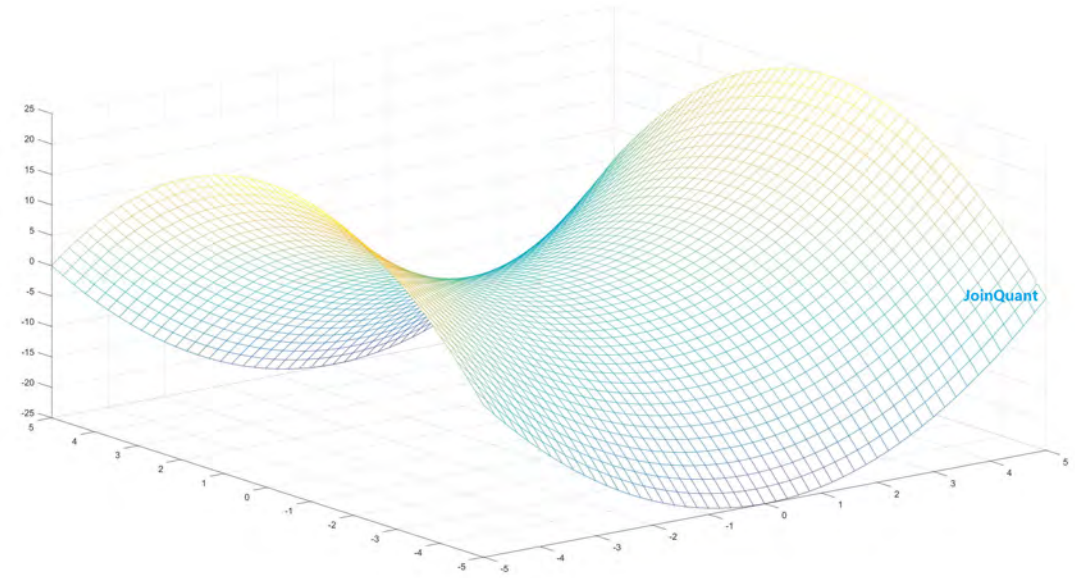
拉格朗日乘子是一个解决这类问题的方法。直接介绍拉格朗日乘子的话并不是很好理解，于是我们先把上述规划问题的解题思路捋一遍，文章最后再介绍拉格朗日乘子的时候，其中的原理就一目了然了。

g 的水平集

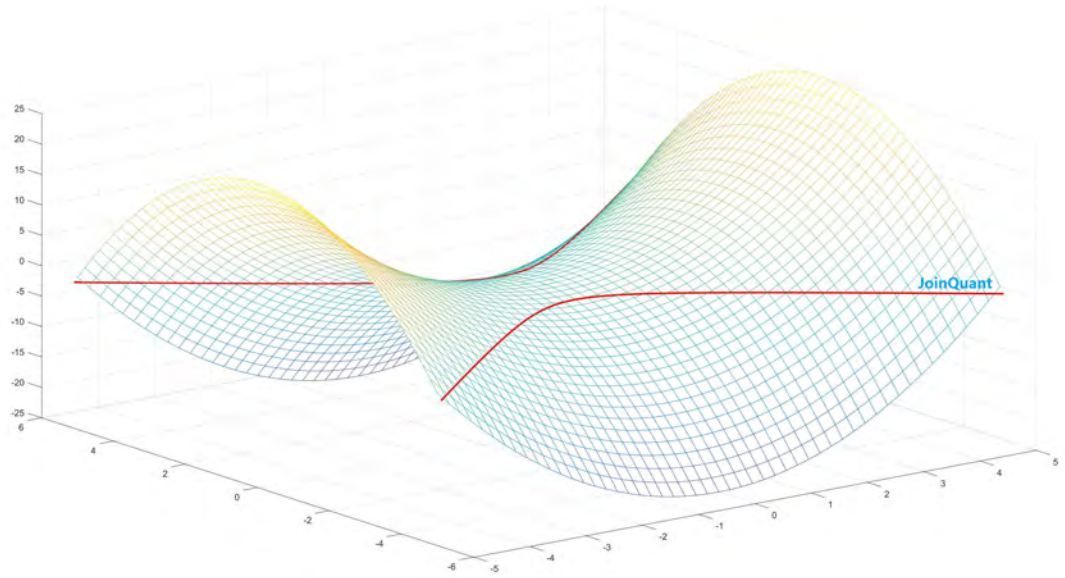
首先我们要了解规划问题的的可行域，也就是所有满足 $g(x) = c$ 的 $x \in \mathbb{R}^n$。下面给这个集合一个定义：
定义. 设 $g : \mathbb{R}^n \to \mathbb{R}$ 是一个函数，我们说 g 在 $c \in \mathbb{R}$ 的*水平集* (level set)是

$$L_c = x \in \mathbb{R}^n : g(x) = c.$$

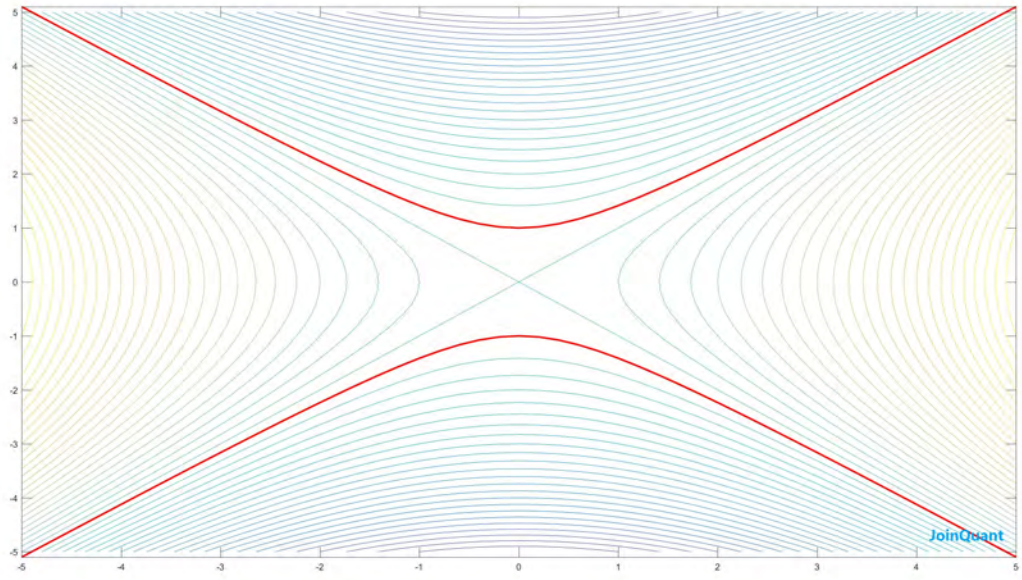
举个一例子。考虑一个固定的 $\mathbb{R}^2 \to \mathbb{R}$ 函数 $\tilde{g}(x, y) = x^2 - y^2$，它的图像如下



我们来选择一个水平集。嗯，c 等于多少好呢... 就 $c=-1$ 吧。看一眼水平集 $\widetilde{L}_{-1}=(x,y)\in\mathbb{R}^2:x^2-y^2=-1$, 如下图红线所示



如果从上方向下看函数 $\tilde{g}$ 的热力图，那么下图中的每一条线是 $\tilde{g}$ 的一个水平集，根据颜色的从紫到黄，依次是 $c=-23,-22,\ldots,22,23$的水平集。其中两条红线是我们的 $c=-1$ 水平集



机智的读者一定已经算出，组成水平集 $\widetilde{L}_{-1}$ 的两条曲线在 (x,y) 坐标系上可以分别用函数

$$y=\sqrt{x^2+1}\qquad 和\qquad y=-\sqrt{x^2+1}$$

的图像表示。

我们来看水平集 $\widetilde{L}_{-1}$ 一个重要的的几何性质。假设从前有座山，山里有座庙，庙前有片空地，地形和图二里的一模一样。你顺着上边的那条红线走呀走，发现自己的 x 坐标和 y 坐标都在变动，可是竖向的 z 坐标一直不变，也就是说海拔不变。你摸着自己地良心问：“我走过了哪里，又将走向哪里？”良心答曰：“你在时间 t 的 (x,y) 坐标是 $p(t)=(t,\sqrt{t^2+1})$。”如梦初醒的你掏出小木棍在土地上一阵划拉，计算出自己在时间 t 的行走方向是 Misplaced & 。

“谢谢你，我的良心。”

“嗯，别烦我了。”

无视了来自良心的蔑视，你又不禁遐想，这片大地的梯度又是多少呢？经过十三年的艰苦钻研，你最终算出了地形的梯度函数 $\nabla\tilde{g}(x,y)=(2x,-2y)$。代入函数 $p(t)$，你得知在时间 t 的时候你的所在地的地形梯度是 $\nabla\tilde{g}(p(t))=(2t,-2\sqrt{t^2+1})$。

于是你一边走一边向 $\nabla\tilde{g}$ 的方向死死盯着不放，最终得了颈椎病。你扶着已经转不回来的脖子，恍然大悟：原来 $\nabla\tilde{g}(p(t))$ 和 Misplaced & 是垂直的。是哪，

Misplaced &

也就是说 $\tilde{g}$ 的水平集的切线和 $\tilde{g}$ 的梯度是永远垂直的!

仔细想一想的话这也是当然。根据线搜索方法文章中所讲，函数 $\tilde{g}$ 在 (x,y) 的梯度 $\nabla\tilde{g}(x,y)$ 是 $\tilde{g}$ 的取值上升最快的方向，而 $-\nabla\tilde{g}(x,y)$ 则是 $\tilde{g}$ 的取值下降最快的方向。水平集是 $\tilde{g}$ 取值不变的集合，那么在水平集上行走的话，行走方向应该和上升方向 $\nabla\tilde{g}$ 之间保持一定角度，这样就不会上升，当然也要和下降方向 $-\nabla\tilde{g}$ 保持一定角度，这样就不会下降。用脚趾头一想，既然 $\nabla\tilde{g}$ 和 $-\nabla\tilde{g}$ 之间的角度是 180 度，那么折中的不上不下的角度应该就是那个的一半吧，也就是说行走方向应该是和 $\nabla\tilde{g}$ 垂直的。

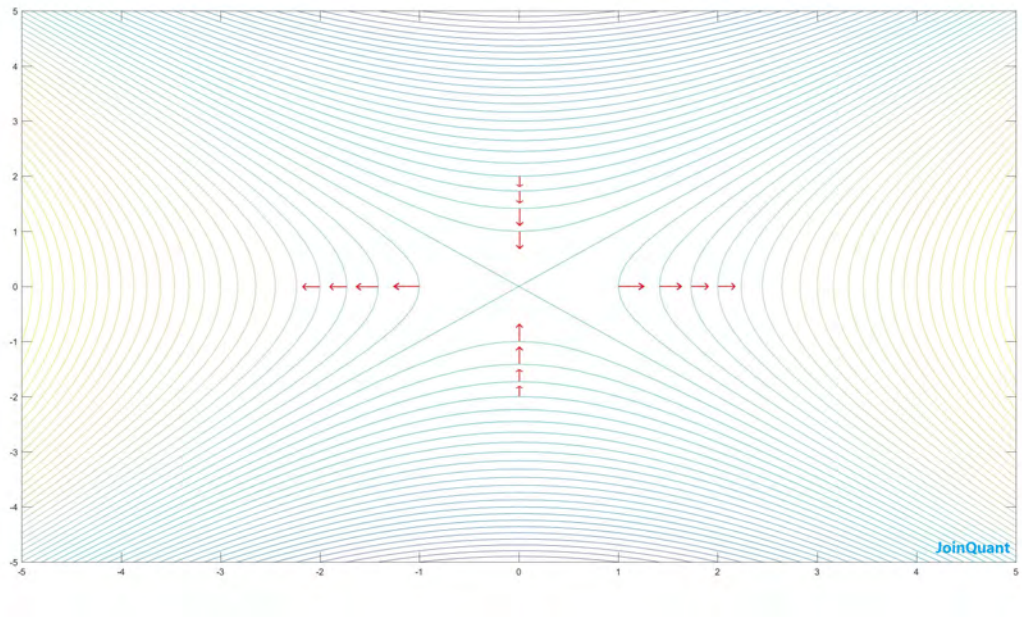
以上的现象用直白的话说，就是梯度和水平集是垂直关系的，这也是多元微积分里的一个基础定理：

定理. 设 $g:\mathbb{R}^n\to\mathbb{R}$ 是一个 $\mathcal{C}^1$ 函数，$c\in\mathbb{R}$ 是一个实数，而 $L_c\subseteq\mathbb{R}^n$ 是 g 取值为 c 的水平集。对于任何一个 Misplaced & 和 $\mathcal{C}^1$ 曲线 $\alpha:(-a,a)\to L_c$，都有 Misplaced & 。

证明. 因为对于所有 $t\in(-a,a)$ 都有 $\alpha(t)\in L_c$，所以也有 $g(\alpha(t))=c$。微分，并且使用链式法则可得

Unknown environment 'align'

是想要的等式。 Undefined control sequence \square
用红色箭头表示函数 $\tilde{g}$ 的梯度的话，它会是像下图中这样，和水平集相垂直。



当然，这个现象不只是针对于例子中这个特定的 g，上面的定理适用于任何一个函数 g，它们的梯度和它们的水平集都是相互垂直的。

有请目标函数 f

我们回顾一下最初要解决的问题

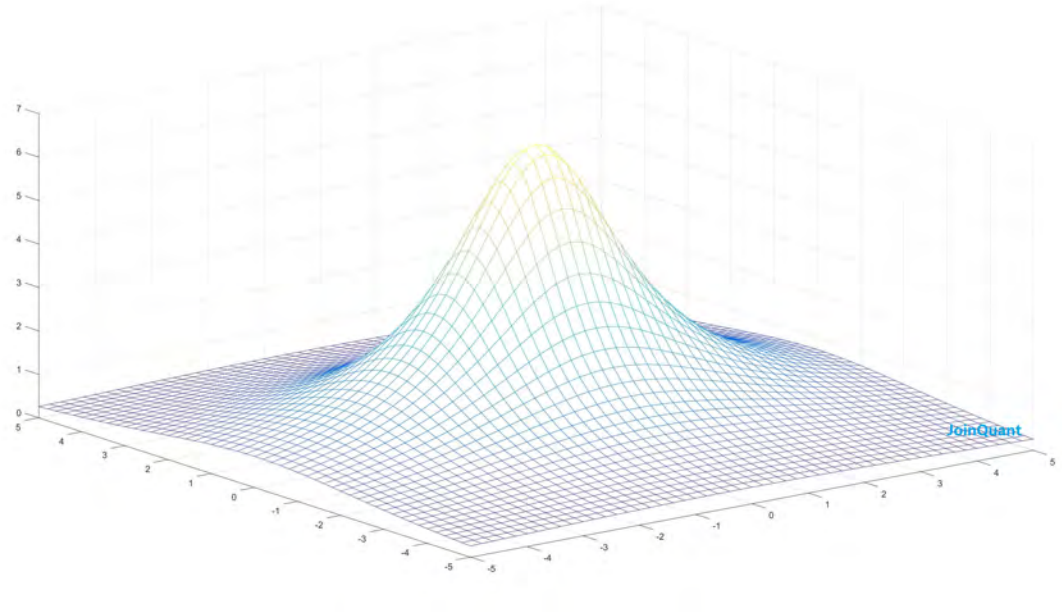
Unknown environment 'align'

在上一节中我们对这个规划问题的可行区域 L_c 有了一个初步的认识。那么接下来，当然是要在可行域上寻找目标函数 f 的最大值了。

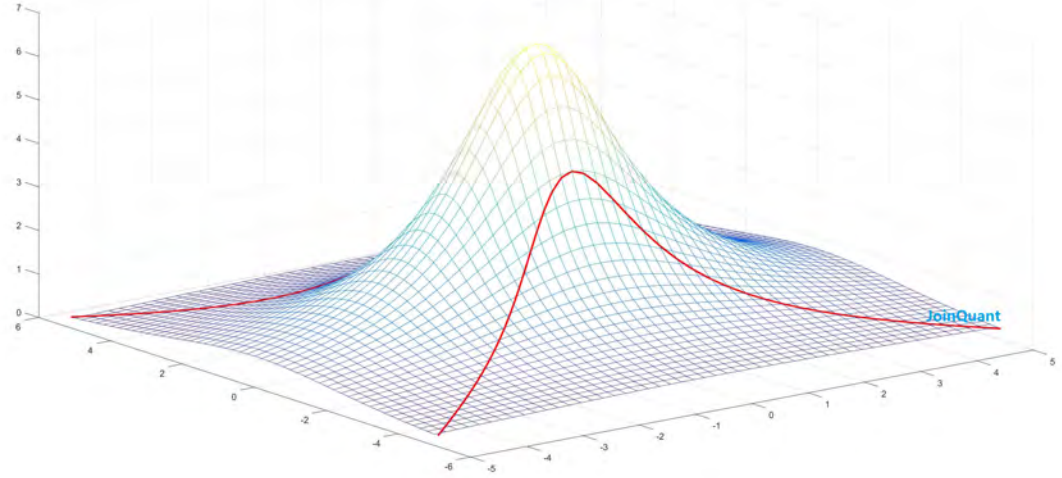
好，我们就设一个目标函数

$$\tilde{f}(x,y)=\frac{20}{3+x^2+2y^2},$$

它长得像这样，中间是一个小山丘。



在规划问题中，我们想要的是在 g 的水平集 $L < em > c$上找到 f 的最大值，那么把上一章节计算出的水平集 $\widetilde{L} < /em > -1$ 画在 $\tilde{f}$ 的图像上，是这样的。



现在就假设你又来了一座山，又发现了一座庙，庙前有个小山坡，长得这个样。站在山脚下，你回想起了以前沿着 $\tilde{g}$ 的水平集走过的路径，也就是 $p(t) = (t, \sqrt{t^2+1})$，你决定还沿着这条路来走这个山坡。

不过这次可和上一次不一样了，虽然走的同样的路径，但是地形不一样，所以海拔不再是一成不变，而是时高时低。你不禁好奇，在这条路上海拔最高的点在哪里？

你摸了摸自己的良心，“良心良心告诉我，……”

“请你不要摸我了好吗？好恶心的。”

“好，”你乖乖地把手拿开，“良心良心告诉我，我的海拔是多少？”

“你在时间 t 的海拔是 $\tilde{f}\left(p(t)\right) = 20/(32+3t^2)$。”

啊！你再次恍然大悟，算出了 $\tilde{f}\left(p(t)\right)$ 的导数并且设它等于零，不就知道海拔高度出现极值的地方了吗？事不宜迟，你掏出了小木棍，又开始在地上胡乱划拉起来。

这一晃又过去了二十三年，你终于长叹一口气，得到了一个如此简练的公式：

Misplaced &

具体的数值什么的，已经都无所谓了（其实是因为你到最后也没搞清楚怎么微分 $1/(32+3t^2)$，不过这种事就让它过去吧）。如果在时间 t 的海拔高度是

最高的，那么上面的这条公式必定被满足。也就是说，对于一个点 $x_0 \in \widetilde{L}_c$，如果 x_0 是 $\tilde{f}$ 在 $\widetilde{L}_c$ 上的一个极大点，那么梯度 $\nabla \tilde{f}(x_0)$ 和 $\widetilde{L}_c$ 在 x_0 的切线一定是垂直的。

“谢谢你，我的良心。”

“你好烦呐。”

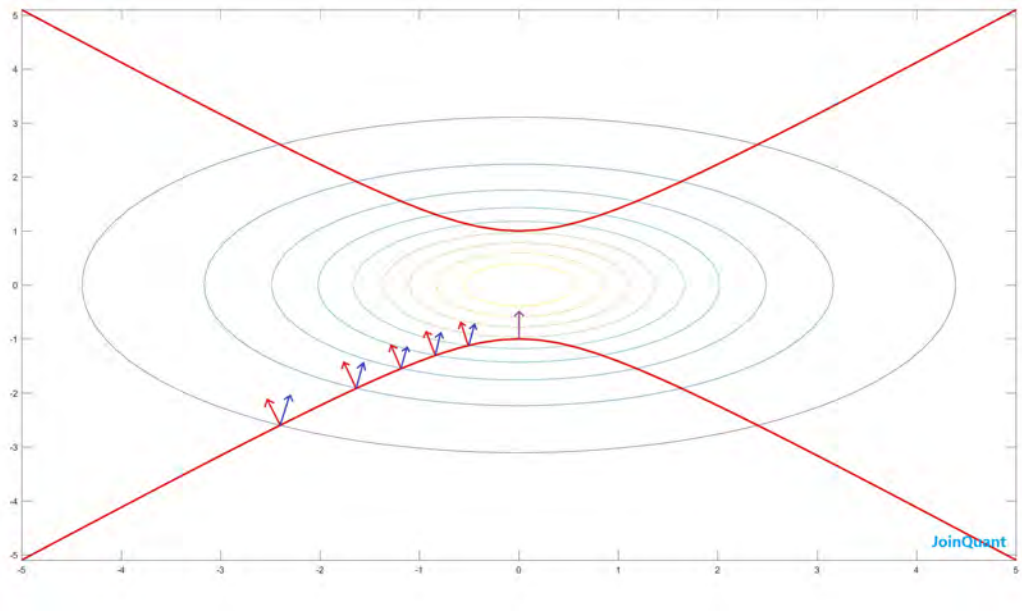
可是，这个公式看着真的是好眼熟啊，眼熟到好像刚刚看过... 是吧，

Misplaced &

$\tilde{g}$ 的梯度和它的水平集永远是垂直的。那么，在极大点上，$\nabla \tilde{g}$ 和水平集垂直，$\nabla \tilde{f}$ 也和水平集垂直，也就是说... $\nabla \tilde{g}$ 和 $\nabla \tilde{f}$ 是平行的！！

事实上，这个现象不只是针对于例子中的 $\tilde{f}$ 和 $\tilde{g}$，对于任何 $\mathcal{C}^1$ 函数 f 和 g 都是这样的。我们有定理如下：
定理. 设 $f, g : \mathbb{R}^n \to \mathbb{R}$ 是 $\mathcal{C}^1$ 函数，$c \in \mathbb{R}$ 并且 $L_c = x \in \mathbb{R}^n : g(x) = c$ 如果 $x_0 \in L_c$ 满足对于所有 $x \in L_c$ 都有 $f(x_0) \geq f(x)$ 或者 $f(x_0) \leq f(x)$，那么存在某个 $\lambda \in \mathbb{R}$ 满足 $\nabla f(x_0) = \lambda \nabla g(x_0)$，满足这个条件的点叫做一个*驻点* (critical point)。
这个定理在 $n = 2$ 情况下的证明正如本文中对 f 和 g 的推导一样，但是在更高的维度里有一些细节会增加证明的难度。不过，你只要坚定地相信这个定理是对的，就一定没有问题了！

那么，对于在上边例子中的函数，如果俯视观看 f 的水平集和行走的路径 $p(t)$，是如下图：



图中红色箭头是 g 的梯度方向，蓝色箭头是 f 的梯度方向，在红线上 f 取值极大点的地方，两者的梯度方向是重叠的，标为紫色。

当应用于更一般性的 f 和 g 的规划问题中，定理告诉我们如果有 $x < em > 0 \in \mathbb{R}^n$ 是 $\max < /em > g(x) = cf(x)$问题的极大点的话，那么它不仅要满足 $g(x_0) = c$，还要存在一个 $\lambda \in \mathbb{R}$ 以至于 $\nabla f(x_0) = \lambda \nabla g(x_0)$。所以如果我们能找出满足这些条件的点，那就可以更简单地找到 x_0。当然，用这个方法算出的点不一定是极大点。它也有可能是极小点或者反射点，这就要用其他的办法来验证了，或者干脆把所有的驻点都算出来，然后比一比哪个大哪个小就知道了。

我们还要拉格朗日乘子干什么?

在以上的计算中，我们没用什么特殊的方法就解决了在 $\widetilde{L}_{-1}$ 上优化 $\tilde{f}$ 的问题，那么为什么还要需要拉格朗日乘子?

上面的例子的一个特点就是，它太简单了，或者说维度太低了，只用两条曲线就完全描述了水平集 $\widetilde{L} < em > -1$，在此之上只需要微积分就可以算出答案。但在更高的维度中，$L < /em > c$是不能用曲线描述的，所以也不会有这么简单的计算方法。但是，前面得出的两个定理在高维中同样适用，并且根据定理，我们知道要找的极大点必须满足

$$(*) \{ g(x) = c, \text{也就是说 } x \text{ 在 } g \text{ 的水平集里; 存在一个 } \lambda \in \mathbb{R} \text{ 满足 } \nabla f(x) = \lambda \nabla g(x).$$

这时，拉格朗日就来了，他给我们一个可以同时囊括上面两个条件的函数。什么函数这么神奇呢? 那就是

$$\mathcal{L}(x, \lambda) = f(x) - \lambda(g(x) - c),$$

这里 $x \in \mathbb{R}^n$ 并且 $\lambda \in \mathbb{R}$，所以 $\mathcal{L}$ 是一个 $\mathbb{R}^{n+1} \to \mathbb{R}$ 的函数。严格来讲，这个函数中的 λ 被称为*拉格朗日乘子* (Lagrange multiplier)，但我们一般说“拉格朗日乘子”的时候指的是这里介绍的方法。

计算 $\mathcal{L}$ 的导数，分别得到

$$\frac{d}{dx_i}\mathcal{L}(x, \lambda) = \frac{d}{dx_i} f(x) - \lambda \frac{d}{dx_i} g(x), \ \forall i = 1, \ldots, n,$$

$$\frac{d}{d\lambda}\mathcal{L}(x,\lambda) = -g(x) + c.$$

或者写作

Undefined control sequence \-

那么，

$$\nabla\mathcal{L}(x,\lambda) = 0 \iff \{\, g(x) = c \; \nabla f(x) = \lambda\nabla g \iff x \text{ 满足}(*).$$

因此，在 g 的水平集上优化 f 的取值时，只需要计算 $\nabla\mathcal{L}(x,\lambda)$ 的零点就可以了。

拉格朗日乘子完全版

拉格朗日乘子的功力还不止于此，它可以应用于有多于一个约束函数的规划问题。
定理. 设 $m \in \mathbb{N}$。对于 $i \in 1, 2, \ldots, m$，g_i 和 f 都是 $\mathbb{R}^n \to \mathbb{R}$ 的 $\mathcal{C}^1$ 函数，并且设 $c_i \in \mathbb{R}$。考虑规划问题

Unknown environment 'align'

定义函数

$$\mathcal{L}(x,\lambda) = f(x) - \lambda_1(g_1(x) - c_1) - \cdots - \lambda_m(g_m(x) - c_m),$$

这里 $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \in \mathbb{R}^m$。那么如果 $\tilde{x} \in \mathbb{R}^n$ 是上述规划问题的极值点，必定存在某个 $\tilde{\lambda} \in \mathbb{R}^m$ 满足

$$\nabla\mathcal{L}(\tilde{x},\tilde{\lambda}) = 0.$$

如果想要理解或者证明这个完整版的定理，我们需要更多的线性代数和微积分的工具，本文中则不进行更深入的讨论。

到此为止我们找到了算出极值点的条件，但是把它算出来终究是个问题。如果 f 和 g_i 都是很丑的函数，那么 $\mathcal{L}$ 必定是其丑无比，计算 $\nabla\mathcal{L}$ 的零点的准确值也是不可能的，所以一般会采取一些数值优化的方法进行估算。一个可行的思路是解决 $\mathbb{R}^{n+m}$ 上的无约束的非线性规划问题

Unknown environment 'align'

而它使用线搜索方法的算法就可以计算。将有约束的规划问题简化成无约束的规划问题，这已经足以体现拉格朗日乘子的价值了。

结语

当然，f 和 g_i 并不一定都非常丑。如果目标函数和约束函数都是二次多项式，求得 $\nabla\mathcal{L}$ 零点的准确值还不是绝望的难，而且很多重要的应用场景都在这个范畴之内，比如... 在量化课堂的下一篇文章中，我们就将针对马科维兹的均值-方差问题使用拉格朗日乘子进行优化，并计算出有效前沿的解析解。

后记

一个白发苍苍的老人独自蹲在荒凉的山脊上，攥着一根细小的木棍，胡乱在土地上划拉着。不论风吹雨打还是春去秋来，他日日都在，一写就是几十载。他书写的的看似是疯狂，又像是真理，模糊、混沌、捉摸不清。一条蜿蜒的红线记载了老人的路程，一头连着彼生，一头连着来世。我们知道，他是一个有良心的人，因为他的良心会说话，你听...

“...艹。”
</br>

金融、经济与市场

【量化课堂】基本面量化研究--钢铁行业景气度预测

前言

亲爱的朋友们大家好，考虑到前期量化小课堂分析文章大多聚焦于公司个股层面的特征性研究，缺少对行业选择的研究，种庄稼都得看行情，分析个股亦该分析行业。因此，接下来这篇文章将给大家换换口味，从宏观角度对行业基本面进行分析。本篇为基本面量化研究系列文章之一，主要逻辑点是从宏观基本面数据出发，找出对行业具有指标性特征的变量，进而对所关心行业的核心变量进行分析预测，为今后深入研究行业轮动与择时打下基础。

作者： 大伟
编辑： 肖睿

导语

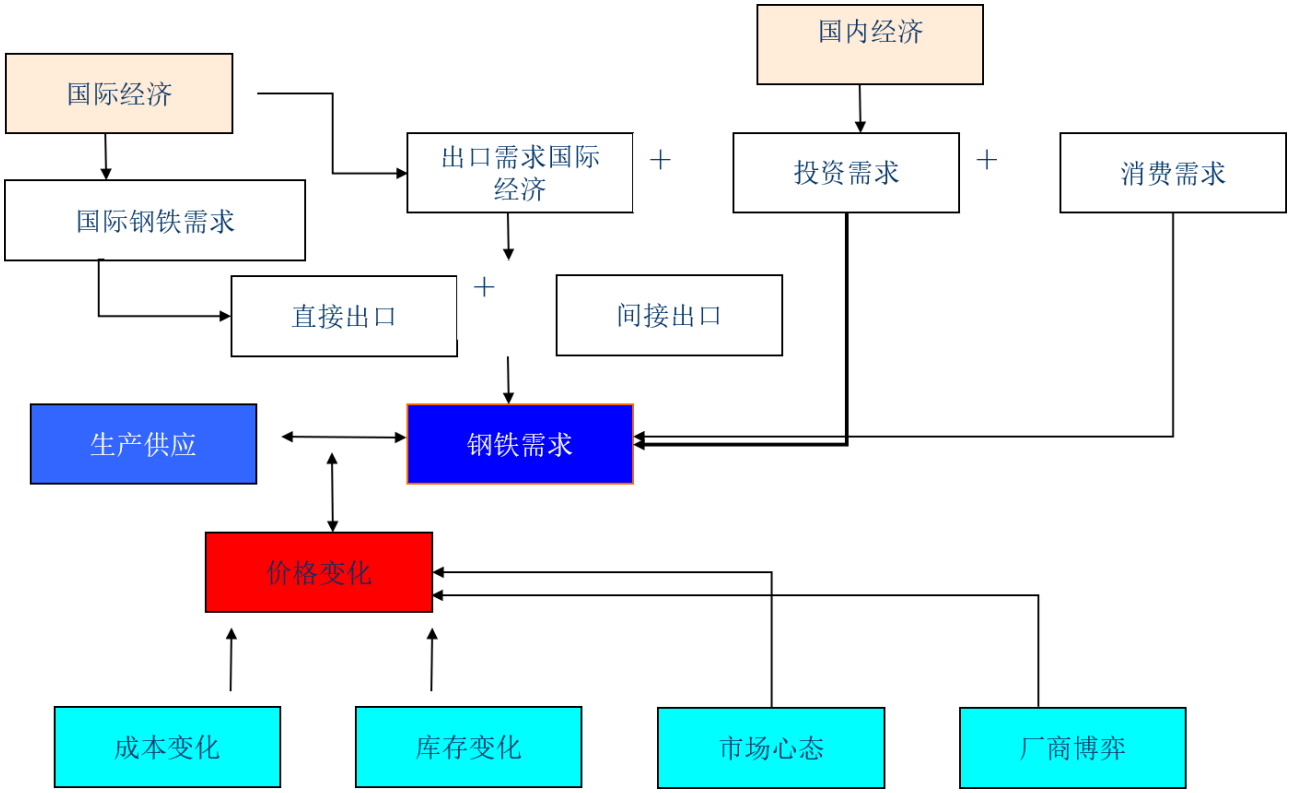
嘿嘿，近期走势最亮眼的莫过于周期钢铁了，因此我们将钢铁行业作为分析标的，钢铁行业作为典型的周期行业，整个行业与经济周期的变动密切相关，同时上下游产业链的走势也影响着钢铁行业的变动，这也决定了钢铁行业的三个属性特征：

1.投资相关性
受到投资拉动作用大，需求受基建、房地产、机械、造船行业等影响

2.周期性
与宏观经济和固定资产投资增速相关

3.区域性
经营环境的差异以及作用方式的不同，导致了钢铁行业区域化特征明显。

钢材市场分析框架



钢铁市场同时受到国内外需求、生产供给、市场博弈等多重因素影响，但核心来看，供需两端的压力变化主导了钢材价格的变动，本篇分析截取一角，主要分析钢铁行业需求的变动，从影响钢铁需求的因素分析，进而构建钢铁行业景气度的预测指标。

具体来说，钢铁需求主要有两个来源，投资需求与消费需求：
投资需求： 主要来自房地产与固定资产投资等
消费需求： 主要来自汽车、机械等

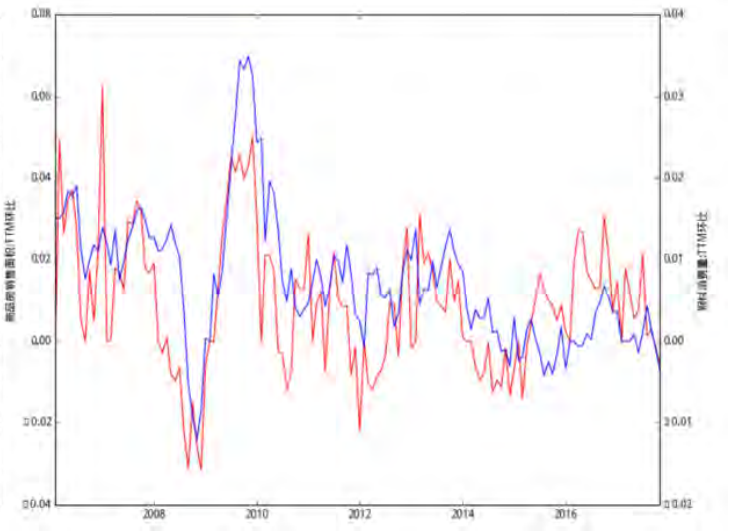
钢铁需求与上下游产业链指标分析

分析钢铁需求，我们就得找一个指标来衡量钢铁需求量。在这里，我们以钢材表观消费量作为钢铁需求的代理变量，将钢材表观消费量与需求影响变量进行不同滞后阶的回归分析，通过比较不同滞后阶回归结果的优劣，我们就能得出表观消费量与其影响指标的领先滞后关系。

需求影响变量取：
房屋新开工面积、商品房销售面积、房屋竣工面积、固定资产投资完成额、基础设施建设投资完成额、房地产投资完成额、汽车产量、M2同比增速。

1.商品房销售面积(TTM环比)与钢材表观消费量(TTM环比)：

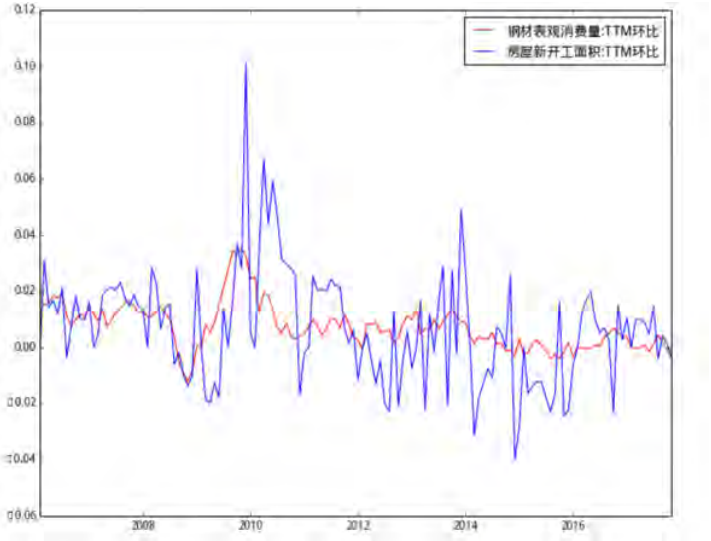
	领先期	相关系数	回归系数	P值	R方
0	-5	0.036177	0.071470	6.747085e-01	0.001309
1	-4	0.156861	0.312623	6.616168e-02	0.024605
2	-3	0.273446	0.549670	1.125678e-03	0.074773
3	-2	0.374056	0.753450	5.298867e-06	0.139918
4	-1	0.514016	1.053551	7.075580e-11	0.264213
5	0	0.600553	1.224321	2.788067e-15	0.360664
6	1	0.631484	1.288070	4.655938e-17	0.398772
7	2	0.610734	1.247909	1.127121e-15	0.372995
8	3	0.573210	1.175172	1.651489e-13	0.328570
9	4	0.496066	1.023276	6.133997e-10	0.246081
10	5	0.440559	0.912868	7.143625e-08	0.194092



上图中，左图为商品房销售面积TTM环比对不同滞后领先期的钢材表观消费量TTM环比做回归得到的系数表，右图为两者数据的时序作图。
左图中，领先期为1，表示商品房销售面积领先1个月的数值与钢材消费量做回归（也就是t-1月的商品房销售面积与t月钢材消费量回归），领先期为-1，表示商品房销售面积滞后1个月的数值与钢材消费量做回归（即t月的商品房销售面积与t-1月钢材消费量做回归，也可以表示为钢材消费量领先商品房销售面积1个月）。
通过比较不同期的R方的大小，判断两者最恰当的领先滞后关系。如上表，在领先期为1、2时回归方程R方达到最大，也就表示商品房销售面积领先钢材消费量1-2个月。

2.房屋新开工面积(TTM环比)与钢材表观消费量(TTM环比)：

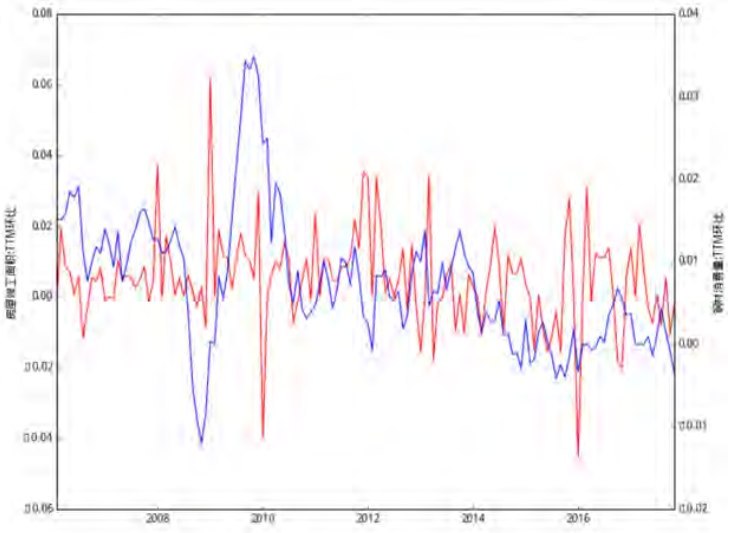
	领先期	相关系数	回归系数	P值	R方
0	-5	0.494652	1.210657	8.057526e-10	0.244680
1	-4	0.511898	1.250358	1.389922e-10	0.262039
2	-3	0.493004	1.204668	7.025351e-10	0.243053
3	-2	0.500122	1.220213	3.145176e-10	0.250122
4	-1	0.514065	1.256256	7.041873e-11	0.264262
5	0	0.470194	1.142141	3.548642e-09	0.221082
6	1	0.393328	0.957595	1.401036e-06	0.154707
7	2	0.343459	0.838657	3.259667e-05	0.117964
8	3	0.188918	0.463060	2.592787e-02	0.035690
9	4	0.140227	0.345719	1.009161e-01	0.019664
10	5	0.108444	0.268907	2.071656e-01	0.011760



可以看出，房屋新开工面积并不是钢材表观消费量的领先指标，更确切的说滞后于钢材表观消费量。可能原因为地产商新开工并不是
一个主动的行为，而且根据经济景气度来定的。 当经济景气度提升，钢铁需求敏感，优先开始提升，而地产商更倾向于当看到经济起来之后才会加大新开工面积。

3.房屋竣工面积(TTM环比)与钢材表观消费量(TTM环比)：

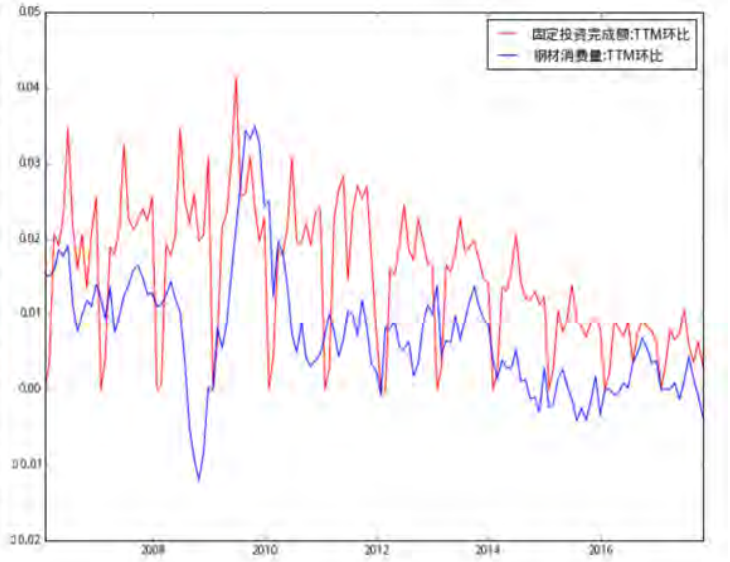
	领先期	相关系数	回归系数	P值	R方
0	-5	0.039678	0.064767	0.645266	0.001574
1	-4	-0.019928	-0.032464	0.816546	0.000397
2	-3	-0.007979	-0.012993	0.925729	0.000064
3	-2	-0.038046	-0.061846	0.655386	0.001447
4	-1	0.008400	0.013662	0.921246	0.000071
5	0	0.104532	0.168999	0.215708	0.010927
6	1	0.079987	0.129620	0.345758	0.006398
7	2	0.164041	0.265412	0.052782	0.026910
8	3	0.145610	0.236506	0.087202	0.021202
9	4	0.173375	0.282578	0.041992	0.030059
10	5	0.174423	0.285966	0.041499	0.030423



从回归的R方我们可以看出，在不同的滞后期，房屋竣工面积对钢材表观消费量的解释能力都不强。从直观逻辑上来讲，房屋竣工面积是事后状态的统计，跟钢材消费量并没有很强的直观逻辑。因此，房屋竣工面积并不是预测钢材消费量的恰当指标。

4.固定投资完成额(TTM环比)与钢材表观消费量(TTM环比)：

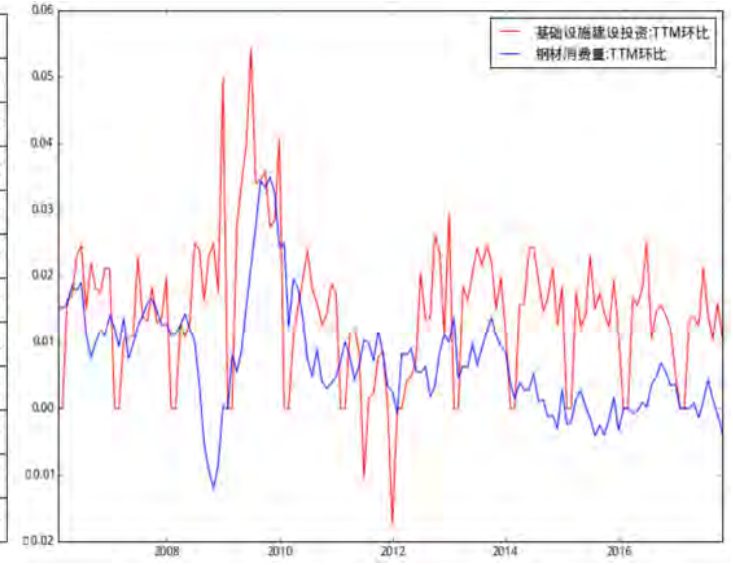
	领先期	相关系数	回归系数	P值	R方
0	-5	0.244045	0.274462	0.004055	0.059558
1	-4	0.261286	0.293987	0.001965	0.068270
2	-3	0.314153	0.353506	0.000166	0.098692
3	-2	0.347938	0.391230	0.000025	0.121061
4	-1	0.361147	0.406961	0.000011	0.130427
5	0	0.384918	0.435056	0.000002	0.148162
6	1	0.369019	0.415936	0.000007	0.136175
7	2	0.338102	0.381084	0.000044	0.114313
8	3	0.320925	0.361044	0.000117	0.102993
9	4	0.301903	0.340731	0.000320	0.091145
10	5	0.311078	0.352965	0.000216	0.096770



可见，固定投资完成额与钢材表观消费量指标基本同步，因此固定资产投资完成额也不是预测钢材消费量的恰当指标

5.基础设施建设投资(TTM环比)与钢材表观消费量(TTM环比)：

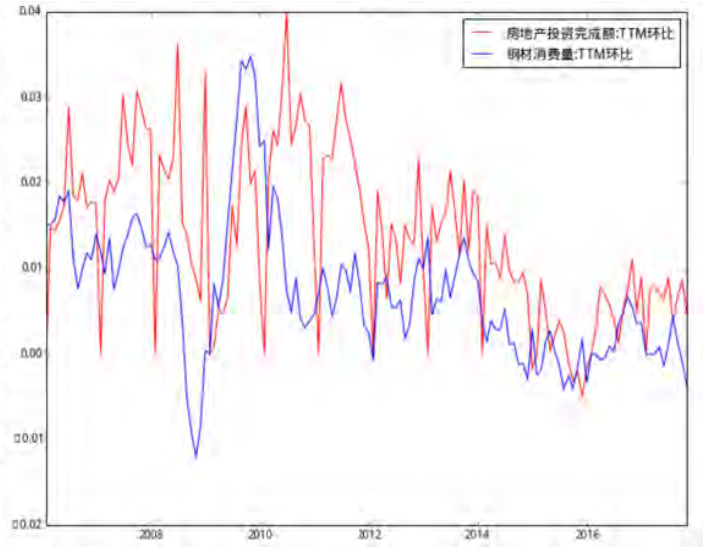
	领先期	相关系数	回归系数	P值	R方
0	-5	-0.008419	-0.011117	0.922218	0.000071
1	-4	0.043222	0.057071	0.614712	0.001868
2	-3	0.088951	0.117406	0.297732	0.007912
3	-2	0.133310	0.175617	0.116363	0.017772
4	-1	0.180527	0.238504	0.032179	0.032590
5	0	0.248761	0.328629	0.002834	0.061882
6	1	0.292400	0.387371	0.000434	0.085498
7	2	0.288777	0.383791	0.000540	0.083392
8	3	0.292671	0.390291	0.000472	0.085657
9	4	0.298607	0.400987	0.000374	0.089166
10	5	0.316240	0.426768	0.000167	0.100007



检验发现，基础设施建设投资对于钢材表观消费量有较好的领先性，领先期为3-5个月。

6.房地产投资完成额(TTM环比)与钢材表观消费量(TTM环比)：

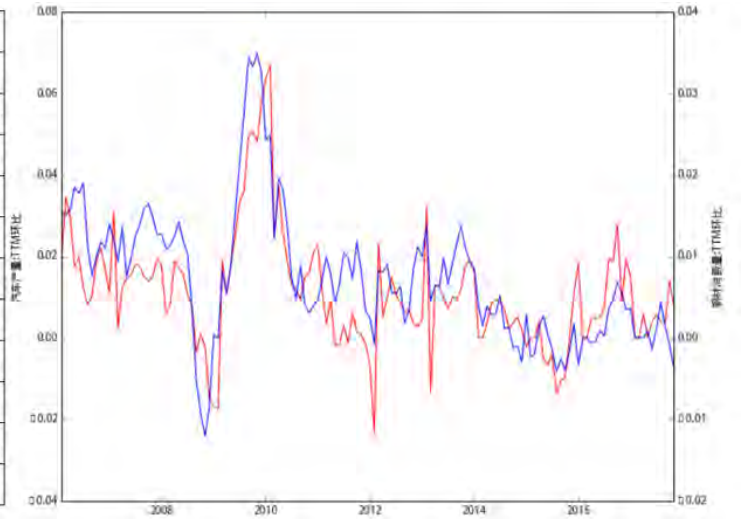
	领先期	相关系数	回归系数	P值	R方
0	-5	0.521876	0.627477	6.164546e-11	0.272354
1	-4	0.486885	0.584353	1.402242e-09	0.237057
2	-3	0.488694	0.586285	1.037837e-09	0.238821
3	-2	0.484789	0.580433	1.282120e-09	0.235020
4	-1	0.477737	0.569918	2.093674e-09	0.228233
5	0	0.455423	0.543601	1.236175e-08	0.207410
6	1	0.395263	0.471935	1.230126e-06	0.156233
7	2	0.330166	0.395118	6.780456e-05	0.109010
8	3	0.227008	0.272194	7.201134e-03	0.051533
9	4	0.170021	0.204532	4.618849e-02	0.028907
10	5	0.121208	0.146595	1.582706e-01	0.014691



房地产投资对钢材消费是一个很关键的变量，普遍认为钢材消费量受到房地产投资影响较大，但检验发现，房地产投资完成额并不是钢材消费量的领先指标，其表现滞后于钢材消费量，因此房地产投资完成额也不是预测钢材消费量的恰当指标，其原因与房屋新开工面积滞后的理由一致。

7.汽车产量(TTM环比)与钢材表观消费量(TTM环比)：

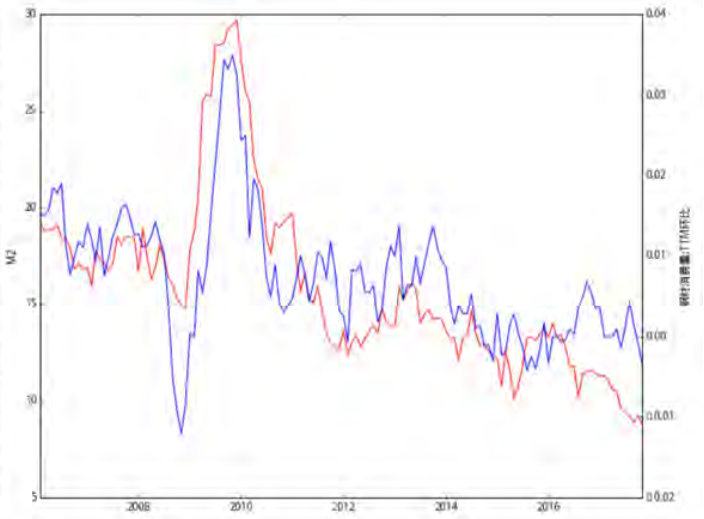
	领先期	相关系数	回归系数	P值	R方
0	-5	0.427195	0.748433	1.928531e-07	0.182495
1	-4	0.525598	0.920078	3.611841e-11	0.276253
2	-3	0.634736	1.111433	4.846082e-17	0.402890
3	-2	0.724027	1.272897	5.128554e-24	0.524216
4	-1	0.736949	1.303056	2.077786e-25	0.543094
5	0	0.785624	1.382278	5.478489e-31	0.617205
6	1	0.690908	1.219318	2.525497e-21	0.477354
7	2	0.648352	1.147791	4.710913e-18	0.420361
8	3	0.545613	0.968759	3.766158e-12	0.297693
9	4	0.461471	0.824318	1.222477e-08	0.212955
10	5	0.346763	0.622978	3.308526e-05	0.120245



汽车是钢铁下游的一个重要机械产业，可以看出，汽车产量同钢材表观消费量表现出较强的同步性特征，回归R方在领先期为零时达到最大值，但并没有明显的领先滞后关系。

8.M2增速(同比)与钢材表观消费量(TTM环比)：

	领先期	相关系数	回归系数	P值	R方
0	-5	0.439704	246.609769	7.622284e-08	0.193339
1	-4	0.531815	298.147224	1.920301e-11	0.282828
2	-3	0.606799	340.547427	2.420984e-15	0.368205
3	-2	0.674200	378.166519	6.840686e-20	0.454546
4	-1	0.728889	407.935127	1.238320e-24	0.531279
5	0	0.761632	424.306728	3.678241e-28	0.580083
6	1	0.770310	426.816460	6.021124e-29	0.593378
7	2	0.769847	424.547981	1.069960e-28	0.592665
8	3	0.749096	411.007586	2.849955e-26	0.561145
9	4	0.720866	394.912805	2.087951e-23	0.519648
10	5	0.669588	366.094684	3.704375e-19	0.448349



M2同比增速是反应央行货币供给量的指标，在一定程度上衡量市场流动性的松紧。货币政策的变化可以对经济基本面产生指导性作用，但整个传导需要一个过程，因此传导至钢铁行业基本面具有时滞性。检验发现其为钢材表观消费量的一个较好的领先指标，领先期为1-2个月。

钢铁行业景气度预测

通过上面的分析我们可以得出：

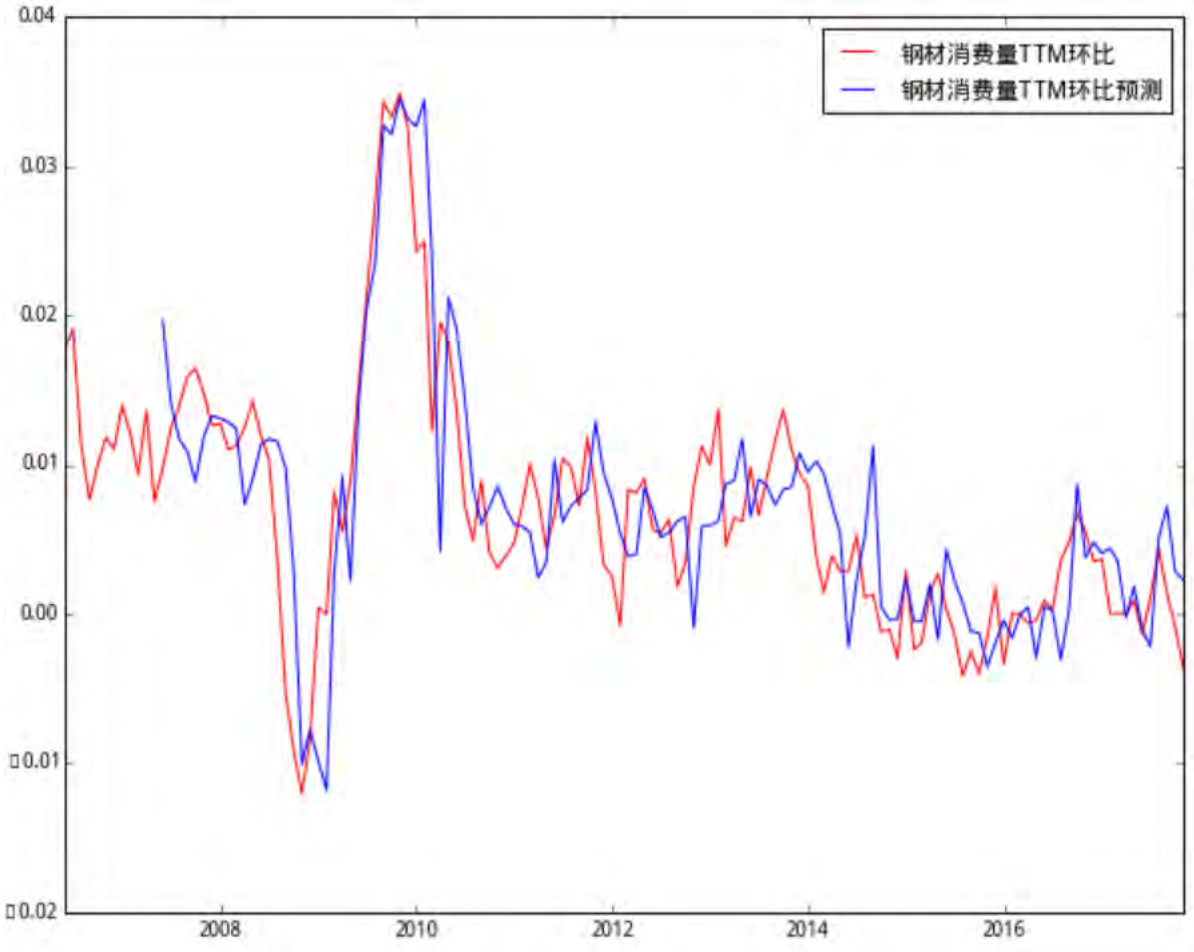
1.新开工房面积、房地产投资完成额都是钢材表观消费量的滞后指标
2.固定资产投资完成额、汽车产量与钢材表观消费量基本同步
3.商品房销售面积，基础设施建设投资、M2为钢材表观消费量的领先指标，领先期分别为1~2个月，3~5个月，1~2个月

要想对钢材需求进行预测，得是用在当期可以获得的数据，来预测未来一段时间的钢材需求。因此，滞后与同步的指标不能选，得选取上述领先指标。通过将三个领先指标与表观消费量进行回归分析，我们可以往前预测两个月的钢材表观消费量TTM环比与钢铁行业营业收入TTM环比（这里的营业收入是指大中型钢铁企业营业收入）

我们拿前一年的滚动样本作回归来衡量系数，以此预测后两个月的钢材表观消费量TTM环比与钢铁行业营业收入TTM环比
例如：
当前为2017年5月，则样本数据为2016年5月至2017年5月，预测值为2017年7月数据。

钢材表观消费量TTM环比预测结果如下：

$$\text{钢材表观消费量TTM环比} < em > T = \beta_1 \cdot \text{商品房销售面积TTM环比} < /em > T-2 + \beta < em > 2 \cdot \text{基础设施建设投资TTM环比} < /em > T-4 +$$



钢铁行业营业收入TTM环比预测结果如下：

Unknown environment 'align'

总结

通过上述模型，我们可以较为有效的预测：
1.钢材表观消费量TTM环比增速（预测期为往后2个月）
2.钢铁行业营业收入TTM环比增速（预测期为往后2个月）
本篇为行业量化基本面的基础性分析，提供基本面量化分析的一种框架，可以将其作为投资股票时行业选择一种思路。对于基于基本面量化的策略性研究，后续我们将继续更新。

注：以上使用数据均来自wind数据库，如需要可以通过链接下载

参考文章：中信建投基本面量化系列研究之八：周期行业基本面量化之钢铁篇

In [1]:

```
import pandas as pd
import numpy as np
from jqdata import gta
import matplotlib.pyplot as plt
import datetime
import statsmodels.api as sm
```

数据的导入¶

（包括钢材表观消费量、M2、房屋新开工面积、房屋竣工面积、固定资产投资完成额、商品房销售面积、基础设施建设投资、房地产开发投资完成额、钢铁产量等的月度数据）

In [7]:

```
steel_fd_data = pd.read_excel('钢铁基本面数据.xls',index_col=0)
###由于数据是wind直接下来的，所以不太干净，先预处理一下、去掉一些注释行
steel_fd_data = steel_fd_data[1:-2]
##数据钢材表观消费量为主，截断此数据开始之前的日期，此数据从2004年1月开始
steel_fd_data = steel_fd_data[steel_fd_data.index>=datetime.datetime(2004,1,1)]
```

因为有些数据是当年累计值（比如房屋新开工面积），为了方便之后做TTM环比比较，先将其处理为当月数据

由于一月份大多属于过年春节期间，因此不少数据在一月份为缺失值

In [14]:

```
need_adjust_columns = ['房屋新开工面积:累计值','房屋竣工面积:累计值','商品房销售面积:累计值','固定资产投资完成额:累计值',
                       '固定资产投资完成额:基础设施建设投资:累计值','房地产开发投资完成额:累计值']

steel_fd_data['month'] = steel_fd_data.index.map(lambda x:x.month)
#这些列都是一月份缺失数据的
for column in need_adjust_columns:
    if column != '固定资产投资完成额:基础设施建设投资:累计值':
        steel_fd_data[column.split(':')[0]] = steel_fd_data[column] - steel_fd_data[column].shift(1)
        steel_fd_data.ix[steel_fd_data['month']<=2,column.split(':')[0]] = steel_fd_data[steel_fd_data['month']<=2][column]
    else:
        steel_fd_data[column.split(':')[1]] = steel_fd_data[column] - steel_fd_data[column].shift(1)
        steel_fd_data.ix[steel_fd_data['month']<=2,column.split(':')[0]] = steel_fd_data[steel_fd_data['month']<=2][column]

###钢铁行业收入一月份没有缺失
column_ = '钢铁行业:大中型企业:销售收入:累计值'
steel_fd_data['钢铁行业销售收入'] = steel_fd_data[column_] - steel_fd_data[column_].shift(1)
steel_fd_data.ix[steel_fd_data['month'] == 1,'钢铁行业销售收入'] = steel_fd_data[steel_fd_data['month']==1][column_]

#取出接下来需要用的主要数据
data_columns_list = steel_fd_data.columns.tolist()
clean_data = steel_fd_data[data_columns_list[:2]+data_columns_list[-7:]+['产量:汽车:当月值']]
#去除第一行
clean_data = clean_data[1:]
```

对数据进行滚动TTM处理，并取其环比值¶

In [17]:

```
clean_data.fillna(0,inplace=True)
TTM_columns = ['表观消费量:钢材:当月值','房屋新开工面积','房屋竣工面积','固定资产投资完成额','基础设施建设投资',\
               '商品房销售面积','房地产开发投资完成额','钢铁行业销售收入','产量:汽车:当月值']
for column in TTM_columns:
    clean_data[column+'TTM'] = pd.rolling_sum(clean_data[column],12)
    clean_data[column+'TTM_环比'] = clean_data[column+'TTM']/clean_data[column+'TTM'].shift(1)-1
```

这里要对钢材表观消费量数据做一下调整：原因是其数据的不规整性：

其数据在15年及之前在1、2月份都是有的，但是16年之后1、2月份都缺失，如果这时候直接TTM环比的话，16年年初的时候将会有一个大幅的下降， 但这其实是由数据的不规整带来的。因此，在这里将16、17年1、2月的TTM环比数据都改为0

同样，汽车产量从14年之后1、2月的TTM环比数据都改为0

截取整段数据从2006年开始

In [18]:

```
clean_data['year'] = clean_data.index.map(lambda x:x.year)
clean_data['month'] = clean_data.index.map(lambda x:x.month)

clean_data.ix[(clean_data['year'].isin([2016,2017]))&(clean_data['month'].isin([1,2])),'表观消费量:钢材:当月值TTM_环比'] = 0

clean_data.ix[(clean_data['year'].isin([2014,2015,2016,2017]))&(clean_data['month'].isin([1,2])),'产量:汽车:当月值TTM_环比'] = 0

clean_data = clean_data[clean_data['year']>=2006]
clean_data.ix[(clean_data['year'] == 2006)&(clean_data['month']==8),'商品房销售面积TTM_环比'] = 0
```

```
/opt/conda/lib/python3.4/site-packages/pandas/core/indexing.py:415: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

In [20]:

```
use_columns = [i+'TTM_环比' for i in TTM_columns]+['M2:同比']
use_data = clean_data[use_columns]
use_data.dropna(inplace=True)

###由于列名太长，这里做一个简写,但其值是表示的为TTM_环比值(但M2为同比值)
use_data.columns = ['钢材消费量TTM环比','新开工房TTM环比','竣工房TTM环比','固投TTM环比',\
                    '基建TTM环比','房销售面积TTM环比','房地产投资TTM环比','钢铁行业收入TTM环比','汽车产量TTM环比','M2']
```

```
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  app.launch_new_instance()
```

钢铁基本面数据之间的关系以及领先与滞后¶

In [21]:

```
def plot_line_chart(x_axis,y1_axis,y2_axis,label1,label2):

    Fig = plt.figure(figsize(10,8))
    Ax = Fig.add_subplot(111)
    lines = Ax.plot(x_axis,y1_axis,'r-',x_axis,y2_axis,'b-')
    lines[0].set_label(label1)
    lines[1].set_label(label2)
    Ax.legend(loc = 0)

def plot_line_chart_double_yaxis(x_axis,y1_axis,y2_axis,label1,label2):

    Fig = plt.figure(figsize(10,8))
    Ax = Fig.add_subplot(111)
    Ax.plot(x_axis,y1_axis,'r-')
    Ax.set_ylabel(label1)

    Ax_ = Ax.twinx()
    Ax_.plot(x_axis,y2_axis,'b-')
    Ax_.set_ylabel(label2)

def calculate_lag(data,column1,column2):

    regre_results=[]
    for i in range(-5,6):
        reg_data = pd.concat([data[column1].shift(i),data[column2]],axis =1)
        reg_data.dropna(inplace=True)
        y=reg_data[column1]
        x=reg_data[column2]
        x=sm.add_constant(x)
        est=sm.OLS(y,x)
        results=est.fit()
        r2 = results.rsquared
        coef = results.params[1]
        p = results.pvalues[1]
        corr = reg_data[column1].corr(reg_data[column2])

        regre_results.append([i,corr,coef,p,r2])

    regre_df = pd.DataFrame(regre_results,columns=['领先期','相关系数','回归系数','P值','R方'])

    return regre_df
```

钢材表观消费量(TTM)与房屋新开工面积(TTM)¶

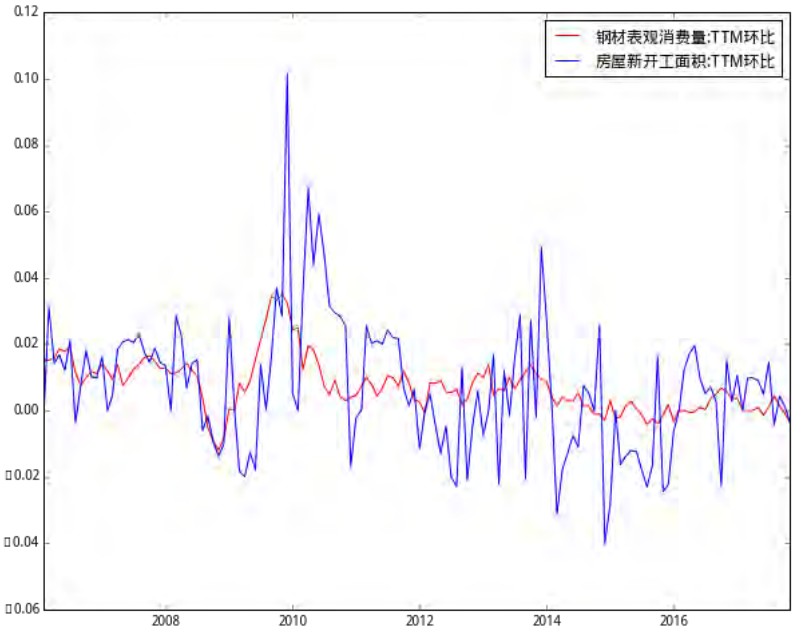
In [22]:

```
plot_line_chart(use_data.index,use_data['钢材消费量TTM环比'],use_data['新开工房TTM环比'],'钢材表观消费量:TTM环比','房屋新开工面积:TTM环比'
chart_xkg_gxf = calculate_lag(use_data,'新开工房TTM环比','钢材消费量TTM环比')
chart_xkg_gxf
```

Out[22]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.494652	1.210657	8.057526e-10	0.244680
1	-4	0.511898	1.250358	1.389922e-10	0.262039
2	-3	0.493004	1.204668	7.025351e-10	0.243053
3	-2	0.500122	1.220213	3.145176e-10	0.250122
4	-1	0.514065	1.256256	7.041873e-11	0.264262
5	0	0.470194	1.142141	3.548642e-09	0.221082
6	1	0.393328	0.957595	1.401036e-06	0.154707
7	2	0.343459	0.838657	3.259667e-05	0.117964
8	3	0.188918	0.463060	2.592787e-02	0.035690

	领先期	相关系数	回归系数	P值	R方
9	4	0.140227	0.345719	1.009161e-01	0.019664
10	5	0.108444	0.268907	2.071656e-01	0.011760



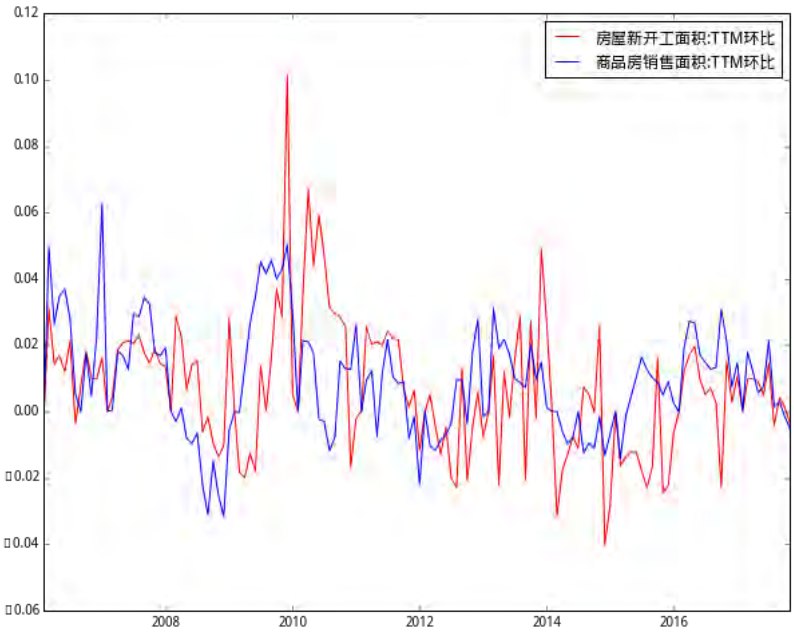
房屋新开工面积(TTM)与商品房销售面积(TTM)¶

In [23]:

```
plot_line_chart(use_data.index,use_data['新开工房TTM环比'],use_data['房销售面积TTM环比'],'房屋新开工面积:TTM环比','商品房销售面积:TTM环比'
chart_xkg_fxs = calculate_lag(use_data,'新开工房TTM环比','房销售面积TTM环比')
chart_xkg_fxs
```

Out[23]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.439651	0.523825	7.653033e-08	0.193293
1	-4	0.430737	0.512420	1.338582e-07	0.185534
2	-3	0.420003	0.499698	2.645205e-07	0.176403
3	-2	0.309651	0.368403	1.969272e-04	0.095884
4	-1	0.253768	0.303059	2.394450e-03	0.064398
5	0	0.369778	0.440596	5.924863e-06	0.136736
6	1	0.147222	0.175434	8.149016e-02	0.021674
7	2	0.070530	0.085811	4.076314e-01	0.004974
8	3	0.058054	0.070915	4.972431e-01	0.003370
9	4	-0.070280	-0.086537	4.127221e-01	0.004939
10	5	-0.101976	-0.126894	2.357181e-01	0.010399



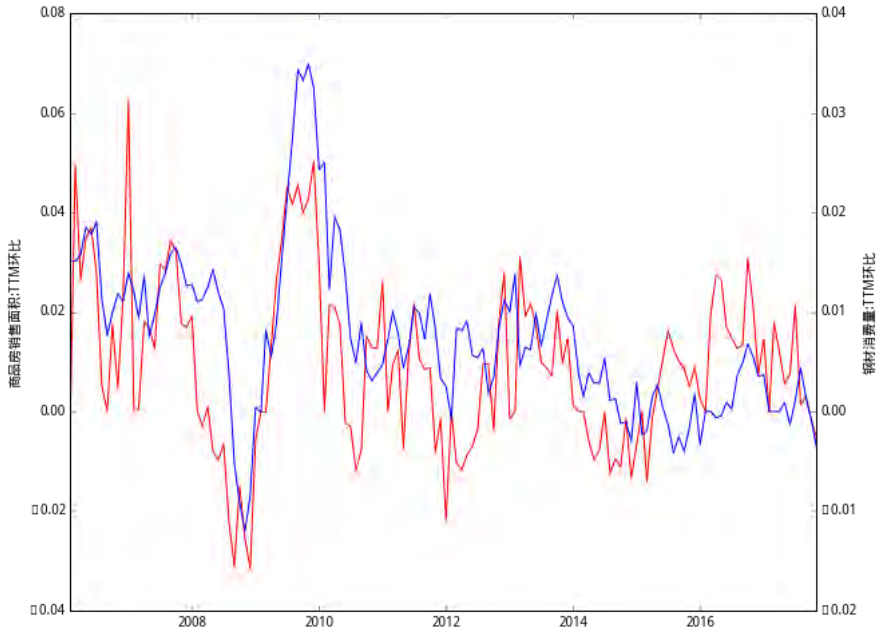
商品房销售面积(TTM)与钢材表观消费量(TTM)¶

In [24]:

```
plot_line_chart_double_yaxis(use_data.index,use_data['房销售面积TTM环比'],use_data['钢材消费量TTM环比'],'商品房销售面积:TTM环比','钢材消费
chart_fxs_gxf = calculate_lag(use_data,'房销售面积TTM环比','钢材消费量TTM环比')
chart_fxs_gxf
```

Out[24]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.036177	0.071470	6.747085e-01	0.001309
1	-4	0.156861	0.312623	6.616168e-02	0.024605
2	-3	0.273446	0.549670	1.125678e-03	0.074773
3	-2	0.374056	0.753450	5.298867e-06	0.139918
4	-1	0.514016	1.053551	7.075580e-11	0.264213
5	0	0.600553	1.224321	2.788067e-15	0.360664
6	1	0.631484	1.288070	4.655938e-17	0.398772
7	2	0.610734	1.247909	1.127121e-15	0.372995
8	3	0.573210	1.175172	1.651489e-13	0.328570
9	4	0.496066	1.023276	6.133997e-10	0.246081
10	5	0.440559	0.912868	7.143625e-08	0.194092



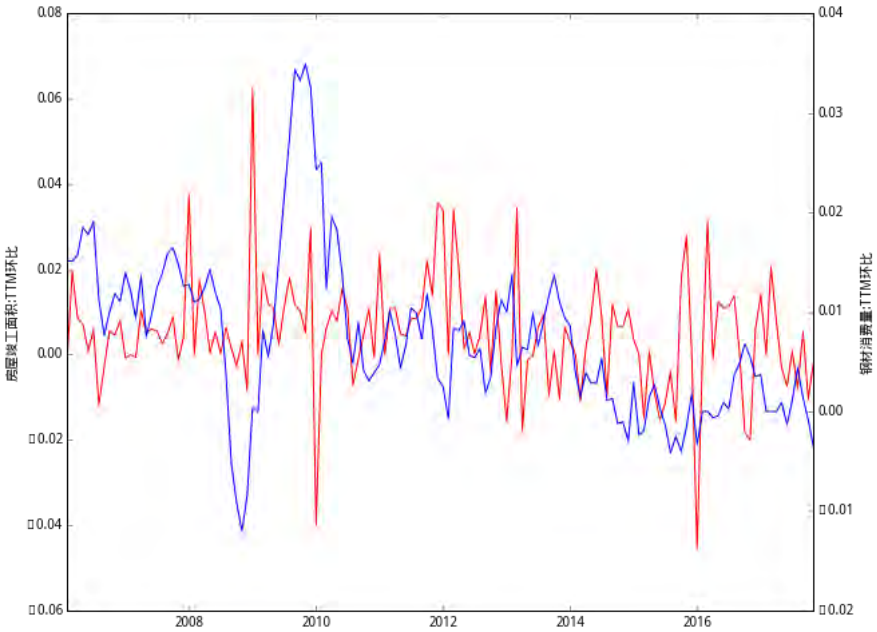
房屋竣工面积(TTM)与钢材消费量(TTM)¶

In [25]:

```
plot_line_chart_double_yaxis(use_data.index,use_data['竣工房TTM环比'],use_data['钢材消费量TTM环比'],'房屋竣工面积:TTM环比','钢材消费量:TTM
chart_jgf_gxf = calculate_lag(use_data,'竣工房TTM环比','钢材消费量TTM环比')
chart_jgf_gxf
```

Out[25]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.039678	0.064767	0.645266	0.001574
1	-4	-0.019928	-0.032464	0.816546	0.000397
2	-3	-0.007979	-0.012993	0.925729	0.000064
3	-2	-0.038046	-0.061846	0.655386	0.001447
4	-1	0.008400	0.013662	0.921246	0.000071
5	0	0.104532	0.168999	0.215708	0.010927
6	1	0.079987	0.129620	0.345758	0.006398
7	2	0.164041	0.265412	0.052782	0.026910
8	3	0.145610	0.236506	0.087202	0.021202
9	4	0.173375	0.282578	0.041992	0.030059
10	5	0.174423	0.285966	0.041499	0.030423



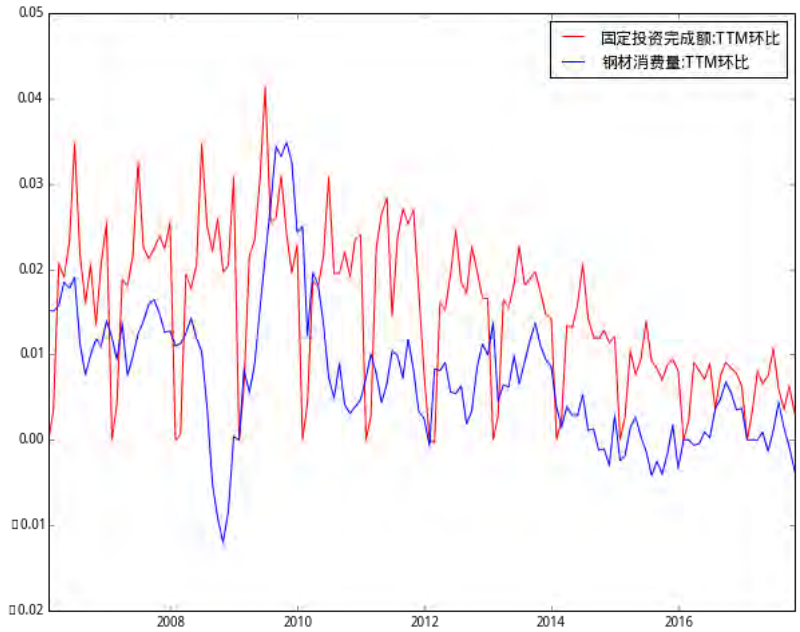
固定投资完成额(TTM)与钢材消费量(TTM)¶

In [26]:

```
plot_line_chart(use_data.index,use_data['固投TTM环比'],use_data['钢材消费量TTM环比'],'固定投资完成额:TTM环比','钢材消费量:TTM环比')
chart_gt_gxf = calculate_lag(use_data,'固投TTM环比','钢材消费量TTM环比')
chart_gt_gxf
```

Out[26]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.244045	0.274462	0.004055	0.059558
1	-4	0.261286	0.293987	0.001965	0.068270
2	-3	0.314153	0.353506	0.000166	0.098692
3	-2	0.347938	0.391230	0.000025	0.121061
4	-1	0.361147	0.406961	0.000011	0.130427
5	0	0.384918	0.435056	0.000002	0.148162
6	1	0.369019	0.415936	0.000007	0.136175
7	2	0.338102	0.381084	0.000044	0.114313
8	3	0.320925	0.361044	0.000117	0.102993
9	4	0.301903	0.340731	0.000320	0.091145
10	5	0.311078	0.352965	0.000216	0.096770



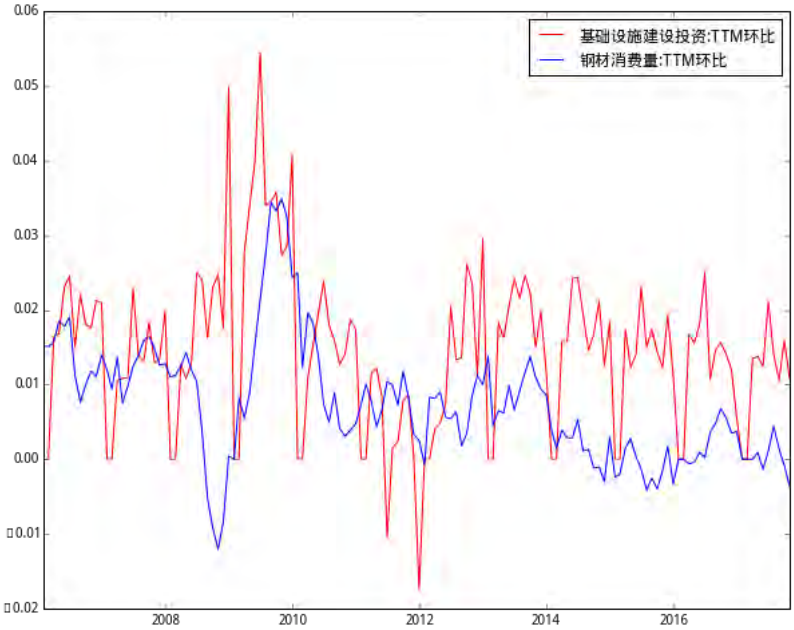
基础设施建设投资(TTM)与钢材表观消费量(TTM)¶

In [27]:

```
plot_line_chart(use_data.index,use_data['基建TTM环比'],use_data['钢材消费量TTM环比'],'基础设施建设投资:TTM环比','钢材消费量:TTM环比')
chart_gt_gxf = calculate_lag(use_data,'基建TTM环比','钢材消费量TTM环比')
chart_gt_gxf
```

Out[27]:

	领先期	相关系数	回归系数	P值	R方
0	-5	-0.008419	-0.011117	0.922218	0.000071
1	-4	0.043222	0.057071	0.614712	0.001868
2	-3	0.088951	0.117406	0.297732	0.007912
3	-2	0.133310	0.175617	0.116363	0.017772
4	-1	0.180527	0.238504	0.032179	0.032590
5	0	0.248761	0.328629	0.002834	0.061882
6	1	0.292400	0.387371	0.000434	0.085498
7	2	0.288777	0.383791	0.000540	0.083392
8	3	0.292671	0.390291	0.000472	0.085657
9	4	0.298607	0.400987	0.000374	0.089166
10	5	0.316240	0.426768	0.000167	0.100007



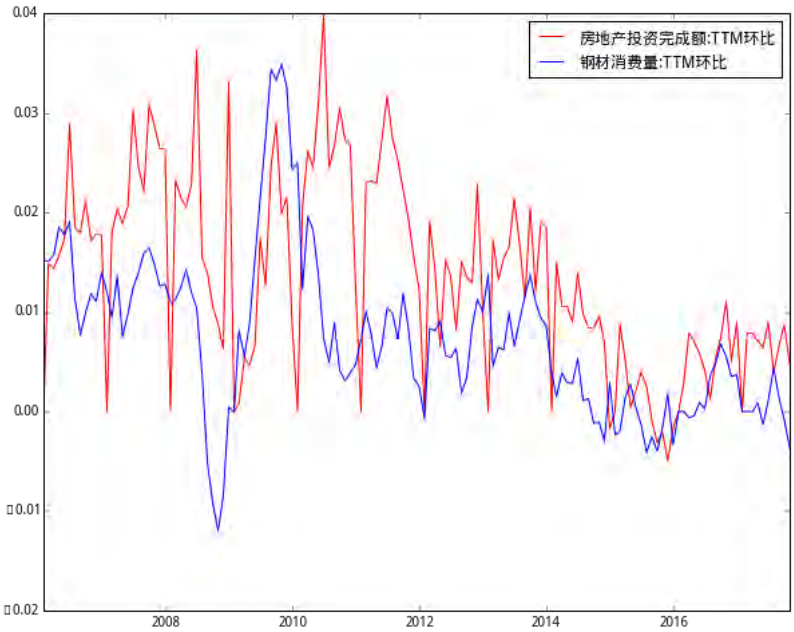
房地产投资和钢材消费量(TTM)¶

In [28]:

```
plot_line_chart(use_data.index,use_data['房地产投资TTM环比'],use_data['钢材消费量TTM环比'],'房地产投资完成额:TTM环比','钢材消费量:TTM环比'
chart_gt_gxf = calculate_lag(use_data,'房地产投资TTM环比','钢材消费量TTM环比')
chart_gt_gxf
```

Out[28]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.521876	0.627477	6.164546e-11	0.272354
1	-4	0.486885	0.584353	1.402242e-09	0.237057
2	-3	0.488694	0.586285	1.037837e-09	0.238821
3	-2	0.484789	0.580433	1.282120e-09	0.235020
4	-1	0.477737	0.569918	2.093674e-09	0.228233
5	0	0.455423	0.543601	1.236175e-08	0.207410
6	1	0.395263	0.471935	1.230126e-06	0.156233
7	2	0.330166	0.395118	6.780456e-05	0.109010
8	3	0.227008	0.272194	7.201134e-03	0.051533
9	4	0.170021	0.204532	4.618849e-02	0.028907
10	5	0.121208	0.146595	1.582706e-01	0.014691



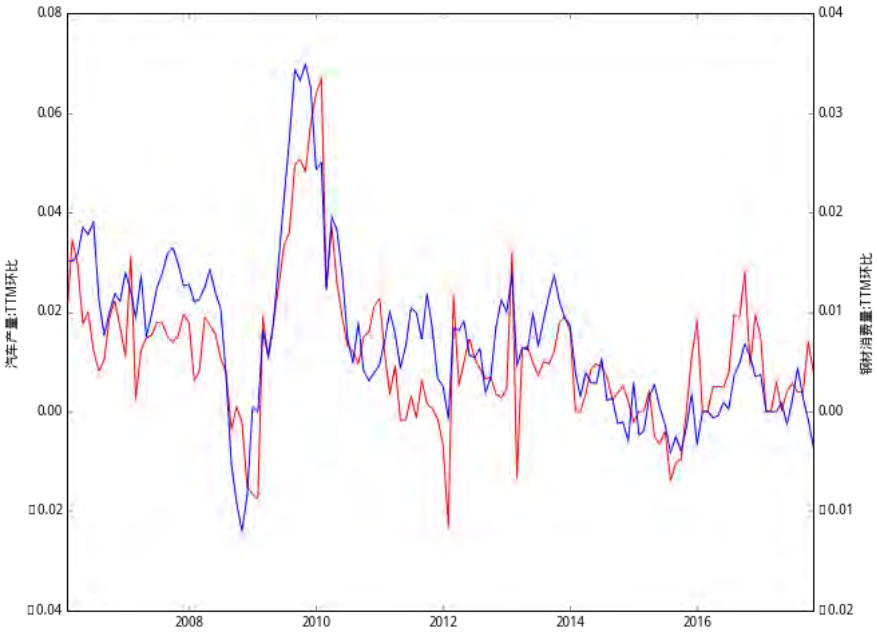
汽车产量(TTM)与钢材消费量(TTM)¶

In [29]:

```
plot_line_chart_double_yaxis(use_data.index,use_data['汽车产量TTM环比'],use_data['钢材消费量TTM环比'],'汽车产量:TTM环比','钢材消费量:TTM环
chart_gt_gxf = calculate_lag(use_data,'汽车产量TTM环比','钢材消费量TTM环比')
chart_gt_gxf
```

Out[29]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.427195	0.748433	1.928531e-07	0.182495
1	-4	0.525598	0.920078	3.611841e-11	0.276253
2	-3	0.634736	1.111433	4.846082e-17	0.402890
3	-2	0.724027	1.272897	5.128554e-24	0.524216
4	-1	0.736949	1.303056	2.077786e-25	0.543094
5	0	0.785624	1.382278	5.478489e-31	0.617205
6	1	0.690908	1.219318	2.525497e-21	0.477354
7	2	0.648352	1.147791	4.710913e-18	0.420361
8	3	0.545613	0.968759	3.766158e-12	0.297693
9	4	0.461471	0.824318	1.222477e-08	0.212955
10	5	0.346763	0.622978	3.308526e-05	0.120245



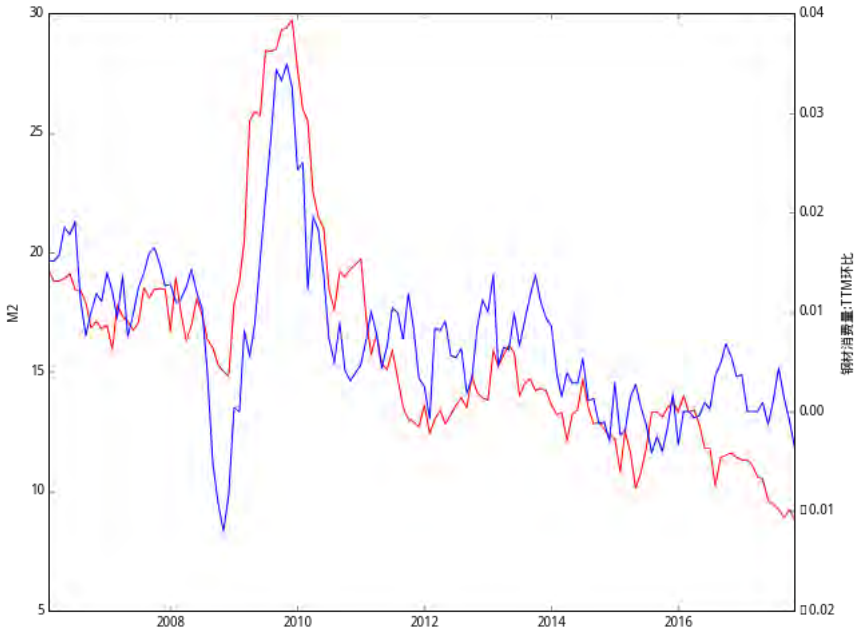
M2与钢材消费量(TTM)¶

In [16]:

```
plot_line_chart_double_yaxis(use_data.index,use_data['M2'],use_data['钢材消费量TTM环比'],'M2','钢材消费量:TTM环比')
chart_gt_gxf = calculate_lag(use_data,'M2','钢材消费量TTM环比')
chart_gt_gxf
```

Out[16]:

	领先期	相关系数	回归系数	P值	R方
0	-5	0.439704	246.609769	7.622284e-08	0.193339
1	-4	0.531815	298.147224	1.920301e-11	0.282828
2	-3	0.606799	340.547427	2.420984e-15	0.368205
3	-2	0.674200	378.166519	6.840686e-20	0.454546
4	-1	0.728889	407.935127	1.238320e-24	0.531279
5	0	0.761632	424.306728	3.678241e-28	0.580083
6	1	0.770310	426.816460	6.021124e-29	0.593378
7	2	0.769847	424.547981	1.069960e-28	0.592665
8	3	0.749096	411.007586	2.849955e-26	0.561145
9	4	0.720866	394.912805	2.087951e-23	0.519648
10	5	0.669588	366.094684	3.704375e-19	0.448349



总结¶

经过前面的一番检验：

1.新开工房面积、房地产投资完成额都是钢材表观消费量的滞后指标

2.固定资产投资完成额、汽车产量与钢材表观消费量基本同步

3.商品房销售面积，基础设施建设投资、M2为钢材表观消费量的领先指标，领先期分别为1~2个月，3~5个月，1~2个月

因此，对于钢铁表观消费量的预测，可以使用商品房销售面积、基础设施建设投资、M2为使用变量

钢材表观消费量TTM环比 预测¶

In [30]:

```
###以前一年数据作为样本进行滚动回归，得到回归系数，以此系数预测下一个月钢材表观消费量TTM环比增速
###商品房销售面积、基础设施建设投资、M2的领先期分别选为2个月、4个月、2个月
testdata = use_data[['钢材消费量TTM环比','房销售面积TTM环比','基建TTM环比','M2']]
```

In [31]:

```
def predict_TTM_growth(testdata,y_var,x_var,x_lag,regre_period = 12):

    x_var_lag = []
    for x,lag in zip(x_var,x_lag):
        testdata[x+'_lag'+str(lag)] = testdata[x].shift(lag)
        x_var_lag.append(x+'_lag'+str(lag))
        use_testdata = testdata[[y_var] + x_var_lag]
        use_testdata.dropna(inplace=True)
        use_testdata[y_var+'_预测'] = np.NaN

    corr_chart = use_testdata[x_var_lag].corr()
    print (corr_chart)
    for i in range(regre_period,len(use_testdata)):

        train_data = use_testdata[i-regre_period:i]
        y = train_data[y_var]
        x = train_data[x_var_lag]
        x = sm.add_constant(x)
        est = sm.OLS(y,x)
        results = est.fit()

        predict_data = use_testdata[i:i+1]
        predict_x = predict_data[x_var_lag]
        predict_x = sm.add_constant(predict_x)


        use_testdata.iloc[i,-1] = results.predict(predict_x)

    return use_testdata
```

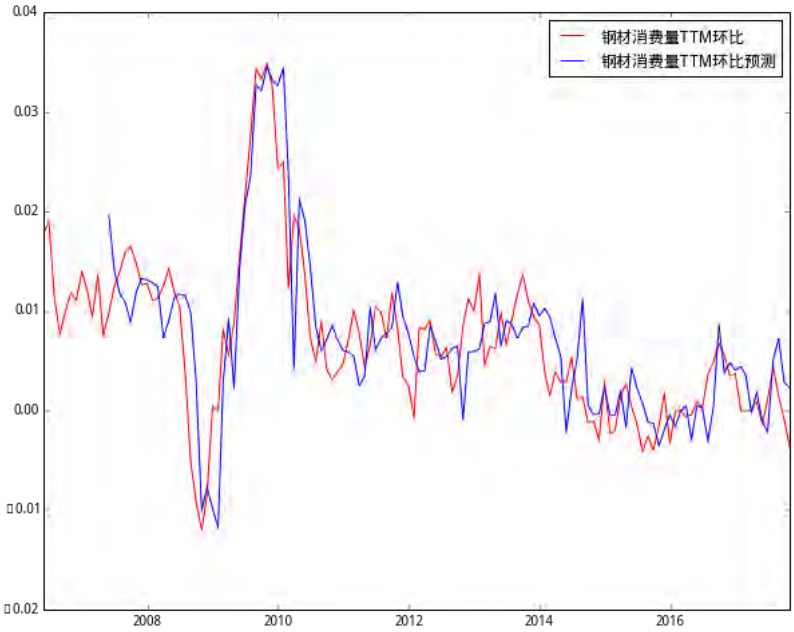
In [32]:

```
predict_data = predict_TTM_growth(testdata,'钢材消费量TTM环比',['房销售面积TTM环比','基建TTM环比','M2'],[2,4,2])
plot_line_chart(predict_data.index,predict_data['钢材消费量TTM环比'],predict_data['钢材消费量TTM环比_预测'],'钢材消费量TTM环比','钢材消费量
```

```
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/conda/lib/python3.4/site-packages/pandas/core/indexing.py:115: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
```

```
                房销售面积TTM环比_lag2  基建TTM环比_lag4   M2_lag2
房销售面积TTM环比_lag2         1.000000      0.198354  0.455816
基建TTM环比_lag4            0.198354      1.000000  0.419496
M2_lag2                0.455816      0.419496  1.000000
```

```
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```



钢铁行业营业收入预测¶

In [63]:

```
steel_price = steel_fd_data[['钢材价格综合指数']]
steel_price.fillna(method='ffill',inplace = True)
steel_price['钢材价格综合指数_lag2'] = steel_price['钢材价格综合指数'].shift(2)
steel_price['钢材价格综合指数_lag14'] = steel_price['钢材价格综合指数'].shift(14)

tdata = use_data[['钢材消费量TTM环比','房销售面积TTM环比','基建TTM环比','M2']]

###以前一年数据作为样本进行滚动回归，得到回归系数，以此系数预测下一个月钢材表观消费量TTM环比增速
testdata = pd.concat([tdata[['房销售面积TTM环比','基建TTM环比','M2']],\
                      steel_price[['钢材价格综合指数']],\
                      clean_data[['钢铁行业销售收入TTM_环比']][:-3]],axis=1)


predict_data_ = predict_TTM_growth(testdata,'钢铁行业销售收入TTM_环比',\
                                   ['房销售面积TTM环比','基建TTM环比','钢材价格综合指数','钢材价格综合指数','M2'],[2,4,2,14,2])
```

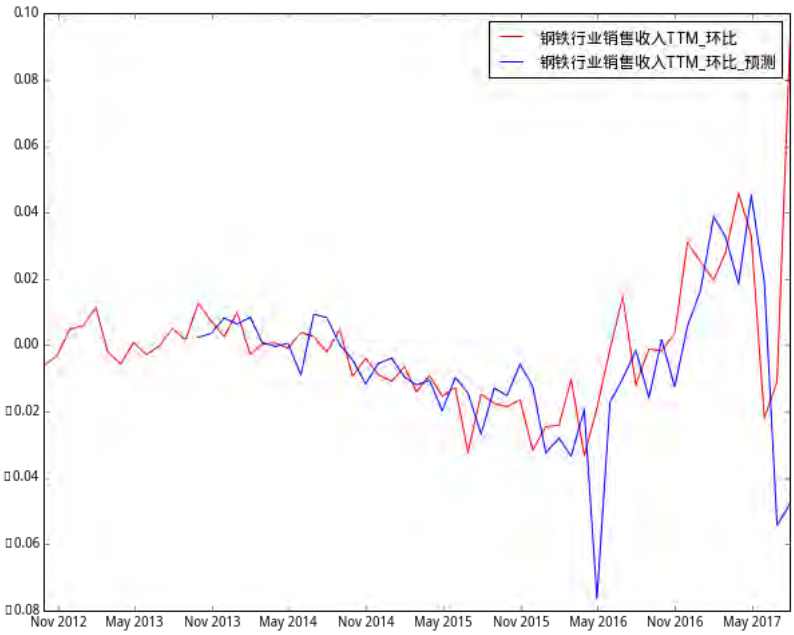
```
plot_line_chart(predict_data_.index,predict_data_['钢铁行业销售收入TTM_环比'],predict_data_['钢铁行业销售收入TTM_环比_预测'],\
                '钢铁行业销售收入TTM_环比','钢铁行业销售收入TTM_环比_预测')
```

```
/opt/conda/lib/python3.4/site-packages/pandas/core/frame.py:2532: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  **kwargs)
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  app.launch_new_instance()
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/conda/lib/python3.4/site-packages/pandas/core/indexing.py:115: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
```

	房销售面积TTM环比_lag2	基建TTM环比_lag4	钢材价格综合指数_lag2	钢材价格综合指数_lag14	\
房销售面积TTM环比_lag2	1.000000	-0.082437	-0.004757	-0.185949	
基建TTM环比_lag4	-0.082437	1.000000	0.029752	0.170184	
钢材价格综合指数_lag2	-0.004757	0.029752	1.000000	0.514030	
钢材价格综合指数_lag14	-0.185949	0.170184	0.514030	1.000000	
M2_lag2	0.083721	0.225676	0.328398	0.690747	

	M2_lag2
房销售面积TTM环比_lag2	0.083721
基建TTM环比_lag4	0.225676
钢材价格综合指数_lag2	0.328398
钢材价格综合指数_lag14	0.690747
M2_lag2	1.000000

```
/opt/conda/lib/python3.4/site-packages/ipykernel/__main__.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```



【量化课堂】效用模型

“钱是钱么？”“额... 应该是吧。”
我们经常以货币来衡量一些物品或者服务的价值。但是仔细想想，货币的价值又该怎么衡量？
“一百万是钱吗？”“额... 好像是吧。”
事实上，金钱的价值是因人而异的，也不是可以简单地说清楚的。
“一百万对巴菲特来说是钱吗？”“额... ”
为什么对于普通人来说数目可观的一百万资金，到了巴菲特手里就变得轻如鸿毛？这就涉及到了经济学中效用的概念。本文将介绍效用的模型，解释我们对价值的衡量标准，并在该模型中分析我们面对不确定性时的决策。

本文由JoinQuant量化课堂推出，难度为进阶上，理解程度为 level-1。
阅读本文需要掌握随机变量（level-0）的知识。

作者：肖睿
编辑：宏观经济算命师

本文是一系列文章中的第一篇。本系列从基础概念入手，推导出 CAPM 模型。系列中共有四篇：

1. 效用模型
2. 风险模型
3. MPT 模型
4. CAPM 模型

效用函数

为了简化问题，我们假设每一个人都有一个*效用函数*(utility function) u，它的输入是一个财富总额，输出是这么多的财富可以给这个人带来多少效用。效用，简单来讲就是指金钱和物质给一个人带来的在生活中的满足感和便利性。每个人的效用函数是不一样的，有些人资产不多但很满足，而有些人却觉得有多少钱都不够。抛开个体的差异，一般来讲，大多数人的效用函数都满足两个性质。

第一个性质是(P1)：如果 $x \leq y$，那么 $u(x) \leq u(y)$。用人话来说，就是钱多总比钱少好。

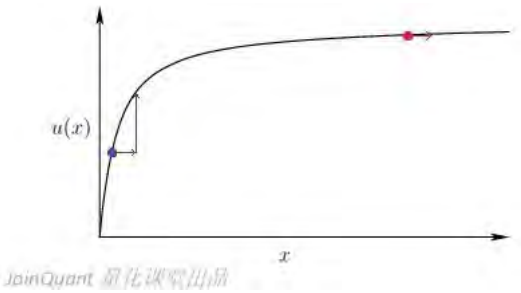
第二个性质是(P2)：如果 $d \geq 0$，并且 $x \leq y$，那么

$$u(x+d) - u(x) \geq u(y+d) - u(y).$$

用人话举个例子，假设 d 是一万元，x 是二十万，y 是二十亿，这就是说当我们的总资产只有二十万的时候，一万元钱提供的效用比我们有二十亿时要大。这也是符合逻辑的，因为当我们只有二十万时，一万的额外资金可以用来做很多事情，但当我们有二十亿的时候，一万元已经无足轻重。

如果一个效用函数满足以上两个性质，我们说它是“正常”的，因为大多数人都要会符合这些性质。并且，在绝大多数经济学理论中，我们都会假设投资者的效用函数是“正常”的。

如果将一个“正常”的效用函数 u 画成图的话，这两个性质在图上表现出来的就是：一，效用曲线是上升的；二，效用曲线的上升越来越慢。如下图所示。



可以假设，图中的蓝点就是你的位置，红点的是巴菲特的位置。两个人在 x 轴上向右移动同样的距离，也就是说你和巴菲特的总资产同时增加同样的数额。这时巴菲特的效用值几乎没有变动，但这些财富却带给了你很多的额外效用。这个模型告诉我们一个好消息，就是因为你很穷，所以增加不多的资产就可以获得很多的满足感。但坏消息是，你很穷。

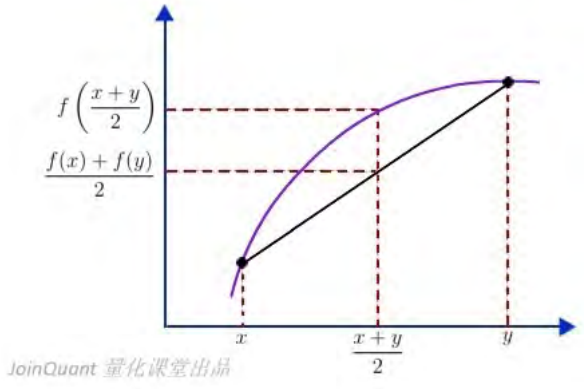
“正常”函数都很“凹”

上一节中的性质(P1)和(P2)有一些等价的性质，同样可以用来描述效用函数。这些额外的描述方式可以方便我们做运算或者在效用函数的模型上做一些证明。

定义：如果一个函数 $f: \mathbb{R} \to \mathbb{R}$ 满足以下条件：对于任何 $x, y \in \mathbb{R}$，有

$$f\left(\frac{x+y}{2}\right) \geq \frac{f(x)+f(y)}{2}.$$

那么我们说 f 是一个*凹函数*(concave function)。



上图是凹函数的示例图，函数在平均点 $(x+y)/2$ 上的取值大于函数在两点上取值的平均 $(f(x)+f(y))/2$。

性质(P2)实际上和凹函数是等同的概念。证明所有(P2)函数都是凹函数比较简单；反之，所有凹函数也都是(P2)函数，但证明起来需要实分析的理论。这里我们给出证明。

命题：所有满足(P2)的函数都是凹函数。
证明：假设 $f:\mathbb{R}\to\mathbb{R}$ 满足(P2)性质，也就是，如果 $d\geq 0$ 和 $a\leq b$，有

$$f(a+d)-f(a)\geq f(b+d)-f(b).$$

那么，给定 $x,y\in\mathbb{R}$，不妨假设 $y\geq x$。定义 $z=\frac{y-x}{2}$，读者可以自行验证 $x+z=\frac{x+y}{2}$。根据(P2)，

$$\begin{aligned}
&< br > f(x+z)-f(x) amp;\geq f(x+z+z)-f(x+z)\\
< br > \quad &< br > f\left(\frac{x+y}{2}\right)-f(x) amp;\geq f(y)-f\left(\frac{x+y}{2}\right) \quad < br >\\
&< br > f\left(\frac{x+y}{2}\right) amp;\geq \frac{f(x)+f(y)}{2} < br >
\end{aligned}$$

由此得知 f 是凹的。Undefined control sequence \blacksquare

此外，了解微积分的读者可能已经发现，如果 u 是二次可导的函数，那么(P1)和(P2)的性质也意味着 Misplaced & 并且 Misplaced &。

预期效用假说

预期效用假说(expected utility hypothesis)假设的是：如果一个投资者的效用函数是 u，面对 n 种选项，并且这些选项的财富值结果可以用随机变量 $X_1,X_2,\ldots,X_n$ 表示，那么该投资者会选择 $E[u(X)]$ 最大的那个选项。

投资者无时不刻不在面临很多选项。有时这些选项可能结果是固定的，比如将钱存入银行；有时可能是不固定的，比如买入风险资产。预期效用假说的意思是，不论投资者的选项是什么，他必定会选择将自己的预期效用最大化的那个选项。

如果抉择的结果全部是固定的，问题并不复杂，我们可以直接选择货币价值最高的那个。但如果结果是不确定的，那么相互比较就比较困难；在这个假说之下，我们可以用效用模型来分析对于不确定性的抉择。

举一个例子。假设一个投资者的效用函数是 $u(x)=\sqrt{x}$（读者可以自行证明这个效用函数是"正常"的），并且他现在有 $x_0=25000$ 的资产。假设他现在有两种选择。一种是什么都不做；另一种是投资于某一项目，该项目有一半的几率成功并且给投资者 15000 的收益，但也有一半的几率会失败并且造成 15000 元的损失。将什么都不做的选项用随机变量 X_0 表示，即为

$$X_0:\ (0,1),$$

即有 $1=100\%$ 的几率让资产变动 0 元。用 X_1 表示投资的选项，则有

$$X_1:\ (15000,1/2),(-15000,1/2).$$

两个选项的预期收益都是 $E[X_0]=E[X_1]=0$。再计算预期效用，得到

$$E[u(x_0+X_0)]=E[u(x_0)]=E[\sqrt{25000}]\approx 158.11,$$

以及

$$\begin{aligned}
&< br > E[u(x_0+X_1)] amp;=\frac{1}{2}\cdot u(x_0+15000)+\frac{1}{2}\cdot u(x_0-15000)\\
< br > \quad &< br > amp;=\frac{1}{2}\cdot\sqrt{40000}+\frac{1}{2}\cdot\sqrt{10000} \quad < br >\\
&< br > amp;=\frac{1}{2}\cdot 200+\frac{1}{2}\cdot 100\\
&< br > amp;=150 \quad lt;158.11. < br >
\end{aligned}$$

得知进行投资的预期效用小于什么都不做的效用，因此不应该选择投资。

结语

效用模型解释了我们对价值的衡量，以及面对不确定性的决策。在下一篇中，我们将解释这个效用模型对于风险的意义，从数学的角度理解为什么“鸡蛋不要放在一个篮子里”。

文章更迭记录：
v1.1，2016-07-26，修正公式
v1.0，2016-07-12，文章上线

【量化课堂】风险模型

导语：在经济学中，风险，就是未来的不确定性，是未知的收益或损失。本篇文章将通过效用模型分析我们对风险的衡量，解释为什么我们要把风险分散化。

本文由JoinQuant量化课堂推出，难度为进阶上，理解程度为 level-1。
阅读本文章需要掌握效用模型（level-0）的知识。

作者：肖睿
编辑：宏观经济算命师

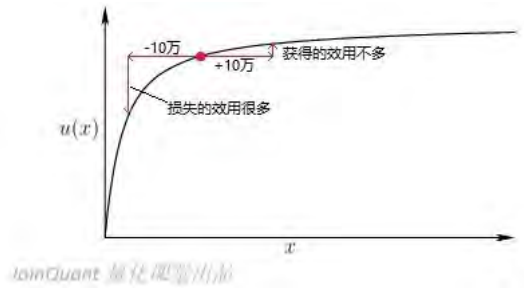
本文是一系列文章中的第二篇。本系列从基础概念入手，推导出 CAPM 模型。系列中共有四篇：

1. 效用模型
2. 风险模型
3. MPT 模型
4. CAPM 模型

风险和收益

我们经常说到“高风险高回报”或者“鸡蛋不应该放在同一个篮子里”的概念。但是这些概念的逻辑究竟是什么呢？这其实可以用效用模型来解释。

我们先假设一个投资项目，这个项目有一半的概率失败，一半概率成功。如果失败的话，投资者损失十万元，成功的话投资者获利十万元。这是不是所谓的“高风险高回报”呢？不是。这是高风险无回报，并且如果一个理智投资者的效用函数符合上一篇的“正常”性质，那么他不会投资这个项目。我们用效用函数来看看为什么。



由于效用函数的增速是递减的，这个投资者在损失十万元时会损失很多效用，但在获利十万元时获得的效用却不多。因此，虽然在账面上这个投资的回报一半几率亏损十万，一半几率获得十万，在概率上是盈亏持平；但在效用的层面，投资者有一半几率损失很多效用，一半几率获得一点效用，平均下来是效用亏损的。因此，如果一个投资者注重的是效用而不是财富的数值，那么他应该回避这项投资。

风险厌恶

通常来讲，假设一项投资 A 的回报可以用随机变量 X 表示，这项投资的回报预期是 $E[X]$。再假设一个投资者具备效用函数 u，并且现有财富是 x_0。那么，该投资者投资于 A 后的财富值可以用随机变量 x_0+X 表示，并且他进行该投资的效用是 $u(x_0+X)$，这项投资带给他的额外效用是 $u(x_0+X)-u(x_0)$。因此，投资于 A 带给投资者的预期效用收益是 $E[u(x_0+X)-u(x_0)]$。如果这个值是负的，说明投资者在预期上是要损失效用的；如果是正的，说明投资者在预期上是得到效用的。所以，“高风险高回报”指的不是最高可能产生的收益 $\max(X)$ 高，而是预期收益 $E[X]$ 高。

定义：如果一个随机变量 R 满足 $E[R]=0$，我们说 R 是一个*零收益投资*。

命题：设 u 为一个“正常”的效用函数，设 $x_0\geq 0$，并且设 R 为一个零收益投资。那么 $E[u(x_0+R)]\leq u(x_0)$。

也就是说，对于一项没有预期收益的投资，一个效用函数“正常”的投资者在预期上是损失效用的，原因正如上一节所讲。因此，我们也可以从风险的角度来理解“正常”，有了以下定义。

定义：如果一个投资者的效用函数 u 是“正常”的，也就是说满足上一篇中(P1)和(P2)的性质，我们说他是*风险厌恶的*(risk averse)。

投资的风险就是它的收益的不确定性。任何投资 X 都可以被写为预期收益 $E[X]$ 和零收益投资 $X-E[X]$ 两部分的加和。其中零收益部分带来预期效用下降，所以需要足够大的 $E[X]$ 来弥补；这里，$E[X]$ 被定义为这项投资的*风险溢价*(risk premium)，只有当风险溢价高于风险所带来的效用折损时，投资者才愿意进行投资。下面举一个例子。

为了保持计算的简便性，我们设定一个简单的风险厌恶的效用函数 $u(x)=\sqrt{x}$。假设一投资的随机变量 X 的收益和概率分配如下

$$X:\ (-10000,2/5),(10000,2/5),(20000,1/5).$$

那么 X 的预期收益是

$$E[X] = \frac{2}{5} \cdot (-10000) + \frac{2}{5} \cdot 10000 + \frac{1}{5} \cdot 20000 = 4000.$$

如果投资者的现有资产是 $x_1 = 10000$，我们看他是否应该进行投资。计算

$$\begin{aligned} < br > E[u(x_1 + X)]amp;= & \frac{2}{5} \cdot u(10000 - 10000) \\ < br >amp; & + \frac{2}{5} \cdot u(10000 + 10000) \\ < br >amp; & + \frac{1}{5} \cdot u(10000 + 20000) \\ < br >amp;= & \frac{2}{5} \cdot \sqrt{0} + \frac{2}{5} \cdot \sqrt{20000} + \frac{1}{5} \cdot \sqrt{30000} \\ < br >amp;\approx & 0 + 56.57 + 34.64 \\ < br >amp;= & 96.21. < br > \end{aligned}$$

这个值小于起始的效用值 $u(10000) = 100$，因此在预期上是损失效用的，不应投资。

再假设投资者的总资产变成了 $x_2 = 20000$，再做一次计算

$$\begin{aligned} < br > E[u(x_2 + X)]amp;= & \frac{2}{5} \cdot u(20000 - 10000) \\ < br >amp; & + \frac{2}{5} \cdot u(20000 + 10000) \\ < br >amp; & + \frac{1}{5} \cdot u(20000 + 20000) \\ < br >amp;= & \frac{2}{5} \cdot \sqrt{10000} + \frac{2}{5} \cdot \sqrt{30000} + \frac{1}{5} \cdot \sqrt{40000} \\ < br >amp;\approx & 40.00 + 69.28 + 40.00 \\ < br >amp;= & 149.28. < br > \end{aligned}$$

这次的计算结果大于 $u(x_2) \approx 141.42$。我们看出，由于总资产增加了，因此投资者可以承受更大的风险，之前预期效用为负的投资变成了预期效用为正，因此可以进行投资。

分散风险

如导语中所述，经济学中的“风险”指的是未来的不确定性；而从概率学的角度来说，一个随机变量的分布越散开，它的确定性就越低。因此，有一个简易的衡量风险的标准，就是收益变量的标准差 σ_X。

一般来讲，在保持 $E[X]$ 不变的情况下，我们希望 σ_X 越低越好。下面就讲一个例子。

继续沿用效用函数 $u(x) = \sqrt{x}$，并且有总资产 $x_0 = 10000$。假设我们的全部资产一万元在河对岸，并且有必要把它们全部运过来。运输本身没有成本，但是在这条河上行驶的船只有 50% 的概率会沉船，两只船的沉与不沉是完全独立的；如果沉船的话会损失该船上所有的资产。我们要决定一个配置方案：将资产平均分配到多少条船上进行运输。

先来计算一条船的情况。我们有一半概率损失所有财产，一半概率保留所有财产。因此运输结果可以用 $X < em > 1$ 表示：

$$X_1 : \ (10000, 1/2), (0, 1/2).$$

有财富预期 $E[X_1] = 5000$，标准差是

$$\sigma < /em > X_1 = \sqrt{\frac{1}{2} \cdot (10000 - 5000)^2 + \frac{1}{2} \cdot (0 - 5000)^2} = 5000,$$

且效用预期是

$$E[u(X_1)] = \frac{1}{2} \cdot \sqrt{10000} + \frac{1}{2} \cdot \sqrt{0} = 50.$$

再者将资产平分于两条船。两条都沉船的概率是 $1/4$，两条都不沉的概率是 $1/4$，只有一条沉的概率是 $1/2$。因此可以用随机变量 $X < em > 2$ 表示：

$$X_2 : \ (10000, 1/4), (5000, 1/2), (0, 1/4),$$

财富预期没有变，$E[X_2] = 5000$，标准差是

$$\sigma < /em > X_2 = \sqrt{\frac{1}{4} \cdot (10000 - 5000)^2 + \frac{1}{2} \cdot (5000 - 5000)^2 + \frac{1}{4} \cdot (0 - 5000)^2} \approx 3535.53.$$

计算与其效用，有

$$\begin{aligned} < br > E[u(X_2)]amp;= & \frac{1}{4} \cdot \sqrt{10000} + \frac{1}{2} \cdot \sqrt{5000} + \frac{1}{2} \cdot \sqrt{0} \\ < br >amp;\approx & \frac{1}{4} \cdot 100.00 + \frac{1}{2} \cdot 70.71 \\ < br >amp;= & 25.00 + 35.35 \\ < br >amp;= & 60.35, < br > \end{aligned}$$

大于 X_1 的预期效用 50。

对于三条船的情况，读者可以自行计算其相对应的随机变量 $X < em > 3$为

$$X_3：(10000, 1/8), (6666.66, 3/8), (3333.33, 3/8), (0, 1/8).$$

同样，$E[X_3] = 5000$，并且计算可得知

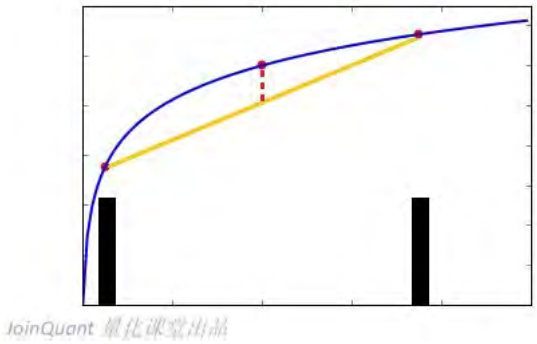
Misplaced &

还有

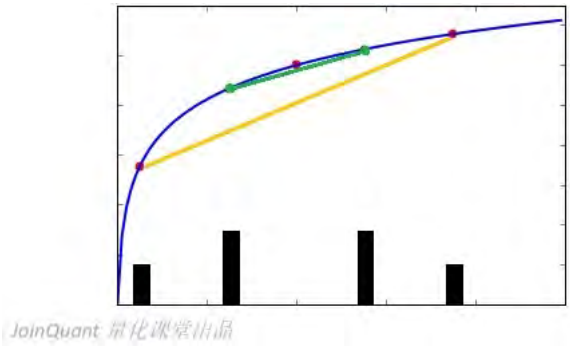
Misplaced &

我们说“将风险分散”了，那么在概率学的层面发生了什么？观察随机变量的分布，发现从 X_1, X_2, X_3，均值保持在 5000 不变，但是概率分布在向均值靠拢。因此，不确定性降低了，或者说，标准差 σ 降低了。

我们画一个简单的图来直观地解释一下为什么预期效用上升了。假设下图中的蓝线是效用函数，两侧的两个红点是投资可能对财造成的变化，下方的黑条是每种可能性的概率分布。



由于风险厌恶的效用函数是凹函数，所以效用函数在两侧的取值的平均要小于它在中间点的取值。



把概率分布向中心聚拢的话，得到离中心更近的两个点，标为绿色。并且，由于效用函数是凹的，它在两个绿点的平均取值要高于外侧两个红点的取值的平均。由于外侧的概率被向绿点位置分配，所以预期的效用较之前的情况是提升了。

在上面的例子中，如果继续计算 $X < em > 4, X_5, \ldots$的话，会发现当 n 越大时， $\sigma < /em > X_n$ 就越接近 0， 并且 $E[u(X_n)]$ 越接近 $u(E[X_n]) = \sqrt{5000} = 70.71$。

对投资者的假设

定义：对于一个投资者，如果任意两个投资回报率的随机变量 X 和 Y 满足 $E[X] \geq E[Y]$ 并且 Misplaced & （也就是说预期收益更大但是风险更小），该投资者会选择X，那么我们说这个投资者是“理智”的。

这个定义的意义在于简化我们对风险的分析。因为效用函数和收益的随机变量有时可能是非常复杂的，但很多情况下，标准差可以合理地衡量风险程度。在金融经济学中著名的现代资产配置理论(MPT)和资本资产定价模型(CAPM)里，一个重要的假设就是，所有投资者都是“理智”的。

结语

通过对效用模型中风险的分析，我们认为投资者应该尽量将预期收益最大化并且将投资波动的标准差最小化。基于这个认知，我们可以建立一个将预期效用最大化的资产配置模型，就是下一篇要介绍的 MPT 模型。

文章更迭记录：
v1.1，2016-07-27，修正公式
v1.0，2016-07-12，文章上线

【量化课堂】股票手续费，一文全学会！！！

导语：在股票交易中，手续费往往是最容易被大家忽视的一笔糊涂账，有时候很多细节甚至连券商的专业人士都一头雾水，本文就针对此问题帮助大家一一梳理，讲讲这其中的来龙去脉。

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。

作者：$BaSO_4$
编辑：宏观经济算命师

我们曾经做过一个小小的调查，请周围的一些散户股民朋友帮忙解答股票交易手续费方面的问题，很多人最开始还能对答如流，然而随着探讨的深入，不出意料地，我们先后得到了以下三个表情：

最初（自信）——

后来（蒙圈）——

最终（试图自尽）——

说到这儿，您大概其也就明白什么意思了。对此，很多人表示不能理解，一脸囧相地问道：“手续费不就那点事儿嘛，有那么难？”您别说，还真有！不信您就去问问度娘，查查教科书，再搜搜官方网站，我保证，要想找出一套完整、权威、最新的解释和数字，您可真得花点时间、费点功夫！

所以，您也知道我们要干嘛了！没错，在这个犬儒主义盛行，人人都不求甚解的大时代中，我们就是要为大家雪中送炭排忧解难！就是要做追求卓越一丝不苟的处女座！这才是聚宽一直追求的工匠精神！（自作多情-_-!）

闲话少说，先来热身。为了有的放矢地阐述问题，我们精心编制了一道计算题作为头盘开胃品——

假设某投资者在上海证券交易所以每股10元的价格买入某A股股票10000股，那么该投资者最低需要以约（ ）元全部卖出该股票才能保本（四舍五入保留2位小数）。若按此价格卖出

A.10.01，164.16　　B.10.02，164.26　　C.10.03，164.36　　D.10.04，164.46

之后呈上汤品，普及几个常见名词——

股票交易手续费：投资者在买卖股票时需要支付的各种费用的总和，无论交易盈亏都要支付。

佣金：买房时中介收取佣金，炒股同样也是一个道理。佣金是投资者在买卖股票时按成交金额的一定比例支付的费用，是证券公司（券商）为投资者提供证券买卖服务收取的费用（包括代收部分）。

过户费：股票成交后，买卖双方为变更证券登记所支付的费用，通俗点说就是股票从一人名下过户到另一人名下的费用。属于中国结算公司的收入，由券商在和投资者清算交割时代为扣收。

结算费：A股的过户费，在B股中叫做结算费，同样也属于中国结算公司的收入。

印花税：根据国家税法规定，在A股和B股成交后对买卖双方投资者按照规定的税率分别征收的税金。（这个税金是交给国家的，类似于赌场的"抽水"，你懂的）

证券公司经纪佣金：券商为投资者提供经纪业务服务时收取的所有佣金中，属于自己的那部分，是券商在这项业务中的主要收入来源。

证券交易所经手费：属于佣金的一部分，券商代收。券商在证券交易所的场内交易成交后，按成交金额的一定比例向交易所交纳的费用。经手费和佣金性质相似（羊毛出在羊身上，下面会做介绍）。

证券交易监管费：也属于佣金的一部分，券商代收。顾名思义嘛，交给证券监管方（证监会）的嘛，监管你们大家很累嘛，不收点好处费怎么对得起自己嘛（还是羊毛出在羊身上）。

成交金额：成交股数×成交价格，如1000股×5元=5000元

成交面额：成交股数×该股票成交面额（一般为1元），如1000股×1元=1000元

汤品过后，主菜献上——

通过脑图大家就会明晰，股票手续费三巨头——佣金、过户费、印花税。

方便起见，我们主要以投资者最多的A股来阐述问题。

一、关于佣金

佣金的构成也是投资者最易混乱的，主要是因为很多券商的工作人员为了减少麻烦，刻意避开了繁琐的解释过程。实际上我们常说的这个佣金费用，主要由三部分组成，由券商统一收取：

1.券商的经纪佣金（属于券商自己的那部分佣金收入）

2.证券交易所经手费（券商交给交易所的手续费，0.0487‰，双向收取）

3.证券交易监管费（证监会收取的费用，0.02‰，双向收取）

1+2+3=佣金，其中2+3=交易规费=0.0487‰+0.02‰=0.0687‰

注：交易规费不能单独收取，否则违反证监会规定。

另外，关于佣金，我们还需注意以下几点：

1.收取标准如何理解：

双向收取，单边费用最高为成交金额的3‰，但不得低于交易规费，起点5元。

交易一只股票——

买入：成交价5元，买入数量100股，成交金额就是5×100=500元。
佣金就是这个成交金额500×3‰=1.5元，不足5元，会按照5元收取。

卖出：计算方法同买入，若不足5元，会按照5元收取。

So，一笔完整的交易，佣金费用=买入佣金+卖出佣金=5元+卖出佣金>5元

2.佣金比例可否降低：
当然可以！佣金比例是有上下限额的，可以浮动的，最高3‰，最低不低于交易规费。那么我们粗略核算一下，考虑交易规费和经营成本，主流券商要想盈利，佣金比例至少要在万分之二以上。而现实也证明了这一点，在异常激烈的竞争环境中，价格斗争的确成为了券商们的常规武器，而大幅降佣也成了大家的无奈之举。早年间，资金量大、交易频繁的客户可自己去和证券部申请佣金优惠，而现如今，业务人员一张嘴便是万三万四或万五，更有甚者万二的比例都不在话下（国金证券的“佣金宝”就曾经遭到了各大券商的围剿以及“川证协”的发难），足见市场厮杀之惨烈（股市持续低迷是罪魁祸首）。当然，各地在佣金方面也略有区别，不能一概而论，总的来说，券商较多的地方，竞争激烈的城市，佣金相对低廉，反之亦然。

但是，价格战一旦祭出“节操碎满地、下限负无穷”的大招，就会出现“杀敌一千自损八百”的悲惨局面，比如2011年之前的很长一段时间，市场上就出现过类似于“零佣金”的畸形竞争，让人一声叹息。当这种情况上升到了“红色”级别，监管部门就不能袖手旁观了。2011年开始，各地监管部门陆续开始出台“佣金限价”政策，打击恶性竞争，保护市场环境。当然，限价的这个“价”也不是随意乱出的，而是由监管部门根据实际情况区别对待的。相关文件大家可以参考《关于进一步加强证券公司客户服务和证券交易佣金管理工作的通知》（中证协发[2010]157号文）

关于进一步加强证券公司客户服务和证券交易佣金管理工作的通知

中证协发[2010]157号

颁布时间：2010-09-30 发文单位：中国证券业协会

各证券公司：

随着多层次证券市场的形成，投资产品的丰富，营销方式的改变，证券公司客户数量迅速增长，客户结构、客户需求日趋复杂和多样，处于同质化竞争阶段的证券经纪业务客户服务和证券交易佣金管理面临严峻的挑战。为推动证券公司增强合规经营意识，提升客户服务水平，保护投资者合法权益，促进证券经纪业务健康、持续发展，现就加强证券公司客户服务和证券交易佣金管理工作的有关问题通知如下：

一、充分认识客户服务工作的重要性和紧迫性

证券经纪业务的本质是为客户提供中介服务，满足客户在投资过程中对金融服务的需求。近年来，在新的市场条件下，证券经纪业务服务模式已从单一化、粗放化向专业化、精细化转变，部分证券公司在客户分类、客户适当性管理、服务团队建设、服务产品化以及信息技术系统支持等方面进行了积极尝试，取得了较好的效果，赢得了客户的信任，初步形成了证券经纪业务品牌效应。但也有一些证券公司未能深刻认识到客户服务对证券经纪业务发展的重要性，长期以来对客户服务工作投入不足，造成了客户服务人员素质不高、专业服务能力不强、服务质量不高、投资咨询产品相对匮乏、信息技术系统配置落后等问题，无法为客户提供丰富的投资产品和专业化服务。一些证券公司采取了以低于成本的证券交易佣金水平、“零佣金”等方式招揽客户，又在客户不知情的情况下随意调高证券交易佣金收取标准，向客户承诺证券买卖收益或赔偿证券买卖损失，贬低同行、进入同行营业场所劝导客户等违法违规和不正当竞争手段，侵害了投资者合法权益，扰乱了市场竞争秩序。

三、加强证券交易佣金的管理，形成科学合理的服务定价机制

证券公司应公平对待所有客户，在对客户进行分类的基础上，按照“同类客户同等收费”、“同等服务同等收费”的原则，制定证券交易佣金标准。证券公司应明确告知客户不同服务内容、服务标准所对应的证券交易佣金标准。证券公司可根据客户交易方式、交易量、资产规模、忠诚度、地域等因素，在服务成本的基础上，确定差别化的分类证券交易佣金收取标准。证券营业部在证券交易佣金管理中还应遵守住所地证券业协会的有关自律规则。

证券公司向客户收取的证券交易佣金应当以证券经纪业务服务成本为基础。证券经纪业务服务成本应采取全成本核算的方式，包括代扣代缴费用、证券营业部运营成本、公司总部和分公司直接为证券经纪业务发生的支出、公司管理与服务部门发生的应由证券经纪业务分摊的支出等。其中，代扣代缴费用包含但不限于：代收的监管、手续费和过户费、营业税金和附加等；证券营业部运营成本包含但不限于：员工工资与福利、租赁费、折旧费、办公费用、业务开发费用和客户维护费用等；公司总部和分公司直接为证券经纪业务发生的支出包含但不限于：投资者保护基金支出、第三方存管费支出、席位及交易单元使用费、直接管理经纪业务部门的支出等；公司管理与服务部门发生的应由证券经纪业务分摊的支出，是指研发、信息技术、财务、清算以及综合管理等部门为证券经纪业务所发生的支出。

二、关于过户费

过户费，顾名思义，就是过户时需要的费用，这个很好理解——当你买入股票后，你就是这家公司的一个股东，这个股东身份由卖出方过户到你的名下，由证券交易所中央结算登记中心办理登记，这个过程中就产生了过户费。由于我国两家交易所不同的运作方式，上海证券交易所采取的是“中央登记、统一托管”，所以此费用只在投资者进行上海股票的交易中才进行支付（双向收取），深圳股票交易时无此费用（已包含在经手费中，不单独收取）。在2015年8月1日之前，沪市是按成交的股数来收取的，每1000股收取0.6元，最低0.6元（低于1000股也收取0.6元），超过1000股的按每100股收取0.06元，即成交面额的0.6‰费率。从2015年8月1日开始，过户费按成交金额的0.02‰来收取。

那新旧政策有什么差别呢，我们假设以下四个情景：

1.投资者以50元的价格买入10000股某A股股票
2015年8月1日之前他需要付出：10000×0.6‰=6元
2015年8月1日开始他需要付出：50×10000×0.02‰=10元

2.投资者以50元的价格买入2000股某A股股票
2015年8月1日之前他需要付出：2000×0.6‰=1.2元
2015年8月1日开始他需要付出：50×2000×0.02‰=2元

3.投资者以5元的价格买入2000股某A股股票
2015年8月1日之前他需要付出：2000×0.6‰=1.2元
2015年8月1日开始他需要付出：5×2000×0.02‰=0.2元

4.投资者以5元的价格买入10000股某A股股票
2015年8月1日之前他需要付出：10000×0.6‰=6元
2015年8月1日开始他需要付出：5×10000×0.02‰=1元

由此判断，新的过户费政策（现行政策）对于低价股的优势显而易见。

三、关于印花税

股票交易印花税是印花税类的一种，从普通印花税发展而来，起源于荷兰，是股票交易时政府向投资者收取的一部分费用。最早的股票交易是纸质凭证，当时缴税时是在凭证上用滚筒推出“印花”戳记以示完税，因此被称为“印花税”。如今的股票交易都是电子版凭证，因此就不见“印花”只见“税”了。

由于税负轻微、税源畅旺、手续简便、成本低廉，所以股票印花税经常被誉为“拔最多的鹅毛，听最少的鹅叫”，深受各国政府喜爱。实际也是如此，中国最早于1990年7月在深圳证券交易所开始征收此税，而不久后的1993年这项税费的收入就达到了22亿元，占全国财政收入的0.51%，而2000年则翻了好几倍，达到了478亿元，占财政收入比重更达到3.57%。就算是税收率极低的今天，按当日两市5000亿的成交金额×1‰（现在是单向收取），政府的日收益也能达到5个亿，相当可观。因此，股票交易印花税可谓是政府增加税收收入的一个超好的手段。

同时，正因为印花税客观上增加了投资者的成本，这也使它自然而然地成为了政府调控股票市场的工具。历史上，我国的印花税税率标准曾经多次调整，几乎每次都令市场指数躁动不安，不信我们立表为证：

调整日期	具体措施	首日表现（沪指）	市场影响
1990年	深交所开证印花税，对买卖双方各征6‰		
1991年	深交所开证印花税，对买卖双方各征3‰		
1991年10月10日	下调：6‰→3‰	上涨1%	深指此前由1000点下跌至400点，印花税下调后1个月指数反弹至700点，2个月内涨至1200点
1997年5月10日	上调：3‰→5‰	上涨2.26%	沪指由1500点下跌至1000点，深指半年内从6000点下跌至4000点，两年内下跌至2500点
1998年6月12日	下调：5‰→4‰	上涨2.65%	沪指在两个月内从1400点下跌至1050点
1999年6月1日	下调：4‰→3‰（B股）	上涨8.55%(B股)	B股此前下跌至历史最低位20点，印花税下调后从40点起步，两年内翻6倍至240点
2001年11月16日	下调：沪深统一下调至2%	上涨1.57%	沪指此前从2200点下跌至1600点，深指此前从5000点下跌至3300点，下调后均继续下跌
2005年1月24日	下调：2‰→1‰	上涨1.73%	下调时沪指1250点，深指3000点，此后沪指最低至998点，深指最低至2590点
2007年5月30日	上调：1‰→3‰	下跌6.5%	上调后一周内股市狂泻，沪指从4300点跌至3400点，深指从13500点跌至10500点
2008年4月24日	下调：3‰→1‰	上涨9.29%	下调后一周内一波反弹，沪指从3000点上涨至3750点，深指从11000点上涨至14000点
2008年9月19日	费率不变，改为单边征收（只对卖方收）	上涨9.45%	下调后一周内再次一波小反弹，沪指从1800点升至2300点，深指从6500点升至7500点

表中可见，短期内，印花税的调整在绝大多数情况下和股市的涨跌成反比关系，但和中长期表现没有特别直接的关联。短期来看，降低了印花税，会降低每次交易的成本，这样会吸引更多投机资金进入股市，同时减少在股市中的资金消耗，理论上为利多措施，反之，则会提高股市交易中的成本，利空市场。

到此为止，我们的讲述就告一段落，最后为您奉上甜品与咖啡——之前那道选择题的答案：

```
解：假设该投资者最低需要以P元全部卖出该股票才能保本。

则卖出收入=P×10000（卖出金额）-P×10000×0.0003(卖出佣金)-P×10000×0.00002（过户费）-P×10000×0.001（印花税）= P×10000×（1-0.0003-0.00002-0.001）=P×9986.

为了保本，必须满足卖出收入≥买入支出，即P×9986.8≥100032，P≥10.0164217≈10.02（四舍五入保留2位小数）

手续费计算：

买入手续费10×10000×0.0003（买入佣金）+10×10000×0.00002（过户费）=32

卖出手续费10.02×10000×0.0003(卖出佣金)+10.02×10000×0.00002（过户费）+10.02×10000×0.001（印花税）=132.264≈132.26（四舍五入保留2位小数）

手续费总计32+132.26=164.26

正确答案：B
```

精明的你，做对了吗？

文章更迭记录：
v1.1，2016-08-10，文章勘误
v1.0，2016-08-02，文章上线

【量化课堂】MPT 模型

导语：资本市场上有诸多风险资产，各有不同的收益率和波动率。为了分散风险，我们一般会同时持有多种不同的资产，但是如何合理地进行配置是个难题。配置不好的话可能不光分散了风险，也对冲没了收益。本文要介绍的，就是著名的现代资产配置（MPT）理论。

作者：肖睿
编辑：宏观经济算命师

阅读本文需要掌握效用和风险模型（level-1）的知识。
本文是一系列文章中的第三篇。本系列从基础概念入手，推导出 CAPM 模型。系列中共有四篇：

1. 效用模型
2. 风险模型
3. MPT 模型
4. CAPM 模型

概述

金融资产配置的目标是将投资资金合理地分配在多种资产上，在将风险控制在一定范围内的同时把收益率最大化。其中最著名的理论是*现代资产配置理论*(Modern Portfoilio Theory)，简称 MPT，由 Markowitz 在1952年提出。MPT 的核心思想是以最小化标准差（或同理的，方差）并最大化预期收益为目标来进行资产配置，有时也称为*均值-方差分析*(Mean-Variance Analysis)，是金融经济学的一个重要基础理论。

模型和假设

在 MPT 模型中，我们假设投资者对一个资产的风险的认知等同于该资产的收益率变量的标准差（见风险模型）。因此，任何一个资产或者资产组合都可以根据其收益和风险被放在一个二维图坐标图上，该图的横轴是收益率的标准差，竖轴是收益率的预期值。

投资者追求的是风险低并且收益高的资产配置。因此，固定收益率不变，我们要将风险降到最低，也就是要得到最小的横轴值；或者固定风险不变，将收益率提升到最高，也就是要得到最大的竖轴值。

假设市场上有 n 种不同的金融资产（可以狭义地想象为股票） $1, 2, \ldots, n$。对于某一资产 i ，用 r_i 表示该资产的收益率的随机变量，$E[r_i]$ 表示收益率的预期，σ_i 表示 r_i 的标准差。

我们将市场上所有收益率方差大于 0 的资产叫做*风险资产*(risky assts)，将收益率没有不确定性的资产叫做*无风险资产*(risk-free assets)。并且，假设市场上所有无风险资产的收益率是一样的，叫做*无风险利率*(risk-free interest rate)，写作 r_f。

一个*风险资产配置*(risky portfolio) P 是由风险资产 $i = 1, 2, \ldots, n$ 按照某个权重比例组成的，每一个资产 i 在 P 中的权重是 $w < em > i$，满足 $\sum < /em > i = 1^n w_i = 1$。我们假设市场是完全开放的，并且可以无限制地买多或卖空，因此 w_i 可以是任何实数。

根据单个资产的收益率，可以计算资产配置 P 的收益变量的一些性质。首先，资产组合收益率的随机变量是

$$r < em > P = \sum < /em > i = 1^n w < em > i r_i,$$

它的预期收益是

$$E[r_P] = E\left[\sum < /em > i = 1^n w < em > i r_i\right] = \sum < /em > i = 1^n w < em > i E[r_i],$$

方差是

$$\mathrm{Var}(r_P) = E[r_P - E[r_P]] = \sum < /em > i = 1^n \sum_{j=1}^{n} w_i w_j \mathrm{Cov}(r_i, r_j),$$

并且有标准差

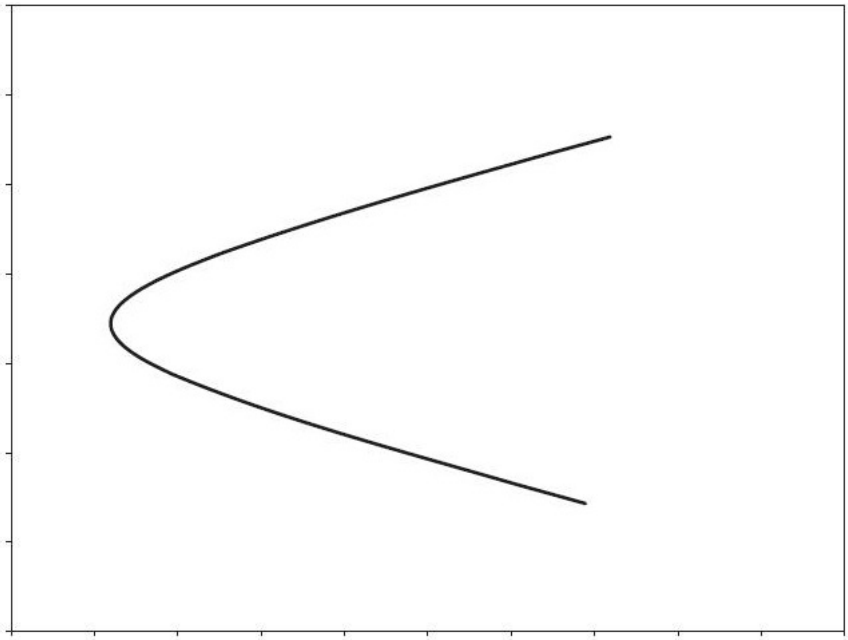
$$\sigma_P = \sqrt{\mathrm{Var}(r_P)}.$$

有效前沿

现在，我们固定预期收益，然后拥有该预期收益，并且标准差最小的资产组合。也就是说，对于任意一个预期收益值 μ，找到一个由配置权重 $w = (w < em > 1, w_2, \ldots, w_n)$定义的资产配置 P，要求 P 的预期收益率为 μ，并且，在所有可以配置出的预期收益为 μ 的组合中，P 的方差是最小的。用最优化问题表示出来的话，就是

Undefined control sequence \<

这个问题的最佳解用 Lagrange 乘子的方法可以找出，但解决过程比较复杂，这里就不多做解释。对于每一个值 μ，我们求得一个风险资产配置 P，满足 $E[r_P] = \mu$，并且 σ_P 是最小的。将这些最优解画成图，在标准差-预期的坐标上得到一条抛物线。根据计算所用到的资产的信息不同，这根曲线会不尽相同，但基本上遵循这个形状。



这条曲线被叫做*有效前沿*(efficient frontier)。由于它的形状像一枚子弹尖，所以有时也叫做*马科维兹子弹*(Markowitz bullet)。有效前沿存在一个波动率最小的位置，也就是图中曲线最靠左的地方，并且在这个点以上的位置才是真正"有效"的；我们是固定预期收益算得的最低风险而得到的这条曲线，如果再固定风险并选择最大的预期收益，则会筛选掉有效前沿的下半部分。所以，很多时候人们所说的"有效前沿"会特指上半部分。

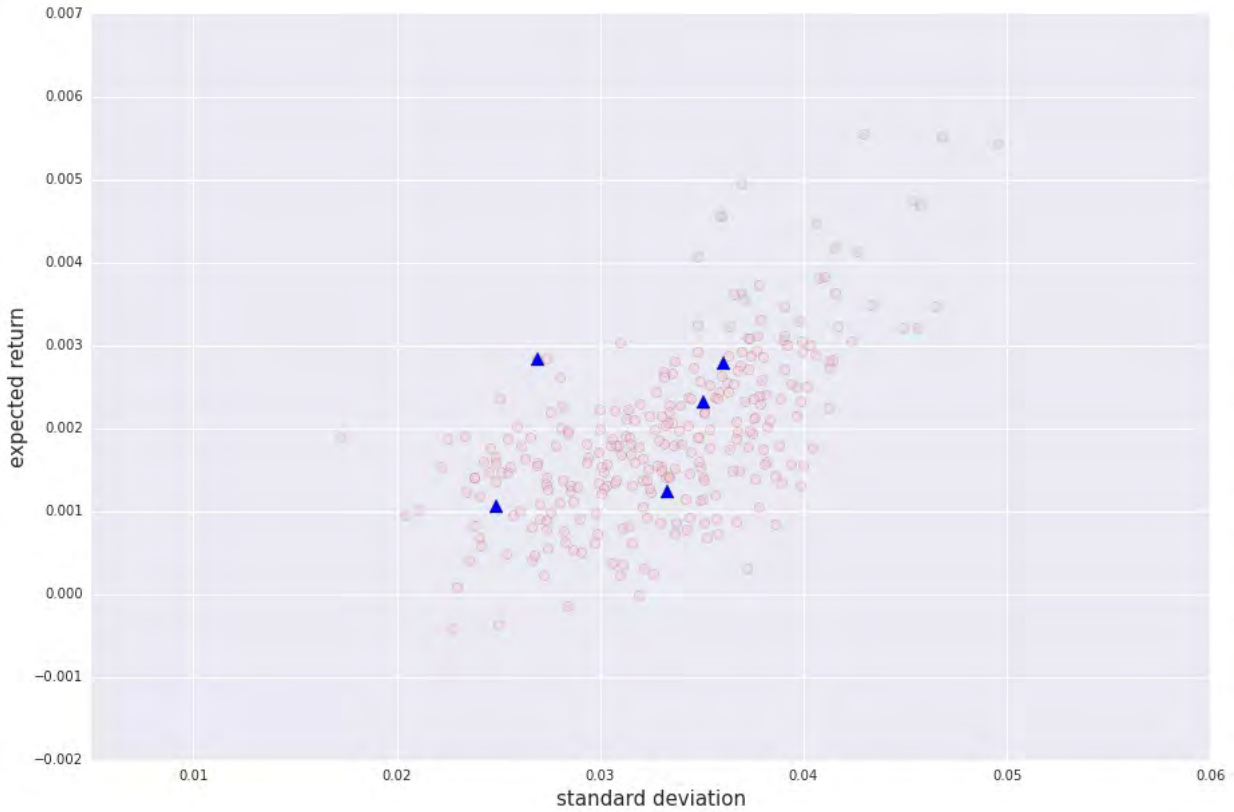
我们举个例子。风险资产采用从沪深三百股票池中随机选出的五支股票，取 2014 年到 2016 年的所有收盘价数据，经计算得出它们的平均日均收益率分别是

$$\mathbf{r} = [0.00106, 0.00283, 0.002323, 0.00279, 0.00125],$$

并且收益率的标准差是

$$\sigma = [0.0249, 0.0270, 0.0351, 0.0361, 0.0333].$$

画在协方差-预期轴上，如下

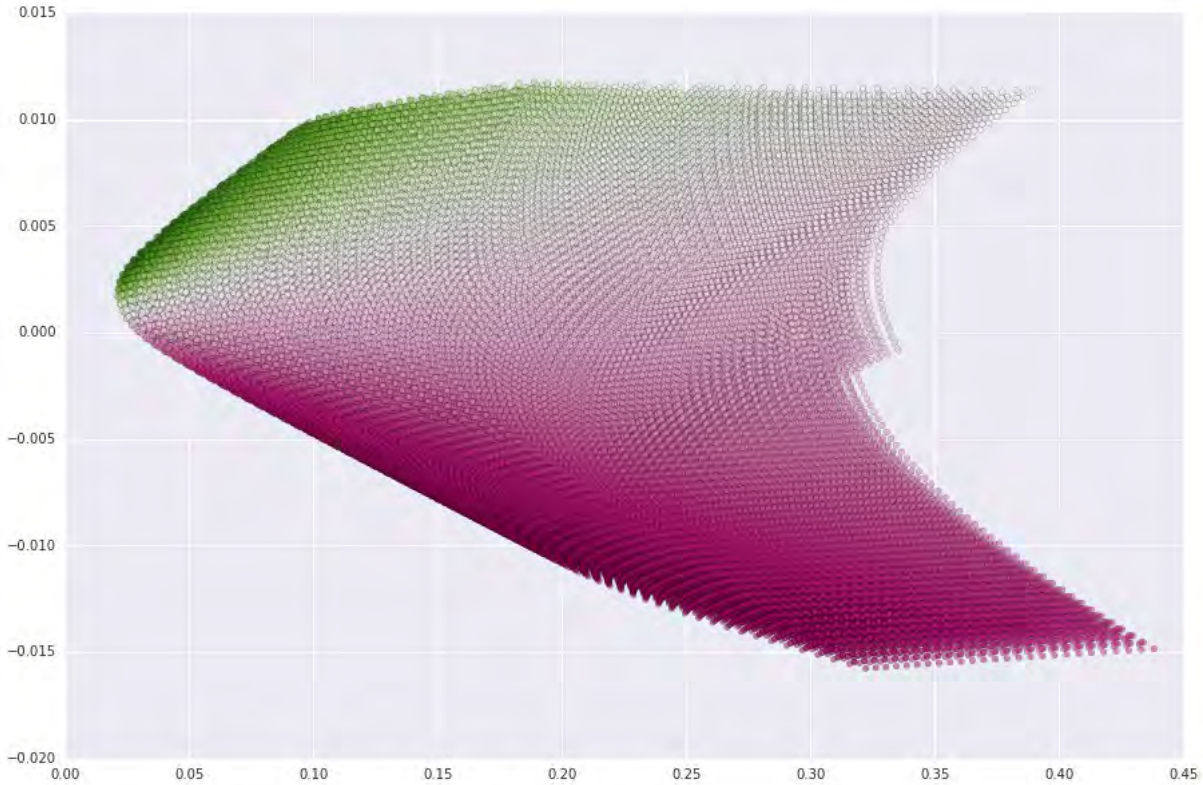


图中的五个蓝色三角是我们模型中选用的五支股票；背景中的粉点是沪深三百中的其他股票，作为参考。

计算这五支股票收益率的协方差，得到矩阵

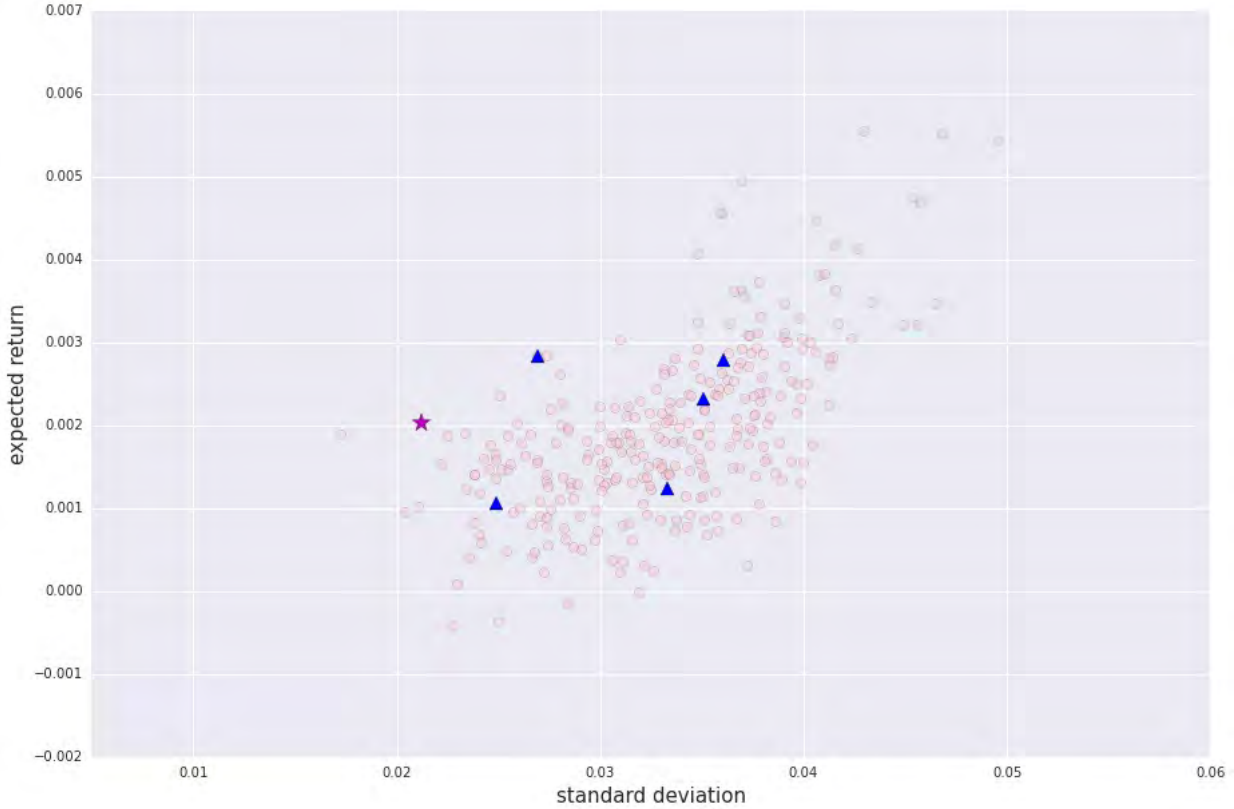
Undefined control sequence \<

这个矩阵的 (i, j) 位置是第 i 和第 j 个资产之间的协方差。通过穷举出用五支股票可以组成的资产配置（允许一定限度的卖空），绘出一个看起来很好吃的图



图中的每一个点代表一个风险资产组合，颜色偏绿代表夏普比率高，偏紫的代表夏普比率低（下一节会介绍夏普比率）。图中最左侧勾勒出来的曲线就是以这五支股票所构成的有效前沿，是在固定收益率的情况下能配置出的最小标准差。

有效前沿上风险最少的配置（最左侧的顶点）有标准差 0.0212，并且有预期收益率 0.00202。



上图中紫色五角星就是最低风险的配置，它的预期收益大约是五支股票的平均，但风险要小于其中的每一支。

理论上的有效前沿是无限延伸的，并且前沿右侧的所有空间都是可以通过配置这五支股票得到。因为计算量的有限性，看上去很好吃的图里只包含了可行资产配置里的一部分，不过其中已经包括了本模型中最重要的一个股票组合。是什么呢？

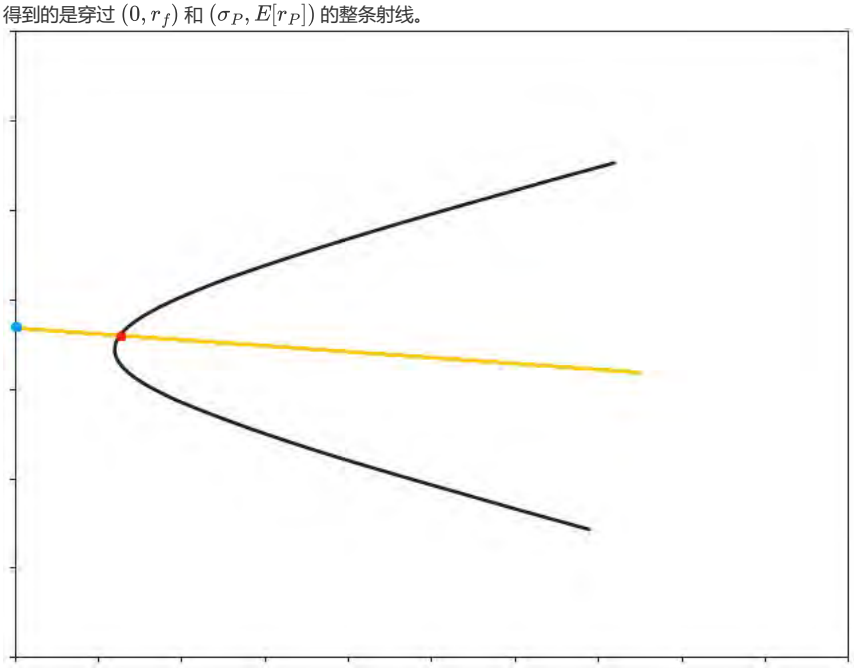
加入无风险资产

有效前沿左侧的区域是通过风险资产无法配置出的。但是！如果把无风险资产加入资产配置，那么左侧的一些位置是可以获取的。

我们选择有效前沿上的一个资产配置 P，并选择比例 $\alpha \geq 0$，将本金的 α 配置于 P，并将 $1-\alpha$ 配置于无风险资产。如果 $\alpha \leq 1$，那么 $1-\alpha \geq 0$，也就是说，我们将 $1-\alpha$ 倍的本金存入银行或买入债券，获取那部分的无风险利率。如果 $\alpha \geq 1$，那么 $1-\alpha \leq 0$，意思是，我们贷款本金 $\alpha-1$ 倍的资金，支付无风险利率，并用贷款连同本金一并配置于 P。如此，以 α 为系数，使用 P 和无风险资产配制出一个组合，我们将它的收益随机变量记为 $r<em>{\alpha}$。计算得到

Undefined control sequence \<

和上面的有效前沿的计算不同，这里 $\sigma < /em > \alpha$和 $E[r_\alpha]$ 不是复杂的多项式关系，而是简单的线性关系。因此，将所有 $\alpha \geq 0$ 所对应的点画出，我们得到的是穿过 $(0, r_f)$ 和 $(\sigma_P, E[r_P])$ 的整条射线。



如上图所示，如果蓝点是无风险利率，红点是一个有效前沿上的风险资产配置 P，那么黄线上的所有点都是可以按照一定比例配置无风险资产和 P 得到的。

使用常用的直线坡度计算方法，所得的黄线的坡度是 P 和 无风险资产的收益差除以它们的标准差的差。也就是，

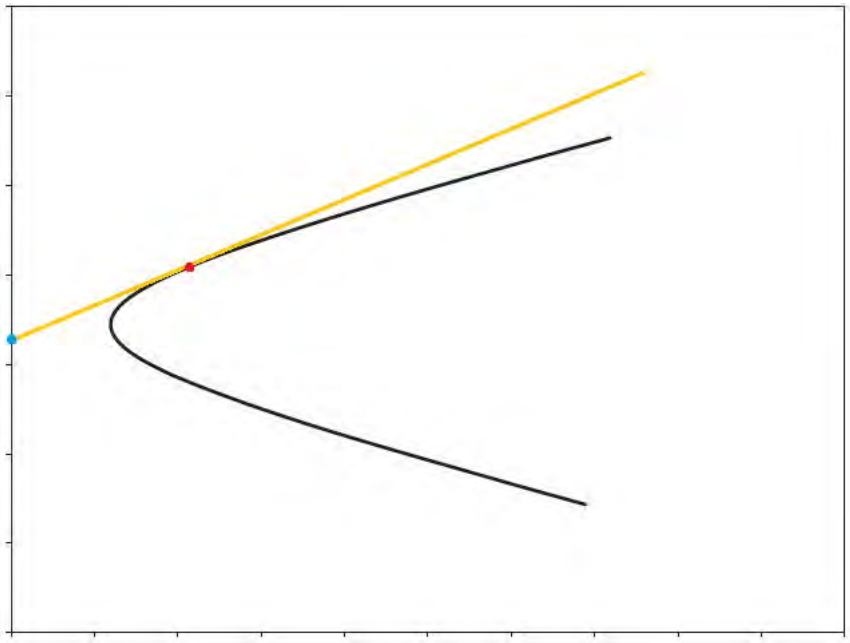
$$S < em > P := \frac{E[r_P] - r_f}{\sigma_P - \sigma < /em > r_f} = \frac{E[r_P] - r_f}{\sigma_P}.$$

这就是著名的*夏普比率*(Sharpe ratio)。

市场组合和资本市场线

经计算可以证明，用市场信息计算得来的有效前沿上必定有一个夏普比率最高的点，我们将其叫做*市场组合*(market portfolio)。

我们知道，一个配置的夏普比率等同于连接无风险资产和这个配置的直线的坡度；那么市场组合的夏普比率最高，就意味着它连接无风险资产的线坡度最陡。因此，这条线必定与有效前沿构成切线关系，如下图所示。



图中的蓝点是无风险资产，红点是市场组合，穿过它们的射线叫做*资本市场线*(capital market line)。红点和蓝点之间的位置是将一份资金存于无风险资产，并将余下资金买入市场组合而得到的；红点右侧的位置是以无风险利率进行一些贷款，并将本金连同贷款全部买入市场组合的配置。

设无风险利率为 r_f，市场组合为 M，并且市场组合的夏普比率为 S_M 的话，资本市场线的公式为

$$\mu = r_f + S_M \cdot \sigma.$$

资本市场线的意义在于，固定标准差，那么市场上收益预期最高的投资组合在这条线上；或者，固定预期收益，那么市场上标准差最低的投资组合在这条线上。所以，资本配置线可以直观地理解为理论上的“最佳配置线”。

实际中，无风险利率不是唯一的，贷款和存款所支付的利率也不是一样的。为了方便起见，我们一般会把国债的利率作为模型中的无风险利率。

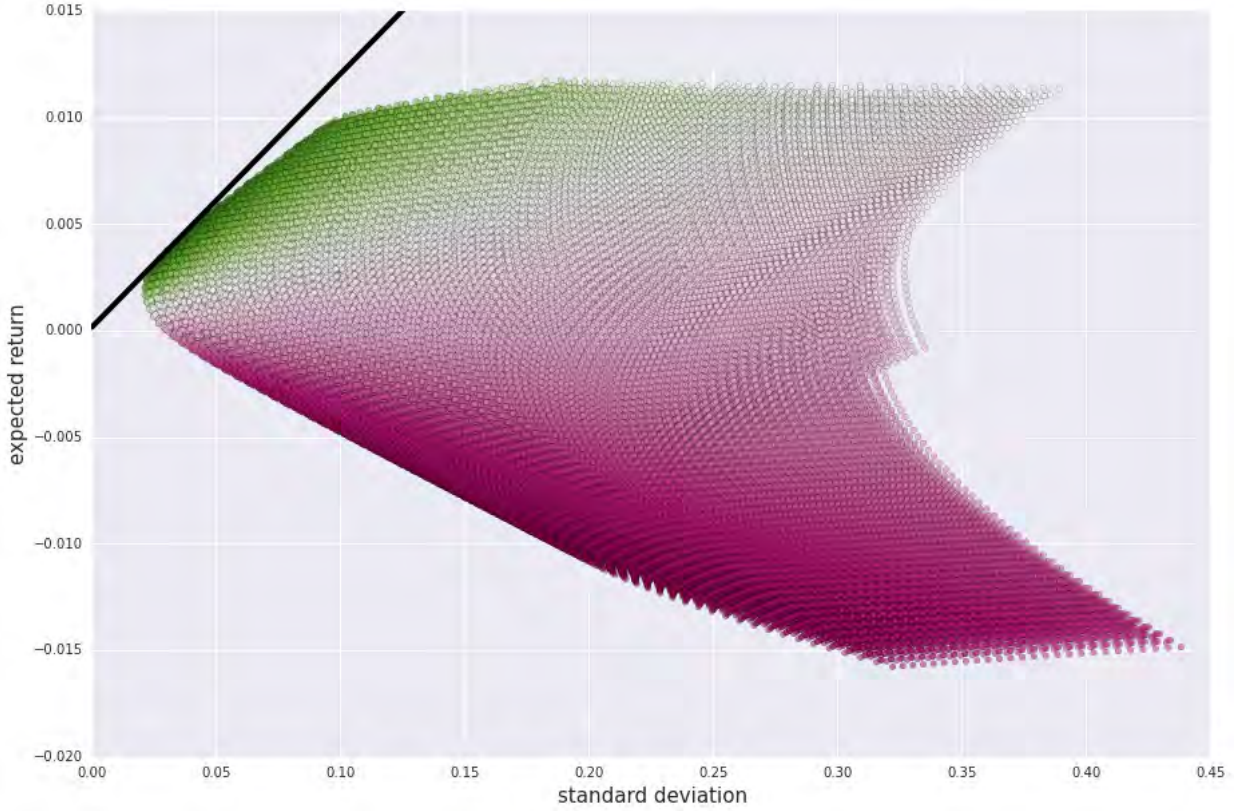
现在，在之前的示例中加入无风险资产进行计算。按国债年化利率为 4% 来计算，无风险的日化利率为

$$(1+0.04)^{\frac{1}{250}}-1\approx 0.00015689.$$

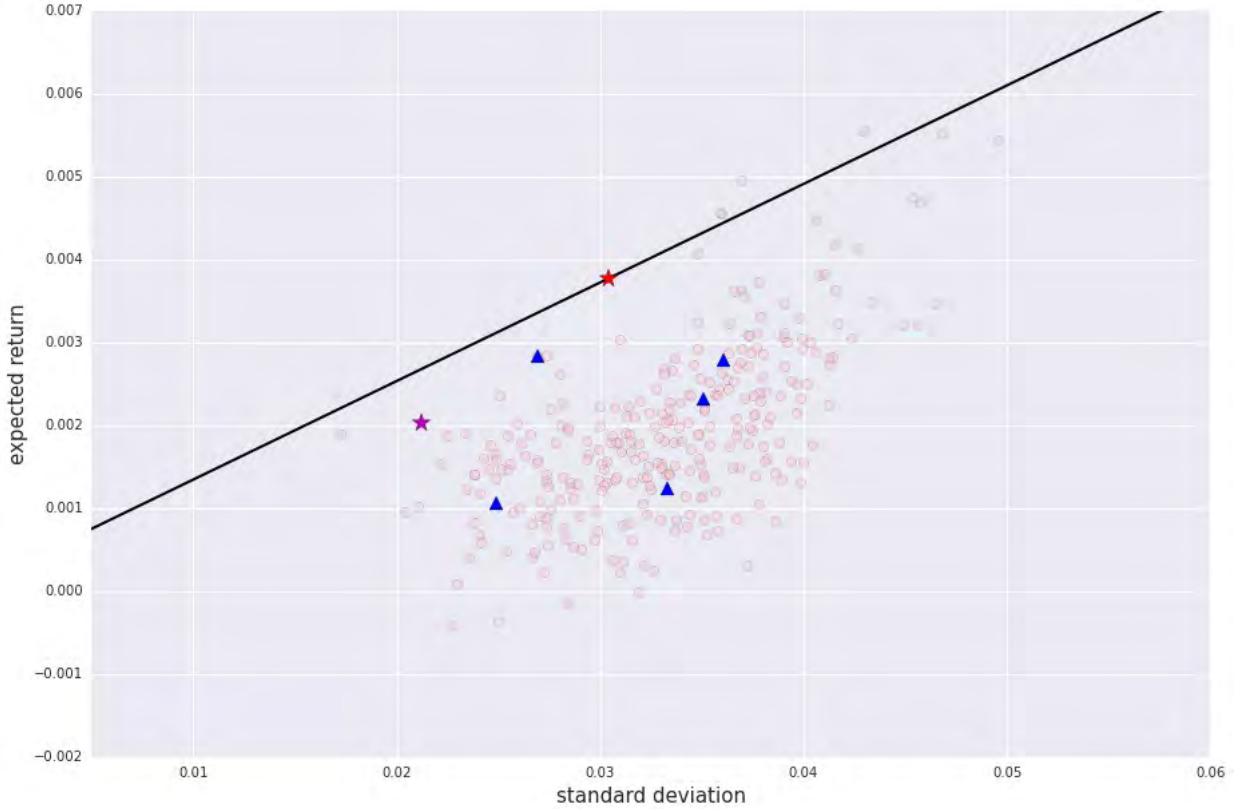
计算得知，有效前沿上最高的夏普比率是 0.11885，由此得知资本市场线的公式为

$$\mu = 0.00015689 + 0.11885\cdot\sigma.$$

将资本市场线画出，如下



资本市场线和有效前沿相切在夏普比率最高的资产配制上，我们将这个组合称作 M。计算可得，该组合的日化预期收益率为 $E[r_M] = 0.0037746$，并且标准差为 $\sigma_M = 0.030439$。将其与原生的五支股票以及最低风险组合进行对比，如下图



图中红色五角星是组合 M，黑线是资本市场线。

再假设，我们想用这五支股票和无风险资产配制出预期日化收益率为 0.003 并且风险最低的组合，那么就应该去资本市场线上找相应的位置。设我们要找的组合是由 α 份 M 和 $1-\alpha$ 份无风险资产构成的，利用资本市场线的公式倒推的标准差

$$\sigma < em > \alpha = \frac{\mu_0 - r_f}{\text{Sharpe}(M)} = \frac{0.003 - 0.00015689}{0.11885} \approx 0.023922.$$

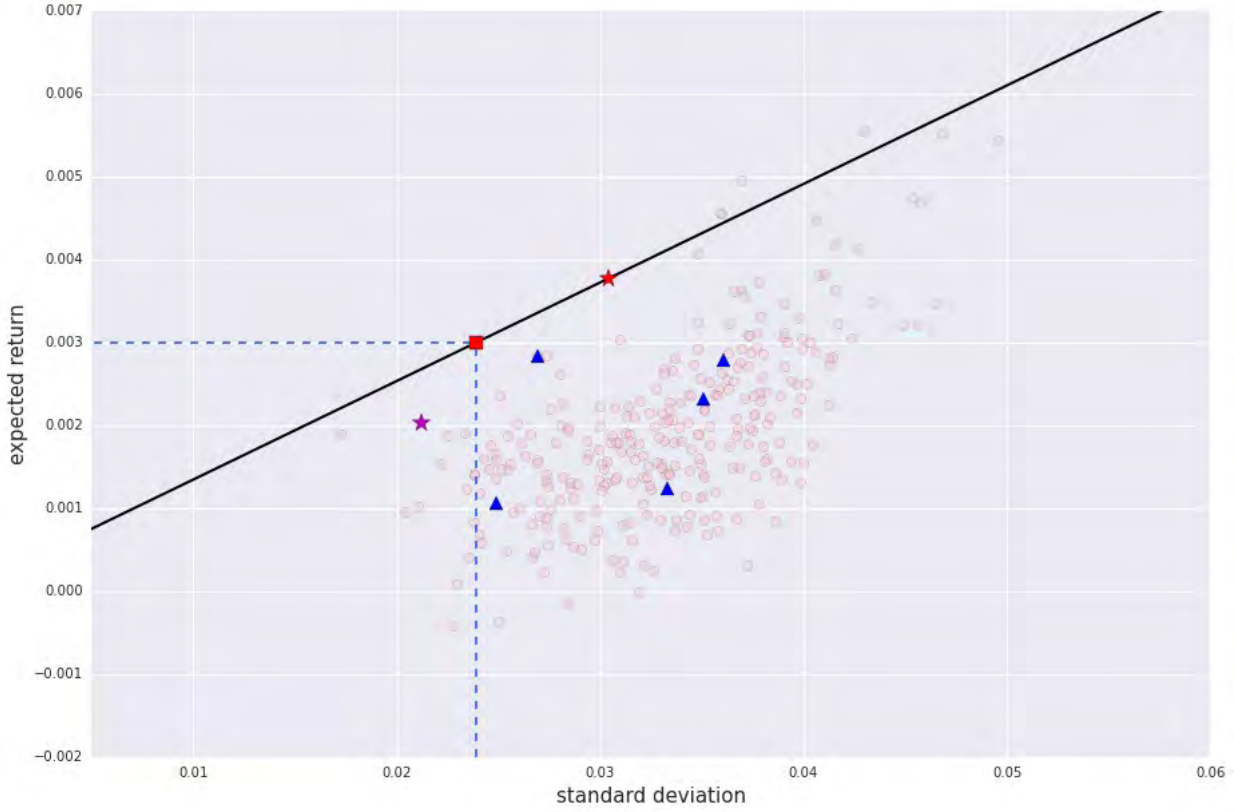
用第四节中所述的公式

$$\sigma < /em > \alpha = \alpha \cdot \sigma < em > M,$$

得出

$$\alpha = \frac{\sigma < /em > \alpha}{\sigma_M} = \frac{0.023922}{0.030439} \approx 0.786.$$

因此，该组合中每个 1 元钱的配置比例为 0.786 的 M 以及 $1-0.786=0.214$ 的国债。对应下图中的红方块。



这个新的股票组合的预期收益率高于它的五个成分股，但风险比其中的每一支都低。

结语

一个资产组合的预期收益是它的成分资产的线性加权平均，但组合的波动性和风险却更复杂，是由成分资产两两之间的相关性决定的。在 MPT 模型的分析中，通过合理地配置资产，可以在保证高收益的同时也降低风险。理论上，最优的风险组合就是夏普比率最高的市场组合。在本系列的下一篇文章中，我们将介绍著名的 CAPM 模型：在已知市场组合的定价的情况下，该如何推算市场上其他风险资产（或风险组合）的收益率。

文章更迭记录：
v1.1，2016-10-13，修正公式，感谢 KTBFFH 指出
v1.0，2016-08-02，文章上线

【量化课堂】“深综君”和“深成君”的前生今世 （追忆篇）

导语：
对于芸芸股民而言，“指数”这个词大家一定并不陌生，但是面对股市万花筒，就算是“老炮儿”也常常遭遇一头雾水不知所措的囧境（俗称“懵X”），比如说拗口难辨的“深证综指”和“深证成指”，就是一个再典型不过的例子了。这不，近期就经常有可爱的同学向我们咨询这二者的联系和区别，以至于我们感觉真的有必要跟大家聊一聊“深证综指”和“深证成指”的个中渊源了。简约起见，我们不妨用“深综君”和“深成君”来区分彼此。

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-1。
阅读本文前需要了解股票交易的基础知识，熟悉各大主流行情软件的基本使用方法，以及适当的独立思考精神。

作者：$BaSO_4$
编辑：宏观经济算命师

谁是“深综君”？谁又是“深成君”？

遥望当年，中国改革开放的总设计师邓爷爷大手一挥，深圳市成为了中国经济先试先行的红色特区，十年之后深圳证券交易所应运而生，中国证券市场由此开启了一段新的篇章。也正是在这个特殊的历史时期，“深综君”和“深成君”就此诞生，成为中国股市开天辟地之际的元老级角色，并承载了托起股市半边天的历史重任，人人敬仰，德高望重。

让我们来一睹他们的真容：

这两位简直是兄弟哇！长得太像了有木有！已然蒙圈了有木有！

没关系，为了方便大家理解，我们制作了一个简单直观的表格加以区分：

	深综君	深成君
指数全称	深圳综合指数	深圳成分指数
代码编号	399106	399001
指数基日	1991 年 4 月 3 日	1994 年 7 月 20 日
初始基点	100	1000
发布日期	1991 年 4 月 4 日	1995 年 1 月 23 日
指数性质	综合类指数	成份股指数
样本范围	深交所上市的全部公司的股票	从上市的所有股票中抽取具有代表性的 500 家公司的股票（2015 年 5 月 20 日之前为 40 家）
成分数量	1821	500
计算方法	1. 以指数股的计算日股份数作为权数进行加权计算 2. 股份数=全部上市公司的总股本数 3. 即时指数=（即日指数股总市值/基日指数股总市值）×基日指数	1. 以指数股的计算日股份数作为权数进行加权计算 2. 股份数=成份股的可流通股本数 3. 即时指数=（即日指数股总市值/基日指数股总市值）×基日指数

这样一来是不是清晰了很多！由此可见，深圳证券交易所并存着两个股票指数，一个是大哥深综君，一个是小弟深成君。

从行情图表来看，这老两位的人生轨迹相差无几，但是随着世事变迁，大哥深综君渐渐销声匿迹，鲜有表现，而小弟深成君却青出于蓝胜于蓝，后来居上地成为了和上证指数并驾齐驱的大红人。

不少同学对此“百思不得其姐”——二人即为兄弟，为何又分立山头，各自为政？莫非个中理由盘根错节不可告人？

其实原因并不复杂——

虽然人们对深综君寄予厚望，但理想丰满，现实骨感。在中国股市实际运作过程和反映股市实际运行状态方面，深综君不断地暴露出明显的先天缺陷，以至于自打生下来那天开始就注定了他的衰败。这种缺陷主要表现在如下方面：

1.不合理的“总股本”权数：中国股市只是个人股市场，国家股和法人股皆不能上市流通，如果使用总股本作为权数，偏差一定是大大的！

2.指数走低，新股上市成罪魁祸首：用全部股票纳入指数计算，新股上市即参与权重，会造成不利影响。比如新股上市后价格不稳，获利套现很多，如果高开低走，往往殃及指数。

3.结构变动频繁：每次新股上市即纳入计算，随着股票数量的增加，每只股票对指数的影响在不断降低，内部结构在不断变动，指数前后的可比性不强。

4.除权时的调整计算有所偏差：由于采用总股本计算除权价，如果国家股、法人股、个人股分红方案不同，则这种除权价对于个人股来说往往不是一个有效的指标，用来计算指数会产生偏差。

5.缺少分类指数（如商业类指数、金融类指数、地产类指数等等），无法多层次地反映市场的主体结构和资金流动状况。

正因为大哥深综君犯下上述“五宗罪”，并有过不幸跌破初始基点的重要前科（1991年月线五连阴，最低点45.66 

），因此深交所不得不狠下决心生下“二胎”深成君，扶持其坐上中国股市的第二把交椅，自此深综君“大将生来胆气豪，腰横秋水雁翎刀”的英雄气概一去不返，“大牌鸡肋”成了其唯一的命运……（自古大哥多磨难啊！怎么有点像黑社会……）

感叹了这么多，继续言归正传！深成君费尽周折登上深交所老大的宝座，自然要拉帮结派苦练内功，于是收买人心搞个名曰“成分股”的领导班子也就成了第一要务。

领导班子嘛，都得是精英，精英嘛，就得流弊闪闪！

那么精英是怎么被挖出来的呢？以表呈现——

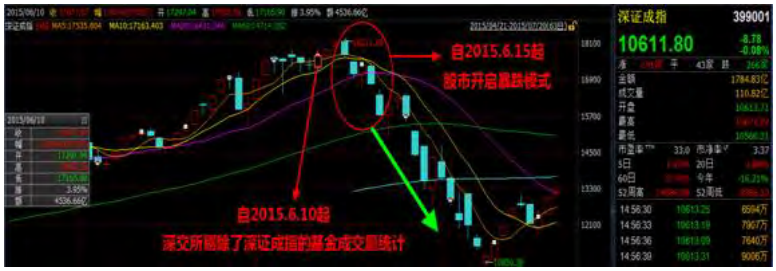
第一步：筛选入围成员	非 ST、*ST 股票；
	有一定上市交易日期（一般为六个月，总市值和自由流通市值综合排名位于深圳市场前 10 名的股票不受此限制）；
	公司最近一年无重大违规、财务报告无重大问题；
	公司最近一年经营无异常、无重大亏损；
	公司有一定的上市规模；
	公司有一定的市场流动性；
	考察期内股价无异常波动。
第二步：确定成分股成员	选样指标为一段时期（前 6 个月）平均总市值比重、平均自由流通市值比重和平均成交金额比重。先计算入围个股平均总市值占市场比重、平均自由流通市值占市场比重和平均成交金额占市场比重，再将上述指标按 1∶1∶1 的权重加权平均。然后将计算结果从高到低排序，选取前 500 名。 若排名相似，综合考虑公司的行业代表性及所属行业的发展前景、公司盈利记录等，优先选取指标优良的上市公司股票。

很严格有木有！另外，为保证筛选机制的客观性和公正性，成份股候选人的选拔不搞“终身制”，深交所定期帮助深成君考察候选人的胜任程度，及时更换能力下降的公司，选入生机勃勃的公司，将新陈代谢进行到底。当然，考虑到稳定性，这种变动不会太过频繁，考察时间为每年的一、五、九月。

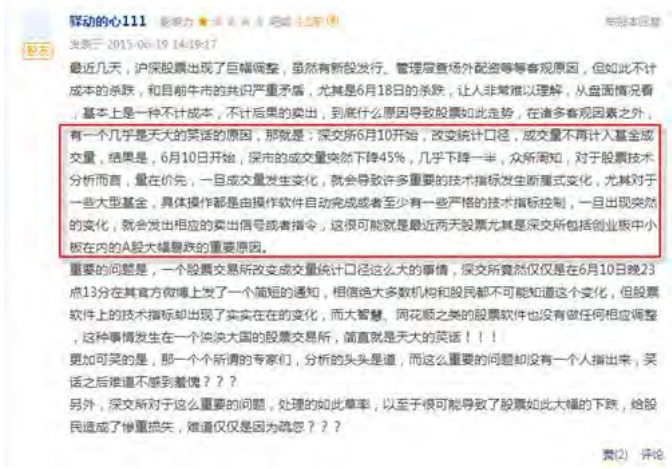
然而，深交所的头把交椅也不是那么好坐的，说了那么多风光的，也得交代交代阴暗的，比如说我们在研究市场的过程中，发现一年以前市场上就发生了一个不大不小的阴谋论风波，以至于深交所和深成君差点成了股灾面前千夫所指的“背锅侠”。

那几日，深成君的成交量大幅萎缩，股民发出强烈呐喊——

时间回到事发时，我们发现散户们总是意料之中的后知后觉——



对此，还未意识到股灾将至的“资深”股民义愤填膺——



驿动的心111

发表于 2015-06-19 14:19:17

最近几天，沪深股票出现了巨幅调整，虽然有新股发行、管理层查场外配资等等客观原因，但如此不计成本的杀跌，和目前牛市的共识严重矛盾，尤其是6月18日的杀跌，让人非常难以理解，从盘面情况看，基本上是一种不计成本，不计后果的卖出，到底什么原因导致股票如此走势，在诸多客观因素之外，有一个几乎是天大的笑话的原因，那就是：深交所6月10开始，改变统计口径，成交量不再计入基金成交量，结果是，6月10日开始，深市的成交量突然下降45%，几乎下降一半，众所周知，对于股票技术分析而言，量在价先，一旦成交量发生变化，就会导致许多重要的技术指标发生断崖式变化，尤其对于一些大型基金，具体操作都是由操作软件自动完成或者至少有一些严格的技术指标控制，一旦出现突然的变化，就会发出相应的卖出信号或者指令，这很可能就是最近两天股票尤其是深交所包括创业板中小板在内的A股大幅暴跌的重要原因。

重要的问题是，一个股票交易所改变成交量统计口径这么大的事情，深交所竟然仅仅是在6月10日晚23点13分在其官方微博上发了一个简短的通知，相信绝大多数机构和股民都不可能知道这个变化，但股票软件上的技术指标却出现了实实在在的变化，而大智慧、同花顺之类的股票软件也没有做任何相应调整，这种事情发生在一个泱泱大国的股票交易所，简直就是天大的笑话！！！

更加可笑的是，那一个个所谓的专家们，分析的头头是道，而这么重要的问题却没有一个人指出来，笑话之后难道不感到羞愧？？？

另外，深交所对于这么重要的问题，处理的如此草率，以至于很可能导致了股票如此大幅的下跌，给股民造成了惨重损失，难道仅仅是因为疏忽？？？

赞(2) 评论

然而，证监会依旧傻了吧唧的“辟谣”再次让市场浮想联翩——

網易財經 网易首页 > 财经频道 > 证券新闻 > 正文

证监会：深证成指成交量统计口径调整影响较小

2015-06-20 03:00:13 来源: 中国证券报-中证网(北京)

分享到：

针对深交所自6月10日起改变深证成指成交量统计口径，不再计入基金成交，证监会新闻发言人邓舸19日表示，证监会关注到了相关情况，经向深交所了解，此次调整目的是为统一行情软件中关于成交量的揭示，方便投资者与其它交易所进行比较。深交所原来关于全市场的总成交包括了股票和基金，本次技术调整后，统计范围包括深市主板、中小板、创业板在内的全部股票的成交，而不再包含基金的成交。调整后，由于基金成交金额比较小，对总成交金额的影响不到5%，对单个股票的成交量和成交金额则没有影响。

邓舸指出，在6月10日调整以后，深成指连续三天上涨，基金等机构投资者总体上净买入，并未出现大量卖出的情况。

邓舸表示，希望广大投资者审慎辨别市场传言，理性投资。

自此之后，股灾1.0，股灾2.0，股灾3.0，暴风，骤雨，无穷，无尽，亦无念——

大江东去浪淘尽，是非成败转头空，青山依旧在，银子全亏空……

如今看来，股灾已成往事，真相也成故事，最终谁也没说清是股灾成就了深成君“削基门”的阴谋论，还是深成君的“削基门”成为了股灾的导火索（之一）。正如《逻辑思维》所说：“常说真理越辩越明。对，短时间内此言不假。但是，很多重要的历史往事，大家不是越辩越明，而是各讲各的故事，看谁的故事更有吸引力、传播力、感召力，最后沦为讲故事的战争……”

因此，我们只相信“价格包含一切市场行为”的基本公理，坚决不做多余评论……

年华似水，匆匆一瞥，多少岁月，轻描淡写。文章到此，“深综君”和“深成君”二十多年的恩怨情长与悲欢离合也讲的差不多了，由于篇幅有限，无数过往只能一带而过，匆匆了之。

在此，为了体现用户至上的真诚态度和孜孜不倦的工匠精神，我们决定在几天之后赠送大家一个“大彩蛋”——《星霜篇》，对一直把本《追忆篇》坚持读完的同学表示严重感谢。

欲知详情，请看下篇。

（友情提示：《星霜篇》篇幅较短，但可能仍会引起您的疑虑和不适，如有顾虑，敬请放弃^_^）

文章更迭记录：
v1.0，2016-08-13，文章上线

【量化课堂】“深综君”和“深成君”的前生今世 （星霜篇）

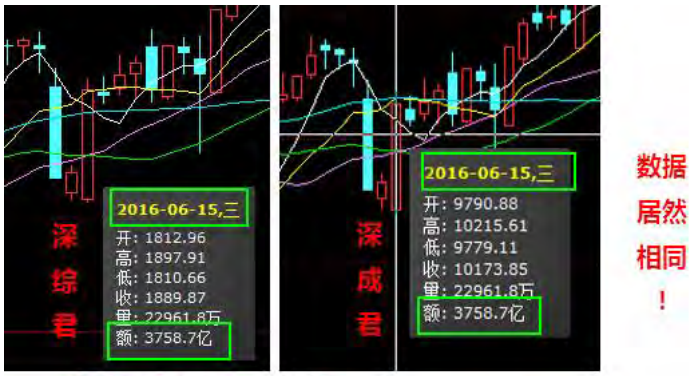
导语：
在此前的《追忆篇》中，我们着重介绍了“深综君”和“深成君”这对兄弟二十多年来的恩怨情长与悲欢离合。文章置笔的那一刻，笔者颇有种“春去秋来红尘中，谁在宿命里安排；红红心中蓝蓝的天，是个生命的开始；前尘红世轮回中，谁在宿命里徘徊；冰雪不语寒夜的你，那难隐藏的光彩……”（91版《雪山飞狐》片尾曲《追梦人》中的歌词）一般的感伤情怀。今天，就让我们把思绪回归当下，来一次小小的“入世”思索。
</br>

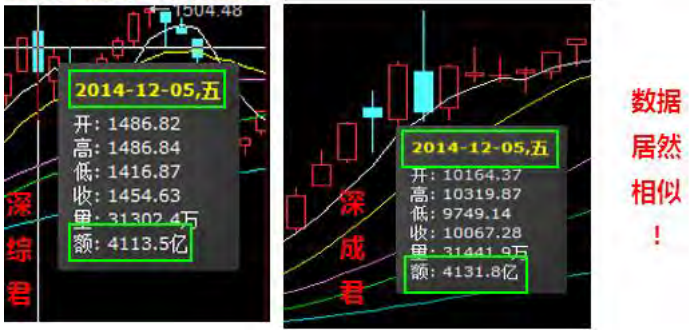
本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-1。
阅读本文前需要了解股票交易的基础知识，熟悉各大主流行情软件的基本使用方法，以及适当的独立思考精神。

作者：$BaSO_4$
编辑：宏观经济算命师

</br>

（友情提示：本文篇幅较短，但可能仍会引起您的的疑虑和不适，如有顾虑，敬请止步^_^）
</br>
众所周知，国内不少金融行情软件向来鲜有职业精神，不说别的，仅仅是在“深综君”和“深成君”这两个的成交额数据上就让人无比蛋疼，我们截图来加以说明：
</br>





</br>
这是传说中“市场占有率位居领先地位”的某行情软件的成交额数据——
</br>

</br>

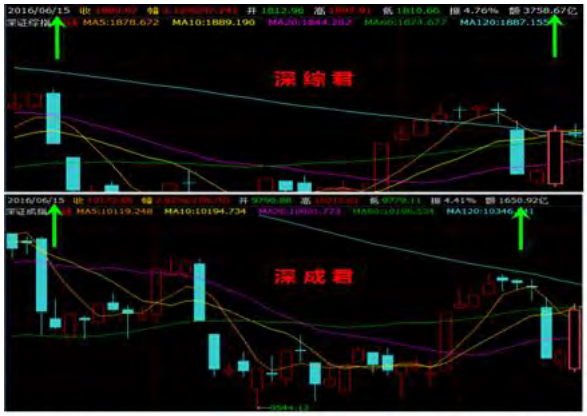
到最后我们发现，对于深综君，人家连数据都不提供给你了（美其名曰“顺应客户需求”），而上图的成交额数据，原来都是“偷梁换柱”（或有其它原因我们不得而知）——

</br>

</br>

更加坑爹的是，我们惊讶的发现，有着类似情况的国内软件居然不在少数，对此我们深感无奈的同时也只能“呵呵”以对了……（欢迎大家验证）那么是否所有的行情客户端都是如此呢？答案是否定的。相比之下，某知名金融数据公司（投资机构方市场占有率大约90%）的厚道和专业就应当给予三十三个赞！有图为证——

</br>



数据不同！
差距合理！
Correct！

数据不同！
差距合理！
Correct！

数据不同！
差距合理！
Correct！

</br>
如此一看，高下立判；孰好孰坏，一目了然！
</br>
然而，“世事难预料”这句话说得一点没错。就在本文准备发稿的当天，我们例行公事般再次核实了该公司软件客户端的两指数据，结果却悲伤地发现，曾经的“阳光少年”，如今竟也被犬儒主义打倒，而经过认真沟通，我们得到的官方解释同样是：顺应客户需求……（若需真实数据，需要另外购买）
</br>
有图为证——
</br>

</br>

于是，我们再一次陷入了深深的思考……

</br>

我们认为，在某种利益诉求的前提下，“市场为王”的经营思维无论如何都无可厚非，从这个角度讲谁也无法超凡脱俗，但同时不得不说，“顺应客户需求”式的“民主”如果走过了头就可能带来“民主的泛滥”和“信息的失真”，最终导致“市场的走偏”和“利益的失衡”，这同样是不争的事实。在这个泥沙俱下物欲横流金钱至上的浮躁当下，能够虚极静笃愿意回归初心的企业和团队已为数不多，而在致力于“无条件将产品做到极致”这一原始问题上，所有人都需要付出更多更多。我们毫无理由地坚信，无论时代怎样发展，社会怎么样进步，“工匠精神”和“共享精神”都应是亘古不变的内在驱动和活力源泉，而这作为专业服务的第一根基，必将成为对于忠诚用户的最大回报以及对于“不忘初心，方得始终”的最好诠释。

</br>

说到这里，我们突然想到弘一大师李叔同先生《晚睛集》中的一段描述：“世界是个回音谷，念念不忘必有回响，你大声喊唱，山谷雷鸣，音传千里，一叠一叠，一浪一浪，彼岸世界都收到了。凡事念念不忘，必有回响。因它在传递你心间的声音，绵绵不绝，遂相印于心。”

（上述思考或有偏颇，欢迎探讨，也欢迎拍砖。）

最后的最后，让我们思绪回归，上段干货：

</br>

```
df=get_price(security=['399106.XSHE','399001.XSHE'],start_date='2010-01-01',end_date='2016-08-19', frequency='daily', fields='money', skip_pause
df.money = df.money.applymap(lambda x: '%.2f亿' % (x/1e8))
df.money
```

</br>

大家可以尝试在研究模块中键入如上代码，看看结果如何。

</br>

没错，正是 “深成君”和“深综君”的真实成交额数据列表（代码中的日期可任意修改）：

</br>

	399106.XSHE	399001.XSHE
2010-01-04	822.79亿	141.99亿
2010-01-05	1023.96亿	224.17亿
2010-01-06	1028.86亿	235.38亿
2010-01-07	1042.56亿	229.67亿
2010-01-08	831.22亿	163.18亿
2010-01-11	1001.92亿	211.21亿
2010-01-12	1144.97亿	243.58亿
2010-01-13	1254.63亿	298.52亿
2010-01-14	1190.06亿	219.46亿
2010-01-15	1060.95亿	182.12亿
2010-01-18	1071.67亿	192.85亿
2010-01-19	1067.42亿	171.02亿
2010-01-20	1291.09亿	215.23亿
2010-01-21	835.15亿	170.27亿
2010-01-22	953.63亿	193.52亿

</br>

2016-07-28	3396.13亿	1689.68亿
2016-07-29	2613.73亿	1265.67亿
2016-08-01	2550.91亿	1219.64亿
2016-08-02	1932.70亿	866.39亿
2016-08-03	2228.11亿	978.84亿
2016-08-04	2515.10亿	1182.13亿
2016-08-05	2404.75亿	1157.59亿
2016-08-08	2421.81亿	1204.07亿
2016-08-09	2756.00亿	1310.77亿
2016-08-10	2839.75亿	1314.15亿
2016-08-11	2704.89亿	1246.00亿
2016-08-12	2473.66亿	1218.52亿
2016-08-15	4231.48亿	1218.52亿
2016-08-16	4010.55亿	2056.63亿
2016-08-17	3751.49亿	2056.63亿
2016-08-18	3783.49亿	1795.60亿
2016-08-19	3245.25亿	1473.51亿

1612 rows × 2 columns
</br>

这些数据，同学们不必有所顾忌，尽可免费获取免费使用，不解释，你懂的！
</br>
所以……就没有所以了！文章到此结束！
</br>
欢迎更多的同学能加入到聚宽的大家庭，做研究，大丰收！
</br>

文章更迭记录：
v1.0，2016-08-19，文章上线

In [1]:

```
df=get_price(security=['399106.XSHE','399001.XSHE'],start_date='2010-01-01',end_date='2016-08-18', frequency='daily', fields='money',
df.money = df.money.applymap(lambda x: '%.2f亿' % (x/1e8))
df.money
```

Out[1]:

	399106.XSHE	399001.XSHE
2010-01-04	822.79亿	141.99亿

	399106.XSHE	399001.XSHE
2010-01-05	1023.96亿	224.17亿
2010-01-06	1028.86亿	235.38亿
2010-01-07	1042.56亿	229.67亿
2010-01-08	831.22亿	163.18亿
2010-01-11	1001.92亿	211.21亿
2010-01-12	1144.97亿	243.58亿
2010-01-13	1254.63亿	298.52亿
2010-01-14	1190.06亿	219.46亿
2010-01-15	1060.95亿	182.12亿
2010-01-18	1071.67亿	192.85亿
2010-01-19	1067.42亿	171.02亿
2010-01-20	1291.09亿	215.23亿
2010-01-21	835.15亿	170.27亿
2010-01-22	953.63亿	193.52亿
2010-01-25	586.64亿	122.37亿
2010-01-26	707.93亿	157.00亿
2010-01-27	589.35亿	145.47亿
2010-01-28	587.97亿	136.24亿
2010-01-29	683.26亿	133.17亿
2010-02-01	728.34亿	135.65亿
2010-02-02	628.16亿	118.38亿
2010-02-03	796.63亿	164.51亿
2010-02-04	808.99亿	130.59亿
2010-02-05	824.78亿	129.99亿
2010-02-08	531.38亿	81.49亿
2010-02-09	467.58亿	77.84亿
2010-02-10	476.39亿	89.88亿
2010-02-11	469.98亿	92.51亿
2010-02-12	464.57亿	83.95亿
...	...	...
2016-07-08	3805.02亿	1784.83亿
2016-07-11	4279.70亿	1998.93亿
2016-07-12	4462.22亿	2194.80亿
2016-07-13	4497.66亿	2347.99亿
2016-07-14	3790.09亿	1910.15亿
2016-07-15	3652.86亿	1803.25亿
2016-07-18	3510.21亿	1733.02亿
2016-07-19	3294.68亿	1628.18亿
2016-07-20	3338.62亿	1532.01亿
2016-07-21	3565.91亿	1665.68亿
2016-07-22	3423.21亿	1647.00亿

	399106.XSHE	399001.XSHE
2016-07-25	3163.45亿	1518.64亿
2016-07-26	2996.54亿	1424.57亿
2016-07-27	4900.87亿	2446.45亿
2016-07-28	3396.13亿	1689.68亿
2016-07-29	2613.73亿	1265.67亿
2016-08-01	2550.91亿	1219.64亿
2016-08-02	1932.70亿	866.39亿
2016-08-03	2228.11亿	978.84亿
2016-08-04	2515.10亿	1182.13亿
2016-08-05	2404.75亿	1157.59亿
2016-08-08	2421.81亿	1204.07亿
2016-08-09	2756.00亿	1310.77亿
2016-08-10	2839.75亿	1314.15亿
2016-08-11	2704.89亿	1246.00亿
2016-08-12	2473.66亿	1218.52亿
2016-08-15	4231.48亿	1218.52亿
2016-08-16	4010.55亿	2056.63亿
2016-08-17	3751.49亿	2056.63亿
2016-08-18	3783.49亿	1795.60亿

1611 rows × 2 columns

【量化课堂】CAPM 模型和公式

导语：α 和 β 你肯定都听说过吧。那么 γ 呢？δ？ε？ζ，η，θ，ι，... ω？？？那好！我们今天就来告诉你..... β 是什么。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，难度为进阶上，深度为 level-2。

阅读本文需要掌握 MPT 模型（level-1）和微积分（level-0）的知识。
本文是一系列文章中的第三篇。本系列从基础概念入手，推导出 CAPM 模型。系列中共有四篇：

1. 效用模型
2. 风险模型
3. MPT 模型
4. CAPM 模型

概述

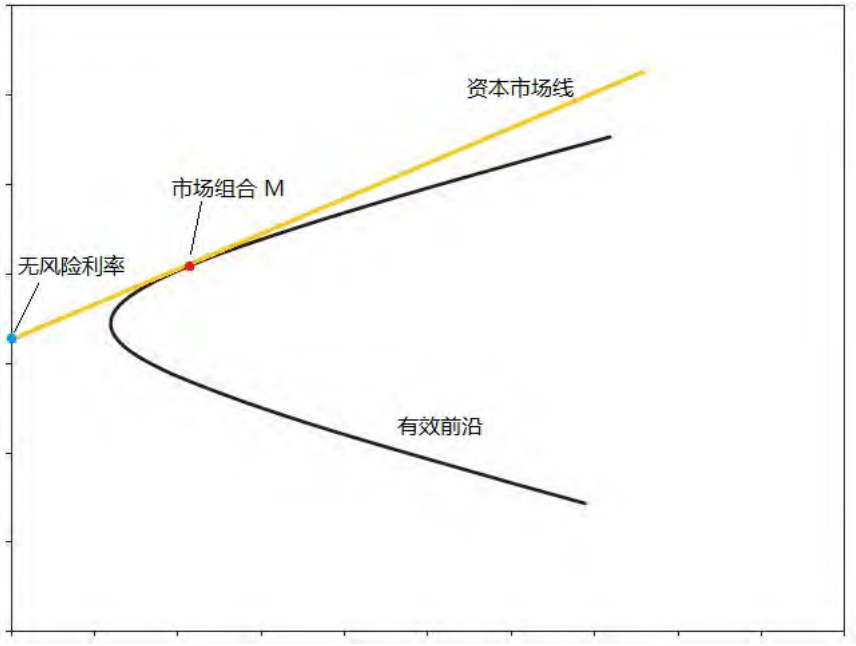
CAPM，全称 Capital Asset Pricing Model，译为资本资产定价模型，是由 Treynor, Sharpe, Lintner, Mossin 几人分别提出。搭建于 Markowitz 的现代资产配置理论（MPT）之上，该模型用简单的数学公式表述了资产的收益率与风险系数 β 以及系统性风险之间的关系。尽管 CAPM 的假设偏于牵强，结论也常与实验证据相悖，但它一直是金融经济学中重要的理论，为更多先进的模型打好了基础。

模型假设

CAPM 是一个理论性很强的模型，它所假设的金融市场有一个非常简单的框架，这样不仅简化了分析的难度，也用非常简练的数学公式表达出结论。

CAPM 假设，市场上所有的投资者对于风险和收益的评估仅限于对于收益变量的预期值和标准差的分析，而且所有投资者都是完全理智的。并且，市场是完全公开的，所有投资者的信息和机会完全平等，任何人都可以以唯一的无风险利率无限制地贷款或借出。

因此，所有投资者必定在进行资产分配时计算同样的优化问题，并且得到同样的有效前沿和资本市场线（见 MPT 模型）。



为了最大化预期收益并最小化标准差，所有投资者必定选择资本市场线上的一点作为资产配置。也就是说，所有投资者都按一定比例持有现金和市场组合 M。因此， M 是名副其实的“市场组合”，因为整个市场都是按照这个组合来分配资产的。所以 M 的波动性和不确定性不单单是市场组合的风险，也是整个市场的风险，叫做*系统性风险*(systematic risk)。

CAPM 公式

CAPM 公式是从以上模型框架推导出的数学表达式，它表达了任何风险资产的收益率和市场组合的收益率之间关系。在这个公式中，任何风险资产的收益率都可以被分为两个部分：无风险收益（利率）和风险收益（β 收益）。我们先看公式。

定理（CAPM 公式）. *对于某一风险资产 S（可以把 S 想象为一种证券），有*

$$E[r_S] = r_f + \beta_S \cdot (E[r_M] - r_f).$$

其中:
– r_S 是组合 S 的收益变量;
– r_M 是市场组合的收益变量,
– r_f 是市场的无风险利率,
– β_S 是组合 S 对于市场风险的敏感度，计算公式为

$$\beta_S = \frac{\mathrm{Cov}(r_S, r_M)}{\mathrm{Var}(r_M)}.$$

在公式中，r_f 是资产的时间价值，是按照无风险利率产生的收益。右边的 $\beta_S \cdot (E[r_M] - r_f)$ 是资产的风险收益，是对投资者所承担的风险的补偿。这里面，$E[r_M] - r_f$ 是市场组合的风险收益，还有 β_S 是组合 S 对系统性风险的敏感系数。可以理解为，资产 S 承担了 β_S 倍的系统性风险，所以将会得到相应倍数的风险补偿。

这里应该指出，风险组合 S 的预期收益是完全由它的 β_S 决定的，与这个资产自己的风险 σ_S 是没有关系的。也就是说，假设风险资产 S 有巨大的风险 σ_S，但是它和市场组合的相关性 β_S 很小，那么 S 预期的收益率其实是很小的。

通过 CAPM 公式，我们还可以推算出资产 S 的夏普比率 $\mathrm{Sharpe}(S)$ 和市场组合 M 的夏普比率 $\mathrm{Sharpe}(M)$ 的关系，如下。

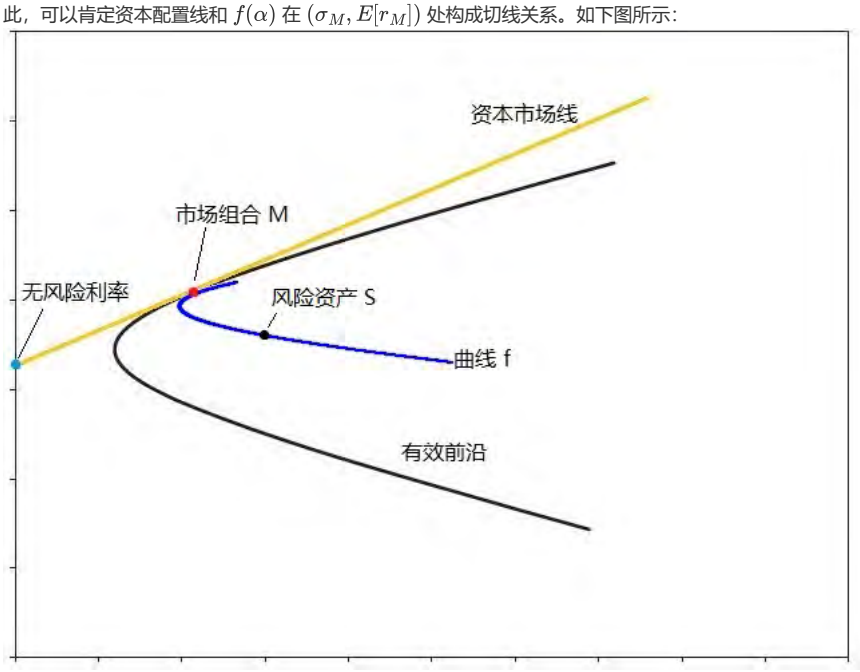
Unknown environment 'align'

也就是说，组合 S 的夏普比率等于 M 的夏普比率乘以 M 与 S 的相关系数。在 MPT 模型中，M 是所有风险组合中夏普比率最高的，也就是最有效的。这个公式告诉我们， S 和 M 的相关性越高，S 的夏普比率就越高，收益比风险的比例也就越大。

CAPM 公式的证明

证明. 对于 $\alpha \in \mathbb{R}$，我们定义 $A < em > \alpha$为按照 α 份 S 比 $1 - \alpha$ 份 M 所构成的资产配置。定义 $\overline{r}(\alpha)$ 和 $\sigma(\alpha)$ 分别为组合 $A < /em > \alpha$的预期收益和波动率。定义函数 $f(\alpha) = (\sigma(\alpha), \overline{r}(\alpha))$，那么 f 是一条在可行资产配置区域内的平滑曲线，并且 $f(1) = (\sigma_S, E[r_S])$， $f(0) = (\sigma_M, E[r_M])$。该曲线在 $\alpha = 0$ 时和资本配置线重叠在市场组合之上；并且，由于所有风险资产配置得位置都在资本配置线的下方，所以曲线 f 不会和资本配置线相交。因

此，可以肯定资本配置线和 $f(\alpha)$ 在 $(\sigma_M, E[r_M])$ 处构成切线关系。如下图所示:



我们知道资本配置线的坡度是 M 的夏普比率

$$< br > \text{Sharpe}(M) = \frac{E[r < em > M] - r_f}{\sigma_M}. < br >$$

那么，曲线 f 的 $\overline{r}$ 坐标相对于 σ 坐标在 $\alpha = 0$ 的导数也等于这个坡度,

$$< br > \frac{d\overline{r}}{d\sigma}\Big| < /em > \alpha = 0 = \frac{E[r < em > M] - r_f}{\sigma_M}. \quad (1) < br >$$

$A < /em > \alpha$ 的收益变量是

$$< br > r < em > A < /em > \alpha = \alpha r < em > S + (1 - \alpha) r_M, < br >$$

因此有

Undefined control sequence \<

代入等式 (2) 和 (3) 并使用初等微积分技能，我们亲手肢解等式 (1) 左侧的表达式，

Undefined control sequence \<

再代入等式 (1) 得到

Undefined control sequence \<

擦干净双手，完成了证明。Undefined control sequence \blacksquare

CAPM 的应用

CAPM 公式的应用在理论上是一个悖论，那是因为在 CAPM 的假设下所有投资者都持有市场组合 M，那么投资者也没有必要去单独计算每一个风险资产的收益率 --- 因为他所持有的资产配置已经是最优的了。但实际上，投资者的效用标准都不一样，资产配置也大相庭径，并不存在一个一致认同的市场组合，这时 CAPM 公式就可以派上用场。

在现实环境里，我们可以将一个概括市场整体的组合（比如大盘指数）作为市场组合，并以其为基准计算每个风险资产的系统性风险 β。这样，我们根据对市场整体趋势的判断以及对风险控制的需要，选择适当的 β 进行资产配置。
举个例子，假设我们以沪深300指数作为市场组合，取过去 500 天的年化日均收益率为 $(r < em > M(t)) < /em > t = 1^{500}$，并且取一只股票 S 过去 500 天的年化日均收益率为 $(r < em > S(t)) < /em > t = 1^{500}$。记它们的均值为

$$\mu < em > M = \frac{1}{500} \sum < /em > t = 1^{500} r < em > M(t),\ \mu_S = \frac{1}{500} \sum < /em > t = 1^{500} r < em > S(t).$$

可以估测出 S 的 β 为

$$\beta_S = \frac{\text{Cov}(r_S, r_M)}{\text{Var}(r_M)} = \frac{\sum < /em > t = 1^{500} (r < em > M(t) - \mu_M)(r_S(t) - \mu_S)}{\sum < /em > t = 1^{500} (r_M(t) - \mu_M)^2}.$$

假设我们选出三支股票，并且计算发现它们的 β 分别为 $\beta_1 = 0.5$， $\beta_2 = 1$ 和 $\beta_3 = 2$，也就是说它们对市场风险的敏感度依次为低、中、高。如果我们通过分析预测大盘在近期会整体趋势向上，那么应该持仓股票3，赚取大盘 2 倍的收益；如果我们认为大盘近期的走势不可判断，那么可以持有股票2，这样收益基本与大盘持平；如果我们认为大盘在近期会整体下跌，那么可以持有股票1，这样损失只有大盘的一半，或者干脆直接空仓。

结语

量化课堂的“从效用到CAPM”系列文章从效用模型开始，介绍了风险分散的原则，再到 MPT 资产配置理论的有效前沿和市场组合，最后本篇讲解了 CAPM 中风险资产的定价公式，这些内容都是金融经济学的核心基础。在以上基础上建立的有 APT 套利定价模型、Fama-French 三因子模型以及其他的诸多理论和模型。读者可以在量化课堂的“策略与应用”栏找到基于这些模型的交易策略，也敬请期待量化课堂未来的经济学文章。

文章更迭记录：
v1.1，2016-09-01，修改公式，感谢 jiziwei 指出
v1.0，2016-08-18，文章上线

【量化课堂】白话分级基金之一——一家“鸡”不说两家话

导语：

炒股时，大家都有如下感受——牛市来了，自己的股票死活跑不赢指数，赚了指数不赚钱；而坑爹的熊市，手里的票又哐哐猛跌，相比指数更加惨不忍睹。每每此时，大家都会泪流满面，不知所措，感觉身体被掏空。那有没有一个方法或品种，能牛市显神威，熊市不亏钱呢？有！那就是中国特色的分级基金。

从本篇开始，我们将用一个系列的文章为投资小白们介绍分级基金这个史上最具可玩性的基金品种。系列文章计划用五个篇章分别讲述分级基金的概念属性、发展历程、分级A、分级B，以及分级套利。表述内容会尽量多用通俗语言，少用晦涩公式，意在抛砖引玉，方便理解。

为了让大家先有个感性的认识，本次我们带来分级基金扫盲第一弹——《一家“鸡”不说两家话》。
</br>

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。
阅读本文前最好对金融市场的大致分类和常规金融理财品种的基本特性有一定了解。

作者：$BaSO_4$
编辑：宏观经济算命师

传送门：《白话分级基金之二——成长的烦恼（上）》

分级基金以相对标准化产品的形式，既提供了较低杠杆的指数化投资途径，也提供了固定收益投资途径，为投资者带来了更丰富多样的投资机会，对投资者有利，对市场有利。 ——银河证券 衍生产品部行政负责人、董事总经理 丁圣元
</br>

分级基金？何方神圣？

分级基金，简单来说，就是把一个基金在内部分离成不同级别的两个基金。

那有人就纳闷了，为何平白无故非得要把人家活生生地拆散呢？真相只有一个：理念不同！

这里说的理念不同，不是说那个基金自己受了刺激“精神分裂”了，而是说选择这个基金的投资者，他们的投资性格大相径庭——

有人信奉“萝卜白菜保平安，老婆孩子热炕头”，他们不想一夜暴富，只想细水长流，“低风险，低收益，我乐意！”；但有人则崇尚“自古富贵险中求，人生不博怎精彩！”，为了牟取暴利，铤而走险也在所不惜，“高风险，高收益，忒刺激！”

当然，截然不同的投资观念无可厚非，无对无错，但把这两类人巧妙地捏合到一起，化学质变就此产生——

激进者逐利心切，但苦于囊中羞涩，怎么办？借嘛。比如，我只有1个亿，牛市中获利翻倍，也只赚到1个亿；但如果我借1个亿，拿2个亿去投资，同样收益100%时我就赚到2个亿，获利暴增（相比自己的本金1个亿）！

在投资领域，这就叫“资金杠杆”，也就是阿基米德口中的“给我一个支点，我能撬起整个地球”。

但是，人家又不搞公益事业，钱不能白借啊，而最终“还本付息”的这个“息”就是你的代价所在。

对此，洞察力敏锐的公募基金公司审时度势，迅速捕捉到创新机遇，发明出一种既能满足“好斗者”又能迎合“老实人”的基金产品——分级基金。

这个基金可以一分为三，一个叫“母基金”，俗称“母鸡”；另外两个是“母鸡”生下的儿子（子基金），分别名曰“分级A”和“分级B”，我们简称“小A”和“小B”。

话说有一天，母鸡把辛辛苦苦攒下的钱，一分为二发给小A和小B（可平分，也可不平分），让它们投资创收。

两个儿子非常开心，但面对现实，小A和小B表现出的脾气秉性却大相径庭。大儿子小A性格保守追求稳健，不敢风险投资；小儿子小B性格激进自信满满，它觉得哥哥太怂，就把它的资金借来一并打理，盈亏自负，并每年给哥哥偿付利息。

于是，兄弟俩心心相印，一拍即合，分级投资之旅就这样愉快开始了。

说到这里，投资小白们满脑袋问号：

“分级基金看上去很美，但安全系数有多高？”（安全性）

“我的钱随时可取吗？“（流动性）

“收益如何呢？“（收益性）

“分级跟其它投资工具相比有什么优势呢？“（独特性）

“适合我们这些理财二把刀吗？”（亲民性）

别着急，咱们细细道来。

资金安全吗？

首先，公募基金公司发行的，至少跑路的事是绝不可能的！

其次，很多人会问，小B这么屌，耍赖不给小A利息怎么办？这点，你想多了。

表面来看，母鸡、小A、小B一分为三，投资权也分归各自，但实际上它们还是相亲相爱的一家人（鸡），资金的生杀大权仍掌握在母鸡手里，它每年会定期直接从小B账户里扣除利息分给小A，雷打不动，风雨无阻，哪怕小B一直亏损，老妈也不会从中插手。因此亲兄弟明算账，对于小B耍赖的担心就成了多余的顾虑。

当然凡事都有个度，那就是极端情况下小B一直亏到连内裤都没了，眼看快要吐血身亡时，老妈这才会母性大发，出手相救。所以只要不陷极端，小A就能旱涝保收。

但是，小B就完全不同了！

小B是个赤裸裸的好斗者，因此我们郑重提示大家，如果你闲钱不多，心理素质不强，那最好这辈子都别搭理小B！理由很简单——它肩扛助涨助跌的杠杆，手握锋利无比的双刃剑，跟它过招，那绝对是“狭路相逢勇者胜”！

所以，没有金刚钻，您可千万别那揽瓷器活。

变现容易吗?

除了风险性，流动性也是必须考虑的。

我们说，分级基金是一种创新型开放式基金，既不失开基的特性，又融入了封基的优点。怎么讲呢?

和开放式基金相比，封闭式基金最大的优点就是它可以在基金交易的二级市场，根据上蹿下跳的市场价格，随时在场内买入卖出（和使用股票软件进行交易一样）。但是普通的开放式基金就没这个待遇了，它只能在场外根据每天市场收盘后算出的净值，供投资者申购赎回。

而分级就是这样的基:

母鸡具有开放式基金的特性，只能场外申赎，不能场内买卖，所以流动性和普通开基毫无二致（当然2015年发行的很多分级基金，如上证50分级母基金也可以场内交易了）；

小A和小B则扮演封闭式基金的角色，只能场内买卖，不能场外申赎（在场外市场，也就是商业银行、众禄基金网、数米基金网、蚂蚁聚宝这些地方，你虽然能查到它们的信息，但无法购买）。因此，小A和小B就像买卖股票一样方便，变现能力自然也就比那些期限固定的债券、信托、银行理财、P2P和私募基金产品都要强，只要不出现涨跌停，资金便可任意进出。

收益怎么样?

首先说说交易成本。

众所周知，昂贵的交易成本将使收益大打折扣，同时让亏损变本加厉，但在小A和小B这里，事情显得令人欣慰。它俩既没有开基那样的申购费和赎回费，也没有股票交易那样的过户费和印花税，万二到万三左右的券商佣金就是其全部成本，何其厚道哇。

但绝对收益方面情况就不同了。

对“以不变应万变”的小A来说，简单的固定收益就足以支撑理想的家园。

你瞧，现在一年期定期储蓄利率才1.5%，余额宝收益也只有2.3%左右，货币基金也大都只在3%以内，银行理财品和国债一般也就3-4.5%，信托8%-10%确实高一点，但动不动就100万的起步价大都是高富帅的选择。而许多P2P的年收益虽然高于10%，但风险却是无法估量，跑路之事成年倍增，对于诸多本分的基友来说，到最后可能就是“玩P玩P，越玩越P”。

但如果把银子直接甩给小B，你每年定期到手至少一年期定存利率+3.5%-5%（即5%-6.5%，此外还有些固息分级产品已达到7%，并且如果考虑小A在场内交易可能存在的折价情况，收益还会更高，后续文章会讲到）的利息（数据截止到2016年8月），虽比上不足，但比下有余，老实人眼里这就不错啦（要啥自行车啊~）。

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率
150106	中小A	1.164	0.09%	77.70	1.0637	-9.43%	7.0%	7.00
150108	同辉100A	1.167	1.92%	2.78	1.0650	-9.58%	7.0%	7.00
150221	中航军A	1.237	0.41%	3305.92	1.0410	-18.83%	+5.0%	6.50
150321	煤炭A基	1.263	-0.16%	132.38	1.0460	-20.75%	+5.0%	6.50
150293	高铁A级	1.110	0.18%	25.12	1.0610	-4.62%	+4.0%	6.25
150331	网金融A	1.145	0.17%	1015.68	1.0426	-9.82%	+4.5%	6.00
150297	互联A级	1.105	0.09%	359.12	1.0679	-3.47%	+4.0%	6.00
150303	创业股A	1.081	0.46%	1025.98	1.0368	-4.26%	+4.0%	6.00
150219	健康A	1.208	0.33%	9.90	1.0380	-16.38%	+4.5%	6.00
150223	证券A级	1.217	0.33%	2471.70	1.0400	-17.02%	6.0%	6.00
150123	建信50A	1.223	0.08%	95.07	1.0375	-17.88%	+4.5%	6.00
150057	中小300A	1.179	-1.50%	4.37	1.0320	-14.24%	5.8%	5.80
150281	金融地A	1.080	-0.55%	166.82	1.0670	-1.22%	+3.5%	5.75
502014	一带一A	1.055	0.09%	468.65	1.0410	-1.34%	+3.5%	5.75
150295	改革A	1.089	0.18%	265.94	1.0636	-2.39%	+3.5%	5.75
150323	环保A端	1.076	0.47%	64.22	1.0338	-4.08%	+4.0%	5.50
150289	煤炭A级	1.080	0.19%	2552.86	1.0370	-4.15%	+4.0%	5.50
150335	军工股A	1.081	0.19%	201.55	1.0370	-4.24%	+4.0%	5.50
150287	钢铁A	1.081	0.46%	8506.47	1.0370	-4.24%	+4.0%	5.50
150263	1000A	1.082	0.46%	20.11	1.0368	-4.36%	+4.0%	5.50
150130	医药A	1.080	0.28%	19608.43	1.0339	-4.46%	+4.0%	5.50
150247	传媒A级	1.083	0.74%	211.25	1.0338	-4.76%	+4.0%	5.50

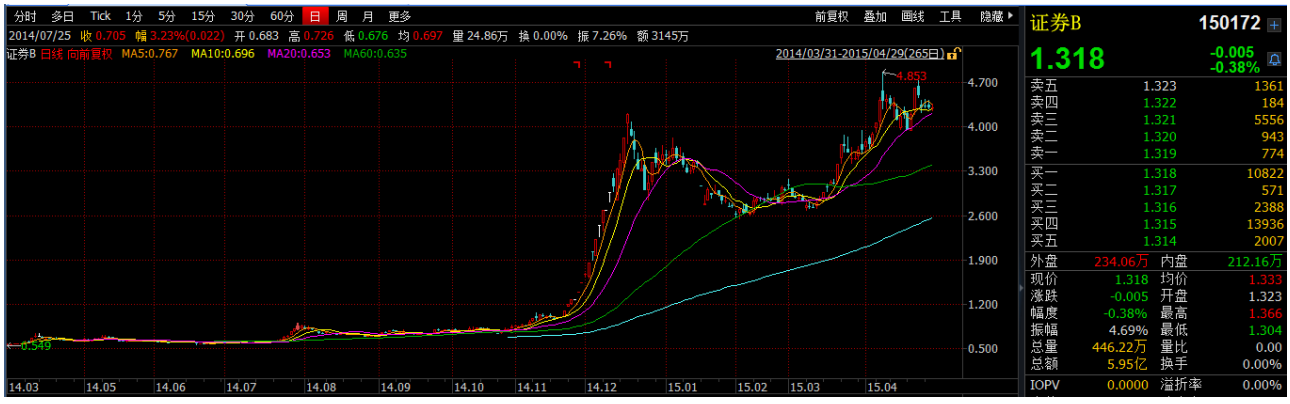
（图片来源：集思录网站）

而对于“赌徒”小B来说，在付完小A的固定利息之后，它享受剩下的所有收益或者承担剩余的全部损失。

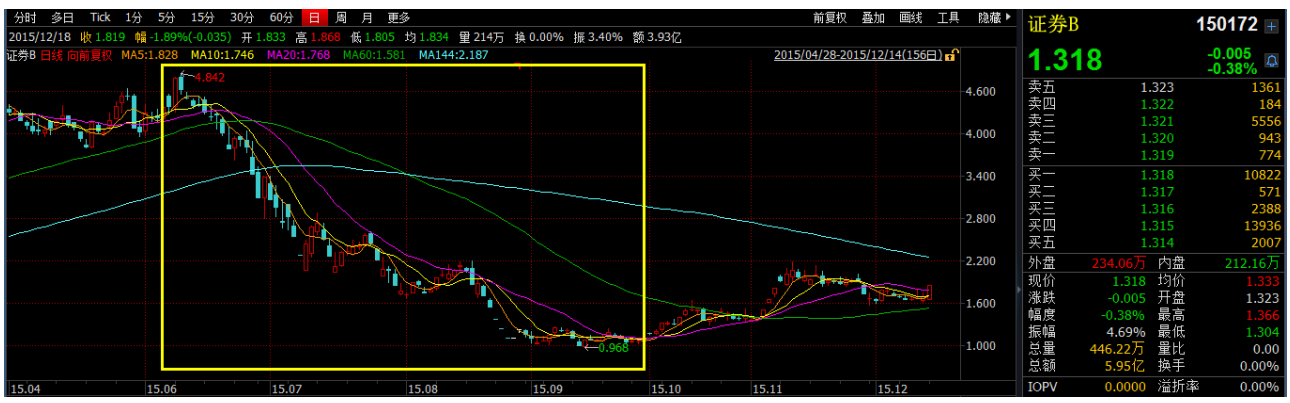
经过2014年底大涨行情的启蒙，2015年初到6月股灾之前各类行业指数分级小B成为了牛市行情的“投资大师”。由于具有突出的进攻性和杠杆性，小B的业绩往往能够跑在行业个股的前面，使得资金增幅可以媲美甚至超过行业牛股的收益。

但是去年股灾期间，小B的下折机制导致资金“亏成狗”的新闻也越来越多，很多人甚至把小B描绘成“亏钱神器”，让人哭笑不得。

因此，简单来说，若碰上牛市小B就赚大发了，若碰上熊市它就亏吐了，这是无法避免的硬伤。



（以波动最大的证券B为例，假如你在2014年4月买入证券B，两眼一闭啥都不管，恭喜你，一年以后你的收益竟然超过600%！好爽！）



（以波动最大的证券B为例，假如你在2016年6月买入证券B，两眼一闭啥都不管，恭喜你，三个月以后你的本金将亏损80%！好爽！）

因此，综上所述，对于小A，安全性和流动性这么高，牺牲一些超额收益也是可以接受的（至少比储蓄、银行理财品、货币基金、国债划算）；而对于小B，它同样可以大施拳脚，大大提高获取超额收益的机会。

其实，小B的做法并不新鲜，有一种叫做“融资融券”的券商业务跟它类似。

融资就是券商借钱给你买股票，让你能加杠杆做多；融券就是券商把自己的股票借给你卖出去，让你能加杠杆做空。

但很遗憾，目前分级市场上还没有推出基友们盼望已久的多空分级基金（A级B级多空份额对赌的分级基金）。尽管市场上呼吁了很多年（中欧基金曾在2013年申报过），但稳妥起见，监管层还没有对其放行。相信在中国金融市场相对健全的那一天，股市再次健康增长的时候，多空分级就会应运而生——心怀希望总归是好的。

分级基金适合投资小白吗?

这个问题问的好！但凡事都要辩证来看。

小A的操作相对简单，但是其中有个折价和溢价的情况，后续文章会加以详述。

小B方面嘛……我们常说没有比较，就没有结论，什么事情都要“相对而言”。

之前我们说过，目前小B的各类属性实际上就跟券商的融资业务非常类似，但相比之下，小B可能更“招人待见”。

从投资门槛上说，目前来看分级基金最低只需100元，而融资的门槛一般都要50万元，屌丝们闹不动哇！

（注： 2016年9月9日，交易所发布《上海证券交易所分级基金业务管理指引（征求意见稿）》，对于“设立投资者30万元证券类资产门槛”公开征求意见。对此我们深表遗憾，同时将会严正关切，并且保留进一步……修改文章的权利。然而，在政策尚未落地之当下，我们决定漠视其的存在，并不加以讨论。）

从融资成本上看，向券商融资的年息一般在8%-10%之间（比较常见的是8.35%）。而反观分级，小B借钱只需5%-6.5%的利息，这不高下立判了吗。

从融资风险上讲，差别更大，因为向券商融资，爆仓风险比较坑爹。怎么讲?

如果你拿自己的钱投资，亏就亏了，只要股票不退市，总有咸鱼翻身时。但是，这钱是券商借你的，你就必须看人家脸色。

举个栗子，你有100万本金，又向券商借了100万炒股，这时共有200万（2倍杠杆）。如果你的股票跌的剩下130万元时，总体只跌了35%，激进者还算能忍，但由于这100万是券商的，绝不能有亏损的危险，所以这时他们就会露出可憎的面目——要么追加资金，要么赶紧走（gun）人（dan）！

根据中国证监会2015年7月1日修订的《证券公司融资融券业务管理办法》的规定，自2015年7月6日起，公司融资融券业务因客户维持担保比例过低而强制平仓的相关规则调整如下：

若客户T日信用账户低于130%的平仓维持担保比例，客户须在T+1日收盘前补充担保物或了结融资融券交易，将维持担保比例提高至不低于150%的补仓维持担保比例。若客户T+1日信用账户维持担保比例低于150%但不低于130%，则公司暂缓对客户信用账户进行强制平仓，客户可自行操作信用账户（包括自行平仓等），但客户应保证，客户T+1日后的每一交易日信用账户的维持担保比例均不低于130%，否则，公司将在该维持担保比例低于130%的次一交易日对客户信用账户进行强制平仓，平仓后维持担保比例不低于150%。

维持担保比例不低于150%以后，公司对客户信用账户解除强制平仓状态。

以上维持担保比例的计算均以日终清算后为准。

如果你没有按时补足保证金，你的股票就要全部被券商卖掉，被强行平仓！更有甚者自己投入的所有本金都会灰飞烟灭，甚至倒欠一屁股债，辛辛苦苦几千年，一夜回到盛唐前。

融资融券被强行平仓，还倒欠证券公司3万块，被追债，我该怎么办？

楼主：注册马甲可真难 时间：2015-09-02 12:21:00 点击：5935 回复：54 脱水模式 给他打赏 只看楼主 阅读设置

8月26号最低点被强平掉的，还剩下一只大跌前就停牌的股票，前几天复牌，天天跌停，终于在27号打开的时候被证券公司强平了，60万资金灰飞烟灭。。。还倒欠3万，证券公司天天打电话让还钱，现在把同一账户下的B股账户资金冻结了，求高手指点现在的情况下应该怎么办？我只是想拿回B股里的钱，就这么还出去太不甘心了，最低点平的，说了会有资金进去的，还是被平掉了，这个损失不能我一个人承担啊。。现在着急上火得几天都没吃过东西了

而且你要记住一句话——成也杠杆，败也杠杆。你融资的杠杆越大，同等仓位下爆仓的可能性也就越大。

而小B虽然肩扛杠杆，有下折机制，但就算从1元发行价跌到只剩一成，也不会爆仓（当然，正常情况下这是不太可能的，后续文章将会详述）。

另外还有一个，就是选股这事儿太坑了！

截止目前，A股市场的股票数量已接近3000，因此对于普通股民而言，甄选股票的任务也就成了最让人头疼的事了。N多股民都有这样的感觉——明明当下大牛市，指数天天往上飙，可自己手里的股票就是丝毫不涨，不动如山，赚了指数不赚钱，这是何等的卧槽啊……

其实这很正常。历史统计显示，在每次大牛市中，总有超过50%的股票的综合收益跑不赢指数，也就是说作为散户，总会有一半以上的概率是买不到“盛世牛股”的，所以还不如选几支指数型基金中长期持有更有赚头儿（指数基金已经包含了某一市场中相对较优秀的股票，牛市中跑赢指数的概率大大增加）。

鉴于市场上大部分分级基金都是指数基金，又细分了行业，精选了成分股，那么牛市中购买小B，风险不就更小了吗？

当然，还有一种和分级基金类似的产品，就是去年异常火爆的结构化阳光私募基金。

这类私募基金也是一分为二——优先级和劣后级。前者和小A一样，相当于你买了个风险很低的固定收益产品；后者和小B一样，相当于借钱投资。在去年6月之前这轮牛市中，多少劣后级都上演了2倍、3倍甚至更高收益的好戏，甚是诱人哇！但很可惜，私募基金的投资门槛一般都在百万以上，一般人……还是洗洗睡吧……-_-!

最后不得不说的，就是“恶贯满盈”的场外配资。

去年6.12股灾之前，场外配资业务可谓是红红火火，如日中天，无数“韭菜”趋之若鹜，无所畏惧。但客观来说，虽然券商的两融业务风险不低，但毕竟还是正规的，而社会上的各种配资公司（尤其是很多P2P公司的股票配资业务），祸害“韭菜”的意图就太明显了。2015年“杠杆牛”最火爆的那段时间，监管机构就向市场和诸多机构明确提示了此配资业务的危险性，而后来A股的一落千丈，积重难返，场外配资绝对“功不可没”。

股票的场外配资罪状几何呢？

一是利息高得离谱，一般配资公司的年利在14%以下都算极其厚道的了，这又是何等的卧槽！

二是融资杠杆高得吓人，5倍，10倍甚至20倍杠杆都不在话下。唯利是图的他们希望你加的杠杆越高越好，因为高杠杆爆仓快，一旦爆仓，这帮饿狼就可以名正言顺地提前收回利息和本金，而你则会一贫如洗，体无完肤。

好在经过一轮股灾的沉淀和监管层的猛击，该业务几乎已经销声匿迹，不再祸害芸芸众生。

综上所述，经过认真比较和冥思苦想，我们发现，分级基金虽有风险，但其中优势也有目共睹，只要投资小白稍加研习，驾驭分级指日可待，所以我们得出结论——至少目前来看（重点强调），不管是直接投资还是作为资产配置的工具，分级基金，你都值得拥有！

关于分级基金的发展历程，以及驾驭它的终极奥义，我们以后详细分解。

文章更迭记录：
v1.0，2016-09-21，文章上线

【量化课堂】白话分级基金之二——成长的烦恼（上）

</br>

导语：
上一篇白话分级基金的文章中，我们用史上最通俗的语言揭开了分级基金的庐山真容。和很多事物一样，分级看上去很美，但成长历程却异常艰辛，极其复杂。古云：以史为鉴，可知兴替。因此，在讲述分级实操之前，我们以几个典型的分级基金为例，普及一下分级基金整体的进化历程和结构由来，以便大家在实践中举一反三，融会贯通。用《成长的烦恼》来命名本篇，应该再合适不过了。

</br>

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。
本篇包含大量的金融知识点，阅读前需先行学习《白话分级基金之一——一家“鸡”不说两家话》。

作者：$BaSO_4$
编辑：宏观经济算命师

</br>
传送门：《白话分级基金之一——一家“鸡”不说两家话》
</br>

</br>

分级基金的特征：标准复制、择时择势、高效流动；

分级基金的特性：行业配置、固收期权、平民杠杆；

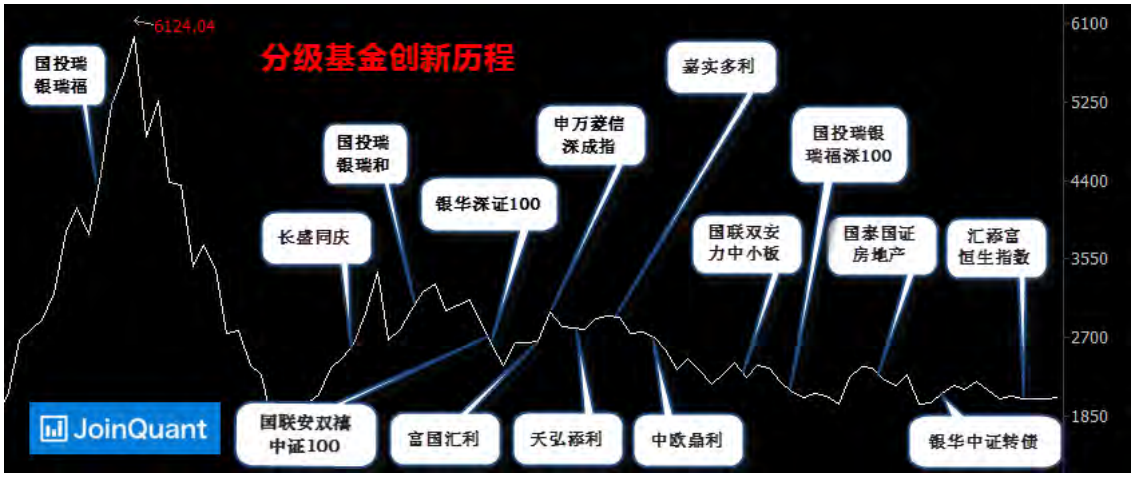
分级基金适应各类投资者，前景广阔，也是公募基金走向工具化发展的积极探索和创新。

——鹏华基金渠道业务部总经理 陈小荣

</br>

自2007年中国第一支分级基金破壳至今已经过去9年多的时间，这期间国内的各大基金公司和不少资管公司都对这一结构化工具不断升级创新，陆续创造出特点鲜明的新型产品，而分级的市场规模也顺理成章地从最初的60亿成功撞破5000亿大关（股灾之后市场规模大幅缩水），让人赞叹连连！

但众所周知，任何事物的发展壮大都要经历螺旋式上升、波浪式前行的艰苦历程，分级基金的成长轨迹充分印证了这一规律。



</br>

（参考来源：集思录）

一、初入江湖（国投瑞银瑞福）

鲁迅先生曾经称赞：“第一个吃螃蟹的人很令人佩服，不是勇士谁敢去吃它呢？”这话的确没错。

螃蟹形状可怕，丑陋凶横，第一个吃螃蟹的人确实需要过人的勇气，而国投瑞银基金公司正是这种鲁迅口中赞不绝口的勇士。

2007年7月17日，在国投瑞银的力促之下，中国第一支主动投资型分级基金默默诞生，它的名字是——国投瑞银瑞福分级基金。

之所以用“默默”一词，是因为那轮史无前例的大牛市中，如日中天的ETF基金才是基金市场的第一主角儿，它红遍大江南北，名扬五湖四海，无人不知，无人不晓。而分级基金在那时却只是一个初出茅庐的新生儿，一个无人问津的“小屁孩儿”，鲜有关注实属正常。

作为第一只分级基金，瑞福分级的结构特点和运作模式大致如下：

1.主动管理型基金；

2.基金分为稳健型的【瑞福优先】和进取型的【瑞福进取】，初始份额比例为5:5;

3.【瑞福优先】和【瑞福进取】的资金分别募集，资产合并管理；

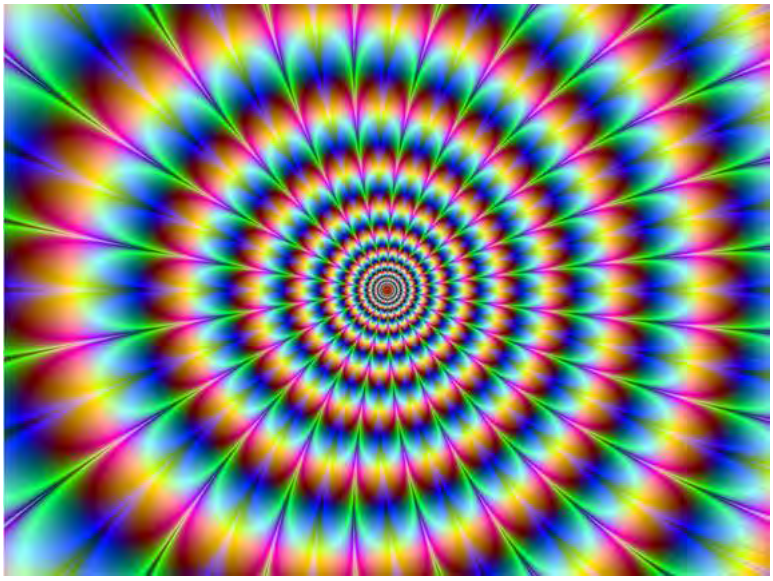
4.【瑞福优先】通过银行渠道销售，每年开放一次，年化收益6%；

5.【瑞福进取】通过券商渠道销售，在深交所挂牌上市交易，封闭式，封闭期为5年；

6.基金有一套特有估值模式和分红策略，具体情况如下两图：

作为分级元老，瑞福虽然开天辟地，但也必然有着先天缺陷，其中几处硬伤不容忽视：

第一，瑞福为主动管理型基金，对基金经理的操作水平要求甚高，较指数跟随型基金风险更大。同时基友们不知道基金经理每天的买卖情况，净值变动也就不像指数型基金那样容易预估。加之其本身一系列繁琐和抽象的估值方法更是让“孩子他爹一头雾水”（如上两图所示），大家计算每天的真实净值就难上加难（如“虚拟清算”之类）。因此，面对这种不接地气的新事物，不明真相的普通群众望而却步也是情理之中了。（9年多过去了，就算许多专业人士回头望月，也仍然感到头晕目眩。-_-!）

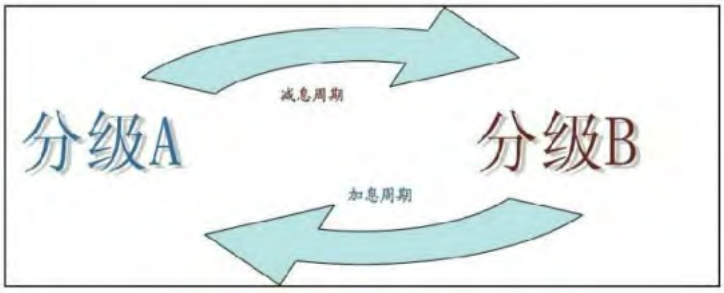


再者，【瑞福优先】每年获得的约定收益率是固定不变的6%，当市场的无风险收益水平随着经济周期的变化而不断变化时，这种设计就会出现问题，并会殃及【瑞福进取】。

逻辑如下：

A．如果央行进入紧缩阶段，以余额宝（货币基金）为代表的市场收益水平就会整体上升，而【瑞福优先】的基友依然只能获得6%的回报，还要承担各种高昂的交易成本，吸引力大大下降，导致优先端可能被大量赎回，资金净流出；同时，由于【瑞福进取】具有杠杆性质，其杠杆也会随着优先端的“失血”而降低，造成进取端对于逐利资本同样“失宠”。

B．相反，如果央行进入宽松周期，整个投资市场的收益水平相应下降（比如余额宝收益只有2%了），这对【瑞福优先】的基友肯定是好事，但买了【瑞福进取】的人就惨了，进取端不但赚不到市场带来的超额收益，还要付出高达6%的融资成本以及其它交易成本，这显然不公平。所以优先端的收益率应当跟随央行利率变动而动态变化，似乎才更加合理。



第三，【瑞福进取】有5年的封闭期（对于中国投资者来说简直是坐牢哇），导致资金流动性严重不足，同时也限制了其规模扩张（后文会用典型案例加以说明）。

综上所述，瑞福分级的结局大家也可想而知了——虽然名气不小，但却有价无市，市场的接受程度较之预期相去甚远，于是在2012年，国投瑞银瑞福分级基金就正式转型成为国投瑞银瑞福深证100 指数分级基金，就是后来市场上最主流的那种分级产品了。

二、二次试水（长盛同庆和国投瑞银瑞和）

2008年，A股暴跌，普天同悲，分级基金的创新脚步也就此暂停。但到了2009年，股市出现了一次大反弹，第二代分级基金借势而生。

</br>

1.长盛同庆可分离交易股票型基金

</br>

长盛同庆出生于2009年5月12日，是瑞福分级的改良版，虽然是如法炮制，但分级模式却简单很多。

与瑞福分级不同的是，长盛同庆是封闭期3年的封闭式基金，运作期满自动转换为上市开放式基金（LOF）。为保证结构化份额符合预订比例，基金采取合并募集，后按4:6的比例拆分为【同庆A】和【同庆B】（即基友们每认购10份长盛同庆基金，将得到4份小A加6份小B，适当降低了后者的杠杆风险），前者每年5.6%的约定收益率，后者获取1.67倍的资金杠杆，这就和今天市场上主流分级基金的相貌很接近了。但还是有几个软肋比较要命：

第一，长盛同庆也是主动投资型基金的身份，这也造成了它和瑞福相似的尴尬；

第二，【同庆A】和【瑞福优先】一样，是固定收益，前文分析的AB逻辑问题凸显无疑；

第三，这个基金还是一个封闭式基金，3年闭关导致“血脉不通”，流动性受阻。

这里重点聊聊流动性的问题。很多时候，流动性不足可能会“秋后算账”要了广大基金的命，长盛同庆就是其中一个苦命的娃。

从2009年5月到2012年5月的3年时间里，A股上蹿下跳，加大了基金经理的操作难度，而且到最后依然是那个没出息的熊样，因此加了杠杆的【同庆B】不出意料亏得体无完肤。悲伤的基友们在封闭期内无计可施，只能憋着大招等待爆豆，结果封转开以后，长盛同庆的持有者以“洪荒之力”大规模赎回基金份额，“同庆”在几天时间里变成“同悲”，缩水大半，基金经理哭成了狗……

所以经过两轮试水，大家发现分级基金不能再玩封闭了，这绝对是弊大于利的事。如果一上市就开放，母基金可以随时申赎，小A和小B可以随意买卖，那长盛同庆式的悲剧是不是就可以避免了呢?

当然，封转开以后，长盛同庆摇身一变，成了长盛同庆中证800指数分级基金，简称同庆800，也和今天的主流分级基金相差不远了。

</br>

2.国投瑞银瑞和沪深300指数分级基金

</br>

对，又是国投瑞银！我们常说在哪跌倒在哪爬起，国投瑞银不但爬起来了，还跑起来了！

作为敢于第一个吃螃蟹的人，又有了自己和长盛的前车之鉴，国投瑞银基金终于也在2009年10月，发行了自己的第二个分级基金——国投瑞银瑞和沪深300指数分级基金。

和长盛同庆相比，瑞和沪深300做了三处重大升级，让人惊喜。

惊喜一：封闭变开放，存续期限改为“不定期”，理论上大大改善流动性；

惊喜二：变主动管理为被动跟随，跟踪基准为沪深300指数。如此一来，普通基友就能根据当天沪深300指数的走势大致估算出基金的实际净值了，大家不再两眼一抹黑，就更愿意对你投怀送抱；

惊喜三：打通了母基金和子基金（【瑞和小康】和【瑞和远见】）之间的配对转换机制。这是分级史上破天荒的重大创新，意味着在任何一个交易日，基友们都可以申请把母基金拆分成子基金，也可以申请把子基金合并成母基金。

这有什么实际意义呢？对！传说中的双向套利！

（双向套利为分级基金的玩法增添了无穷的色彩，一直为众多分级基友和金融机构所津津乐道。但由于这其中涉及到了太多关于价格、净值、溢价、折价、套利规则方面的知识点，所以我们只能以“篇幅有限”为由，在后续文章中详细描述。）

然而，创新归创新，突破归突破，再好的东西也可能遭遇生不逢时的遗憾。由于长盛同庆的阴影始终挥之不去，瑞和沪深300也跟着吃了瓜落儿，最终的发行并不成功，但是指数化投资策略和配对转换机制为下一代分级基金留下了宝贵的经验。
到了2010年，集长盛同庆和瑞和沪深300的优点于一身的第三代分级基金诞生了。

三、新益求新（国联安双禧）

长盛同庆和瑞和沪深300这两个第二代分级基金都在第一代分级基金国投瑞银瑞福的基础上做了不少改良和创新，但都还是有各自的缺点。所以到了2010年4月，融合了两代前辈优点的第三代分级基金出现了，它的名字叫做“国联安双禧中证100指数分级基金”。

它有这么几个主要优点：

1.被动跟随型分级基金，跟踪中证100指数；

2.无封闭期，一上市就和普通的开放式基金一样随时可以申购赎回；

3.【双禧A】每年的约定收益是一年期定期存款利率+3.5%，解决了前两代AB份额共同的软肋；

4.保留了瑞和沪深300可以配对转换套利的机制，大大激活了资金参与投机的冲动和热情；

5.更重要的是，国联安双禧开创了基金份额的“向下不定期折算”机制（简称下折机制），让极端情况的风险得到一定释放。

下折机制是什么鬼？

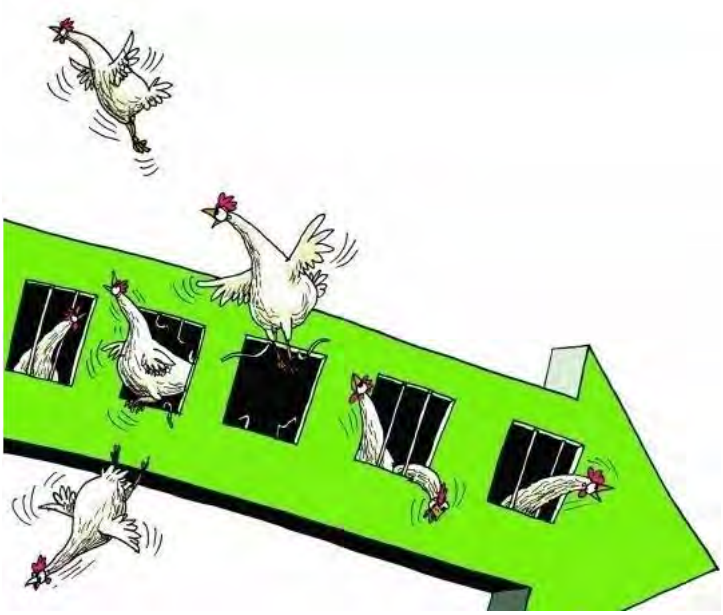
上篇文章中我们讲过，正常情况下小A不用担心小B耍赖，后者会在每年约定时间给予小A利息，这叫定期折算机制。但碰见一种情况就完蛋了——小B遭遇股灾之类（比如2016年6月12日以后那种情况）亏成了狗，眼看就要破产失去翻本机会。这时母鸡就会顾全大局，暂停小B的常规操作并采取补救措施，避免大家都陷入颗粒无收的尴尬境地。

这个补救措施就是国联安双禧就想出的办法——基金份额下折机制。

其作用主要在于：

1.保证小A净值的安全。当小B的净值跌破0.15元的阈值，整个国联安双禧基金的三类份额——母基金、双禧A和双禧B就会同时启动下折程序。不管这三类份额原来净值几何，下折后它们都会重新回归1元的初始净值，小A多余的份额将会被其持有人以母基金份额的形式获得，投资者可以赎回兑现这部分收益，而小B的份额将按照一定比例缩减，价值不变。

2.调整小B的杠杆倍数以降低风险。随着小B净值的下跌，其杠杆倍数快速放大，风险放大。经过下折，小B恢复为初始杠杆倍数，其风险也随之降低，起到悬崖勒马的作用。



做了这样破天荒的重大升级，国联安基金理所当然自信爆棚，但当其沾沾自喜准备分享市场蛋糕之时，仅仅过了一个月，银华基金又横空出世，在国联安双禧的基础上推出了最具里程碑意义的第四代分级基金——银华深证100指数分级基金。

预知后事如何，且听下回分解……

文章更迭记录：
v1.0，2016-10-13，文章上线

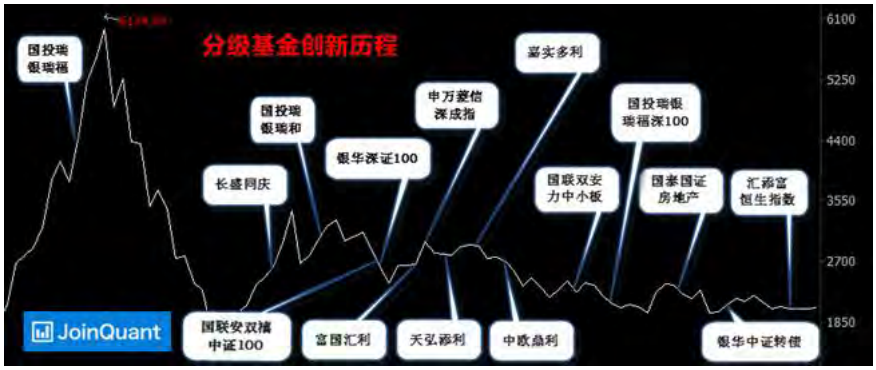
【量化课堂】白话分级基金之二——成长的烦恼（下）

导语：
在《白话分级基金之二——成长的烦恼 （上）》中，我们详述了前三代分级基金艰苦卓绝的成长历程。就如同“西天取经多磨难”一般，分级发展之路也充满了荆棘，厚重和深沉。在后来的几年之中，分级的江湖又上演了怎样的血雨腥风、恩怨情仇，请看《白话分级基金之二——成长的烦恼（下）》。

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。
本篇包含大量的金融知识点，阅读前需先行学习：
1.《白话分级基金之一—一家“鸡”不说两家话》
2.《白话分级基金之二—成长的烦恼（上）》。

作者：$BaSO_4$
编辑：宏观经济算命师

传送门1：《白话分级基金之一——一家“鸡”不说两家话》
传送门2：《白话分级基金之二——成长的烦恼（上）》



分级基金在中国证券市场里就像荒野中的那朵野玫瑰，她神秘而独特的魅力只有愿意探索和发现的投资者才能真正体会。——集思录创始人 周天舒

上文我们说到，国联安双禧中证100指数分级基金做了破天荒的重大升级（尤其是下折机制），但当其踌躇满志地准备分享市场蛋糕之时，仅仅过了一个月（2010年5月），银华基金又横空出世，推出了里程碑意义的第四代分级基金——银华深证100指数分级基金。

这位大神在中国分级史上可是非常了得！它练就了何种神功呢？请往下看——

四、精益求精（银华深证100）

银华的深证100指数分级基金在国联安双禧的基础上主要做了两处改动：

1.把【银华稳进】和【银华锐进】的AB份额比例从双禧的4:6变为5:5，这也是今天大多数分级基金的标准配比。

这么做有两个好处：

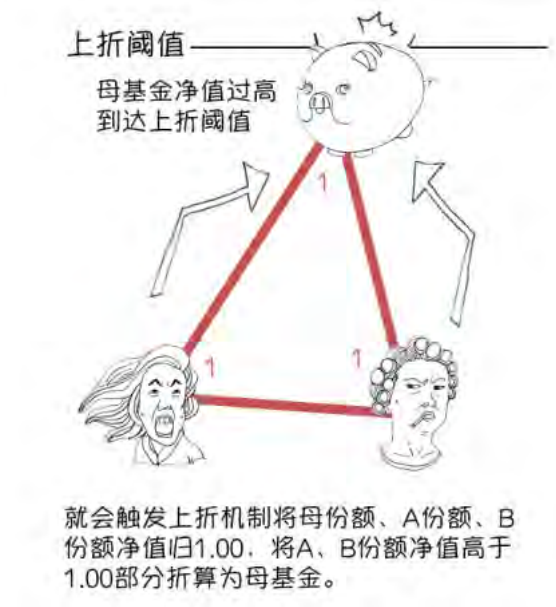
A.初始杠杆比例从原来的1.667倍提高到了2倍，杠杆更大，分级B的吸引力就会更大。

B.配对转换套利的资金分配更容易计算了，只要1:1等权配比AB基金就可以了。

2.不但设置了小B净值跌破0.25元的下折程序，还开创了母基金净值涨到2元以后启动向上不定期折算（上折）的机制。上折也是重新把三类份额的净值回归到1元（小A和小B净值超过1的部分都折算为母基金的份额），只不过相比下折时基金份额的缩水，上折会使母基金、小A和小B的份额都有所增加。

为什么一定要设置上折机制呢？

当母基金净值涨到2元时，分级B的杠杆就从2倍降低到了约1.33倍，杠杆优势逐步消失，对于投机资金的吸引力也就消失了。通过上折，分级B恢复到了初始2 倍杠杆，基友们欢天喜地，其交易功能也就能发挥出来了。



这两个改动其实并不算惊艳，但正是这看似不惊艳的上折，让分级基金这支航天飞船的进化得以升华——银华深证100指数分级基金在其后几年间大放异彩，傲视群雄，最高规模达到200多亿元，甚至在某一段时间，它占领了分级市场75%以上的规模，是当之无愧的分级老大。所以，如果你听到“在分级发展史上，银华深证100永远是不可错过的神话！”这样的评论，请一定不要惊讶。

但凡事都有个例外。有句广告词叫“不是所有的牛奶都叫特仑苏”，同样，也不是所有的分级基金都有上下折，比如奇葩的150023——申万菱信深成指B。它就不信“上下折”这个邪，而是刀走偏锋采用了另一种奇葩的手段——当小B净值跌破0.1元时，小A就被无情地拉下水，跟小B同涨同跌！也就是说，母基金、

小A和小B之间的净值涨跌幅同步了，这个分级基金暂时变成了普通的指数基金。这也算是分级史上的一个小插曲，如果大家感兴趣，请自行查阅资料，此处不做赘述。

（图片来源：万得）

五、隐忍前行

1.政策引导

我们常说树大招风，这话一点不假。随着分级基金的队伍逐渐庞大，各种问题也慢慢浮出水面。2011 年 12 月，证监会基金部出台了《分级基金产品审核指引》，首次正式从产品设计、募集方式、最大杠杆倍数、最低认购金额等方面对分级基金进行规范。

指引规定明确指出：

股票型分级基金采用合并募集并分拆的方式，B类份额初始杠杆不超过2倍，净值杠杆不超过6倍；

债券类分级基金采用分开募集资金的方式，B类份额初始杠杆不得超过10/3倍，净值杠杆不超过8倍。

另外，《指引》还规定了分级基金单笔认/申购金额不得低于5万元的限制，初步设置了投资门槛。

2.银华风波

很不幸，政策出台还不到一年的时间，银华中证等权90分级基金就成为历史上第一个触发下折机制的基金（2012年8月31日）。

自银华中证90成立之日起（2011年3月17日），A股就始终没有逃离熊市的阴霾，所以被动跟随型的银华中证90的小B一直处于跌势之中。但是有了之前银华深证100的成功，加上分级杠杆的吸引力，以及市场中“越跌越买”之言论误导的原因等等，许多激进型投资者就坚定地判断它而会绝地反击，起死回生，不会下折，于是不顾一切跳进火海，渴望赚到超额收益，结果导致小B的溢价非常之高。

但下折是按照基金净值而不是市场价来计算的，所以只要遇到极端行情下折程序被启动，那一夜之间那些高溢价买入小B的赌徒就可能亏到只剩内裤（后续文章会加以详述）。

最后，墨菲定律应验了，这群赌徒失败了。

亏成了狗，心里自然不爽，就开始在网上驴唇不对马嘴地抨击银华中证90有设计缺陷。他们认为，上下折的阈值极不合理，理由很简单：高溢价的风险我们认了！但是，当小B的0.25元下折机制被触发时，实际上下跌了75%（母基金的净值是0.625元，才跌了37.5%），但上折启动的条件，却是小B必须上涨200%（母基金从1元涨到2元，涨幅100%）。下折容易上折难，挨打容易吃肉难，这也太不公平了吧！

这一系列的吐槽看似只是无端发泄，但仔细想一想这上下折的制度确实有失公平，所以到了2013年，信诚沪深300分级基金率先修改合同，把触发上折时母基金的净值从过去的2元修改为1.5元，即小B只要上涨100%（母基金只要涨50%），上折便如约启动。

这样的改良引发了市场的无数共鸣，于是此后新发行的分级基金，大多都把上折的母基金净值设置为1.5元。

3.调息插曲

另外还有一段小插曲，就是小A的约定收益率一直在提升。

当年很多小A产品的约定收益率大都是一年期定存利率+3%左右，但以银华为代表的基金公司在发行产品之后发现了一个问题：无论股市牛熊，小A总是折价，小B总是溢价，这不科学啊！

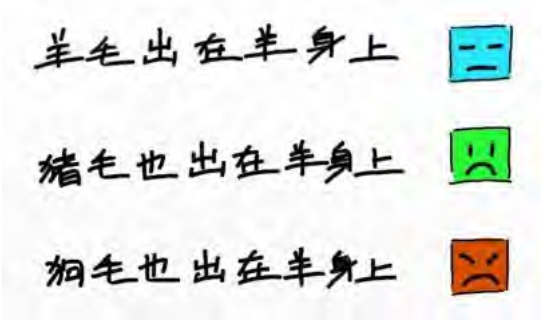
经过调研，基金公司发现，反科学的现象是因为小A的约定收益率太低，受到市场冷落了！于是基金大佬们迅速调整三观，竞相提升小A的约定利率。后来这成了各个公司的竞争常态，以至于2015年成立的富国中证煤炭和前海开源中航军工指数分级基金的A类份额约定收益率，已经达到了一年期定存利率+5%，也就是6.5%之高。（此外还有一些固定利息品种，如中小A和同辉100A，目前收益率为7%）

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率
150106	中小A	1.164	0.09%	77.70	1.0637	-9.43%	7.0%	7.00
150108	同辉100A	1.167	1.92%	2.78	1.0650	-9.58%	7.0%	7.00
150221	中航军A	1.237	0.41%	3305.92	1.0410	-18.83%	+5.0%	6.50
150321	煤炭A基	1.263	-0.16%	132.38	1.0460	-20.75%	+5.0%	6.50
150293	高铁A级	1.110	0.18%	25.12	1.0610	-4.62%	+4.0%	6.25
150331	网金融A	1.145	0.17%	1015.68	1.0426	-9.82%	+4.5%	6.00
150297	互联A级	1.105	0.09%	359.12	1.0679	-3.47%	+4.0%	6.00
150303	创业股A	1.081	0.46%	1025.98	1.0368	-4.26%	+4.0%	6.00
150219	健康A	1.208	0.33%	9.90	1.0380	-16.38%	+4.5%	6.00
150223	证券A级	1.217	0.33%	2471.70	1.0400	-17.02%	6.0%	6.00
150123	建信50A	1.223	0.08%	95.07	1.0375	-17.88%	+4.5%	6.00
150057	中小300A	1.179	-1.50%	4.37	1.0320	-14.24%	5.8%	5.80
150281	金融地A	1.080	-0.55%	166.82	1.0670	-1.22%	+3.5%	5.75
502014	一带一A	1.055	0.09%	468.65	1.0410	-1.34%	+3.5%	5.75
150295	改革A	1.089	0.18%	265.94	1.0636	-2.39%	+3.5%	5.75
150323	环保A鹏	1.076	0.47%	64.22	1.0338	-4.08%	+4.0%	5.50
150289	煤炭A级	1.080	0.19%	2552.86	1.0370	-4.15%	+4.0%	5.50
150335	军工股A	1.081	0.19%	201.55	1.0370	-4.24%	+4.0%	5.50
150287	钢铁A	1.081	0.46%	8506.47	1.0370	-4.24%	+4.0%	5.50
150263	1000A	1.082	0.46%	20.11	1.0368	-4.36%	+4.0%	5.50
150130	医药A	1.080	0.28%	19608.43	1.0339	-4.46%	+4.0%	5.50
150247	传媒A级	1.083	0.74%	211.25	1.0338	-4.76%	+4.0%	5.50

（数据来源：集思录）

然而，基金公司频繁升息到底为哪般？难道是钱赚嗨了要回报投资者？并不是……

你琢磨啊，一方面约定收益率提高会加大小A的吸引力，引发其更容易在二级市场出现溢价，另一方面小B因为杠杆的关系本身就特招人待见容易溢价（尤其在牛市中），因此分级基金就顺理成章地更容易出现整体溢价，吸引套利资金蜂拥而至。然而，之前我们讲过，溢价套利首先要去申购母基金，这一环节至少要交1%的申购费，于是事情就明晰了——你想溢价套利，先得掉一层皮。因此，羊毛出在羊身上，这是亘古不变的真理。



六、兴衰之叹

1.扶摇直上

正如广大人民群众料想的那样，分级基金并没有因为银华中证90那次小小的挫折而停下进取的脚步。

在那之后的日子里，各大基金公司相继推出市场主流的分级品种，将蛋糕越做越大。尤其是2014下半年新一轮牛市开启的分级盛宴，以中欧盛世成长分级基金为代表的多位“杰出青年”借势纷纷上折（2014年8月22日中欧盛世成长成为中国基金业史上首只向上折算的分级基金），拉开了分级扩张的序幕，这让无数的投资者加入到了追逐分级基金（分级B）的基友大军之中。

一方面，在这种历史背景下，除了国投瑞银、长盛、国联安、银华这些分级领域起步较早的基金公司以外，鹏华、申万菱信、国泰也都加快了大规模扩容的步伐，加上易方达、南方基金这种传统大牛后来崛起，分级市场呈现出一片群雄并起逐鹿中原的繁荣景象。

另一方面，分级基金市场从过去把深证100、沪深300、中证500等普通成分指数作为热点，到后来把兴趣逐渐转变为诸如地产分级、医药分级、证券分级、军工分级、白酒分级、一带一路分级等热门行业指数，再到后来投资海外股市的QD分级、投资债券的纯债型、一级债、二级债分级、以及能够上市交易的母基金的出现，可以说，分级的进取脚步一直都没有停歇。截止2015年6月30日，分级基金规模已突破5000亿元大关，超越了公募基金整体的同期发展速度将近两倍，让人叹为观止。

分级基金概况一览

基金类型	2014年（全年）		2015年（上半年）		变动情况	
	基金规模（亿元）	基金数量	基金规模（亿元）	基金数量	数量变动	规模变动
指数型	1460.60	58	4535.24	113	55	3074.64
纯债型	216.93	29	187.68	27	-2	-29.25
一级债	182.41	22	171.64	15	-7	-10.77
股票型	46.93	5	30.33	4	-1	-16.6
二级债	27.30	5	63.07	6	1	35.77
QDII	13.35	3	182.50	3	0	169.14
总计	1947.52	122	5170.45	168	46	3222.93

数据来源：济安金信基金评价中心　　截止日期：2015年6月30日　　JoinQuant

2.卧薪尝胆

然而，美好的事物往往都很短暂，随着去年6.12之后牛市的陨落，当年7月，分级基金不出意料地惊现“下折风波”，分级基金的火爆告一段落——

2015年6月26日，银华中证转债增强分级基金B级份额净值跌幅超50%，成为上轮牛市首只下折的分级基金。随后，市场纷纷上演“下折潮”，许多基友一夜之间的亏损过半。

紧接着就出现了投资者维权事件，像“无良分级B基金坑我血汗钱”之类的讨伐之声屡见不鲜，一时间分级基金成了洪水猛兽。
2015年8月21日，监管层暂缓分级基金的审批，分基遭遇冷落；

今年8月19日，监管层表示，考虑到分级基金比较复杂，投资者难以理解，证监会决定暂停分级基金的注册工作；

随后的9月9日，交易所发布《上海证券交易所分级基金业务管理指引（征求意见稿）》，对于“设立投资者30万元证券类资产门槛”公开征求意见，分级基金（可能）就此步入散户时代的霜冻期……

3.好事多磨

说到这里，我们突然意识到，此情此景和去年9月股指期货被错杀的情形是何其相似啊，这不禁让人想到了狄更斯《双城记》中的那段醒世恒言——

时之圣者也，时之凶者也；

此亦蒙昧世，此亦智慧世；

此亦光明时节，此亦黯淡时节；

此亦笃信之年，此亦大惑之年；

此亦多丽之阳春，此亦绝念之穷冬；

人或万物具备，人或一事无成；

我辈其青云直上，我辈其黄泉永坠。

在我们的记忆中，公募基金风风雨雨十八载，这似乎是监管层第一次如此重视地去干预某一类细分领域的金融产品。从一开始曲高和寡的小众玩物，到后来的飞入寻常百姓家的大众产品，分级基金艰难地走到了第十个年头。回顾这十年，分级承受了股市大跌与杠杆“原罪”之痛，但也表现出了不屈不挠的战斗力、激荡力和想象力，这也一度成为了金融行业最亮眼的标志，成为了这个时代金融创新的动力源泉。

我们更愿意相信，监管层的步步决策，纯粹是对于市场的保护，这种保护虽然有些“粗鲁”，却也着实急切与真挚。但就本质而言，分级基金并无对错，它本身或许只是中国金融市场混沌时代的又一只替罪羊而已。因此，我们迫切地希望，分级市场再一次“面朝大海，春暖花开”的那个“最好的时代”，能够早日到来。

拓展阅读：《一位分级爱好者致交易所的信：分级基金征求意见稿》
http://fund.jrj.com.cn/2016/09/18101021490632.shtml

文章更迭记录：
v1.0，2016-11-10，文章上线

【量化课堂】白话分级基金之三——让小A飞

导语：

“人生本无定数，回首已是天涯，但五味杂陈的烈酒，总好过温吞水一杯吧。”带着这样的感慨，我们从上一篇的分级基金成长史中脱离思绪，步入实战部分。今天，我们就从驾驭分级A开始。

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。

本篇包含大量的金融知识点，需先行学习以下两篇文章：

1.《白话分级基金之一——一家“鸡”不说两家话》

2.《白话分级基金之二——成长的烦恼》

作者：$BaSO_4$

编辑：肖睿

传送门1：《白话分级基金之一——一家“鸡”不说两家话》

传送门2：《白话分级基金之二——成长的烦恼（上）》

传送门3：《白话分级基金之二——成长的烦恼（下）》

指数分级的本质是市场经济的选择，也是金融市场发展中指数化投资的必经之路，已经成为普通投资者转变为专业投资者的重要工具之一。 ——鹏华基金量化及衍生品部总经理 王咏辉

一、小A的魅力所在

我们常说小A特别适合理财小白和稳健型基友，理由有三：

1.投资属性

兼具安全性和流动性，同时其约定收益率优于银行定存、余额宝、货币基金、国债和银行理财品，吸引力明显。

2.结算机制

分级基金每天净值结算时，规定先对小A的净值进行处理，优先保证其每日约定收益，并会以定期折算（通常在12月或1月）的形式通过母基金发放给基友，这种结算机制进一步降低了小A的风险系数。

小A的净值收益=1.00+（已计息天数/应计息天数）*约定收益率
（已计息天数为距离上次定期折算或不定期折算的自然天数，应计息天数为当年总天数，一般为365）

3.可操作性

并不是说固定收益，小A就没得玩了，搞好了也能高回报。比如去年股灾期间，沪指从5178点跌到3357点，跌幅高达35%，但就是有民间高手规避了风险，在小A身上薅到了20%多利润。所以，玩法的多样性和灵活性也增加了其投资魅力。

二、小A的收益率谜团

有人说收益率有啥可讲的，闭着眼睛买几个固定利息最高的长期拿着不就成了。

然而，事实还真没那么简单，先来普及几个概念——

1.净值 & 价格

母基、小A、小B都有两个标价，基金净值（简称“净值”）和交易价格（简称“价格”）。

净值，即基金申购和赎回时的参考数值，是基金背后对应的真实资产价值。

价格，即二级市场的场内投资者对该基金的激烈博弈和重新定价，跟买卖股票道理一样。换句话说，就是“你认为”这个资产应该值多少钱。

在申购分级基金时，母基、小A、小B的净值和价格的关系是：

A净值+B净值=2*母基金净值≈A价格+B价格

净值方面：

新发行的基金（母基）净值是1元，拆分后小A和小B的净值也各为1元。

假如小A净值变成1.05元，小B净值0.95元，根据上述公式，母基净值还是1元（1.05+0.95=2；2/2=1）；如果母基净值涨到1.5元，小A还是1.05元，小B净值就会大幅飙到1.95元，涨了将近一倍（1.5*2=3；3-1.05=1.95）！所以母基和俩儿子之间的净值关系永远对等。

指数型分级母基和小B（在杠杆作用下）的净值根据所跟踪的指数涨跌来决定，而小A的净值则是根据每天小B支付的利息来决定，这都没问题。但是，正因为有了价格，事情复杂了。

价格方面：

价格是二级市场资金博弈产生的，所以波动可能较大，常常偏离净值（偏离较大时会有套利行为在较短时间使价格回归平衡，所以价格与母基净值是约等于的关系），价格＜净值叫折价，价格＞净值叫溢价。

若溢价买入，实际回报就会<约定回报（东西买贵了，亏了）；

若折价买入，实际回报就会>约定回报（打折买东西，赚了）。

2.约定收益率 & 实际收益率

比如，中航军A和煤炭A基，因为约定收益率6.5%高出别人一截，就会像美女一样在二级市场受人追捧供不应求导致溢价，实际收益率（隐含收益率）就会下降而低于6.5%。反之，供大于求导致折价，实际收益率就会提高，如深成指A和互利A。

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率
150221	中航军A	1.259	-0.32%	6279.97	1.0550	-19.34%	+5.0%	6.50
150321	煤炭A基	1.289	-0.23%	202.45	1.0610	-21.49%	+5.0%	6.50

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率
150022	深成指A	0.863	0.47%	10169.99	1.0384	16.89%	+3.0%	4.50
150066	互利A	0.922	0.11%	6.72	1.0250	10.05%	+1.5%	3.00

（数据来源：集思录 2016-11-8）

因此所见未必即所得，约定利息高的不一定真划算，还需一番计算：

小A实际收益率（年化）=小A约定收益率（年化）/（小A价格-（小A净值-1））

试着用中航军A和深成指A套用公式：

中航军A实际收益率（年化）=6.5%/（1.259-（1.055-1））≈5.399%

深成指A实际收益率（年化）=4.5%/（0.863-（1.0384-1））≈5.457%

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率	下期利率	修正收益率
150221	中航军A	1.259	-0.32%	13861.54	1.0550	-19.34%	+5.0%	6.50	6.50	5.399%
150022	深成指A	0.863	0.47%	10169.99	1.0384	16.89%	+3.0%	4.50	4.50	5.457%

（数据来源：集思录 2016-11-8）

上图的“修正收益率”即“实际收益率（年化）”。结果显而易见，前者的约定收益率虽然比后者高出不少，貌似很有吸引力，但考虑折溢价情况，后者的实际收益率反而占据上风，印证了刚才的说法。

3.集思录 & JoinQuant 聚宽

说到这里，你就明白了，要选实际收益率高的。但很多基友头疼了——全市场100多只小A，要是按实时变动的折溢价自己计算实际收益率，那还不疯X！别怕，有一个捷径助你发家致富：

首先，打开一个叫“集思录”的分级基金页面（www.jisilu.cn/data/sfnew/#tlink_3）。这里详列了分级的各类信息，包括小A的实时价格、每日净值、约定收益、实际收益等，多种数据，任君挑选。

然后，打开JoinQuant聚宽的网页（www.joinquant.com），结合自己的交易思想，做研究，写策略！Look，分级的人生多美妙——

（网址：https://www.joinquant.com/post/dd62a1059e97bf604e6a8fd358a5fd26）

（网址：https://www.joinquant.com/post/3132?tag=new）

三、小A的实战玩法

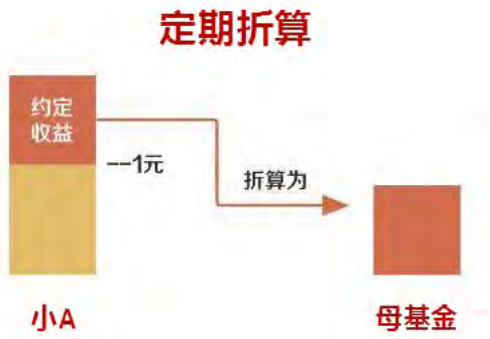
重点来了，我们总结了三套玩法，都很实用——

1.配置型玩法：

前篇说过，供求关系的影响下，牛市配小B，熊市配小A。所以如果你是个怕麻烦的人（不喜欢套利或场内波段交易），熊市中踏踏实实吃点利息（最好折价买入），这是最省心的。

而且你甚至可以多一份奢望，即通过对于牛熊转换的行情判断（如判断市场行情过热时）提前布局小A，以更高的折价买进并长期持有，获取更高的实际收益率。

但有人问，如果我看错了方向怎么办？没关系，就算你买入后行情继续上涨，小A价格继续下跌出现浮亏，那也不用怕，因为每年都会有一次定期折算，届时小A的折溢价都会归零，所以浮亏只是暂时的，只要拿得住，就能等到天亮的一天。



2.交易型玩法：

不喜欢长期持有？那就玩短线交易！

我们都知道套利行为会在短期内平衡价格，所以牛市中小B的高溢价和小A的高折价一般都不会维持很久，这就给了小A短线操作的机会，即高折价买入后，等待小A短期内（几周甚至一、两周）迅速反弹，然后果断卖出，获取可观的利润（有时甚至会比约定收益率高出很多）。之后，再去寻找新的投资洼地，继续上述短线交易策略，循环往复（即使时运不济一时被套，扛到最后定折吃利息就行了）。

另外，股市大涨可能造成频繁的溢价套利，导致小A被频繁抛售，短时间也可能出现高折价，这时也是买入的好时机。

以上方式进可攻，退可守，一年赚个20%、30%不在话下，相当适合“不安分”的基友。

3.下折型玩法：

当你判断小B狂跌以至于其净值很可能触发下折时，迅速入手小A。

一方面，小B净值越接近下折，小A越容易上涨，你就越容易从中获利。原因有三：

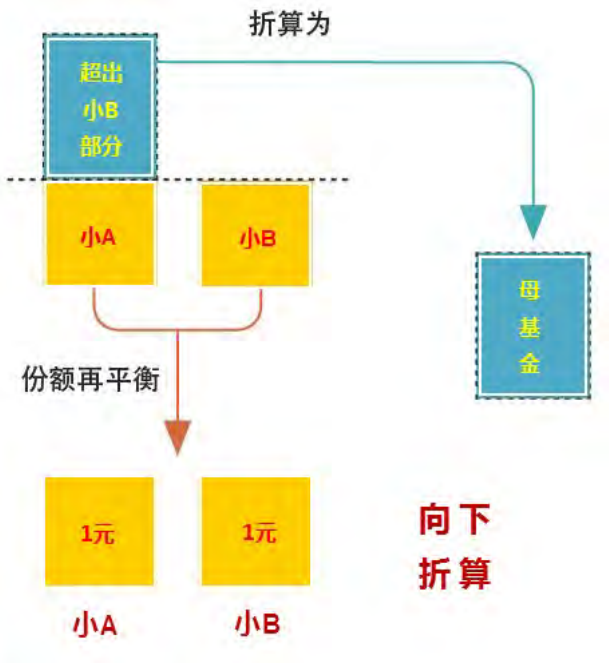
（1）牛熊转换的供求关系导致避险资金涌向小A；

（2）小B狂跌的过程可能会造就折价套利的难得机会，很多套利资金会买入小A进行配对转换，并场外赎回母基金，短期推升小A价格；

（3）小B的投资者为了避免更大的损失，很可能被迫去买入小A后合并成母基赎回，这也会间接抬高小A。

另一方面，一旦下折成真，小A净值多于小B净值（一般是0.25元）的那部分，都会折算成母基金返还给基友。这看似是左右装右兜的事（甚至还损失了母基金的赎回费用），但实际上如果你之前以很低的折价买入了小A，下折后也可以享受不错的折价收益。

当然下折型玩法相对激进，有一定风险（后文会讲），需要熟悉行情和实时监控，更需要传说中的智慧和勇气。



四．小A的选择之道

当然，不管你待见哪种玩法，都要有个更加细致的选A原则作为指南针。
1.成交额原则：

选择交易量大的小A。

学过金融的朋友都明白这是一条重要的原则，交易量越大，交投越活跃，价格越有效，冲击成本也就越低。如今分级基金在去年股灾之后万夫所指，交投低迷，但依然有一些大而不倒的"擎天柱"支撑着市场的门面，比如下面这些，你就可以重点考虑：

代码	名称	现价	涨幅	成交额(万元)
150175	H股A	1.029	-1.15%	24427.97
150200	券商A	1.069	-0.37%	11641.09
150018	银华稳进	1.077	-0.09%	10538.67
150022	深成指A	0.862	-0.12%	10109.46
150205	国防A	1.067	-0.28%	9354.26
150130	医药A	1.126	-0.09%	7378.86
150221	中航军A	1.254	-0.32%	4946.44
150209	国企改A	1.083	-0.09%	3402.44
150152	创业板A	1.101	-0.36%	3333.62
150194	互联网A	1.066	-0.19%	3238.64
150227	银行A	1.036	-0.19%	2987.97
150117	房地产A	1.142	0.26%	2893.56

（数据来源：集思录 2016-11-9）
而下面这些，就干脆别碰为妙：

代码	名称	现价	涨幅	成交额(万元)
150225	证保A级	1.141	-0.17%	0.10
150263	1000A	1.178	0.00%	0.12
150145	高贝塔A	1.084	-0.64%	0.22
150140	国金300A	1.117	0.18%	0.26
502001	500等权A	1.116	5.78%	0.28
150039	鼎利A	1.086	-0.09%	0.36
150057	中小300A	1.186	0.17%	0.37
150053	泰达500A	1.090	0.00%	0.44
150327	新能A级	1.188	0.00%	0.47
150083	深证100A	1.076	0.37%	0.53
502037	网金A	1.112	-2.28%	0.56
502057	医疗A	1.149	-2.79%	0.67

（数据来源：集思录 2016-11-9）

2.存续期原则：

选择存续期限为“永续”的小A。

市场上的分级基金可根据存续期限分为两类：一是有期限的，比如3年或5年就到期，之后有的分级基金的AB子基金会终止上市，有的则会把AB全部合并成一个LOF基金。这类分级基金的期限一天天缩短，计算实际收益率时比较复杂，是当前市场中的极少数（目前只有9只基），我们不予讨论。

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率	下期利率	修正收益率	剩余年限
150188	转债优先	1.059	0.00%	0.00	1.0500	-0.86%	其它	5.50	5.50	-3.764%	0.10
150133	德信A	1.048	0.00%	0.00	1.0570	0.85%	+1.2%单	3.70	3.70	5.502%	0.46
150085	中小板A	1.072	-0.19%	69.04	1.0251	-4.58%	+3.5%	5.00	5.00	-4.115%	0.49
150039	鼎利A	1.086	-0.09%	0.36	1.0960	0.91%	+1.0%单	4.00	4.00	5.272%	0.60
150096	商品A	1.130	1.07%	15.21	1.0430	-8.34%	+3.5%	5.00	5.00	-8.004%	0.63
150108	同辉100A	1.080	0.00%	0.00	1.0110	-6.82%	7.0%	7.00	7.00	-1.064%	0.84
150016	合润A	1.066	0.00%	48.77	1.0000	-6.60%	无约定	0.00	0.00	-2.584%	2.44
150106	中小A	1.103	-0.27%	163.16	1.0093	-9.28%	7.0%	7.00	7.00	3.502%	2.86
150135	国富100A	1.086	0.00%	108.60	1.0430	-4.12%	+3.5%	5.00	5.00	3.260%	3.38

（数据来源：集思录 2016-11-9）

另一类则是永续型的小A，类似于永续债券，计算实际收益率比较方便，又经常和十年期优质企业债券相提并论，同时刚上市时也容易在二级市场出现折价（理论上永续债券只付息不还本，会有一定的风险补偿），因此强烈推荐。

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率	下期利率	修正收益率	剩余年限
150175	H股A	1.029	-1.15%	24427.97	1.0469	1.71%	+3.5%	5.00	5.00	5.091%	永续
150200	券商A	1.069	-0.37%	11641.09	1.0410	-2.69%	+3.0%	4.50	4.50	4.377%	永续
150018	银华稳进	1.077	-0.09%	10538.67	1.0380	-3.76%	+3.0%	4.50	4.50	4.331%	永续
150022	深成指A	0.862	-0.12%	10109.46	1.0385	17.00%	+3.0%	4.50	4.50	5.464%	永续
150205	国防A	1.067	-0.28%	9354.26	1.0430	-2.30%	+3.0%	4.50	4.50	4.395%	永续
150130	医药A	1.126	-0.09%	7378.86	1.0466	-7.59%	+4.0%	5.50	5.50	5.095%	永续
150221	中航军A	1.254	-0.32%	4946.44	1.0560	-18.75%	+5.0%	6.50	6.50	5.426%	永续
150209	国企改A	1.083	-0.09%	3402.44	1.0400	-4.13%	+3.0%	4.50	4.50	4.314%	永续
150152	创业板A	1.101	-0.36%	3333.62	1.0430	-5.56%	+3.5%	5.00	5.00	4.726%	永续

（数据来源：集思录 2016-11-9）

3.实收率原则：

相同条件下，选择折价更高的小A。

个中原因前文已经论证，结合我们之前的玩法介绍，相信你已经得出结论：相同条件下，高折价就意味着更大的折扣买入小A，实际收益率也就会更高了。

代码	名称	现价	涨幅	成交额(万元)	净值	折价率	利率规则	本期利率	下期利率	修正收益率
150133	德信A	1.048	0.00%	0.00	1.0570	0.85%	+1.2%单	3.70	3.70	5.502%
150022	深成指A	0.862	-0.12%	10109.46	1.0385	17.00%	+3.0%	4.50	4.50	5.464%
150331	网金融A	1.160	-0.17%	461.28	1.0564	-9.81%	+4.5%	6.00	6.00	5.437%
150221	中航军A	1.254	-0.32%	4946.44	1.0560	-18.75%	+5.0%	6.50	6.50	5.426%
150321	煤炭A基	1.285	-0.31%	21.99	1.0610	-21.11%	+5.0%	6.50	6.50	5.310%
150297	互联A级	1.125	-0.35%	331.35	1.0817	-4.00%	+4.0%	6.00	5.50	5.273%
150039	鼎利A	1.086	-0.09%	0.36	1.0960	0.91%	+1.0%单	4.00	4.00	5.272%
150323	环保A端	1.103	-0.63%	139.30	1.0464	-5.41%	+4.0%	5.50	5.50	5.205%
150335	军工股A	1.108	-0.63%	195.64	1.0500	-5.52%	+4.0%	5.50	5.50	5.198%
150303	创业股A	1.110	-0.45%	1674.85	1.0506	-5.65%	+4.0%	6.00	5.50	5.194%

（数据来源：集思录 2016-11-9）

五、小A的投资风险

前篇我们说小A是无风险投资品，主要是从其债券属性的角度考虑的，实际上作为可以场内交易的品种，小A多少也会有一些【Dangerous Times】需要基友们重点关注。

1.反弹风险

前文讲玩法时说到，强烈的下折预期通常会推升小A的价格，让你有利可图。但市场不会以投资者的意志为转移，如果小B触发下折失败，绝地反弹了，那溢价买入小A的人就可能面临亏损。同时随着反弹级别的提升，可能出现的溢价套利行为也将会短期强力打压小A，造成价格下跌风险。

2.定折风险

下折时小A净值>小B净值的那部分都会折算成母基金返还给基友，但母基金最快也要T+2日才能赎回，如果过程中遭遇市场大跌，那实际收益率将会大打折扣。

3.挤兑风险

若市场遭遇极端行情，基金公司可能很难应付基民们的巨额赎回，造成挤兑风险，但此种情况概率较小。

看到这里，分级小A已成功被你驯服了。让我们荡起双桨，哦不，是稍事休息，准备迎接下一个主角——分级小B。

注：以上关于折溢价套利的内容，后续文章会有详述，读者们不必着急。

文章更迭记录：
v1.0，2016-11-17，文章上线

【量化课堂】白话分级基金之四 ——进击的小B

导语：
如果说小A如谦谦君子，温良如玉，那么小B则像脱缰的野马般狂放不羁、个性张扬。今天就让我们来做个勇敢的尝试——驯服“进击的小B”。

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。
本篇包含大量的金融知识点，需先行学习以下两篇文章：
1.《白话分级基金之一——一家“鸡”不说两家话》
2.《白话分级基金之二——成长的烦恼》
3.《白话分级基金之三——让小A飞》

作者：$BaSO_4$
编辑：肖睿

传送门1：《白话分级基金之一——一家“鸡”不说两家话》

传送门2：《白话分级基金之二——成长的烦恼（上）》

传送门3：《白话分级基金之二——成长的烦恼（下）》

传送门4：《白话分级基金之三——让小A飞》

分级基金是中国资本市场独步全球的产品，是中国金融的自主创新，是中国人的骄傲。——方正证券研究所所长助理，金融工程首席分析师 高子健

一、小B的魅力所在

《白话分级基金一》已详析了小B的优势（上图红框所示），我们做个简单总结和延伸：

1.分级杠杆带你飞

杠杆理念作为人类历史上最伟大的发明之一，被广泛应用到金融投资领域，其属性使小B在上涨市中跑赢指数的机会大大增加，资金效率和收益水平明显提升。

2.只赚指数不赚钱？

指数跟随型策略可降低投资风险，获得平均收益，大大降低了主动管理型投资的不确定性。目前大多数分级都是指数型，只赚指数不赚钱？就让它“往事随风”吧。

3.选股烦恼不再有

股民常叹息：“后宫三千佳丽（近3000只股票），让朕如何挑选？”不必担心，小B的服务一应俱全——划分行业，细分主题，精选个股，事半功倍。何以解忧？唯有小B。

4.相比竞品优势大

投资门槛极为亲民（目前是这样），理财三性相对平衡，这就是小B相比券商融资、个股期权、场外配资、结构化产品等杠杆类投资方式更为厚道的一面。

5.韭菜命运去不留

庄家说股市的散户就是任人宰割的小韭菜，但小B却不同意。原因有二：1.没人能操纵指数，也就没人能操纵小B；2.套利行为会平衡小B的二级市场价格，最终让市场回归本质。

二、小B的杠杆结构

小B的杠杆可以从三个角度去理解：

1.份额杠杆=（小A份额+小B份额）/小B份额

此杠杆也叫初始杠杆，是发行分级基金时小A小B初始份额比例下的杠杆大小，如5：5的话，份额比例=（5+5）/5=2，小B的杠杆就是2倍。

其作用，一是套利做配对转换时方便计算小A和小B的买入份额，二是作为后面两类杠杆的推导基础。

2.净值杠杆=（小A净值+小B净值）/小B净值=母基净值*份额杠杆/小B净值

由公式可以看出，净值杠杆与小B净值成反比，小B越跌，净值杠杆越高，风险越大；反之亦然。

另外，由于小B不能在二级市场按照净值单独申购赎回，所以净值杠杆实际上是理论杠杆，反映的是真实价值的变动速率。

3.价格杠杆=（小A净值+小B净值）/小B价格=母基净值*份额杠杆/小B价格

由公式可以看出，价格杠杆与小B价格成反比，小B越跌，价格杠杆越高，风险越大；反之亦然。不难看出，在实际交易中，小B可以理解为收益和风险并不对等。

另外，由于价格可以理解为小B在二级市场的实际买卖成本，所以价格杠杆比净值杠杆更具有参考价值，对盈亏的判断帮助更大。

三、小B的折算机制

“是福不是祸，是祸躲不过”，这应该是对折算机制最好的形容。其目的是：

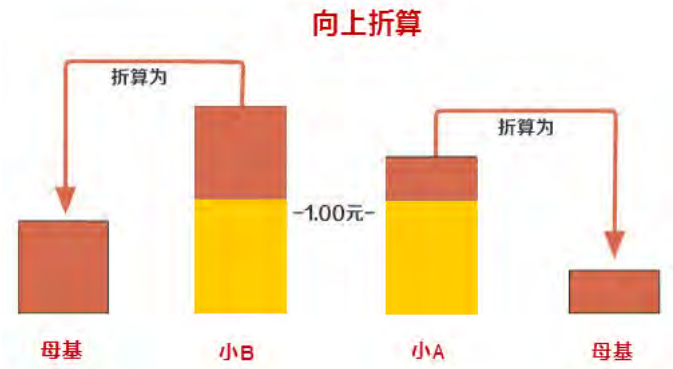
定期折算（定折）：兑现小A的约定收益。（详见《白话分级基金三》）

向上折算（上折）：提高小B的杠杆以增加收益效应和投资吸引力。（详见《白话分级基金二（下）》）

向下折算（下折）：降低小B的杠杆以降低投资风险，保护小A利益。（详见《白话分级基金二（上）》）

定折与小B关系不大，我们重点来说上下折。

1.上折（福）



当母基净值涨到阈值（一般为1.5元或2元）时上折启动，母基、小A和小B的净值都重新归1（小A和小B净值超过1的部分都折算为母基的份额发放给基友），母基的份额也会增加。

上折流程：

上折	小A	小B	母基
T日（折算基准日）	正常交易	正常交易	暂停申购赎回，场内买入A和B不能合并
T+1日（折算执行日）	暂停交易	暂停交易	暂停申购赎回
T+2日（结果公告日）	10:30恢复交易	10:30恢复交易	暂停申购赎回，A和B持有人收到母基份额

上折举例：

上折前	净值	份额	上折后	净值	份额
母基	1.5元	2000份	母基	1元	3000份+50份A+950份B
小A	1.05元	1000份	小A	1元	1000份
小B	1.95元	1000份	小B	1元	1000份

50份A=（1.05-1）*1000份

950份B=（1.95-1）*1000份

接下来你就该想了，这“幸福的烦恼”（多出来的母基份额）该怎么面对呢？方法有三：

（1）继续持有。

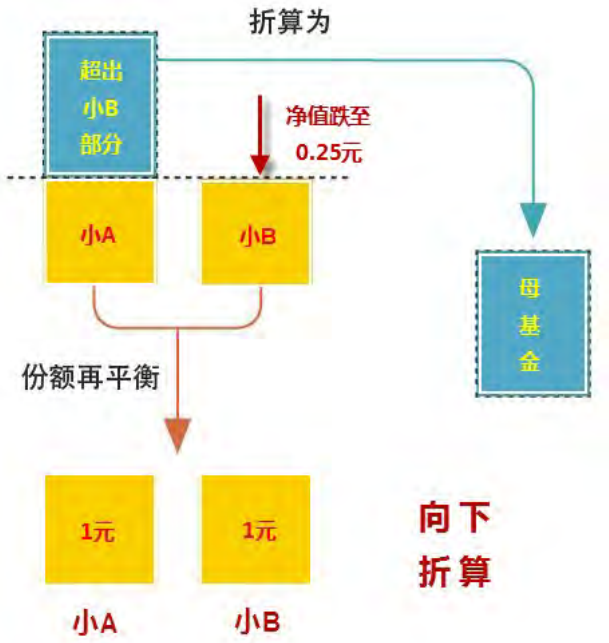
（2）直接赎回。但需要支付一定的赎回费，成本较高，不太推荐（除非你不在乎）。

（3）“肢解”母基。把母基拆分成场内的小A和小B，根据当时的实际行情进行处理——看好后市，则卖A卖B或卖A留B，反之亦然。

注：拆分不收费，当天就能搞定。

2. 下折（祸）

如果不幸遭遇跌跌不休的熊市，甚至是股灾，“恐怖の下折”就会被无情地触发。



当小B净值跌破阀值（一般为0.25元）时下折启动。母基、小A和小B的净值都重新归1（小A净值多于小B净值的那部分都折算成母基的份额发放给基友），三者的份额都将相应缩减。

下折流程：

下折	小A	小B	母基
T日（折算基准日）	10:30开始交易	10:30开始交易	暂停申购赎回，场内买入A和B不能合并
T+1日（折算执行日）	暂停交易	暂停交易	暂停申购赎回
T+2日（结果公告日）	10:30恢复交易	10:30恢复交易	暂停申购赎回，A持有人收到母基份额

下折举例：

下折前	净值	份额	下折后	净值	份额
母基	0.65元	2000份	母基	1元	1300份+800份A
小A	1.05元	1000份	小A	1元	250份
小B	0.25元	1000份	小B	1元	250份

800份A=（1.05-0.25）*1000份

下折到底哪儿恐怖?

一方面，二级市场的场内交易针对价格有涨跌停板制度（和股票一样是10%），但净值是没有的；

另一方面，行情连续下跌时，小B的净值持续缩水，净值杠杆不断放大，净值加速下跌，每天的跌幅可能超过10%甚至达到20%，让小B的价格望尘莫及。

因此，溢价不断提高，溢价亏损的窟窿也就越来越大，终于“砰”地一声，人还在，钱没了……
让我们一起感受下这酸爽的滋味：

假设某基友在某交易日以0.85元的价格买入小B期待其触底反弹，当日小B净值为0.8元（溢价较小）。

之后不幸股市继续暴跌，小B的每日净值跌幅超过10%，而价格由于10%的跌停板限制始终跟不上净值的跌幅，加之下跌中杠杆加大，溢价幅度越来越高，触发下折时小B的价格为0.45元，净值为0.22元（溢价率明显扩大）。

按价格计算，小B只亏损了（0.85-0.45）/0.85*100%=47%，

但下折是按净值计值，小B最终实际亏损达到了（0.85-0.22）/0.85*100%=74%

另外，有一个规则大家要格外注意：

下折被触发后，即使下一交易日小B涨停，下折也不能停止。但因为这天10:30之后还能继续交易，又没有下折公告之类的提示，就让不少“不明真相的群众”在这天躺枪买入这种小B，导致一天就亏损三到四成，比窦娥还冤。

不过，凡事都要辩证来看。说下折恐怖，其实单纯是从过程和结果来分析的，如果积极来看这也是好事，毕竟此机制降低了小B的交易风险（杠杆恢复到合理水平，资金亏成0的概率大大降低），也保护了小A的投资利益（保证兑现其约定收益），增加了分级两端的投资吸引力。

四、小B的投资策略

1.牛市行情

众所周知，突出的进攻性和杠杆性会让小B在牛市中如虎添翼，其业绩增幅往往可以媲美甚至超过行业牛股的收益。因此当你判断大势向好时，赶快分析哪个行业或主题更有投资价值，果断买入，长期持有，等待阳光灿烂的日子！（对于散户来说，趋势行情下，长期持有比波段交易更稳定）

以2014-2015年牛市中波动最大的证券B为例，假如你在14年10月底建仓，两眼一闭啥都不管，恭喜，半年不到你将获得超过500%的收益！太爽了！

2.非牛行情

即震荡和熊市行情。

当趋势明确下行或并不明朗时，我们建议别碰小B，因为机会很难把握，动不动就可能深陷泥潭。但如果你财大气粗心比天高非要下海一搏，那就请遵循以下原则：

（1）快进快出，切勿恋战；

（2）控制仓位，切忌重仓；

（3）严格止损，杜绝“裸奔”；

（4）板块轮动，进退得当。

3.暴跌行情

暴跌最让人蛋疼，唯一的办法就是——跑！

有人说撞到15年6月以后那种变态的暴跌行情，导致手中的小B连续跌停想跑跑不掉怎么办？可能减少损失的方法有四：

（1）虔诚祈求行情否极泰来（阿弥陀佛……）；

（2）挂单，以小B下一个交易日的跌停价卖出；

（3）立刻买入相同份额的小A，尽快合并赎回母基；

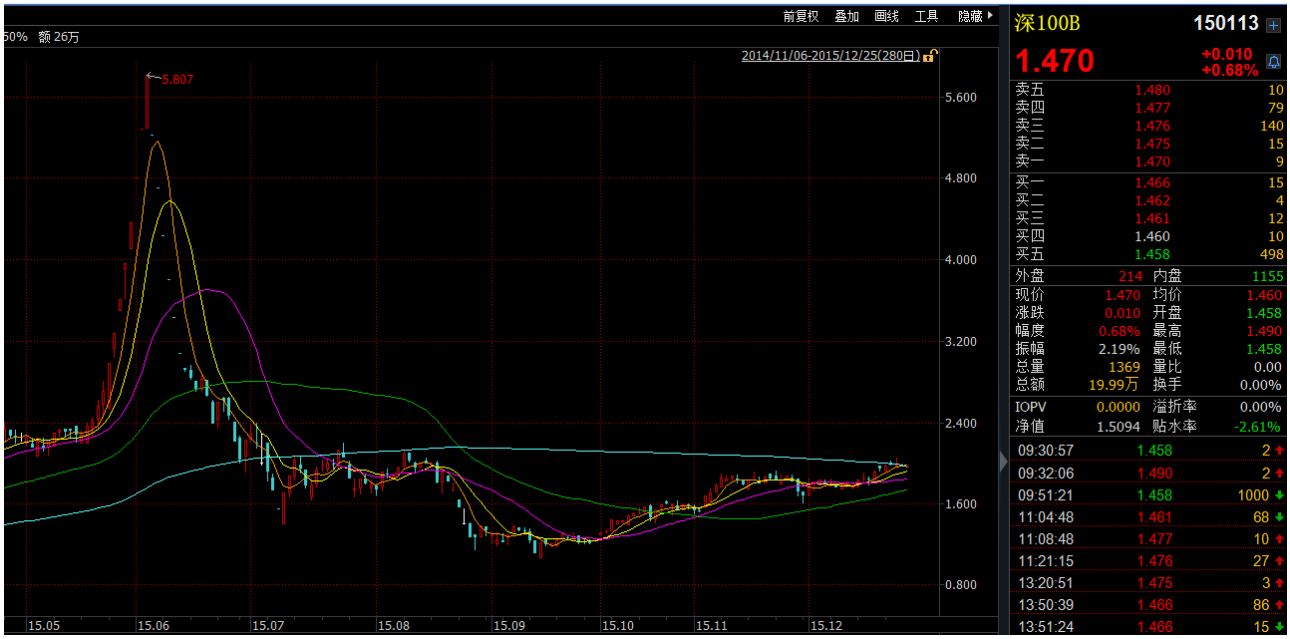
（4）若小A有大幅折价则选择买入，一旦下折成真，即享受上篇文章所说的额外份额的折价收益，降低小B的亏损（聊胜于无）。

4.套利行情

当小A和小B的价格之和大于母基，溢价套利资金就会如期而至，母基被大量申购，在场内快速拆分。

如果利润空间够大，套利者就会争先恐后地将小B大量抛售，甚至造成踩踏，导致其价格疯狂下跌。

等到行情企稳，你逐步建仓的好时机，也许就会如期而至！



5.临折行情

即“临近下折搏反弹”！这绝对是刀山火海的勇敢者游戏！

因为小B越跌杠杆越大，非常接近下折时，杠杆已接近最大，此时预判“行情必反弹”的基友选择建仓，未来可能的收益将是最大的。而且，只要不是暴跌行情，理论上小B接近下折阀值的过程不会太过迅猛，反复探底的情况会有很多，这也给了基友们多次试错的机会。

但必须强调的是，该法一旦失败，前文讲过的下折高溢价亏损将成为你的噩梦，所以请谨慎选择。

五、小B的选择之道

1.看指数类型

分级基金绝大多数都是指数基金，所以买B之前，先要甄选其生存土壤——基准指数。

目前分级基金的跟踪指数有三种：

（1）成分指数：如上证50、沪深300、中小板指数、创业板指数等；

（2）行业指数：如券商、军工、环保、地产、互联网等；

（3）概念指数：如国企改革、新能源汽车、一带一路等。

具体选哪种指数标的，完全看基友的判断和喜好，没有一定之规。

比如，你对行业轮动毫无研究，那就在牛市中选覆盖所有行业的成分指数；发现国企改革和一带一路是近期国家倡导的热点，那就重点布局这两个板块，等等。

总之，选择指数是一门功夫，需要你具备一定的经济学常识，并经常关心国家政经大事。

代码	名称	现价	涨幅	成交额(万元)	估值	净值	溢价率	剩余年限	利率规则	价格杠杆	净值杠杆	融资成本	参考指数
150206	国防B	0.497	-0.20%	28484.26	0.500	0.5020	-0.68%	永续	+3.0%	3.107	3.086	6.09%	中证国防
150023	深成指B	0.419	0.96%	8489.15	0.247	0.2410	69.92%	永续	+3.0%	3.069	5.214	7.00%	深证成指
150324	环保B端	0.534	0.75%	107.48	0.590	0.5849	-9.52%	永续	+4.0%	3.066	2.775	6.76%	中证环保
150248	传媒B级	0.537	1.90%	1851.74	0.595	0.5882	-9.82%	永续	+4.0%	3.059	2.759	6.76%	中证传媒
150131	医药B	0.543	1.31%	12864.83	0.604	0.5995	-10.06%	永续	+4.0%	3.041	2.735	6.78%	国证医药
150149	医药800B	0.544	0.00%	235.93	0.604	0.5990	-9.91%	永续	+3.2%	3.027	2.727	6.08%	800医药
150146	高贝塔B	0.545	-0.37%	30.59	0.599	0.6040	-8.94%	永续	+3.5%	3.017	2.748	6.37%	300高贝
150136	国富100B	0.543	0.00%	4.95	0.593	0.5940	-8.47%	3.36	+3.5%	3.015	2.760	6.17%	中证100
150292	银行B份	0.556	-0.18%	156.79	0.618	0.6180	-10.03%	永续	+4.0%	3.000	2.699	6.78%	中证银行
150288	钢铁B	0.578	-1.87%	8527.45	0.659	0.6680	-12.29%	永续	+4.0%	2.957	2.593	6.71%	国证钢铁
150222	中航军B	0.646	0.62%	16039.13	0.839	0.8370	-22.98%	永续	+5.0%	2.935	2.260	7.02%	军工指数
150336	军工股B	0.576	0.35%	742.34	0.634	0.6330	-9.10%	永续	+4.0%	2.925	2.659	6.82%	中证军工
150264	1000B	0.607	1.85%	67.45	0.724	0.7157	-16.17%	永续	+4.0%	2.923	2.451	6.57%	中证1000
150134	德信B	1.297	-0.31%	2.16	1.288	1.2870	0.72%	0.44	+1.2%单	2.896	2.917	4.88%	中高企债
150308	体育B	0.573	2.14%	863.49	0.602	0.5890	-4.77%	永续	+3.0%	2.870	2.733	6.07%	中证体育
150276	一带一B	0.579	0.35%	9618.13	0.573	0.5770	1.02%	永续	+3.0%	2.788	2.816	6.22%	一带一路

（数据来源：集思录 2016-11-15）

2.看规模大小

选择成交额大的。（原因上篇已讲，不再赘述）

代码	名称	现价	涨幅	成交额(万元)
150201	券商B	0.664	-1.19%	97033.09
150153	创业板B	0.968	3.20%	69928.72
150172	证券B	1.374	-0.79%	37779.02
150195	互联网B	0.486	3.85%	30986.56
150206	国防B	0.497	-0.20%	28484.26
150197	有色B	1.280	-4.41%	26298.69
150019	银华锐进	0.858	0.00%	18403.48
150222	中航军B	0.646	0.62%	16039.13
150290	煤炭B级	0.669	-3.88%	15686.20
150224	证券B级	1.110	-1.25%	15600.22
150210	国企改B	0.709	0.00%	14850.04
150131	医药B	0.543	1.31%	12864.83
150182	军工B	1.156	-0.34%	12349.51
150276	一带一B	0.579	0.35%	9618.13
150176	H股B	0.790	1.28%	9440.56
150288	钢铁B	0.578	-1.87%	8527.45

（数据来源：集思录 2016-11-15）

3.看溢价程度

选择溢价率不高的，否则回落和亏损风险大大增加（前文也已详述）。

像下图这么高的，还是先避避吧……

代码	名称	现价	涨幅	成交额(万元)	估值	净值	溢价率
150023	深成指B	0.419	0.96%	8489.15	0.247	0.2410	69.92%
150067	互利B	1.495	1.01%	0.01	1.229	1.2290	21.68%
150244	创业B	1.154	10.01%	3788.87	1.024	1.0080	12.72%

（数据来源：集思录 2016-11-15）

4.看价格杠杆

前文讲过，买卖小B是按价格而非净值，所以价格杠杆更有实际意义。

同等条件下，优先选择价格杠杆高的，涨起来才比个股更带劲嘛！（不过跌下去可就完蛋了）

代码	名称	现价	涨幅	成交额(万元)	估值	净值	溢价率	剩余年限	利率规则	价格杠杆
150033	多利进取	1.036	-1.05%	308.37	1.060	1.0596	-2.23%	永续	5.0%	5.008
150165	可转债B	0.815	-0.12%	212.17	0.871	0.8780	-6.41%	永续	+3.0%	4.040
150144	转债B级	0.879	-0.45%	216.27	0.922	0.9300	-4.70%	永续	+3.0%	3.818
150189	转债进取	0.974	-0.41%	328.66	0.976	0.9830	-0.19%	0.08	其它	3.517
150282	金融地B	0.472	-0.21%	178.50	0.529	0.5270	-10.77%	永续	+3.5%	3.411
150214	成长B级	0.464	3.11%	5088.15	0.494	0.4850	-6.07%	永续	+3.5%	3.315
150208	地产B端	0.470	3.75%	742.78	0.495	0.4810	-4.97%	永续	+3.0%	3.267
150095	泰信400B	0.490	0.00%	22.91	0.555	0.5530	-11.71%	永续	+3.5%	3.261
150118	房地产B	0.515	4.67%	8217.94	0.607	0.5913	-15.12%	永续	+4.0%	3.212
150298	互联B级	0.504	4.13%	268.30	0.525	0.5205	-4.07%	永续	+4.0%	3.191

（数据来源：集思录 2016-11-15）

5.看下折趋势

虽然上文出了个“临近下折搏反弹”的幺蛾子，但稳妥起见，对于眼看着就要下折的小B，稳健的基友还是别冒险了，如果把你拖进“无尽的深渊”，我们也会很心痛……

6.看融资成本

融资成本就是小B给小A的约定收益率。

同等条件下，融资成本低的小B就意味最终赚到了更多，虽然影响可能不大，但蚂蚁再小也是块肉啊。

六、小B的投资风险

以下风险前文大都提及，不再详述：

1.系统性风险

除非空仓，否则谁也躲不过去，如去年股市暴跌造成的千股跌停、千股停牌等大风大浪大灾难。

2.流动性风险

交易量越小，交投越清淡，价格就越无效，买卖就越困难、冲击成本也就越大。

3.溢价风险

高溢价风险提了无数次（上折下折都存在），大家一定要谨记在心，睡觉都不能忘。

4.杠杆风险

高杠杆的双刃剑导致净值和价格双双加速涨跌，正可谓一念天堂，一念地狱，一念成佛，一念成魔。

5.临折搏反弹风险

纯属火中取栗、刀口舔血，把持不好，就会“稀拉里”。

6.溢价套利风险

溢价套利行为会导致A B价格均被快速打压，如果你高价买入小B，亏损便是大概率事件。

文章到此，分级野马小B也成功被你降服了。下一篇，我们将强势攻克——分级套利！

文章更迭记录：
v1.0，2016-11-25，文章上线

【量化课堂】白话分级基金之五——攻克套利

导语：

二战后期，前苏联红军经过顽强斗争，一举攻下希特勒的柏林老巢，取得了反法西斯卫国战争的最终胜利。

作为《白话分级基金》最后一篇，我们也会坚定地将伟大的红军精神传承到底，为了自由，《攻克套利》！

本文由JoinQuant量化课堂推出，难度为入门，理解程度为level-0。

本篇包含大量的金融知识点，需先行学习以下四篇文章：

1.《白话分级基金之一——一家“鸡”不说两家话》
2.《白话分级基金之二——成长的烦恼》
3.《白话分级基金之三——让小A飞》
4.《白话分级基金之四——进击的小B》

作者：$BaSO_4$

编辑：肖睿

传送门1：《白话分级基金之一——一家“鸡”不说两家话》

传送门2：《白话分级基金之二——成长的烦恼（上）》

传送门3：《白话分级基金之二——成长的烦恼（下）》

传送门4：《白话分级基金之三——让小A飞》

传送门5：《白话分级基金之四——进击的小B》

分级基金是中国资本市场独步全球的产品，是中国金融的自主创新，是中国人的骄傲。——方正证券研究所所长助理，金融工程首席分析师 高子健

分级套利是个好玩的东西!

但在正式讲解前，大家需要回忆几个概念（前篇有详述，此处不赘述）：

1.配对转换机制

2.净值和价格

3.折价和溢价

一、套利原理

1.理论基础

所谓套利，简单说就是寻找两个市场之间的差价，然后买便宜的卖贵的，分级基金套利也是如此。

从供求关系角度讲，小A小B的价格由市场预期（投资者情绪的直接反映）决定——

一般来说，在牛市中，乐观情绪（投资者认为获得超额收益的机会很大）使得看涨者远超看跌者，小B遭疯抢出现溢价，而小A则无人问津出现折价，反之亦然。总体来说，小A和小B相反的折溢价关系，大致就像跷跷板一样此消彼长。

但是，如果小A 与小B的价格之和与2倍母基的净值相比依然出现了一定的折溢价，尤其是偏离较大时，就会大规模“引狼入室”。

无数套利资金会迅速套走这部分折溢价，使得小A与小B价格之和继续约等于2倍母基金的净值，价格和净值最终回归平衡。

折价套利机会：小A价格+小B价格<母基净值*2
溢价套利机会：小A价格+小B价格>母基净值*2
（未考虑交易成本）

2.操作方法

折价套利机会出现时，场内买入小A和小B，通过配对转换合并成母基，全部赎回实现获利；

溢价套利机会出现时，申购母基，通过配对转换在场内拆分成小A和小B，全部卖掉实现获利。

分级基金具有份额配对转换机制，这是分级基金套利的基础机制。
基金份额的配对转换有分拆和合并两种转换行为。

套利者是真实价值的发掘者，也是市场价格体系的维护者。正是由于这些人的存在，与真实价值出现偏移（不合理）的价格才会重新“返璞归真”。从这个角度讲，套利的行为促进了市场的健康运转。

二、套利实操

1.套利流程

套利案例（以溢价套利为例）

某分级基金，该基金的小A、小B份额比例为5：5。

T日，基金出现溢价套利机会，于是我们申购了1000份母基金，净值1.1元，即市值1100元；

T+2日，确认基金到账后申请等比例分拆为子基金，获得相同份额的小A和小B；

T+3日，将小A小B全部卖出，当时价格分别为0.9元和1.4元，则共获得500*0.9+500*1.4=1150元。

不考虑交易成本的话，我们最终获得1150-1100=50元利润，50/1100≈4.55%即本次溢价套利收益率。

申购/赎回、分拆/合并、买入/卖出、盲拆/盲合，整个过程必须折腾好几天，还没考虑各种实操细节……没错，聪明的基友已看出端倪——分级套利绝并不简单的是空手套白狼的数字游戏，否则谁还去费尽心思研究它呢？

所以，让我们条分缕析，讨论以下几个重点环节。

2.特别关注

（1）时间风险（价格波动风险）

整个套利过程的时间空挡所带来的的价格波动风险是套利的第一天敌。

完成折价套利需3个交易日，完成溢价套利需4个交易日。这几天中，股市涨跌皆有可能，套利利润就成了不确定因素。

假如在T日你发现套利空间是5%，但之后几天股市连跌，基金跌幅超过了5%，套利即宣告失败，况且这还没算上交易费用。

（2）盲拆与盲合

风险无法完全消除，但可以适当规避，于是针对（1）的情况，就有部分“文艺券商”推出了“盲拆”和“盲合”（如华泰、广发、中信、方正证券等）。

盲拆就是在T+1日，母基还没到账时，手工计算自己应得的母基份额进行分拆，分拆后T+2日即可卖出，比“普通券商”早一天完成溢价套利。

T日申购母基 → T+1日盲拆为小A和小B → T+2日卖出

盲合是一样的道理，也可以提前一天完成折价套利。

T日买入小A和小B → T日盲合成母基 → T+1日赎回母基

这样的操作降低了套利流程中的价格波动风险，但需特别注意以下三点：

A.合并拆分一定要看小A小B的份额比例，5:5就按5:5来配对，4:6就以4:6来配对。

B. 对于盲拆业务，T+1日母基份额尚未登记在账户中，需自行计算可分拆份额，公式为：

母基份额 = 申购金额 / 申购当日净值 /（1+申购费率）

如分拆申报的数量大于实际申购的母基数量，分拆申报将会无效而导致失败。

C.母基最好通过场内申购，因为场外转托管到场内通常需要2个交易日，资金使用效率大大降低，价格波动风险进一步增加。

（3）交易成本

除了价格波动风险，交易成本也是分级套利的一大天敌，不可忽略，所以之前的那套公式就该写成：

折价套利机会：小A价格+小B价格<母基净值*2-交易成本（赎回费）
溢价套利机会：小A价格+小B价格>母基净值*2+交易成本（申购费）

申购费通常是按资金量呈阶梯递减的（请见下表），赎回费一般是0.5%。

50万元以下	1.2%
50万元（含）以上200万元以下	1.0%
200万元（含）以上500万元以下	0.5%
500万元（含）以上	1000元/笔

所以不管是溢价套利还是折价套利，这种交易成本都是躲不掉的，尤其是溢价套利时申购母基的1.2%更是让小散们相当蛋疼（不过目前很多分级基金为了鼓励套利和投资，已经取消了申购费），因此选择套利之前必须要计算清楚。

母基代码	母基名称	母基净值	估算净值	跟踪指数	指数涨幅	估值仓位	前日仓位	申购费	赎回费
502013	一带一路	0.8510	0.8428	CSSW丝路	-1.02%	95%	99%	1.2%	0.5%
160420	创业50	0.8403	0.8385	创业板50	-0.23%	95%	92%	0	0.5%
160638	带路分级	0.8150	0.8063	一带一路	-1.12%	95%	92%	0	0.5%
164818	传媒母基	0.8126	0.8125	中证传媒	-0.51%	95%	95%	0	0.7%
160637	创业指基	1.0210	1.0188	创业板指	-0.23%	95%	76%	0	0.5%
161725	白酒分级	0.9930	0.9881	中证白酒	-0.52%	95%	86%	1.0%	0.5%
160629	传媒分级	1.1700	1.1643	中证传媒	-0.51%	95%	96%	0	0.5%
167503	一带分级	0.8080	0.8073	一带一路	-1.12%	95%	92%	0.5%	0.5%
160630	国防分级	0.7730	0.7701	中证国防	-0.40%	95%	124%	0	0.5%
160633	券商指基	1.0680	1.0491	证券公司	-1.86%	95%	129%	0	0.5%
168204	煤炭母基	0.8860	0.8695	中证煤炭	-1.96%	95%	121%	0	0.7%
164705	添富恒生	1.0140	1.0125	恒生指数	0.37%	95%	-125%	0	0.5%
161831	H股分级	0.9029	0.9021	恒生国企	0.24%	95%	63%	1.2%	0.5%
160632	酒分级	0.9290	0.9236	中证酒	-0.61%	95%	104%	0	0.5%

（数据来源：集思录 2016.11.18）

（4）母基净值的估算

其实，要计算清楚的不只是交易成本，还有母基的动态净值。

小A和小B在交易日的价格是不断变化的，但母基的净值在盘中却不会被公示，而是当天收盘后才公布。盘中的整体折溢价率无法准确获取，那套利就只能蒙着来了？（靠谱的孩子，不靠谱的妈啊 -_-!）

其实不用担心，这是可以估算的，公式如下：

盘中某时刻的母基净值=母基上一交易日净值*（1+基金所跟踪指数的涨跌幅度*基金仓位估算值）
（其中，基金仓位估算值=上一交易日母基涨跌幅度/基金所跟踪指数的涨跌幅度*100%）

例如，某分级基金跟踪的是沪深300指数，上一交易日该指数上涨2%，但分级基金净值只上涨1%，净值为1元。则基金仓位估算值=1%/2%*100%=50%（即基金昨日只采用了一半仓位）

简单说，就是通过昨日的静态净值估算今日的动态净值，这样计算折溢价的套利空间就安全多了。

不过随时要算，太麻烦了！好在已经有人为大家铺好了光明大道：

母基代码	母基名称	母基净值	估算净值	跟踪指数	指数涨幅	估值仓位	前日仓位
502013	一带一路	0.8510	0.8428	CSSW丝路	-1.02%	95%	99%
160420	创业50	0.8403	0.8385	创业板50	-0.23%	95%	92%
160638	带路分级	0.8150	0.8063	一带一路	-1.12%	95%	92%
164818	传媒母基	0.8126	0.8125	中证传媒	-0.51%	95%	95%
160637	创业指基	1.0210	1.0188	创业板指	-0.23%	95%	76%
161725	白酒分级	0.9930	0.9881	中证白酒	-0.52%	95%	86%
160629	传媒分级	1.1700	1.1643	中证传媒	-0.51%	95%	96%
167503	一带分级	0.8080	0.8073	一带一路	-1.12%	95%	92%
160630	国防分级	0.7730	0.7701	中证国防	-0.40%	95%	124%

（数据来源：集思录 2016.11.18）

（5）上交所的新玩法

有基友问，分级基金发展到现在，有没有一种更为粗暴便捷的套利制度呢？有。

2015年上交所开创了一种分级基金新机制——场内T+0，这与深交所形成了鲜明对比，也对套利产生了不小的影响。

	上交所	深交所
代码标识	母基以50开头，且与A、B代码连号	母基以160/161/162/163开头，A、B以150开头
上市交易原则	母基、小A、小B皆可上市交易	只有小A、小B可上市交易
合并/分拆门槛	申报数量应为100份的整数倍，且不低于5万份（以母基份额计算）	申报数量不低于100份
合并/拆分制度	变相实现T+0 **拆分**：T日买入母基，T日可拆分，拆分后得到的小A和小B，T日可卖出。 **合并**：T日买入小A和小B，T日可合并，合并后得到的母基，T日可卖出或赎回。	非T+0 **拆分**：T日分拆母基金，T+1日获得A、B后才可卖出。 **合并**：T日合并小A和小B，T+1日获得母基后才可以赎回。

上交所虽然资金门槛较高，但其拆分/合并的制度更加开放和灵活，场内T+0机制使得价格风险进一步降低。

但是，如果套利的风险变小了，大家趋之若鹜，还会有明显的套利空间吗？这点需要大家思考。

三、风险对冲

有风险，就有对冲风险的方法，比较主流的有两种：

1.股指期货对冲

为了降低套利过程中价格下跌导致套利空间收窄的风险，在T日进行套利第一步操作的那个时间点，可以同时做空股指期货。

以溢价套利为例（未考虑手续费）：

以溢价对冲套利的下列两个场景为例（未考虑手续费）：

情景	基金整体溢价率	股指期货波动情况	套利类型	对冲套利理论收益
X日	2%	下跌1%	不对冲套利	2%-1%=1%
			对冲套利	2%
Y日	3%	上涨1.5%	不对冲套利	3%+1.5%=4.5%
			对冲套利	3%

注：为方便计算，假设该分级基金所跟踪的指数与股指期货的波动情况一致。

由此可见，利用股指期货对冲价格风险的做法，可以锁定当前的套利利润，但可能会也牺牲未来指数上涨所带来的更高收益，适合于稳健型投资者，而激进者则需要根据具体的市场情况选择性使用。

2.底仓套利（变相对冲和T+0交易）

顾名思义，就是始终持有底仓（母基或子基）的套利模式。

溢价底仓套利流程：

（1）按比例配对买入某分级小A和小B，一直持有；

（2）当出现整体溢价时，将小A和小B全部卖掉；

（3）与此同时，申购相对应和相应数量的母基。

折价底仓套利流程：

（1）申购某分级母基，一直持有；

（2）当出现整体折价时，将母基全部赎回；

（3）与此同时，按比例配对买入相对应和相应数量的小A和小B。

注意事项：

1.上述2和3两步在折溢价套利中全都需要同步完成，这样才能规避普通套利原本需要2-3天的价格波动风险，成为变相的对冲和T+0交易。

2.底仓套利实际上是把钱一分为二不断地进行“换仓”操作，所以资金利用效率降低了一半，不喜者勿用。

3.底仓套利不是无风险套利。在牛中，指数上涨会使分级基金整体收益上涨，获得除了套利收益以外的杠杆收益，但在熊市中，分级基金的整体下跌会大幅吞噬套利收益，使得整体收益入不敷出，甚至大幅亏损。因此，底仓套利是把双刃剑，择时使用才是明智的选择。

到此为止，我们终于攻克分级套利！以及《白话分级基金》系列最终完结。

分级基金是一个复杂的理财品种，以至于我们花了大量篇幅加以讲解，也只能触其皮毛。金融市场日新月异，创新脚步永不停歇，作为能攻能守的投资利器，分级基金必将在不久的将来脱胎换骨，重获新生。

愿这一系列《白话分级基金》能够抛砖引玉，为你发掘更为广阔的投资道路带来启发。

文章更迭记录：
v1.0，2016-11-26，文章上线

【量化课堂】MPT 模型的解析解（上）

导语：本篇文章将用拉格朗日乘子法来计算马科维兹的方差最小化问题。

作者：肖睿
编辑：宏观经济算命师

本文由JoinQuant量化课堂推出，难度为进阶（上），深度为 level-1。

阅读本文前需要掌握线性代数、多元微积分、MPT 模型以及拉格朗日乘子法的基础知识。

前言

量化课堂的 MPT 模型文章介绍了马科维兹的现代资产配置理论 (MPT, modern portfolio theory) 的模型和理论，其中讲解了重要的有效前沿和资本市场线的重要概念，但并没有解释有效前沿的计算方法。在拉格朗日乘子文章中我们介绍了使用拉格朗日乘子 (Lagrange multiplier) 来解决非线性规划问题的方法，并提到了该方法可以用来解决现代资产配置理论中的优化问题，这便是本篇文章探讨的主题。

MPT 模型可以分没有无风险资产和有无风险资产两个版本，这两个版本的解决思路相似但在细节上有一些差异，故本系列文章分为上下两篇，上篇主要探讨没有无风险资产的情况，而下篇解决有无风险资产的情况。本篇为上篇。

回顾

MPT 模型

在 MPT 模型中，我们假设金融市场上有风险资产 $i \in 1, 2, \ldots, n$，其中的每个资产的收益率都随机变量 $r < em > i$表示，有数学期望值 $\overline{r_i} = \mathbb{E}[r_i]$以及标准差 $\sigma_i = \sqrt{\mathrm{Var}[r_i]}$。这些资产之所以称为风险资产，因为它们的标准差 σ_i 都大于零。在没有无风险资产的 MPT 模型中，我们想要按照一定权重将资金配置于风险资产，如果分配于资产 i 的权重是 w_i，那么 $\mathbf{w} = (w_1, \ldots, w_n)$代表整个资产组合的配置比重。注意这里必须满足
$\sum < /em > i = 1^n w_i = 1$; 当 $w_i < 0$时意味着要卖空第 i 种金融资产。

我们用 $r < em > w$表示按照 w 配置资产得出的资产组合的收益率变量。经过计算，发现 r_w 的期望值和方差分别满足以下等式

Undefined control sequence \<

根据分散风险的方针，资产配置的一个目标是在固定预期收益的前提条件下把收益率的方差最小化，也就是对于一个固定的期望收益 μ 解决下面的最小化问题

Unknown environment 'align'

得到一个最小的方差 $V(\mu)$，对应着标准差 $\sigma(\mu) = \sqrt{V(\mu)}$。所有的期望和最小标准差的二元组 $(\mu, \sigma(\mu))$ 组成一条曲线，叫做有效前沿 (efficient frontier)。本文将致力于解决上述的方差最小化问题，并计算出有效前沿的闭合式公式。

拉格朗日乘子法

在拉格朗日乘子的文章中，我们提到了一个重要的定理：
定理. 设 $n, m \in \mathbb{N}$。对于 $i \in 1, 2, \ldots, m$, g_i 和 f 都是 $\mathbb{R}^n \to \mathbb{R}$ 的 $\mathcal{C}^1$ 函数。并且设 $c_i \in \mathbb{R}$。考虑规划问题

Unknown environment 'align'

定义函数

$$\mathcal{L}(x, \lambda) = f(x) - \lambda_1(g_1(x) - c_1) - \cdots - \lambda_m(g_m(x) - c_m),$$

这里 $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \in \mathbb{R}^m$。如果 $\tilde{x} \in \mathbb{R}^n$ 是上述规划问题的极值点，那么必定存在某个 $\tilde{\lambda} \in \mathbb{R}^m$ 满足

$$\nabla \mathcal{L}(x, \lambda) = 0.$$

我们就将使用这个定理来解决 MPT 模型的方差最小化问题。

解决马科维兹最优化问题

我们稍微更改原本的问题：

Unknown environment 'align'

目标函数中的 $\frac{1}{2}$ 不改变问题的本质，但它可以让后边的计算过程更干净。上边的问题也可以改写成矩阵的形式

Unknown environment 'align'

在这些符号中，

$\mathbf{\Sigma} = \begin{bmatrix} \mathrm{Cov}(r_1, r_1) & amp; \ldots & amp; \mathrm{Cov}(r_1, r_n) \\ \vdots & amp; \ddots & amp; \vdots \\ \mathrm{Cov}(r_n, r_1) & amp; \ldots & amp; \mathrm{Cov}(r_n, r_n) \end{bmatrix}$; $\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$; $\mathbf{r} = \begin{bmatrix} \overline{r_1} \\ \vdots \\ \overline{r} \end{bmatrix}$

当然，我们知道 $\mathrm{Cov}(r_i, r_i) = \mathrm{Var}[r_i]$。
在解题之前我们确认这个规划问题的极小点是存在的。首先，目标函数（也就是方差）是一个凸函数：对于任何两个随机变量 X 和 Y 有

Unknown environment 'align'

步骤 $(*)$ 是因为

Unknown environment 'align'

在此之上，规划问题的可行集 Missing or unrecognized delimiter for \left 是由两个线性约束决定的，因此它是一个凸集。根据数学规划简介所述，一个凸函数在一个凸集上必定有极小点，即 $\min < /em > \mathbf{w} \in \Delta \frac{1}{2}\mathrm{Var}(r_{\mathbf{w}})$ 是有解的。*以上逻辑所涉及到的理论会在以后的凸优化文章中更详细地解释。*

下面我们着手解决上述规划问题。根据拉格朗日乘子定理，我们定义函数

$$\mathcal{L}(\mathbf{w}, \lambda < em > 1, \lambda_2) := \sum < /em > i = 1^n \sum < em > j = 1^n w_i w_j \mathrm{Cov}(r_i, r_j) - \lambda_1 \left(\sum < /em > i = 1^n w < em > i\overline{r_i} - \mu \right) - \lambda_2 \left(\sum <$$

并计算它的各个偏导，

Unknown environment 'align'

上面的第一条可以改写成矩阵的形式，表示为

Unknown environment 'align'

将梯度 $\nabla \mathcal{L}$ 设为 0，即

Unknown environment 'align'

假设 $\mathbf{\Sigma}$ 是可逆的，由 (1) 可以得出 $\mathbf{w}$ 与 λ_1 和 λ_2 的关系

$$\mathbf{w} = \lambda_1 \mathbf{\Sigma}^{-1}\mathbf{r} + \lambda_2 \mathbf{\Sigma}^{-1}\mathbf{1}_n \qquad (\dagger)$$

下面还需要解出 λ_1 和 λ_2 根据 (2) 和 (3) 两个等式，又有

Unknown environment 'align'

代入 (†)，有

Unknown environment 'align'

这里注意因为协方差矩阵 $\mathbf{\Sigma}$ 是对称的，根据线性代数和对称矩阵的一些基本性质，有 $(\mathbf{\Sigma}^{-1})^{\mathsf{T}} = \mathbf{\Sigma}^{-1}$.

上面的两个等式可以写为矩阵形式，即

$$[\, \mu \; 1 \,] = \begin{bmatrix} \mathbf{r}^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{r} & amp; \mathbf{1}_n^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{r} \\ \mathbf{r}^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{1}_n & amp; \mathbf{1}_n^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{1}_n \end{bmatrix} [\, \lambda_1 \; \lambda_2 \,].$$

那么，如果我们设 $a = \mathbf{r}^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{r}, b = \mathbf{1}_n^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{r}, c = \mathbf{1}_n^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{1}_n$ 可以解出

$$[\, \lambda_1 \; \lambda_2 \,] = [\, a \quad amp; b \; b \quad amp; c \,]^{-1} [\, \mu \; 1 \,], \qquad (\dagger\dagger)$$

然后利用 (†) 的等式

$$\mathbf{w} = \lambda_1 \mathbf{\Sigma}^{-1}\mathbf{r} + \lambda_2 \mathbf{\Sigma}^{-1}\mathbf{1}_n$$

就可解出梯度函数 $\nabla \mathcal{L}$ 的一个零点 $\mathbf{w}^{<}em >$；如果矩阵 $[\, a \quad amp; b \; b \quad amp; c \,]$ 和 $\mathbf{\Sigma}$ 都是可逆的话，根据矩阵的列秩的性质，$\mathbf{w}^{<}/em >$ 便是 $\nabla \mathcal{L}(\mathbf{w}, \lambda_1, \lambda_2) = 0$ 的唯一解。我们知道原规划问题是有极小点的，根据拉格朗日乘子定理那些极小点都是 $\nabla \mathcal{L}$ 的零点，然而如果 $\nabla \mathcal{L}$ 只有一个零点 $\mathbf{w}^{<}em >$，那么 $\mathbf{w}^{<}/em >$ 必定是规划问题的唯一极小点。接下来我们需要确认 $[\, a \quad amp; b \; b \quad amp; c \,]$ 和 $\mathbf{\Sigma}$ 在什么情况下是可逆的，并且对于不可逆的情况给出解决方法。

两个矩阵的可逆性

在这一节中我们将发现有两个假设可以保证矩阵 Σ 和 $[\, a \quad amp; b \; b \quad amp; c \,]$ 的可逆性。它们是：

1. 如果任何一组风险资产都不能配置出无风险资产，那么 Σ 是可逆的，即公式 (†) 成立；
2. 如果不是所有风险资产的收益率期望都是相等的，那么 $[\, a \quad amp; b \; b \quad amp; c \,]$ 是可逆的，即公式 (††) 成立。

接下来将展示上述结论的推导，这需要使用正定矩阵和半正定矩阵的相关理论。在线搜索方法的文章中我们介绍过正定矩阵和半正定矩阵的定义，这里重温一下：

定义. 设 $A \in \mathbb{R}^{n \times n}$ 是一个对称矩阵，即 $A^{\mathsf{T}} = A$。如果对于任何一个 $x \in \mathbb{R}^n$ 都有 $x^{\mathsf{T}}Ax \geq 0$，那么 A 是一个*半正定矩阵* (positive semidefinite matrix)。如果对于任何一个 $x \in \mathbb{R}^n \backslash \mathbf{0}_n$ 都有 Misplaced &，那么 A 是一个*正定矩阵* (positive definite matrix)。

首先我们考虑协方差矩阵 $\mathbf{\Sigma}$。根据概率论的基础理论，任何的协方差矩阵都是一个半正定矩阵。我们需要用到线性代数的理论中关于半正定矩阵可逆性的命题：

定理. 设 $A \in \mathbb{R}^{n \times n}$ 是一个半正定矩阵，那么下面两个陈述是等价的：

1. A^{-1} 存在；
2. 对于每一个 $x \in \mathbb{R}^n \backslash \mathbf{0}_n$，都有 Misplaced &。

假设半正定矩阵 $\mathbf{\Sigma}$ 不是可逆的，根据以上定理，可以找出某个非零的 $\mathbf{w} \in \mathbb{R}^n$ 以满足 $\mathbf{w}^{\mathsf{T}}\mathbf{\Sigma}\mathbf{w} = 0$。那么有

Unknown environment 'align'

那么，按照 $w_1, \ldots, w_n$ 的比例配置 $(r_1, \ldots, r_n)$，会得到一个零方差的随机变量，对应着一个无风险的投资组合。这种情况可以在下一篇的有无风险资产的 MPT 模型中解决；本篇中不妨假设无风险资产不能由风险资产配置得来，这样 Σ 必定是可逆的。

接下来我们需要保证矩阵 $[\, a \quad amp; b \; b \quad amp; c \,] = \begin{bmatrix} \mathbf{r}^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{r} & amp; \mathbf{1}_n^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{r} \\ \mathbf{r}^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{1}_n & amp; \mathbf{1}_n^{\mathsf{T}}\mathbf{\Sigma}^{-1}\mathbf{1}_n \end{bmatrix}$ 是可逆的。根据行列式的定理，只需要证明 $b^2 - ac \neq 0$ 即可。

首先，我们需要假设 $a\mathbf{1}_n - b\mathbf{r} \neq \mathbf{0}_n$。这是一个合理的假设，因为如果上述的差等于零，那么所有的 $\overline{r_i}$ 都是相等的，也就是说所有的风险资产的收益率都是相同的，很显然这在现实中是几乎不可能发生的，这种情况也会导致有效前沿变成一条横向直线而不是一条曲线。

根据前一段的结论，我们知道 $\mathbf{\Sigma}$ 是一个正定矩阵，并且根据下面的定理，
定理. 如果 $A \in \mathbb{R}^{n\times n}$ 是一个正定矩阵，那么 A^{-1} 也是一个正定矩阵。
可以得知 $\mathbf{\Sigma}^{-1}$ 也是正定矩阵。

那么根据正定矩阵的性质和 $a\mathbf{1}_n - b\mathbf{r} \neq \mathbf{0}_n$ 的性质，有

Unknown environment 'align'

由此可见 $ac - b^2 \neq 0$，得知 $[\, a \quad amp; b \, b \quad amp; c \,]$ 是可逆的。

有效前沿和资本市场线

有效前沿

在马科维兹优化问题中，给定每一个期望收益率 μ 都有一个对应的最小标准差 $\sigma(\mu)$，所有这些 $(\mu, \sigma(\mu))$ 的二元组构成了一条曲线，叫做有效前沿。接下来我们将计算出这条曲线的公式。

根据第二节中的拉格朗日乘子方法，标准差最小的配置权重向量满足

$$\mathbf{w} = \lambda_1 \mathbf{\Sigma}^{-1}\mathbf{r} + \lambda_2 \mathbf{\Sigma}^{-1}\mathbf{1}_n$$

于是有

Unknown environment 'align'

再代入

$$[\, \lambda_1 \; \lambda_2 \,] = [\, a \quad amp; b \, b \quad amp; c \,]^{-1} [\, \mu \; 1 \,]$$

的结论，有

Unknown environment 'align'

那么得出有效前沿的公式

$$\sigma(\mu) = \sqrt{\frac{1}{ac - b^2}(c\mu^2 - 2b\mu + a)}.$$

最小风险组合

下面我们计算有效前沿上风险最小的组合。为了寻找函数 $\sigma(\mu)$ 的极小点，我们取其导数并将导数设为 0，

$$0 = \frac{d\sigma}{d\mu} = \frac{1}{2\sigma(\mu)}(2c\mu - 2b).$$

由于 Misplaced &，那么上面等式成立当且仅当 $c\mu = b$，也就是说最小的标准差存对应着收益期望 $\mu = b/c$，将这个数代进马科维兹优化问题就可以算出最小方差组合的权重了。

举例的时间到了

为了举例的简便性，我们只选择五个风险资产进行计算。这五个资产是从沪深 300 成分股中随机选出的五支股票，使用两年的日收益率数据来计算收益率和标准差和协方差。这五支股票的日平均收益率和日收益率标准差分别是

Undefined control sequence \2

这些数据放到标准差-均值坐标图上如下图的蓝色三角所示，图中的粉点是同期其他的沪深 300 成分股作为参考对比。

并且，我们计算这五支股票收益率的协方差矩阵为

Undefined control sequence \<

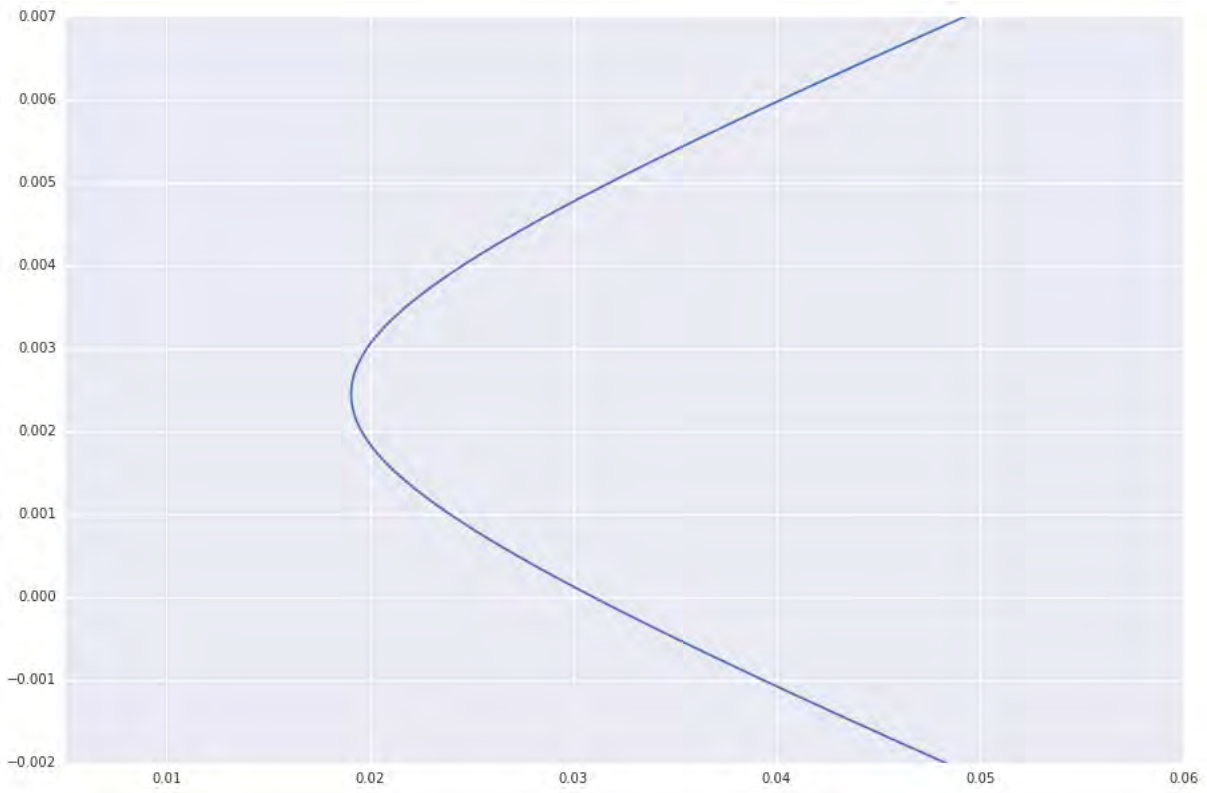
使用 Numpy 计算 a, b, c 的值：

Unknown environment 'align'

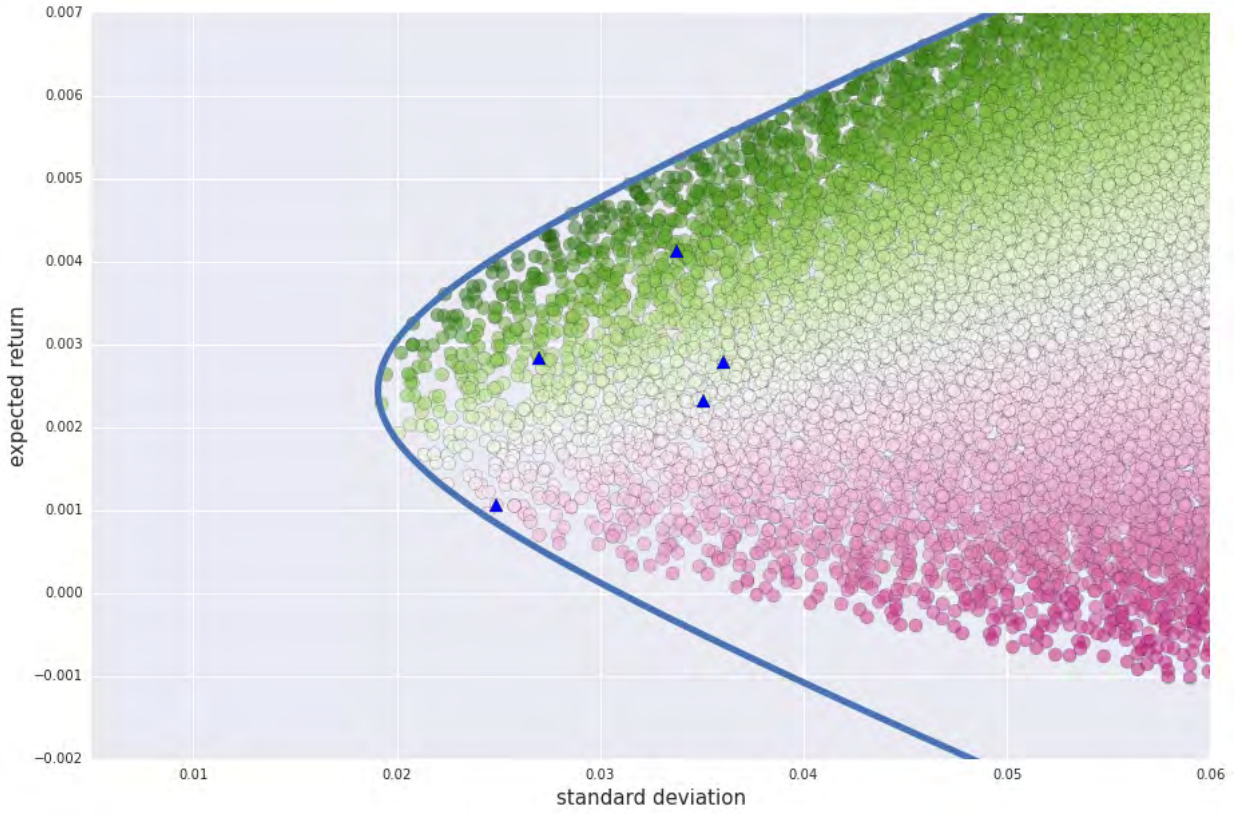
那么计算出有效前沿曲线的公式为

$$\sigma(\mu) = \sqrt{\frac{1}{ac-b^2}(c\mu^2 - 2b\mu + a)} = \sqrt{99.41\mu^2 - 0.4865\mu + 9.593 \cdot 10^{-4}},$$

在 (σ, μ) 坐标图上的曲线如下



同时，我们也可以通过穷举列出一些加和为 1 的权重向量，让后计算按照这些权重配置出的资产组合的标准差和收益预期，并将这些数据也画在坐标图上，得到下面的图。图中的蓝线是之前算出的有效前沿，圆圈是穷举出的组合，五个蓝色三角形对应的是五个原生股票。



可以看出，所有的组合都在有效前沿的右边，并且标准差最小的组合正好就在有效前沿上。
下面我们想计算整个有效前沿上风险最小的投资组合。根据之前的分析，我们知道最小标准差组合的收益期望是

$$\mu_{\min} = \frac{b}{c} = 2.447 \cdot 10^{-3}.$$

那么建立马科维兹优化问题

Unknown environment 'align'

我们用公式 (††) 先计算出拉格朗日乘子的值

Unknown environment 'align'

然后再使用公式 (†) 计算出权重向量

Unknown environment 'align'

这个向量代表着五支股票配置的权重比例，其中第五支股票的权重是负数，说明需要做空。

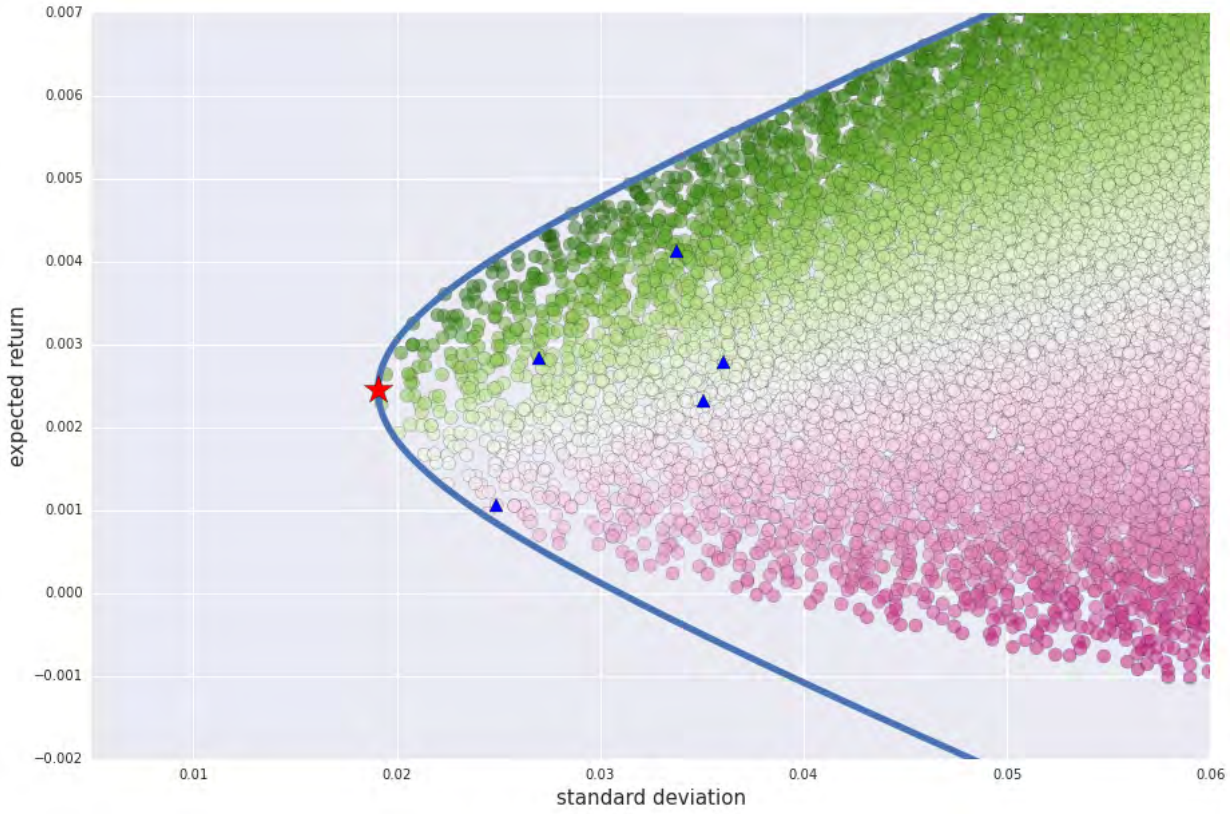
接下来计算这个组合的期望收益和标准差，有

Unknown environment 'align'

以及

Unknown environment 'align'

将 $(1.908 \times 10^{-2}, 2.477 \times 10^{-3})$ 的点画在坐标图上，是下图中的红色猩猩，



它的确是在整个有效前沿曲线的最顶端，这就对了。

结语

本文使用拉格朗日乘子法解决了没有无风险资产的 MPT 模型的优化问题，算出了最小方差组合以及有效前沿还有的闭合式公式，以此进行计算要比穷举和使用蒙特卡罗法更快也更准确。在本系列的下一篇文章中，我们将使用拉格朗日乘子法来解决有无风险资产情况，计算出资本市场线的闭合公式，分析它和有效前沿的关系，并得到市场组合的计算方法。

v1.0，2017-01-19，文章上线