真格量化完整策略学习——期货进阶篇（二）——多项式拟合策略开发流程

一个策略从想法到实现，究竟是一个什么流程呢？许多初入门量化交易的“小白”对策略的开发过程并不熟悉。因此，我们以“多项式拟合策略”为例，展示整个策略的开发流程。

策略的产生，一般来说是数据驱动的。通过对金融市场数据、图表的观测，发现其中可能含有的半定量规律。这些通过数据产生的想法就是一个策略的萌芽。接下来，我们需要思考这种“规律”是否有背后的逻辑，通常我们喜欢采用有较为明确的经济含义的策略；但随着数据挖掘技术的演进，一些仅依靠相关性做出的策略也有可能得到很好的表现。下面是对整个策略做定量化处理，便于其工程实现；定量化处理的过程中，要设置策略可供调整的参数，分析策略可能存在的风险因素，并将策略算法化。最后，是将数量化策略转化为工程代码，并处理可能出现的代码问题等。

一、策略想法与基本逻辑

“历史会重复”是许多量化从业人员的一个基本信念。换言之，当我们对于历史信息有了足够充分的了解后，我们就可以对于未来的可能性有了更加精确的预判。期货价格序列是最直接体现市场参与者观点的交易数据，同时也是大部分预测模型的预测目标，因此值得量化交易爱好者去重点研究。

在观察期货数据 K 线图时，我们常常能够发现，在某一段时间内期货价格是有趋势性的，但是在短期内却有较大的波动。因此我们可以设想，期货的价格数据是在一只“看不见的手”的引导下在附近做随机游走。这只“看不见的手”可能是当前的期货标的现货市场行情，可能是今天的天气，甚至有可能是复仇者联盟 4 的上映，但是这些都是确定性的，是含有金融市场信息的因素。而随机游走就是不确定性的因素，反映了短期内多空双方的博弈和交易者的心理因素。这种想法就自然而然地衍生出一个策略：我们能否将短期的波动“平滑化”，这样我们就可以专注于期货的趋势。从而我们可以在上涨趋势下多头持仓，下跌趋势下空头持仓，获取较大收益。

那么，我们应当采用什么方法处理信息和噪声的耦合呢？如果我们能将价格序列中所有有用的信息提取出来，那么理论上我们可以对未来趋势得到精准的预测。联想到计算数学中的插值拟合方法：有限样本点的情形下我们总是可以利用某个多项式函数完美逼近这些样本。如果降低这个多项式函数的次数，那么我们拟合出来的就是某种“模式”，这种模式就对应了金融市场当前的信息。而样本偏离这个多项式的部分，就是所谓的“噪声”。根据大数定律，当样本点足够的时候，我们拟合出的多项式曲线与真实曲线间的差异将可以达到足够小。因此我们设想，多项式拟合或许可以起到价格序列的“平滑”作用。

接下来，我们将我们的想法进行定量化处理。这里我们不加证明地简要介绍此策略的理论基础，对于理论不感兴趣的读者可以直接跳过这一部分。

我们假设期货价格数据具有如下的形式：

$$p_t = f(p_{t-1}, t) + \varepsilon_t$$

其中，p_t 为 t 时刻期货的价格，$\{\varepsilon_t\}_{t=0}^{\infty}$ 是白噪声过程，即零均值的正态分布。利用这种关系，我们就把期货的价格数据进行了分离，$f(p_{t-1}, t)$ 是信息项，ε_t 是噪声项。

事实上，这是一个递推函数，当给出期货的初始价格 $p_0 = P_0$（通常是当天的开盘价）时，利用递推的方式可以得到多层复合函数如下：

$$p_t = f(p_{t-1}, t) + \varepsilon_t = f(f(p_{t-2}, t-1) + \varepsilon_{t-1}, t) + \varepsilon_t = \cdots = f(f(\cdots f(P_0, t_0) \cdots, t-1) + \varepsilon_{t-1}, t) + \varepsilon_t$$

容易看出，上式的信息项仅和t有关，因此可以改写成为

$$p_t = \xi(t) + \mu_t$$

这里$\{\mu_t\}$仍为噪声序列，但是不再符合正态分布。但是，如果$f(p,t)$是一个多项式函数，那么μ_t就可以表示为$\{\varepsilon_t\}$及其整数方幂的线性组合，从而μ_t仍然满足正态分布，便于我们的处理。当价格数据满足弱平稳条件时，我们可以将一般函数进行泰勒级数展开，并舍去高阶项。而泰勒级数就是多项式函数，因此误差项服从均值为μ，方差为σ^2的正态分布。这便为我们多项式拟合奠定了理论基础。

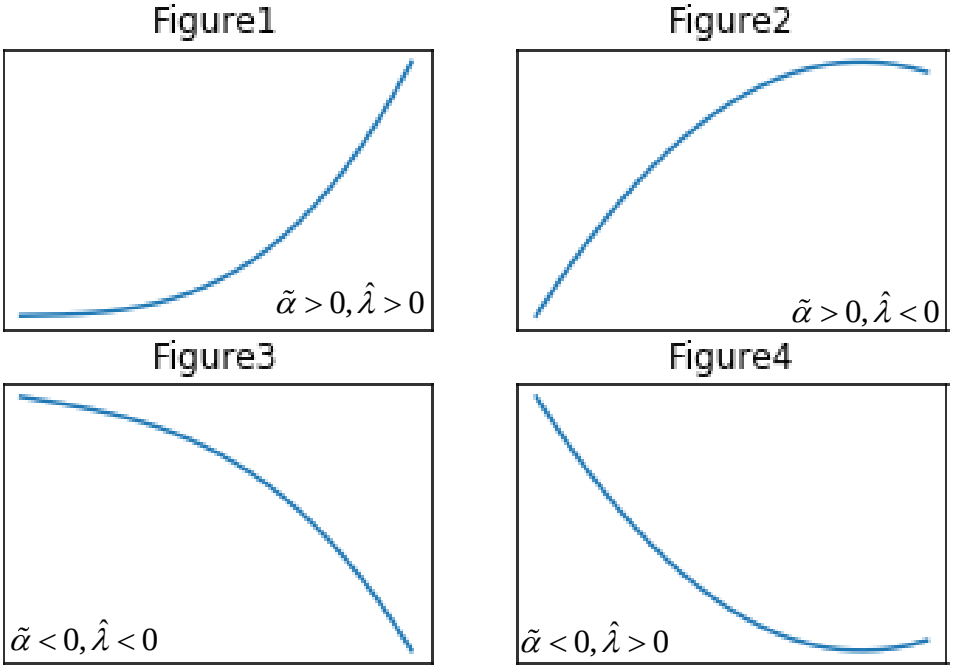
二、策略的实现方法

通过上述理论分析，我们知道，短时间的期货价格数据与时间之间存在着函数关系。因此，将p_t对$1, t, t^2, \cdots, t^n, \cdots$进行拟合，可以得到期货价格的短期预测曲线。一般来说，我们选取有经济含义的拟合方式，而忽略更高阶次。在本策略中，我们采用了两种拟合方式，一次拟合和二次拟合。即

$$p_t = \tilde{\alpha} t + \tilde{\beta}$$

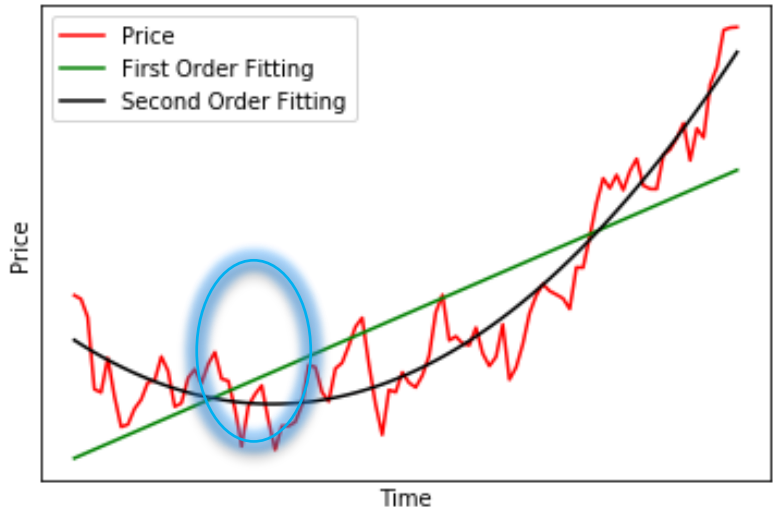
$$p_t = \hat{\lambda} t^2 + \hat{\alpha} t + \hat{\beta}$$

一次拟合的经济含义在于寻找当前趋势：$\tilde{\alpha} > 0$表示当前期货价格总体走高（在这里已经中和了随机项的影响）；$\tilde{\alpha} < 0$表示当前价格持续走低；二次拟合的经济含义在于寻找当前趋势的快慢：$\hat{\lambda} > 0$表示当前上升趋势有加快态势或当前下降趋势有减弱态势；$\hat{\lambda} < 0$表示当前下降趋势有加快态势或当前上升趋势有减弱态势。我们可以利用示意图区分这四种不同的状态：

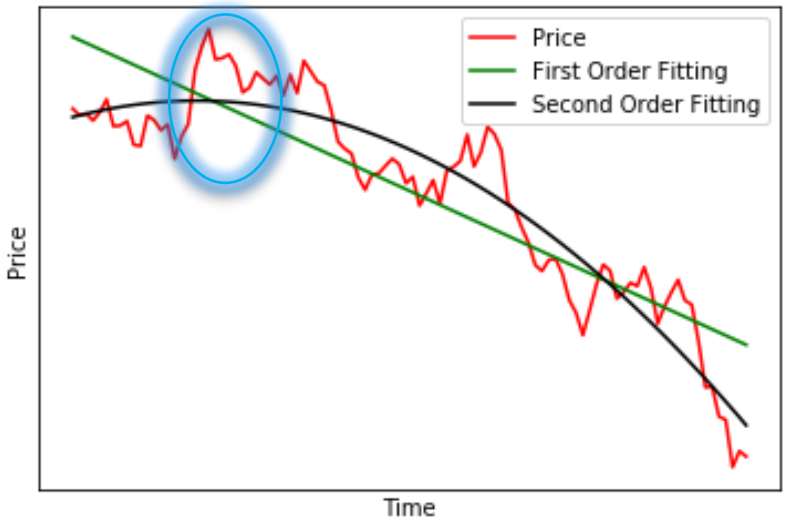


开仓与平仓条件如下：

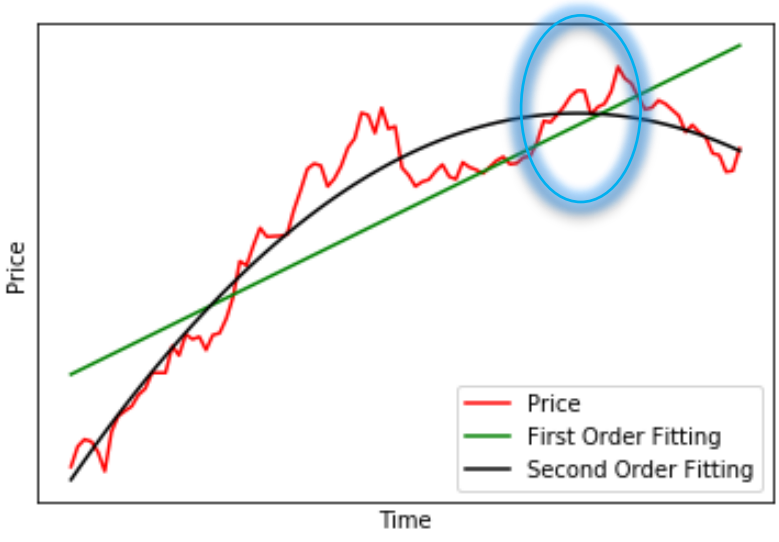
（1）当$\tilde{\alpha} > 0$，$\hat{\lambda} > 0$时，当前期货正处于加速上升状态，多头买入；



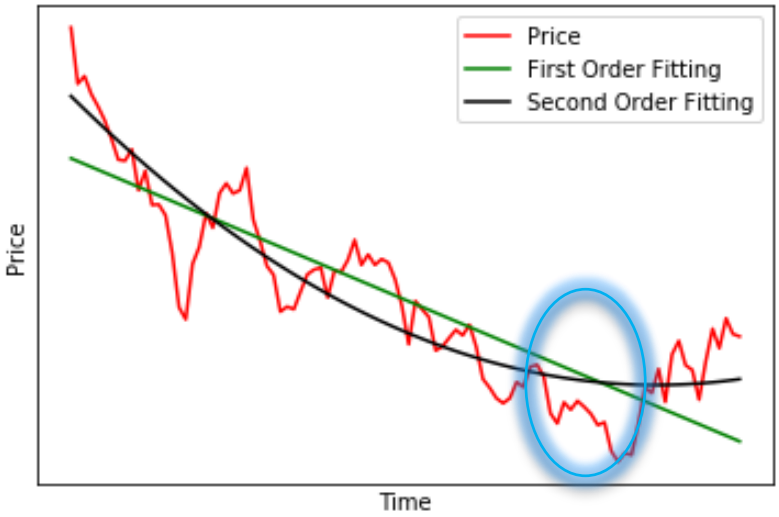
（2）当$\tilde{\alpha}<0$，$\hat{\lambda}<0$时，当前期货正处于加速下降状态，空头买入；



（3）当$\tilde{\alpha}>0$，$\hat{\lambda}<0$时，当前期货正处于减速上升状态，多头平仓；



（4）当$\tilde{\alpha}<0$，$\hat{\lambda}>0$时，当前期货正处于减速嘉奖状态，空头平仓。



总体而言，多项式拟合策略就是在追逐加快的趋势。读者可以很容易地看出，拟合的过程实际上是降噪的过程，因为每一个价格序列的噪音项被长期趋势“平滑“掉了。

但是，为了使得策略效果表现优异，我们拟合的时间长度应当选取适当。拟合时间过短，根据大数定律，随机项的影响仍然较为明显，所得到的态势可能有误；拟合时间过长，可能会导致没有足够的时间获得趋势收益，从而减小预期的收益。入场的时间点一般选取开盘之后 1-2 小时内，具体的时间可以通过回测调参获得。

三、策略的工程实现

（一）策略整体框架

首先，我们应当明确整个策略的基本框架。一个完整的策略应当能够初始化参数，并在实时行情到来的时候处理行情数据并完成相应的策略。因此，我们的策略基本框架是这样的。

```
#开始这个策略，用于初始化一些参数
def OnStart(context) :
    return

#实时行情事件，当有新行情出现时调用该事件
def OnBar(context,code,bartype) :
    return
```

（二）策略核心逻辑的实现

首先，我们介绍策略核心逻辑的工程实现。这个策略实质就是在不断监控实时趋势，当监测到加速型趋势的时候开仓，当该趋势趋于平缓时及时平仓。为了保证我们的实时监测，交易策略主要是在 Onbar 函数中实现的。Onbar 函数，顾名思义，就是在每一个 K 线行情到来的时候自动调用这一个函数。注意 Onbar 是根据根据订阅 K 线的频率触发的：比如我们选择订阅的是 3 分钟 K 线（见上图 BarType.Min3），则每根 3 分钟 K 线刚出现的时候,调用 Onbar。

完成 Onbar 事件时，首先要过滤到我们不需要的信息，比如其他合约的 K 线信息、其他账户的信息、检测的时间等等。在这个策略中，我们只关注银主力合约（即 g.code 内的内容），且只有当前可交易（g.initial_flag = 1）的时候才监测实时行情。

```
#过滤掉不需要的行情通知
if code != g.code:
    return
#开盘前、收盘后不再进行交易
if g.initial_flag == 0:
    return
```

一般来说，Onbar 函数中要完成调仓（即资产的转换）、择时入场、出场和仓位控制部分。下面我们来一一介绍如何完成这些功能。

1、 调仓

在这个策略中，我们选用了银主力合约作为持仓，那么为了保证流动性、控制整体风险，当主力合约发生变更的时候，要及时改变我们的持仓标的。

那么问题就是，如何知道我们的主力合约发生了变化呢？我们当然可以在 Onbar 事件中不断监测当前的主力合约，但是主力合约一般都在白盘和夜盘行情初始化的时候变更，因此我们不需要实时监听主力合约。为此，我们需要新建一个行情初始化事件 OnMarketQuotationInitialEx。在每次白盘或者夜盘初始化的时候监测当前主力合约是否变更。

```
def OnMarketQuotationInitialEx(context, exchange,daynight):
    if exchange != 'SHFE':
        return

    codel = GetMainContract('SHFE','ag',20)

    if g.code!= codel:
        g.change_flag = 1
        UnsubscribeBar(g.code,BarType.Min3)  #取消订阅原K线数据
        g.code = codel  #变更合约
        SubscribeBar(g.code, BarType.Min3) #订阅K线数据，用于驱动OnBar事件
        print "当前交易合约为"+str(g.code)
```

需要注意的是，当主力合约发生变更（即 g.code!=codel）的时候，我们要取消订阅原品种的行情数据，取而订阅新品种的行情数据。同时，注意到我们并没有将原来的持仓平仓，因此设计一个标记 g.change_flag = 1 提醒我们在 Onbar 中进行调仓处理。

在 Onbar 中，如果接收到了 g.change_flag = 1，那么说明需要进行调仓。将之前的仓位（记录在 g.code_position 中）平掉（注意多头和空头两个方向），并注意及时改变相关参数的值。如下所示

```
#换期时平掉之前的仓位
if g.change_flag == 1:
    QuickClosePosition(context.myacc,g.code_position,'buy',
                       max_volume=-1, today_first=True)
    QuickClosePosition(context.myacc,g.code_position,'sell',
                       max_volume=-1, today_first=True)
    g.change_flag = 0
    g.code_position = g.code
```

2、 择时入场

择时入场是期货交易的精髓。一般来说，我们要通过资产的历史数据获取交易信号，择时入场。在这个策略中，我们的入场信号按照如下方式确定：

对资产在 10：30 到当前的开盘价时间序列 $\{p_t\}_{t=0}^{T}$ 对时间戳进行一阶和二阶拟合，得到

$$p_t = \tilde{\alpha}t + \tilde{\beta}$$

$$p_t = \hat{\lambda}t^2 + \hat{\alpha}t + \hat{\beta}$$

如果满足如下条件之一：

（1）$\frac{dp_t}{dt} = \tilde{\alpha} > \alpha_0$ 且 $\frac{d^2 p_t}{dt^2} = 2\hat{\lambda} > 2\lambda_0$

（2）$\frac{dp_t}{dt} = \tilde{\alpha} < -\alpha_0$ 且 $\frac{d^2 p_t}{dt^2} = 2\hat{\lambda} < -2\lambda_0$

其中 $\alpha_0 > 0, \lambda_0 > 0$ 是给定的两个阈值，一般根据市场结构的调整而发生变化。

则持仓入场。在情况（1）下多头持仓，在情况（2）下空头持仓。

严格来说，对于时间序列数据的拟合问题，必须要考虑其自相关性，但是我们在这里只考虑其变动趋势，不考虑其他因素。因此，我们获取当前标的历史分钟线数据，并进行拟合。

```
#获取当天相关数据
dyndata = GetQuote(g.code)

df = GetHisDataAsDF(code,bar_type = BarType.Min3,start_date = start_time(),
                    end_date = GetCurrentTime())

price = np.array(df["open"])

#多项式拟合
Y = price - np.mean(price)  #去中心化处理,作为拟合用的因变量

X = np.linspace(1,len(Y),len(Y)) #拟合用的自变量

poly_1 = np.polyfit(X,Y,deg=1) #一阶拟合
poly_2 = np.polyfit(X,Y,deg=2) #二阶拟合

print "拟合结果："
print poly_1,poly_2
```

对价格序列的处理一般采用去中心化或者取对数等。通过调用 numpy 中的 polyfit 函数，可以快速获得拟合系数 poly_1 和 poly_2。接下来，获取我们需要的参数（即 poly_1 和 poly_2 列表中的第一个参数），并与我们设定的阈值进行比较。若满足开仓条件，则进行开仓处理。具体流程如下。

```
#在可交易时间段中,且当天没有进行过交易
if time_now == "10:57:00" or time_now == "11:00:00" and g.has_contract == 0:
    if poly_1[0]>3 and poly_2[0]>0.03:        #加快增长趋势
        QuickInsertOrder(context.myacc,g.code,'buy','open',
                         dyndata.now,200)    #开多头
        g.has_contract = 1
        print "多头加速，入场"

    elif poly_1[0]<-3 and poly_2[0]<-0.03:     #加快下跌趋势
        QuickInsertOrder(context.myacc,g.code,'sell','open',
                          dyndata.now,200)    #开空头
        g.has_contract = 2
        print "空头加速，入场"
```

在这里，我们控制交易时间是在 10：57 或者 11：00，这个入场时间参数是可以自行调整的。注意要加入识别标记，即 g.has_contract = 0，表示当前不持有任何合约。当开多仓时，g.has_contract = 1；当开空仓时，g.has_contract = 2。

3、 出场

3.1、主动出场

出场策略也是基于相应的信号。在这里，我们设置的信号是价格趋势出现了反转，即：

（a）若以 1 方式开仓，则 $\frac{dp_t}{dt} = \tilde{\alpha} < \alpha_1$ 或 $\frac{d^2 p_t}{dt^2} = 2\hat{\lambda} < 2\lambda_1$ 时平仓；

（b）若以 2 方式开仓，则 $\frac{dp_t}{dt} = \tilde{\alpha} > -\alpha_1$ 且 $\frac{d^2 p_t}{dt^2} = 2\hat{\lambda} > -2\lambda_1$ 时平仓。

其中 $|\alpha_1| < |\alpha_0|, |\lambda_1| < |\lambda_0|$ 是给定的两个阈值，一般根据市场结构的调整而发生变化。

按照同样的逻辑，可以写出如下代码，其中要注意平仓数量时当前账户中持有的数量。

```
#当天进行过交易之后，寻找合适的出场信号
if g.has_contract == 1:
    if poly_2[0]<=0 or poly_1[0]<=0.2:    #增长速率减缓

        g.buy_flag = 4  #平多仓标志
        position = context.myacc.GetPositions()
        volume = 0
        for pos in position:
            volume += pos.volume
        QuickInsertOrder(context.myacc,g.code,'sell','close',
                         dyndata.now,volume)    #平仓
        g.has_contract = 0
        print "多头减速，出场"

if g.has_contract == 2:
    if poly_2[0]>=0 or poly_1[0]>=-0.2:    #下跌趋势减缓

        g.sell_flag = 4  #平空仓标志
        position = context.myacc.GetPositions()
        volume = 0
        for pos in position:
            volume += pos.volume
        QuickInsertOrder(context.myacc,g.code,'buy','close',
                         dyndata.now,volume)    #平仓
        g.has_contract = 0
        print "空头减速，出场"
```

3.2、被动出场

写完上述代码之后，我们需要思考：这段代码就能够实现一个日内策略了吗？显然是不能的，因为如果全天的数据都无法满足平仓条件，则该期货将会持有至下一天。为了避免这种情况，我们需要在 14：45 时做平仓设置，将未平仓的合约全部平掉，如下图。由于持仓手数较大，为了保证能够平仓，不发生隔夜仓，因此将报价进行一定的调整。

```
if time_now == "14:45:00" and g.has_contract == 1:

    position = context.myacc.GetPositions()
    volume = 0
    for pos in position:
        volume += pos.volume
    QuickInsertOrder(context.myacc,g.code,'sell','close',
                     dyndata.bidprice(0) - 1*g.step_unit,volume)    #平仓
    print "收盘平仓"
    g.has_contract = 0


if time_now == "14:45:00" and g.has_contract == 2:

    position = context.myacc.GetPositions()
    volume = 0
    for pos in position:
        volume += pos.volume
    QuickInsertOrder(context.myacc,g.code,'buy','close',
                     dyndata.bidprice(0) + 1*g.step_unit,volume)    #平仓
    print "收盘平仓"
    g.has_contract = 0
```

聪明的读者可能发现了，编写策略的时候有非常多的细节需要我们去考虑。一旦有一招不慎，可能会导致满盘皆输，将一个很好的交易想法变成了一个亏钱的策略。

4、仓位控制

仓位控制在高风险产品交易中是一个重要的课题。比如在这个策略中，买入合约的数量一定是 200 手吗？这个数量是否应当与当前趋势强烈度有关呢？平仓的时候是否一定要全部平仓呢？还是根据当前参数改变量设置平仓窗口？仓位的控制一般能够极大的改善策略的回测指标，从而大大降低系统性风险的发生，但同时可能会小幅减小预期收益率。

关于这部分的实现，就留给读者自己开发。

（三）交易初始化

完成多项式拟合策略时，我们需要对参数进行初始化、登陆交易账户，同时设置我们的投资标的（银主力合约）。这些功能在 Onstart 函数中实现。

首先，我们先登陆账户选用的是期货交易，并选择“回测”账号。

```
context.myacc = None
if context.accounts.has_key("回测期货") :
    print "登录交易账号[回测期货]"
    if context.accounts["回测期货"].Login() :
        context.myacc = context.accounts["回测期货"]
```

接下来，我们就要选择投资标的了。由于期货品种、数量都较为单一，因此我们选用了银的主力合约作为我们的投资标的。在策略开发过程中，读者也可以采用其他技术进行标的选取。

```
#选取主力合约作为交易合约
g.code = GetMainContract('SHFE','ag',20)
g.code_position = g.code
print g.code
```

在这里需要提醒的是，合约的代码应当保存在全局变量中，因为在后续的交易过程中，合约的代码会经常用到。由于主力合约可能会发生更换，因此我们常常需要在另一个全局变量（在这里是

g.code_position）中储存当前主力合约的代码，便于未来更换合约时候进行处理。

接下来，我们需要订阅交易合约的 K 线数据。如果没有订阅当前合约的 K 线数据，那么是无法完成下单的。

```
#订阅K线数据，用于驱动OnBar事件
SubscribeBar(g.code, BarType.Min3)
```

最后，我们需要在 Onstart 函数中初始化一些参数。这个部分随着后续策略的完善而随时更新。比如，在我们的策略中设置了四个全局参数。

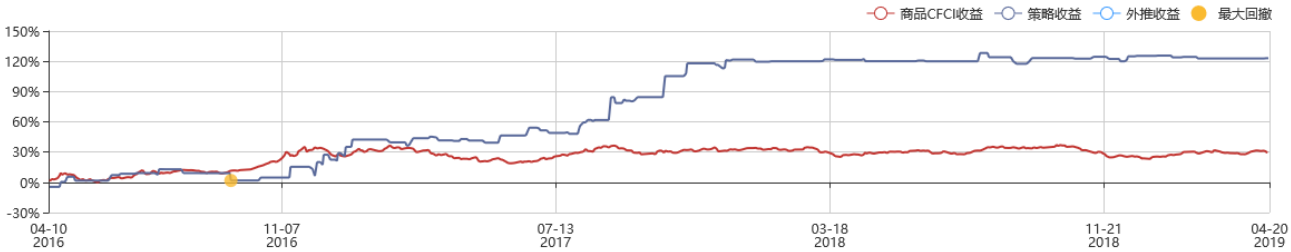
```
g.change_flag = 0 #用于判断是否发生了主力合约的更换
g.has_contract = 0 #判断当前是否持仓
```

到此，我们的 Onstart 函数就基本上完成了。将 Onstart 函数和 Onbar 函数结合起来，这个策略在分钟线上就可以完成回测了。

四、策略总结与回测

下面我们通过分钟线回测观察一下策略表现。

年化收益达到 30.31%，最大回撤为 9.90%，夏普比率为 1.26，信息比率为 0.90，总体来说还是一个比较不错的策略。当然，读者可以通过风控手段进一步增加策略的夏普比率。



到此为止，我们已经完成了多项式拟合策略的全部开发流程。读者可以仔细回顾一下，我们是如何通过观测 K 线找到想法，并将这种想法进行数量化，最终完成代码开发的。