量化投资 实践篇(1) ¶

MVO(上)

> 在量化投资技术篇中我们介绍了Markowitz的现代投资理论中MVO(Mean Variance Optimization)的基本原理；在时序分析系列文章中我们也介绍了一些时序预测技术。在本实践篇中，我们将开始使用MVO技术做资产组合配置的优化。

首先，让我们来简要回顾一下MVO技术：

MVO在实践中一般可以转化为如下问题，

- 假设有n个资产，试图求解资产的权重为向量W
  $$min_W \; W^T \Sigma W - W^T \mu / \gamma$$
  $$\text{条件为 } W \in C$$
  式中Σ为资产的协方差矩阵，$W^T \Sigma W$是组合收益的方差，μ为资产的期望收益率向量，γ为风险厌恶系数，$W^T \mu$则为组合期望收益率，C为权重的约束条件。
  该问题成为一个有约束的二次规划问题。

从上式中，我们可以看到实际上MVO就是最小化风险与收益的差，当风险恒定时，收益越大越好；当收益恒定时，风险越小越好。

- MVO的核心要素有三个
  期望收益率 μ
  协方差矩阵 Σ
  约束条件 C

我们先来看一下约束问题，即资产权重应该满足某种限制条件，例如在实际做资产配置组合中，很可能资产不能做空，这意味着权重必须大于0；又比如基于风控考虑，任何资产的最少比例不能小于5%。除此之外，约束条件还有很多种，例如正则化约束、交易约束、政策约束、参数化组合约束、风险暴露约束等等。

对于期望收益率和协方差，我们希望其估算误差越小越好。

在本实践中，我们采用如下方法来估算期望收益率和协方差矩阵。
期望收益率：KDE+Monte Carlo Sampling 协方差矩阵: EWMA
约束条件：$W_i > 0.05$
令$\gamma = 1$

导入python包，并得到收益率数据

In [3]:

```
import warnings
warnings.simplefilter('ignore')
```

In [4]:

```
import pandas as pd
import numpy as np
%matplotlib inline

from fintechtools.backtest import *
from fintechtools.datasource import *
from fintechtools.SimuMultiTest import *
#from lib.portfolio import DailySimulator
#from lib.experiment import Experiment

import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
from arch import arch_model
#sns.set_context("talk")
import matplotlib
import matplotlib as mpl
from matplotlib.ticker import FuncFormatter
#mpl.style.use('classic')

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['font.serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
import seaborn as sns
sns.set_style("whitegrid",{"font.sans-serif":['simhei', 'Arial']})
sns.set_context("talk")

#zhfont1 = matplotlib.font_manager.FontProperties(fname='C:\Users\ktwc37\Documents\ZNTG\notebooks\S


%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [5]:

```
pv = load_csv(file_path='./data/all_funds_2017-03-17.csv')
start = '2012-01-01'
end = '2017-03-01'
```

In [6]:

```
pv.dropna(axis=1,how='all',inplace=True)
pv.dropna(how='all',inplace=True)

pv_sub = pv.loc[start:end,:].copy()
pv_sub.dropna(axis=1,thresh=30,inplace=True)

pv_sub.fillna(axis=0,method='ffill',inplace=True)
pv_sub.fillna(axis=0,method='bfill',inplace=True)
```

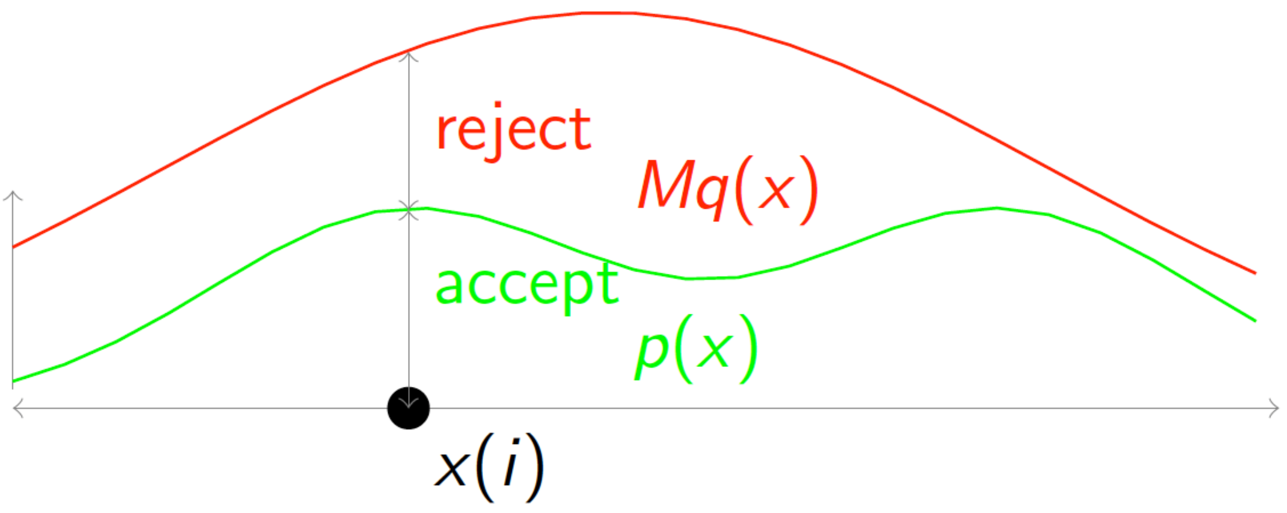
In [7]:

```
indexs = pd.read_excel('./data/华夏指数.xlsx')
indexs_pv = indexs.pivot_table(index='日期', columns= '简称', values= '收盘价(元)')
indexs_pv.index = pd.to_datetime(indexs_pv.index, unit='d')
```

期望收益率估算

在时序分析系列中，我们介绍了多种收益率估算方法。这一次我们采用KDE来平滑资产的历史收益率的概率分布，并采用蒙特卡洛采样从平滑后的收益率概率密度函数中得到预期收益率。
我们将针对周收益率进行建模。
我们将比较不同的蒙特卡洛采样方法，包括：

- Direct : 直接从经验公式计算
  将历史收益的均值作为期望收益率。
- PDF Sampling:
  KDE平滑后得到的概率密度分布中采样计算均值作为期望收益率。
- Monte Carlo Inverse CDF
  计算机模拟技术只能对均匀分布进行采样，而资产的收益分布肯定不是均匀分布，这会导致出现较大的采样误差。但连续型随机变量的累积分布函数是在[0,1]之间呈现均匀分布的。 简单来说，这种方法就是当已知随机变量的CDF（累积分布函数)后，我们可以得到其反函数CDF^{-1}。该函数的定义域是均匀分布的，我们在其上采样而得到的分位数的抽样分布和原随机变量的分布是一致的。 细节请参见 https://en.wikipedia.org/wiki/Inverse_transform_sampling (https://en.wikipedia.org/wiki/Inverse_transform_sampling)
- Monte Carlo Accept Rejection
  当随机变量的累积分布函数比较难求的时候，我们可以选择采用接受拒绝采样来得到其抽样分布。该方法的核心思想是：寻找一个可被采样的函数$q(x)$,它可以完全覆盖待采样的函数$q(x)$，或者我们说$M * q(x) \geq p(x)$，其中$p(x)$为待采样函数，M为一定值，如下图所示



重复如下步骤，直到获得m个被接受的采样点
1)从$q(x)$中获得一个随机采样点x_i
2)对于x_i计算接受率

$$\alpha = \frac{p(x_i)}{Mq(x_i)}$$

3)从均匀分布Uniform(0,1)中随机生成一个值μ
4)如果$\alpha > \mu$，则接受x_i作为一个采样点，否则拒绝x_i
5)重复
此方法的主要缺点是很多时候拒绝率太高，致使整个过程收敛太慢。这时候我们需要采用Gibbs采样或者自适应拒绝采样等。这些并非本文所讨论的范围。

- 计算周收益率

In [8]:

```
index_ret_w = indexs_pv.dropna().resample('W').apply(lambda x: (x[-1] - x[0])/ x[0] if len(x) > 0 e
index_ret_w.dropna(inplace=True)
```

```
- KDE平滑
采用scott准则估算bandwidth参数
```

In [10]:

```
fund_ret = index_ret_w[['中证全指']].as_matrix()
fund_ret = fund_ret[fund_ret != 0]

from scipy import stats
from scipy.stats import norm

kde1 = stats.gaussian_kde(fund_ret)

print(kde1.factor)
```

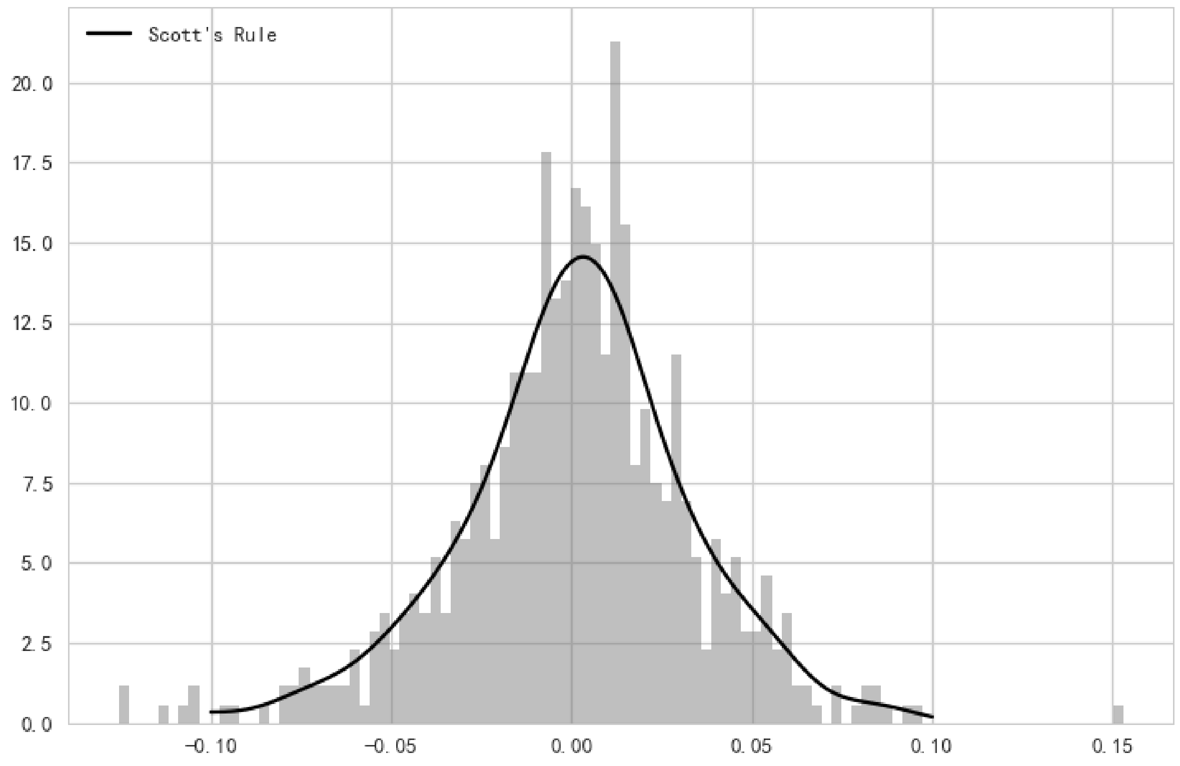
```
0.2760343198607585
```

在（-0.1，0.1）之间选取100000

In [15]:

```
x_grid = np.linspace(-0.1, 0.1, 100000)
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)

ax.plot(fund_ret, np.zeros(fund_ret.shape), 'b+', ms=20)  # rug plot
ax.hist(fund_ret, 100, fc='gray', histtype='stepfilled', alpha=0.5, normed=True)
Scott = ax.plot(x_grid, kde1(x_grid), 'k-', label="Scott's Rule")
plt.legend(loc='upper left')
plt.show()
```



In [13]:

```
pdf1= kde1.pdf(x_grid)/sum(kde1.pdf(x_grid))
cdf1 = np.cumsum(pdf1)
```

```
直接计算法得到
周收益率均值：0.000846
周收益率方差：0.001095
月收益率均值：0.003390
```

In [16]:

```
def method_theory(x,pdf,interval):
    # 理论公式计算周收益率期望
    ex = np.dot(x, pdf)/np.sum(pdf)
    ex2 = np.dot(np.square(x), pdf) / np.sum(pdf)
    return ex
def method_sampling(x,bw,repeat=100000):
    # pdf采样计算周收益率期望
    kde = stats.kde.gaussian_kde(x, bw_method=bw)
    sample = kde.resample(repeat)
    return np.mean(sample)
def cal_month(ex,interval=4):
    # 从周收益率期望计算月收益率期望
    return (1+ex)**interval - 1
```

In [17]:

```
E_week = np.mean(fund_ret)
print("Week mean=",E_week)
print("Week variance=",np.var(fund_ret))
```

```
Week mean= 0.0008463509485831558
Week variance= 0.0010954534664965557
```

In [18]:

```
E_month = (1+E_week)**4-1
print("Month mean=",E_month)
```

```
Month mean= 0.0033897040794133027
```

In [19]:

```
method_compare_df = pd.DataFrame()
method_compare_df['Direct'] = np.array([E_week,E_month])
method_compare_df.index = ['Week','Month']
```

In [20]:

```
method_compare_df
```

Out[20]:

	Direct
Week	0.000846
Month	0.003390

KDE估算法得到
周收益率 scott : 0.00156
月收益率 scott : 0.00625

In [21]:

```
dic = {'Scott':[pdf1,cdf1,kde1]}
output_week = {key:method_theory(x_grid,value[0],4) for key,value in dic.items()}
output_month = {key:cal_month(value,4) for key,value in output_week.items()}
print(output_week)
print(output_month)
```

```
{'Scott': 0.0015592206938296602}
{'Scott': 0.006251484959179043}
```

In [22]:

```
method_compare_df['KDE Scott'] = np.array([0.00156,0.00625])
```

In [24]:

```
method_compare_df
```

Out[24]:

	Direct	KDE Scott
Week	0.000846	0.00156
Month	0.003390	0.00625

PDF采样法得到
周收益率 scott: 0.000968
月收益率 scott: 0.003878

In [25]:

```
Scott_ = np.mean(kde1.resample(100000))
kde_t = stats.kde.gaussian_kde(fund_ret, bw_method=0.002)
sample = kde_t.resample(100000)
output_week_resampling = {'Scott':Scott_}
output_month_resampling = {key:cal_month(value,4) for key,value in output_week_resampling.items()}
print(output_week_resampling)
print(output_month_resampling)
```

```
{'Scott': 0.0007088847020347067}
{'Scott': 0.0028385553384238094}
```

In [26]:

```
method_compare_df['PDF Sampling Scott'] = np.array([0.000968,0.003878])
```

In [27]:

```
method_compare_df
```

Out[27]:

	Direct	KDE Scott	PDF Sampling Scott
Week	0.000846	0.00156	0.000968
Month	0.003390	0.00625	0.003878

Monte Caelo 方法（Inverse CDF)

In [28]:

```
def inverseCDF(x,cdf,repeat=100000):
# random sampling from (0,1)
    np.random.seed(123)
    rs = np.random.uniform(0,1,repeat)
# Calculate mean
    value = 0
    for p in rs :
        idx = np.argmin(np.abs(cdf - p))
        idx = idx if cdf[idx] - p < 0 else idx + 1
        #value += ( cdf[idx]-cdf[idx-1 if idx > 0 else 0] ) * x[idx]
        value += x[idx]
    return value/rs.size
```

In [29]:

```
output_week_MCDF = {'Scott':inverseCDF(x_grid,cdf1,cdf1.size)}
output_month_MCDF = {key:cal_month(value,4) for key,value in output_week_MCDF.items()}
print(output_week_MCDF)
print(output_month_MCDF)
```

```
{'Scott': 0.0016101838418384265}
{'Scott': 0.006456308224946472}
```

In [30]:

```
method_compare_df['MCIC Scott'] = np.array([0.00161,0.006456])
```

In [31]:

```
method_compare_df
```

Out[31]:

	Direct	KDE Scott	PDF Sampling Scott	MCIC Scott
Week	0.000846	0.00156	0.000968	0.001610
Month	0.003390	0.00625	0.003878	0.006456

Monte Carlo Accept Rejection方法
采样1000000
Scott 接受率：0.933395

In [32]:

```
def MCAJ(x,pdf,ceiling=0.7,num=1000000):
    randnum=np.random.uniform(0,ceiling,num)
    result=[]
    for axi in range(len(pdf)):
    #print(pdf[axi],"     ", randnum[axi])
        if( pdf[axi] > randnum[axi] ):
            result.append(x[axi])
    #print(np.mean(result))
    #print(np.var(result))
    print('Accept Rate ',len(result)/num)
    return np.mean(result)
```

In [33]:

```
num = 1000000
x_grid_ = np.random.uniform(-0.1, 0.1, num)
#kde3_ = KernelDensity(kernel='gaussian', bandwidth=0.015).fit(fund_ret.reshape(-1, 1))
#pdf3_ = np.exp(kde3_.score_samples(x_grid_[:, None]))
pdf1_ = kde1.pdf(x_grid_)
#pdf1_= kde1.pdf(x_grid_)/sum(kde1.pdf(x_grid_))
```

In [34]:

```
output_week_MCAJ = {'Scott':MCAJ(x_grid_,pdf1_,0.7,num)}
output_month_MCAJ = {key:cal_month(value,4) for key,value in output_week_MCAJ.items()}
```

```
Accept Rate  0.933437
```

In [35]:

```
print(output_week_MCAJ)
print(output_month_MCAJ)
```

```
{'Scott': -0.00024516438515175 17}
{'Scott': -0.0009802969660919203}
```

In [36]:

```
method_compare_df['MCAJ Scott'] = np.array([-0.000315,-0.00126])
```

各种方法比较
周收益率相差在万分之一到万分只八左右
Monte Carlo 接受拒绝采样法与其他方法有较大差距，估计是因为没有找到合适的包络函数所造成的。

```
In  [37]:
```

```
method_compare_df.T
```

```
Out[37]:
```

	Week	Month
Direct	0.000846	0.003390
KDE Scott	0.001560	0.006250
PDF Sampling Scott	0.000968	0.003878
MCIC Scott	0.001610	0.006456
MCAJ Scott	-0.000315	-0.001260

随机抽取50只基金，计算几种方法的RMSE

In [38]:

```
def method_theory(x,pdf,interval=None):
    # 理论公式计算周收益率期望
    ex = np.dot(x, pdf)/np.sum(pdf)
    ex2 = np.dot(np.square(x), pdf) / np.sum(pdf)
    return ex
def method_sampling(x,bw,repeat=100000):
    # pdf采样计算周收益率期望
    kde = stats.kde.gaussian_kde(x, bw_method=bw)
    sample = kde.resample(repeat)
    return np.mean(sample)
def inverseCDF(x,cdf,repeat=100000):
# random sampling from (0,1)
    np.random.seed(123)
    rs = np.random.uniform(0,1,repeat)
# Calculate mean
    value = 0
    for p in rs :
        idx = np.argmin(np.abs(cdf - p))
        idx = idx if cdf[idx] - p < 0 else idx + 1
        #value += ( cdf[idx]-cdf[idx-1 if idx > 0 else 0] ) * x[idx]
        value += x[idx]
    return value/rs.size
def cal_month(ex,interval=4):
    # 从周收益率期望计算月收益率期望
    return (1+ex)**interval - 1
def cal_kpc(x,rule='scott',num=100000):
    x_grid = np.linspace(min(x)-0.01, max(x)+0.01, num)
    if rule == 'GridSearch':
        grid = GridSearchCV(KernelDensity(), {'bandwidth': np.arange(0.001, 0.5, 0.001) }, cv=5)
        grid.fit(x.values.reshape(-1,1))
        kde_ = grid.best_estimator_
        bandwidth = grid.best_params_['bandwidth']
        kde_ = stats.gaussian_kde(x,bw_method = bandwidth)
        pdf_ = kde_.pdf(x_grid)/sum(kde_.pdf(x_grid))
    else:
        kde_ = stats.gaussian_kde(x,bw_method = rule)
        bandwidth = kde_.factor
        pdf_ = kde_.pdf(x_grid)/sum(kde_.pdf(x_grid))
    cdf_ = np.cumsum(pdf_)
    e_week_direct = np.mean(x)
    e_week_theory = method_theory(x_grid,pdf_)
    e_week_pdf_sampling = method_sampling(x,bandwidth)
    e_week_MCDF = inverseCDF(x_grid,cdf_)
    e_week = [e_week_direct,e_week_theory,e_week_pdf_sampling,e_week_MCDF]
    e_month = [ cal_month(x) for x in e_week ]
    def _get_month():
        sample = kde_.resample(4)+1
        return sample.prod() - 1
    e_month_sampling = np.mean([_get_month() for i in range(1,100000)])
    #print(rule,bandwidth,e_week,e_month,e_month_sampling)
    return e_week,e_month,e_month_sampling
def cal_month_sample(x,bw,week_n=4,count=10000):
    kde = stats.kde.gaussian_kde(x, bw_method=bw)
    month_rate = 1
    def _get_month(month_rate):
        sample = kde.resample(week_n)+1
        return sample.prod() - 1
    space = [_get_month(month_rate) for i in range(1,count)]
    #print(space)
    return np.mean(space)
def estimate_(x,xrange,method='Scott'):
    dic_week = { key+'_method_theory':method_theory(xrange,pdf,4),
                 key+'_method_sampling':method_sampling(x,bandwidth),
                 key+'_method_MCDF':inverseCDF(xrange,cdf,cdf.size),
                 key+'_method_MCAJ':MCAJ(xrange,pdf,0.7),
                 }
    dic_month = { k.replace('week','month'):cal_month(v) for k,v in dic_week.items()}
    dic_month['month_sampling_'+method] = cal_month_sample(x,bandwidth,week_n=4,count=10000)
    dic_all = dict(dic_week)
    dic_all.update(dic_month)
    return dic_all
```

In [39]:

```
pv_sub1 = pv.loc['2014-01-01':'2016-4-30',:].copy()
pv_sub1.dropna(axis=1,thresh=30,inplace=True)
pv_sub1.fillna(method='ffill',inplace=True)
pv_sub1.fillna(method='bfill',inplace=True)
pv_sub1.drop(pv_sub1.std()[pv_sub1.std() < 0.000001].index.values,axis=1,inplace=True)
```

In [41]:

```
category = load_category()
stock = get_fund_by_type(pv_sub1,category,type='股票型基金')
pv_stock = pv_sub1[stock]
#pv_stock.head()
```

In [42]:

```
stock_ret_w = pv_stock.dropna().resample('W').apply(lambda x: (x[-1] - x[0])/ x[0] if len(x) > 0 el
stock_ret_w.dropna(inplace=True)
```

随机抽取50支基金 估算基金周收益率和月收益率

In [43]:

```
ri = np.random.randint(1,stock_ret_w.shape[1],50)
RS = stock_ret_w.iloc[:,ri]
```

In [44]:

```
kpc_scott = RS.apply(cal_kpc)
```

In [45]:

```
scott_df = kpc_scott.to_frame(name='info_tuple')
scott_df['week'],scott_df['month'],scott_df['month_sampling'] = zip(*scott_df.info_tuple)
scott_df['week_direct'],scott_df['week_theory'],scott_df['week_pdf_sampling'],scott_df['week_MCDF']
scott_df['month_direct'],scott_df['month_theory'],scott_df['month_pdf_sampling'],scott_df['month_MCD
scott_df.drop(['info_tuple','week','month'],inplace=True,axis=1)
```

In [46]:

```
week_real = pv_sub.loc['2016-05-01':'2016-05-07',RS.columns].apply(lambda x: (x[-1] - x[0])/ x[0] i
month_real = pv_sub.loc['2016-05-01':'2016-06-01',RS.columns].apply(lambda x: (x[-1] - x[0])/ x[0]
```

In [47]:

```
scott_df['week_real'] = week_real
scott_df['month_real'] = month_real
```

In [48]:

```
scott_df.head()
```

Out[48]:

	month_sampling	week_direct	week_theory	week_pdf_sampling	week_MCDF	montl
165310.OF	0.007011	0.001759	0.001761	0.001929	0.001832	0
001410.OF	-0.000106	-0.000052	-0.000065	-0.000086	-0.000007	-0
519032.OF	0.004176	0.001032	0.001015	0.001036	0.001081	0
001631.OF	0.002004	0.000499	0.000501	0.000409	0.000534	0
001592.OF	0.000274	0.000010	-0.000123	-0.000021	-0.000064	0

计算RMSE

In [49]:

```
from sklearn.metrics import mean_squared_error as mse
```

In [50]:

```
week_metrics = ['week_direct','week_theory','week_pdf_sampling','week_MCDF']
month_metrics = ['month_sampling','month_direct','month_theory','month_pdf_sampling','month_MCDF']
```

In [51]:

```
scott_week_mse ={k:mse(scott_df['week_real'],scott_df[k]) for k in week_metrics}
scott_month_mse = { k:mse(scott_df['month_real'],scott_df[k]) for k in month_metrics }
```

In [54]:

```
scott_week_mse.update(scott_month_mse)
scott_mse_df = pd.DataFrame.from_dict(scott_week_mse,orient='index')
scott_mse_df.rename(columns={0:"scott_rmse"},inplace=True)
```

In [58]:

```
#result = pd.concat([scott_mse_df,gridSearch_mse_df],axis=1)
scott_rmse_df = scott_mse_df.applymap(np.sqrt).sort_index()
```

In [60]:

```
scott_rmse_df
```

Out[60]:

	scott_rmse
month_MCDF	0.017818
month_direct	0.017911
month_pdf_sampling	0.017975
month_sampling	0.017870
month_theory	0.017859
week_MCDF	0.011571
week_direct	0.011551
week_pdf_sampling	0.011554
week_theory	0.011537

In [61]:

```
scott_rmse_df.head(5).std()
```

Out[61]:

```
scott_rmse    0.000059
dtype: float64
```

In [62]:

```
scott_rmse_df.tail(4).std()
```

Out[62]:

```
scott_rmse    0.000014
dtype: float64
```

从周收益率上看，从历史收益率上直接计算均值取得了最小的RMSE。
从月收益率上看，蒙特卡洛Inverse CDF获得了最小的RMSE.
当然，因为这里主要是为了演示方法，所以只去了一周的样本外数据进行RMSE的计算，所以不足以说明问题。

未完待续.....