时序分析(12) -- 基于回归的平滑技术

如无特殊说明，本系列文章中的数据将使用2012~2017年，分别代表国内股票、香港股票、国内债卷和国内货币的四个指数数据。

上一篇文章中，我们着重探讨了金融时序分析中应用非常广泛的指数平滑技术。本篇文章我们补充介绍在工业中也起着重要作用的基于回归方法的平滑技术，主要包括：

- Spline Smoothing
- LOESS/LOWESS Smoothing
- Kernel Smoothing

- Spline Smoothing

Spine的核心目标是为了最小化下式：

$$J[s] = \alpha \int (\frac{ds^2}{d^2t})^2 dt + (1-\alpha) \sum_i w_i (y_i - s(x_i))^2$$

式中，s是平滑后的时序，α是混合因子，w_i是权重参数，y_i是实际时序。 仔细观察后可以发现，第二项代表的是平方误差和；而第一项中的二阶导数代表曲线的曲率，也就是说第二项代表平滑程度。

刚才介绍的是最常见的单变量三次Spline平滑，下面我们来深入讨论一下spline平滑的思想：

什么是Spline呢？我们说k阶spline是一个分段k阶多项式连续函数，在其结点处有具有连续的1到$k-1$阶导数。

即函数$f: \mathbb{R} \to \mathbb{R}$为一个$k$阶spline，其拐点为$t_1 < \ldots < t_m$，那么$f$在每一段间隔$(-\infty, t_1], [t_1, t_2], \ldots, [t_m, \infty$是一个$k$阶多项式，且$f$的0到$k-1$阶导数在结点$t1, \ldots, t_m$上连续。

最常见的情况是$k=3$，也就是刚才所讲的cubic spline.

当然，spline平滑不仅可以应用于单变量，也可应用于多变量。

实践中，spline的参数多是通过交叉验证得到的。

- LOESS/LOWESS Smoothing

spline具有全局优化目标，造成其对局部细节信息不敏感。而LOESS(Local Weighted Regression Spline)解决了这个问题。

LOESS通过低阶多项式(通常是线性的)回归来近似数据的局部特征，其权重分配的原则是离待平滑点近的点的权重要高于距离较远的点，这种方式称为Local Weighting.

让我们先考虑一个简单的一阶LOESS:

本地近似采用线性形式，$a + bx$，我们需要通过最小化下式，

$$\chi^2 = \sum_i w(x - x_i; h)(a + bx_i - y_i)^2$$

来找到参数a, b.
这里，w是权重函数，它应该是平滑的且具有很强的尖峰。基本上是一个核函数(关于核函数，我们将在下一节讨论)。
在LOESS中，经常采用的核函数为
$K(x) = (1 - |x|^3)^3$，当$x < 1$时；否则，$K = 0$。
h控制了kernel的宽度。

以上就是LOESS的基本思想，我们可以看到这个方法时很容易扩展的，例如更改$a + bx$为高阶关系或者更改核函数等。
需要注意的是，LOESS是计算敏感的，每一次计算平滑值时都需要所有点参与计算。

- Kernel Smoothing

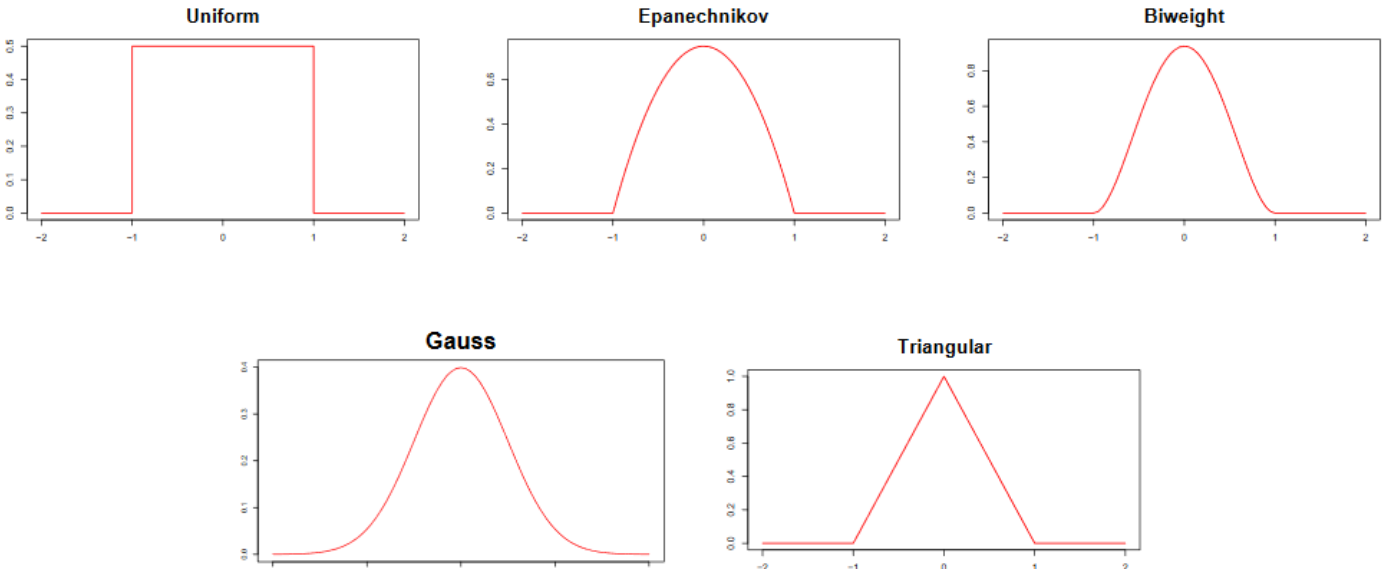
Kernel Smoothing又称为KDE(Kernel Density Estimator),它与前面所讲的平滑技术有所不同。一般的平滑技术都是用来平滑时序数据的观测值，而KDE多用来对某随机变量的概率密度函数进行平滑和降噪。
前面提到过，核函数一般是一个具有较强的尖峰平滑函数。构建KDE一般就是在每一个点上放置一个Kernel，然后把所有Kernel的贡献加起来构成一个平滑曲线。 我们来看一下常见的三种Kernel:

$$K(x) = \begin{cases} 1/2 & |x| < 1 \\ 0 & else \end{cases} \quad box\ kernel$$

$$K(x) = \begin{cases} \frac{3}{4}(1 - x^2) & |x| < 1 \\ 0 & else \end{cases} \quad Epanechnikov\ kernel$$

$$K(x) = \frac{1}{2\pi} exp(-\frac{1}{2}x^2) \quad Gaussian\ kernel$$

一些kernel的图像如下：



KDE的公式为：

$$D_h(x;\{x_i\}) = \sum_{i=1}^{n} \frac{1}{h} K(\frac{x - x_i}{h})$$

h代表kernel的宽度。 通过上式，我们可以计算出任何一点的平滑值。

可以看出，和LOESS类似，KDE的计算要求较高。

实践

In [4]:

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

1. 导入python包

In [5]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%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\Si


%load_ext autoreload
%autoreload 2
```

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

2. 读入数据

In [6]:

```
start = '2012-01-01'
end = '2017-02-05'
```

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')
```

In [8]:

```
indexs_pv.columns = ['国内债券', '国内股票', '香港股票', '国内货币']
indexs_pv = indexs_pv[['国内债券', '国内股票', '国内货币', '香港股票']]
indexs_pv.fillna(axis=0,method='bfill',inplace=True)
indexs_sub = indexs_pv.loc[start:end,]
```

国内债卷：中债综合财富(总值)指数
国内股票：中证全指
香港股票：恒生指数
国内货币：货币基金

In [9]:

```
indexs_sub.head()
```

Out[9]:

日期	国内债券	国内股票	国内货币	香港股票
2012-01-04	141.5160	2571.951	1166.7726	18727.31
2012-01-05	141.5501	2513.699	1166.9696	18813.41
2012-01-06	141.7277	2527.247	1167.1185	18593.06
2012-01-09	141.8669	2619.638	1167.5058	18865.72
2012-01-10	142.0118	2713.529	1167.6330	19004.28

In [10]:

```
indexs_logret = indexs_sub.apply(log_return).dropna()
```

In [11]:

```
indexs_logret.head()
```

Out[11]:

日期	国内债券	国内股票	国内货币	香港股票
2012-01-05	0.000241	-0.022909	0.000169	0.004587
2012-01-06	0.001254	0.005375	0.000128	-0.011782
2012-01-09	0.000982	0.035906	0.000332	0.014558
2012-01-10	0.001021	0.035214	0.000109	0.007318
2012-01-11	0.000188	-0.002115	0.000113	0.007740

- Spline Smoothing
  我们用指数函数加上随机抖动模拟原数据

In [71]:

```
from scipy.interpolate import UnivariateSpline
from numpy import linspace,exp
from numpy.random import randn
x = linspace(-3, 3, 100)
y = exp(-x**2) + randn(100)/10
s = UnivariateSpline(x, y, s=1)
```

In [72]:

```
xs = linspace(-3, 3, 1000)
ys = s(xs)
plt.plot(x, y, '.-',label='Raw')
plt.plot(xs, ys,label='Spline')
plt.legend()
plt.show()
```

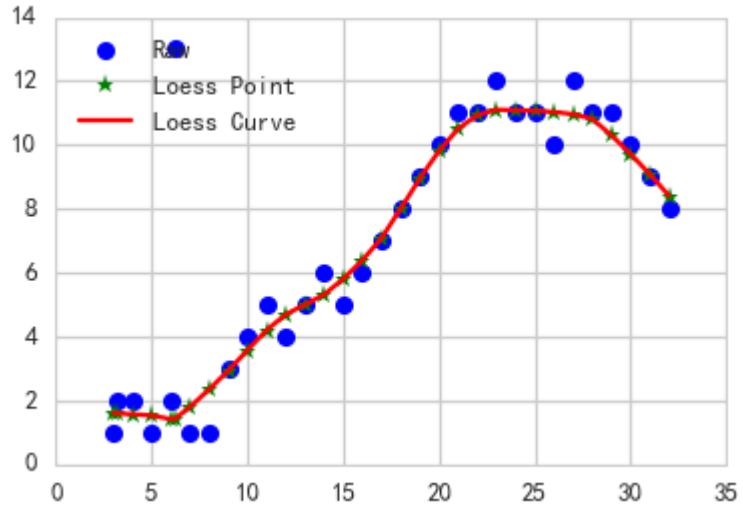
- LOESS Smoothing
  我们假造一些数据来模拟原数据

In [75]:

```
from scipy.interpolate import interp1d
import statsmodels.api as sm

# introduce some floats in our x-values
x = list(range(3, 33)) + [3.2, 6.2]
y = [1,2,1,2,1,1,3,4,5,4,5,6,5,6,7,8,9,10,11,11,12,11,11,10,12,11,11,10,9,8,2,13]

# lowess will return our "smoothed" data with a y value for at every x-value
lowess = sm.nonparametric.lowess(y, x, frac=.3)

# unpack the lowess smoothed points to their values
lowess_x = list(zip(*lowess))[0]
lowess_y = list(zip(*lowess))[1]

# run scipy's interpolation. There is also extrapolation I believe
f = interp1d(lowess_x, lowess_y, bounds_error=False)

xnew = [i/10. for i in range(400)]

# this this generate y values for our xvalues by our interpolator
# it will MISS values outsite of the x window (less than 3, greater than 33)
# There might be a better approach, but you can run a for loop
#and if the value is out of the range, use f(min(lowess_x)) or f(max(lowess_x))
ynew = f(xnew)


plt.plot(x, y, 'o',label='Raw')
plt.plot(lowess_x, lowess_y, '*',label='Loess Point')
plt.plot(xnew, ynew, '-',label='Loess Curve')
plt.legend(loc='upper left')
```

Out[75]:

<matplotlib.legend.Legend at 0x12582a07cc0>



- KDE
  我们将会重点展示KDE的实现，不但是因为这个方法有非常广泛的应用，而且在《量化投资技术篇》中我们将会采用此技术作为估算收益率的方法之一。 主要步骤如下：

  1. 计算周收益率
  2. 得到周收益率的直方图和经验分布

3. 估算参数
4. 得到KDE

1. 计算周收益率

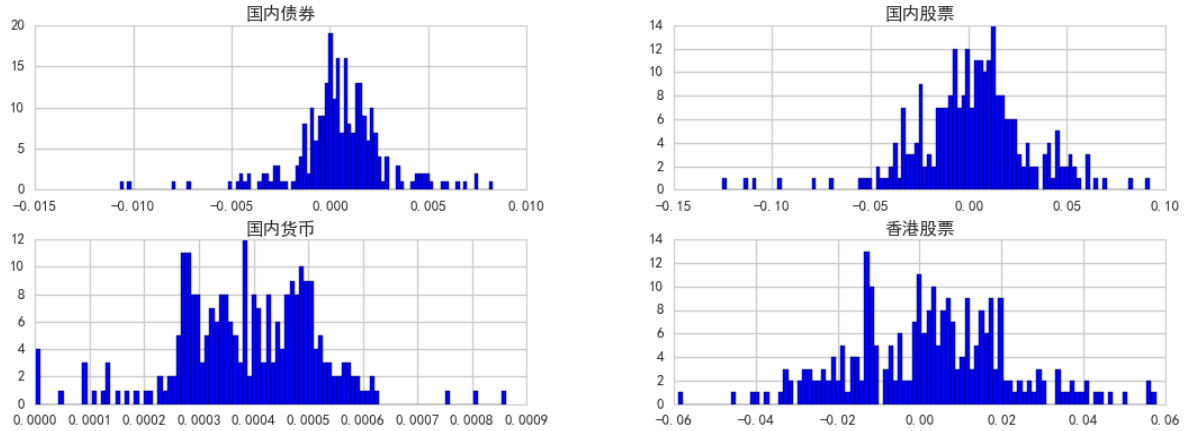
In [77]:

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

2. 得到直方图和经验分布

In [79]:

```
_ = index_ret_w.hist(bins=100, figsize=(18, 6))
```



3. 估算bandwidth参数

In [80]:

```
fund_ret = index_ret_w[['国内股票']].as_matrix()
fund_ret = fund_ret[fund_ret != 0]
```

In [81]:

```
from sklearn.model_selection  import GridSearchCV
from sklearn.neighbors import KernelDensity
```

gridSearch搜索参数

In [82]:

```
grid = GridSearchCV(KernelDensity(), {'bandwidth': np.arange(0.001, 0.3, 0.001) }, cv=5)
grid.fit(fund_ret.reshape(-1,1))
print(grid.best_params_)
```

```
{'bandwidth': 0.016}
```

Scott和Sliverman准则估算参数

In [83]:

```
from scipy import stats
from scipy.stats import norm
```

In [84]:

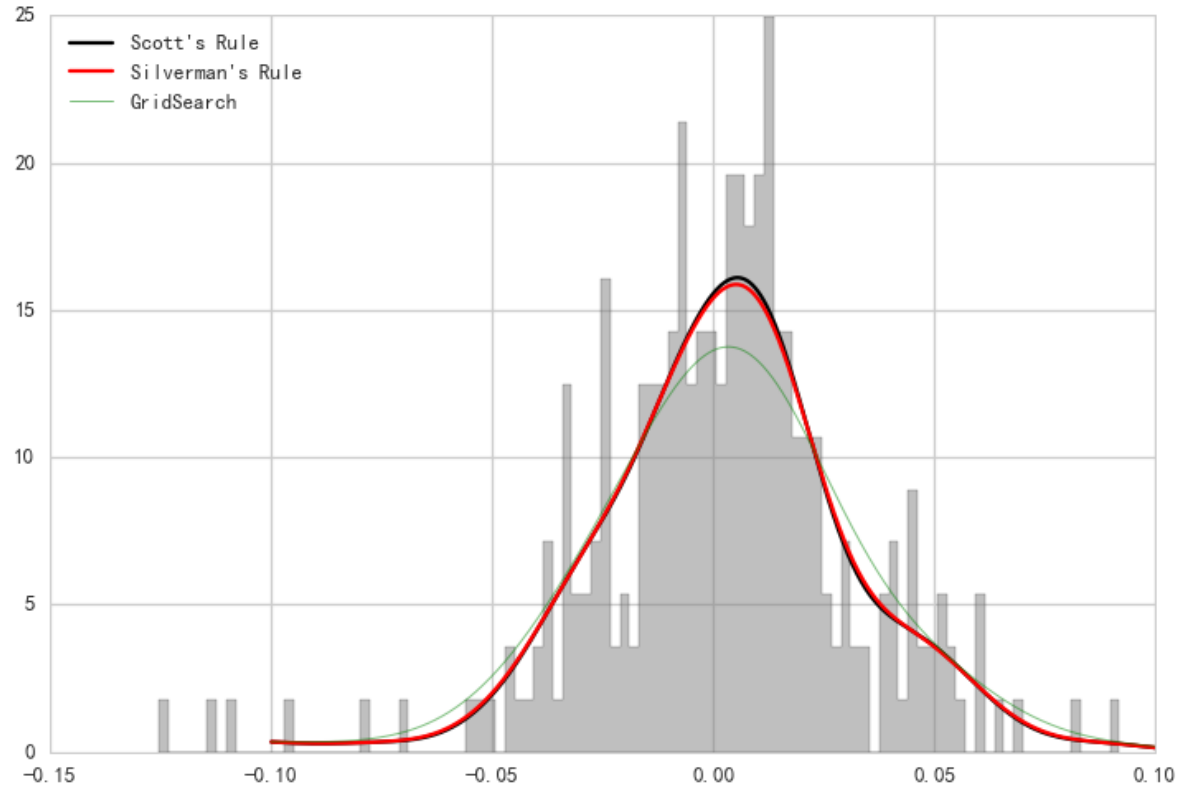
```
kde1 = stats.gaussian_kde(fund_ret)
kde2 = stats.gaussian_kde(fund_ret, bw_method='silverman')

print("scott rule band width:",fund_ret.size**(-1./(1+4)))
print("Sliverman rule band width:",(fund_ret.size * (1 + 2) / 4.)**(-1. / (1 + 4)))
print(kde1.factor,kde2.factor)
```

```
scott rule band width: 0.329363947250484
Sliverman rule band width: 0.34887014530965604
0.329363947250484 0.34887014530965604
```

三种方式估算参数的KDE比较

In [86]:

```
x_grid = np.linspace(-0.1, 0.1, 100000)

kde3 = grid.best_estimator_
pdf3_ = np.exp(kde3.score_samples(x_grid[:, None]))

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
#x_eval = np.linspace(-0.015, 0.015, num=500)
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")
Silverman = ax.plot(x_grid, kde2(x_grid), 'r-', label="Silverman's Rule")
GridSearch = ax.plot(x_grid, pdf3_, linewidth=1, alpha=0.5, label='GridSearch' % kde3.bandwidth)

plt.legend(loc='upper left')
plt.show()
```



本文介绍了平滑技术中的基于回归的平滑，包括Spline,LOESS,KDE。当然前两者在时序数据平滑中很少采用，而后者主要用在概率密度函数平滑中。其实平滑的核心作用是为了降噪，在实现的代码和图形展示中我们已经看到了这一点。