法律声明

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

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

Python库

邹博

主要内容

- 信息摘要与安全哈希算法MD5/SHA1
  - Message Digest Algorithm/Secure Hash Algorithm
- 统计量：均值/方差/偏度/峰度
- 多元高斯分布
- 阶乘的实数域推广：Γ函数
- Pearson相关系数的计算
- 快速傅里叶变换FFT与信号滤波
- 奇异值分解SVD与图像特征
- 股票数据相关：收盘价格曲线、滑动平均线、K线图
- 图像的卷积与卷积网络CNN
- 蝴蝶效应：Lorenz系统的曲线生成

MD5/SHA1

```python
import hashlib


if __name__ == "__main__":
    md5 = hashlib.md5()
    md5.update('This is a sentence.')
    md5.update('This is a second sentence.')
    print u'不出意外，这个将是“乱码”：', md5.digest()
    print u'MD5:', md5.hexdigest()

    md5 = hashlib.md5()
    md5.update('This is a sentence.This is a second sentence.')
    print u'MD5:', md5.hexdigest()
    print md5.digest_size, md5.block_size
    print '------------------'

    sha1 = hashlib.sha1()
    sha1.update('This is a sentence.')
    sha1.update('This is a second sentence.')
    print u'不出意外，这个将是“乱码”：', sha1.digest()
    print u'SHA1:', sha1.hexdigest()

    sha1 = hashlib.sha1()
    sha1.update('This is a sentence.This is a second sentence.')
    print u'SHA1:', sha1.hexdigest()
    print sha1.digest_size, sha1.block_size
    print '======================'
```

5.1.hashlib

```
不出意外，这个将是“乱码”： +? I QUL ?k )$?q
MD5: 2bfe14491651554c05996b1629248571
MD5: 2bfe14491651554c05996b1629248571
16 64
------------------
不出意外，这个将是“乱码”： ??JH??a e%?+?? 1?: ,
SHA1: 99a74a48d5fb611a6525962ba3d51131cb3a1d2c
SHA1: 99a74a48d5fb611a6525962ba3d51131cb3a1d2c
20 64
```

MD5

- MD5(Message Digest Algorithm)，消息摘要算法，为计算机安全领域广泛使用的一种散列函数，用以提供消息的完整性保护。
  - 文件号RFC 1321(R.Rivest,MIT Laboratory for Computer Science and RSA Data Security Inc. April 1992)
- 对于任意长度的信息，经过MD5算法，得到长度为128bit的摘要。

MD5的框架理解

- 对于长度为512bit的信息，可以通过处理，得到长度为128bit的摘要。
- 初始化摘要：

  0x0123456789ABCDEFFEDCBA9876543210
  - A=0x01234567　　B=0x89ABCDEF
  - C=0xFEDCBA98　　D=0x76543210
- 现在的工作，是要用长度为512位的信息，变换初始摘要。

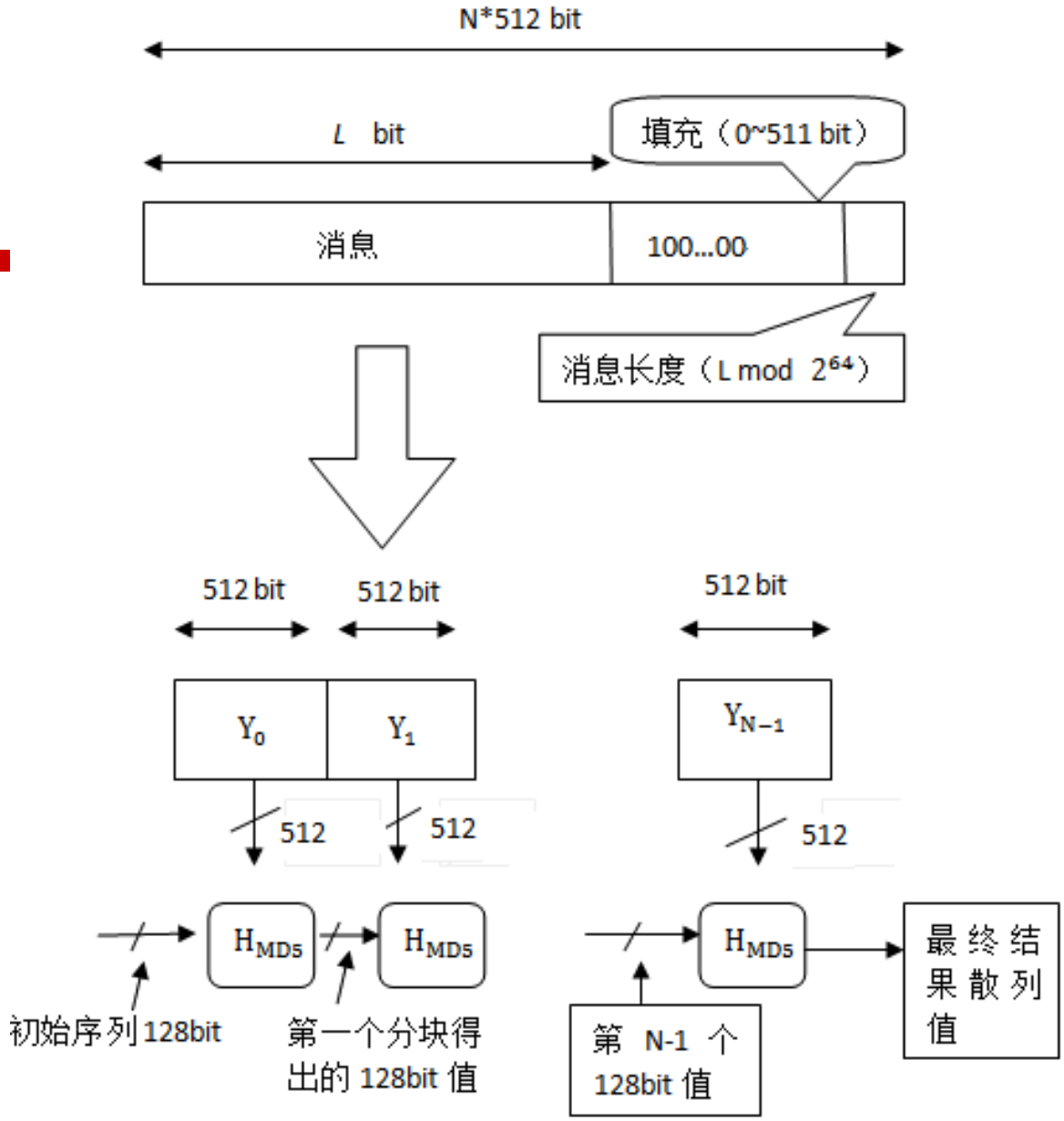
MD5的总体理解

- 定义变量a,b,c,d,分别记录A,B,C,D;
- 将512bit的信息按照32bit一组，分成16组；分别记为M_j (0≤j≤15)；
- 取某正数s、t_k，定义函数：
  $FF(a,b,c,d,M_j,s,t_k)=(a+F(b,c,d)+M_j+t_k)<<s$
- 利用M_j分别进行信息提取，将结果保存到a
  - 其中，F(X,Y,Z) =(X & Y) | (~X & Z)

MD5的总体理解

- 经过以上16次变换，a,b,c,d带有了M_j的信息
- 事实上经过四轮这样的变换(4轮*16次=64次)
- 经过64次变换后，将a,b,c,d累加给A,B,C,D
- 此时，完成了512bit信息的提取；进行下一个512bit信息的相同操作

MD5框架


N*512 bit
L bit
填充（0~511 bit）
消息
100...00
消息长度（L mod 2^64）
512 bit
512 bit
512 bit
Y0
Y1
YN−1
512
512
512
HMD5
HMD5
HMD5
初始序列128bit
第一个分块得出的128bit值
第 N-1 个128bit值
最终结果散列值

MD5细致算法

- 在算法中，首先需要对信息进行填充，使其长度对512求余的结果等于448。
- 填充的方法如下，在信息的后面填充一个1和若干个0，直到满足上面的条件。然后，在这个结果后面附加一个以64位二进制表示的原始信息长度Length。
- 经过这两步的处理，数据总长度为N*512+448+64=(N+1)*512，即长度恰好是512的整数倍。

MD5的细致算法

- 定义四个非线性函数：
  - F(X, Y, Z)=(X&Y)|((~X)&Z)　　H(X, Y, Z)=X^Y^Z
  - G(X, Y, Z)=(X&Z)|(Y&(~Z))　　I(X, Y, Z)=Y^(X|(~Z))
- M_j表示数据的第j个子分组(0≤j≤15，循环表示)，常数t_k是2^{32}*abs(sin(k))的整数部分，1≤k≤64，单位是弧度。
- FF(a,b,c,d,M_j,s,t_k)=(a+F(b,c,d)+M_j+t_k)<<s
- GG(a,b,c,d,M_j,s,t_k)=(b+G(b,c,d)+M_j+t_k)<<s
- HH(a,b,c,d,M_j,s,t_k)=(b+H(b,c,d)+M_j+t_k)<<s
- II(a,b,c,d,M_j,s,t_k)=(b+I(b,c,d)+M_j+t_k)<<s

64次操作

第一轮

FF(a, b, c, d, M0, 7, 0xd76aa478)
FF(d, a, b, c, M1, 12, 0xe8c7b756)
FF(c, d, a, b, M2, 17, 0x242070db)
FF(b, c, d, a, M3, 22, 0xc1bdceee)
FF(a, b, c, d, M4, 7, 0xf57c0faf)
FF(d, a, b, c, M5, 12, 0x4787c62a)
FF(c, d, a, b, M6, 17, 0xa8304613)
FF(b, c, d, a, M7, 22, 0xfd469501)
FF(a, b, c, d, M8, 7, 0x698098d8)
FF(d, a, b, c, M9, 12, 0x8b44f7af)
FF(c, d, a, b, M10, 17, 0xffff5bb1)
FF(b, c, d, a, M11, 22, 0x895cd7be)
FF(a, b, c, d, M12, 7, 0x6b901122)
FF(d, a, b, c, M13, 12, 0xfd987193)
FF(c, d, a, b, M14, 17, 0xa679438e)
FF(b, c, d, a, M15, 22, 0x49b40821)

第二轮

GG(a, b, c, d, M1, 5, 0xf61e2562)
GG(d, a, b, c, M6, 9, 0xc040b340)
GG(c, d, a, b, M11, 14, 0x265e5a51)
GG(b, c, d, a, M0, 20, 0xe9b6c7aa)
GG(a, b, c, d, M5, 5, 0xd62f105d)
GG(d, a, b, c, M10, 9, 0x02441453)
GG(c, d, a, b, M15, 14, 0xd8a1e681)
GG(b, c, d, a, M4, 20, 0xe7d3fbc8)
GG(a, b, c, d, M9, 5, 0x21e1cde6)
GG(d, a, b, c, M14, 9, 0xc33707d6)
GG(c, d, a, b, M3, 14, 0xf4d50d87)
GG(b, c, d, a, M8, 20, 0x455a14ed)
GG(a, b, c, d, M13, 5, 0xa9e3e905)
GG(d, a, b, c, M2, 9, 0xfcefa3f8)
GG(c, d, a, b, M7, 14, 0x676f02d9)
GG(b, c, d, a, M12, 20, 0x8d2a4c8a)

第三轮

HH(a, b, c, d, M5, 4, 0xfffa3942)
HH(d, a, b, c, M8, 11, 0x8771f681)
HH(c, d, a, b, M11, 16, 0x6d9d6122)
HH(b, c, d, a, M14, 23, 0xfde5380c)
HH(a, b, c, d, M1, 4, 0xa4beea44)
HH(d, a, b, c, M4, 11, 0x4bdecfa9)
HH(c, d, a, b, M7, 16, 0xf6bb4b60)
HH(b, c, d, a, M10, 23, 0xbebfbc70)
HH(a, b, c, d, M13, 4, 0x289b7ec6)
HH(d, a, b, c, M0, 11, 0xeaa127fa)
HH(c, d, a, b, M3, 16, 0xd4ef3085)
HH(b, c, d, a, M6, 23, 0x04881d05)
HH(a, b, c, d, M9, 4, 0xd9d4d039)
HH(d, a, b, c, M12, 11, 0xe6db99e5)
HH(c, d, a, b, M15, 16, 0x1fa27cf8)
HH(b, c, d, a, M2, 23, 0xc4ac5665)

第四轮

II(a, b, c, d, M0, 6, 0xf4292244)
II(d, a, b, c, M7, 10, 0x432aff97)
II(c, d, a, b, M14, 15, 0xab9423a7)
II(b, c, d, a, M5, 21, 0xfc93a039)
II(a, b, c, d, M12, 6, 0x655b59c3)
II(d, a, b, c, M3, 10, 0x8f0ccc92)
II(c, d, a, b, M10, 15, 0xffeff47d)
II(b, c, d, a, M1, 21, 0x85845dd1)
II(a, b, c, d, M8, 6, 0x6fa87e4f)
II(d, a, b, c, M15, 10, 0xfe2ce6e0)
II(c, d, a, b, M6, 15, 0xa3014314)
II(b, c, d, a, M13, 21, 0x4e0811a1)
II(a, b, c, d, M4, 6, 0xf7537e82)
II(d, a, b, c, M11, 10, 0xbd3af235)
II(c, d, a, b, M2, 15, 0x2ad7d2bb)
II(b, c, d, a, M9, 21, 0xeb86d391)

思考

新浪微博长URL的压缩（存储、查找相关）

取消关注 | 14 ...

【题目】

新浪微博发布内容要求字符不超过140，但是用户如果在发布内容中有很长的url时，会认为是很多字符。所以新浪上发布内容包含一个URL时，时把他压缩成一个TinyURL(缩小)。比如：

(因为无法发站外链接，所以去掉了链接中的http://头)

输入：zhidao.baidu.com/search?ct=17&pn=0&tn=ikaslist&rn=10&word=helloworld&ie=utf-8&fr=wwwt

实际显示：//asdfa.cn/ak78ss

前面asdfa.cn是对应域名zhidao.baidu.com，后面长长的字符串被压缩成ak78ss。

【问题】

现在让你来设计TinyURL的实现，以下问题要怎么设计：

(1)：域名后面的编码如何实现?

(2)：对于已经映射过的一个URL，怎么查找已存在的TinyUrl?

(3)：有10亿个url，一个服务上存不下，需要多台服务器，怎么设计实现

(4)：让你来设计这样一个服务，最大的问题是什么?

【探讨】

暂时的思路(根据网友的回答汇总)，欢迎探讨：

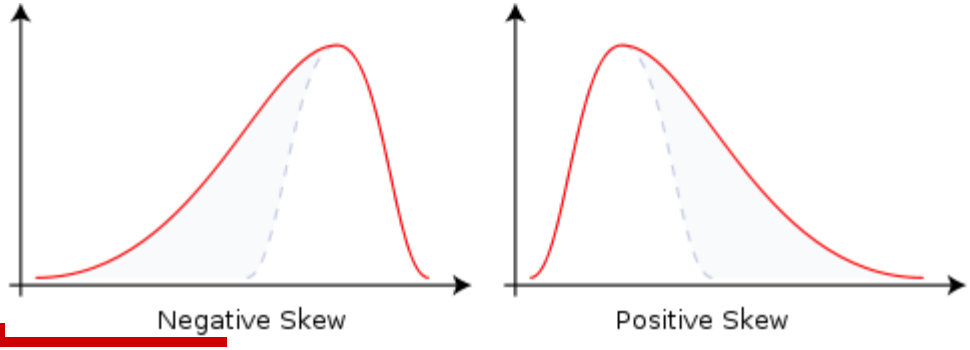
(1)：两种思路，一种是把真实网址存数据库，然后取自增id，做个哈希或者进制转换之类的，生成短网址，用的时候查一下就行了；另一种是直接真实网址哈希然后截取特定位数，6位的话 `（26+26+10）^6` 种组合，应该够用了，实在不行再做一次碰撞检测

(2)：直接key-value存储查询

(3)：二次哈希， 根据ak78ss这样的值映射到不同机器上，hash或者字母序层次下去

(4)：查询速度？响应时间？还有过期的url在浪费存储空间？

偏度



- 偏度衡量随机变量概率分布的不对称性，是相对于均值不对称程度的度量。
  - 偏度的值可以为正，可以为负或者无定义。
  - 偏度为负(负偏)/正(正偏)表示在概率密度函数左侧/右侧的尾部比右侧的长，长尾在左侧/右侧。
  - 偏度为零表示数值相对均匀地分布在平均值的两侧，但不一定意味着一定是对称分布。

偏度公式

- 三阶累积量与二阶累积量的1.5次方的比率。
  - 偏度有时用Skew[X]来表示。

$$\gamma_1 = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{E\left[(X-\mu)^3\right]}{\left(E\left[(X-\mu)^2\right]\right)^{3/2}} = \frac{\kappa_3}{\kappa_2^{3/2}}$$

$$\gamma_1 = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{E\left[X^3\right]-3\mu E\left[X^2\right]+2\mu^2}{\sigma^3} = \frac{E\left[X^3\right]-3\mu\sigma^2-\mu^3}{\sigma^3}$$

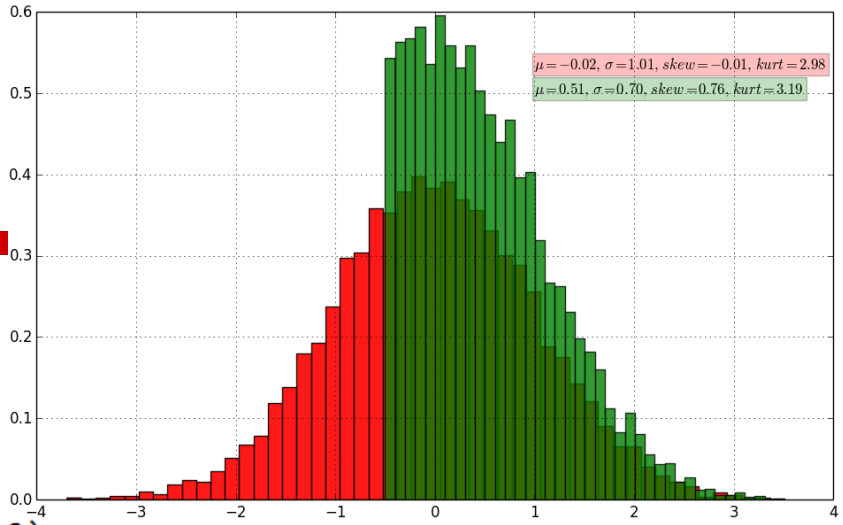
峰度 $\frac{\mu_4}{\sigma^4}$

- 峰度是概率密度在均值处峰值高低的特征，通常定义四阶中心矩除以方差的平方减3。

$$\gamma_2 = \frac{\kappa_4}{\kappa_2^2} = \frac{\mu_4}{\sigma^4} - 3 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^2} - 3$$

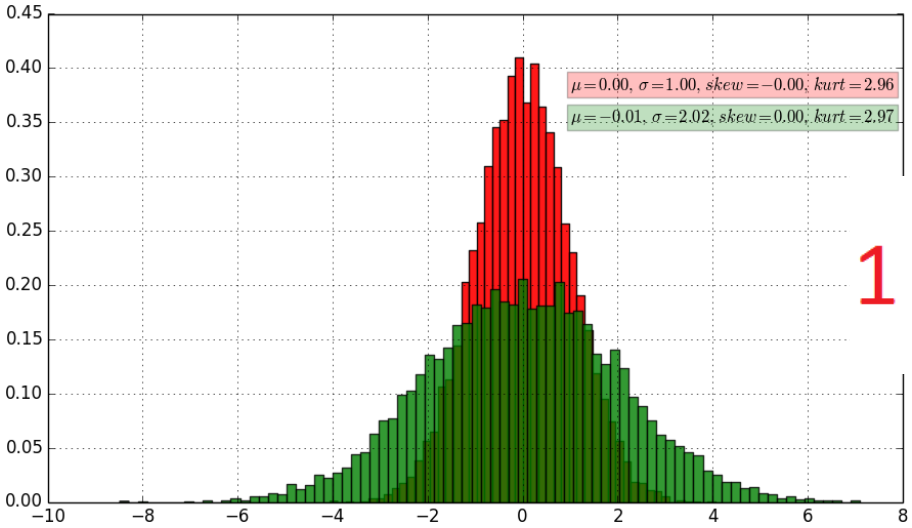
- $\frac{\mu_4}{\sigma^4}$ 也被称为超值峰度(excess kurtosis)。
  - “减3”是为了让正态分布的峰度为0。
  - 超值峰度为正，称为尖峰态(leptokurtic)
  - 超值峰度为负，称为低峰态(platykurtic)

统计量Code



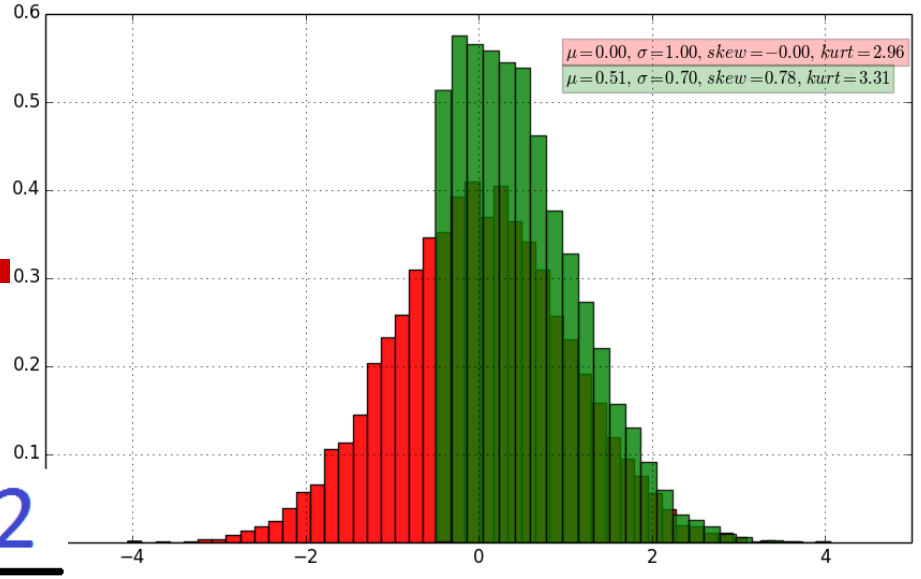

```
if __name__ == "__main__":
    data = list(np.random.randn(10000))
    data2 = list(2*np.random.randn(10000))
    data3 = [x for x in data if x > -0.5]
    data4 = list(np.random.uniform(0, 4, 10000))
    [niu, sigma, skew, kurt] = calc_stat(data)
    [niu2, sigma2, skew2, kurt2] = calc_stat(data2)
    [niu3, sigma3, skew3, kurt3] = calc_stat(data3)
    [niu4, sigma4, skew4, kurt4] = calc_stat(data4)
    print niu, sigma, skew, kurt
    print niu2, sigma2, skew2, kurt2
    print niu3, sigma3, skew3, kurt3
    print niu4, sigma4, skew4, kurt4

    info = r'$\mu=%.2f,\ \sigma=%.2f,\ skew=%.2f,\ kurt=%.2f$' % (niu, sigma, skew, kurt)
    info2 = r'$\mu=%.2f,\ \sigma=%.2f,\ skew=%.2f,\ kurt=%.2f$' % (niu2, sigma2, skew2, kurt2)
    plt.text(1, 0.38, info, bbox=dict(facecolor='red', alpha=0.25))
    plt.text(1, 0.35, info2, bbox=dict(facecolor='green', alpha=0.25))
    plt.hist(data, 50, normed=True, facecolor='r', alpha=0.9)
    plt.hist(data2, 80, normed=True, facecolor='g', alpha=0.8)
    plt.grid(True)
    plt.show()
```

数据分析

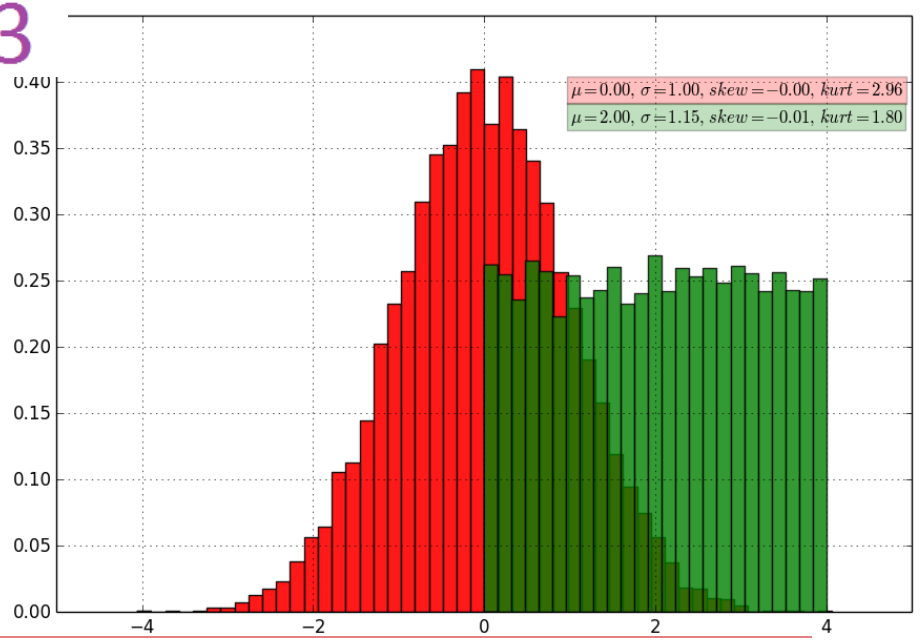


1

2

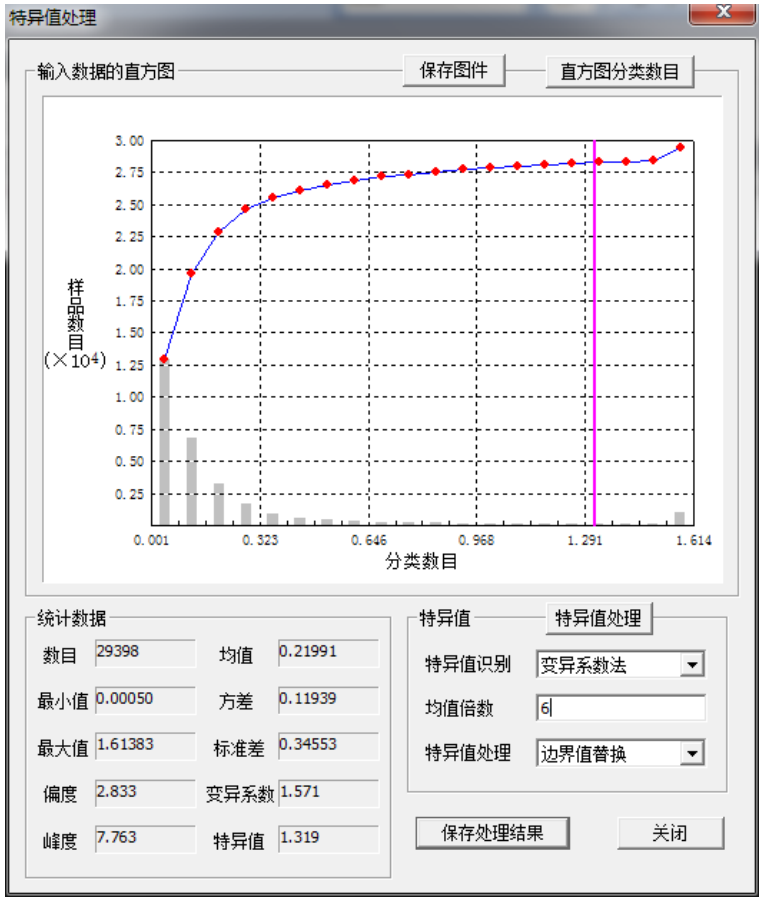


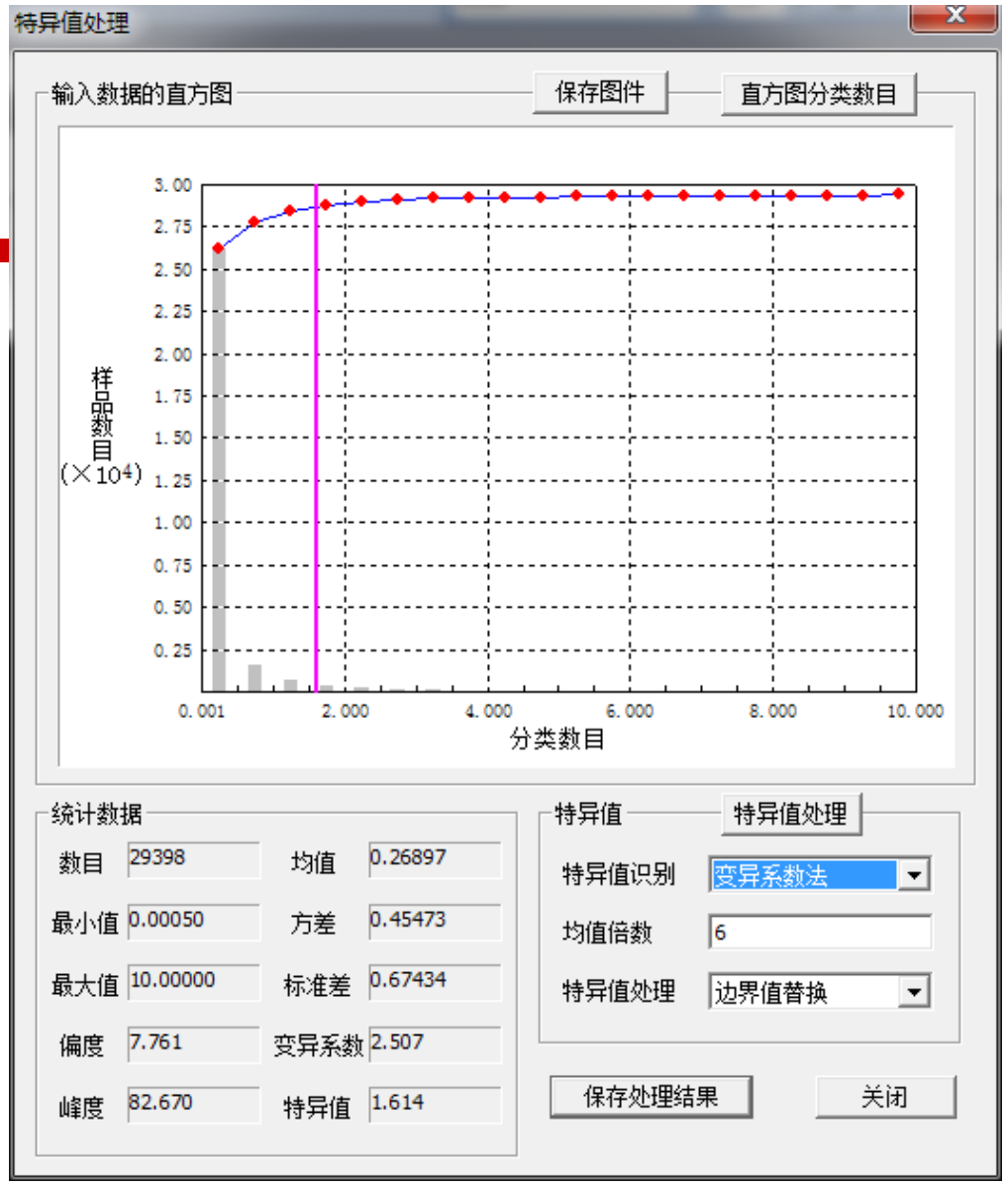
3



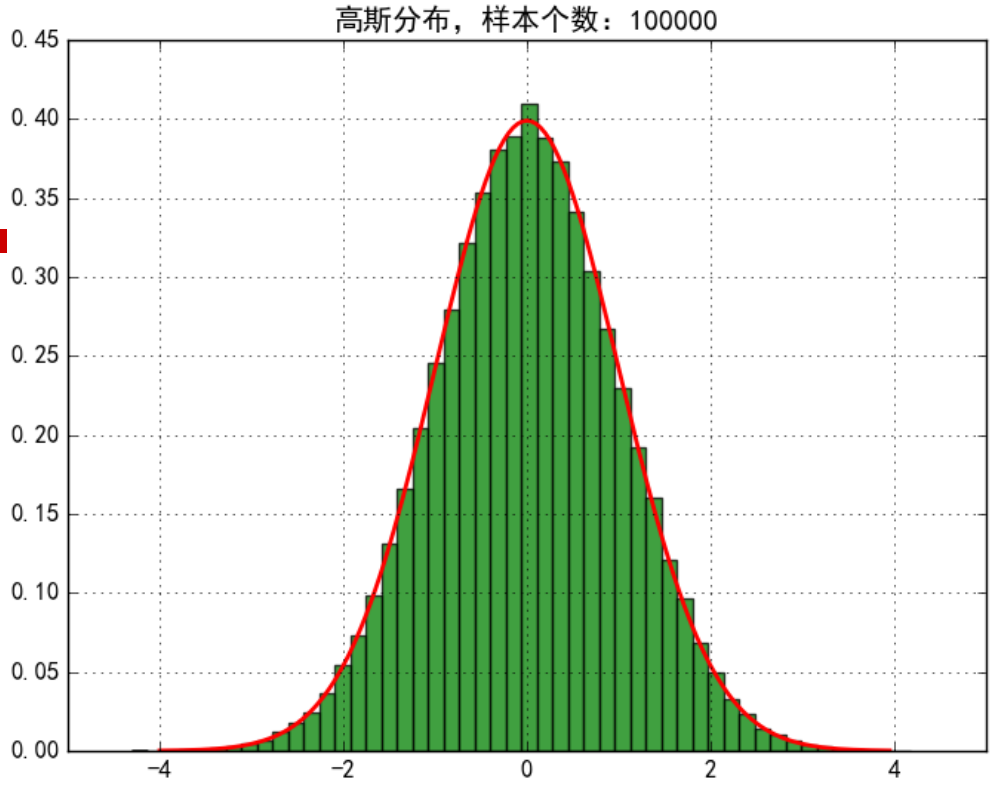
0.0029 1.0001 -0.0040 2.9643
-0.0137 2.0208 0.0046 2.9748
0.5134 0.6962 0.7825 3.3052
2.0016 1.1548 -0.0107 1.8012

实践中的例子


特异值处理
输入数据的直方图
保存图件
直方图分类数目
样品数目 (×10⁴)
分类数目
统计数据
数目 29398
均值 0.21991
最小值 0.00050
方差 0.11939
最大值 1.61383
标准差 0.34553
偏度 2.833
变异系数 1.571
峰度 7.763
特异值 1.319
特异值
特异值处理
特异值识别 变异系数法
均值倍数 6
特异值处理 边界值替换
保存处理结果
关闭


特异值处理
输入数据的直方图
保存图件
直方图分类数目
样品数目 (×10⁴)
分类数目
统计数据
数目 29398
均值 0.26897
最小值 0.00050
方差 0.45473
最大值 10.00000
标准差 0.67434
偏度 7.761
变异系数 2.507
峰度 82.670
特异值 1.614
特异值
特异值处理
特异值识别 变异系数法
均值倍数 6
特异值处理 边界值替换
保存处理结果
关闭

统计量



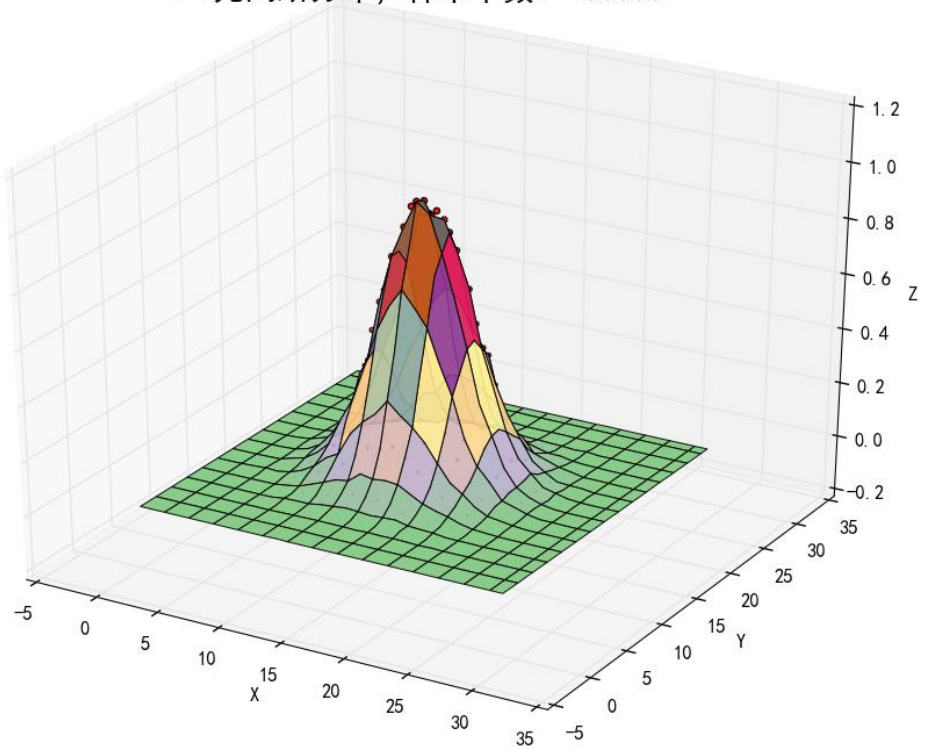

```
def calc_statistics(x):
    n = x.shape[0]  # 样本个数

    # 手动计算
    m = 0
    m2 = 0
    m3 = 0
    m4 = 0
    for t in x:
        m += t
        m2 += t*t
        m3 += t**3
        m4 += t**4
    m /= n
    m2 /= n
    m3 /= n
    m4 /= n

    mu = m
    sigma = np.sqrt(m2 - mu*mu)
    skew = (m3 - 3*mu*m2 + 2*mu**3) / sigma**3
    kurtosis = (m4 - 4*mu*m3 + 6*mu*mu*m2 - 4*mu**3*mu + mu**4) / sigma**4 - 3
    print '手动计算均值、标准差、偏度、峰度：', mu, sigma, skew, kurtosis

    # 使用系统函数验证
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0)
    skew = stats.skew(x)
    kurtosis = stats.kurtosis(x)
    return mu, sigma, skew, kurtosis
```

手动计算均值、标准差、偏度、峰度： -0.00232018730484 1.00220229337 0.0070687774347 0.0174102810253

函数库计算均值、标准差、偏度、峰度： -0.00232018730484 1.00220229337 0.0070687774347 0.0174102810253

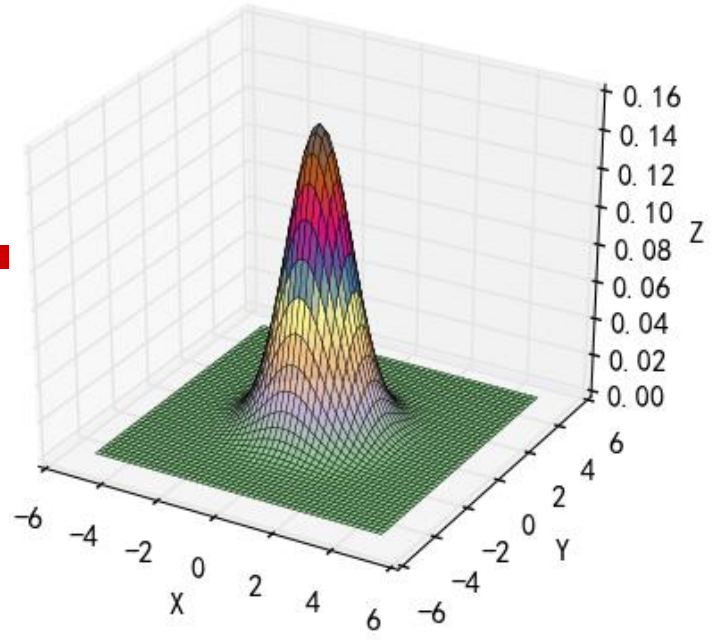
二元高斯分布

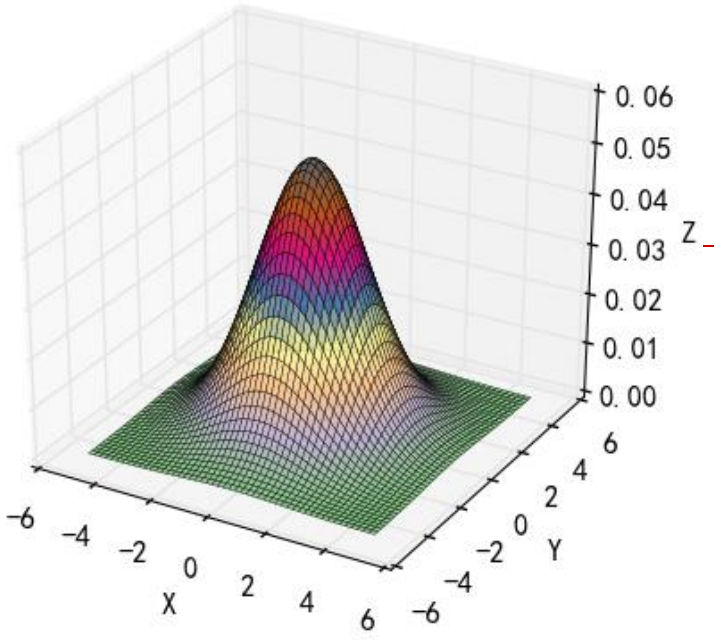
二元高斯分布，样本个数：100000

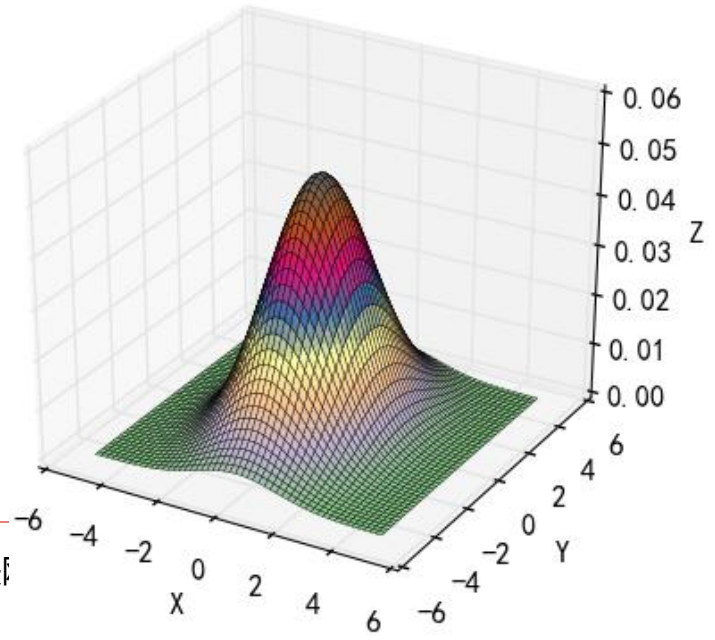


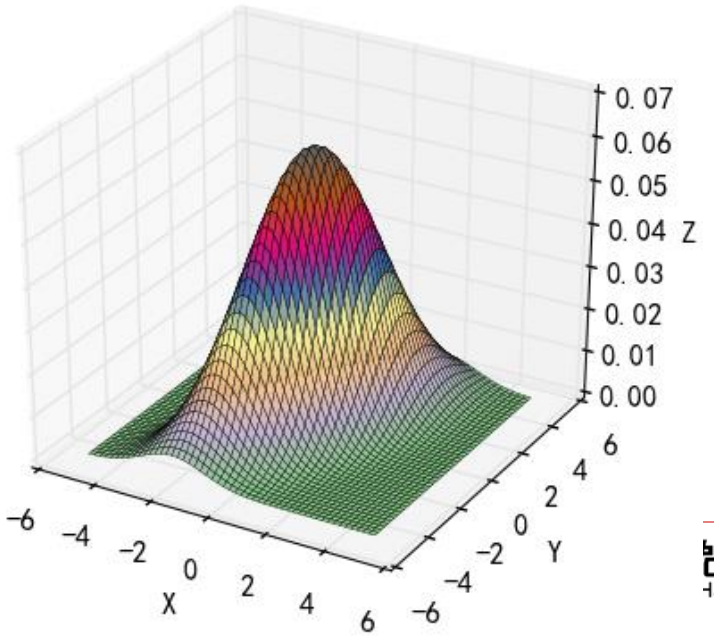

```
d = np.random.randn(100000, 2)
mu, sigma, skew, kurtosis = calc_statistics(d)
print '函数库计算均值、标准差、偏度、峰度：', mu,
# 二维图像
N = 30
density, edges = np.histogramdd(d, bins=[N, N])
print '样本总数：', np.sum(density)
density /= density.max()
x = y = np.arange(N)
t = np.meshgrid(x, y)
fig = plt.figure(facecolor='w')
ax = fig.add_subplot(111, projection='3d')
ax.scatter(t[0], t[1], density, c='r', s=15*density, marker='o', depthshade=True)
ax.plot_surface(t[0], t[1], density, cmap=cm.Accent, rstride=2, cstride=2, alpha=0.9, lw=0.75)
ax.set_xlabel(u'X')
ax.set_ylabel(u'Y')
ax.set_zlabel(u'Z')
plt.title(u'二元高斯分布，样本个数：%d' % d.shape[0], fontsize=20)
plt.tight_layout(0.1)
plt.show()
```

二元高斯分布方差比较


0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Z
X
Y


0.06
0.05
0.04
0.03
0.02
0.01
0.00
Z
X
Y


0.06
0.05
0.04
0.03
0.02
0.01
0.00
Z
X
Y

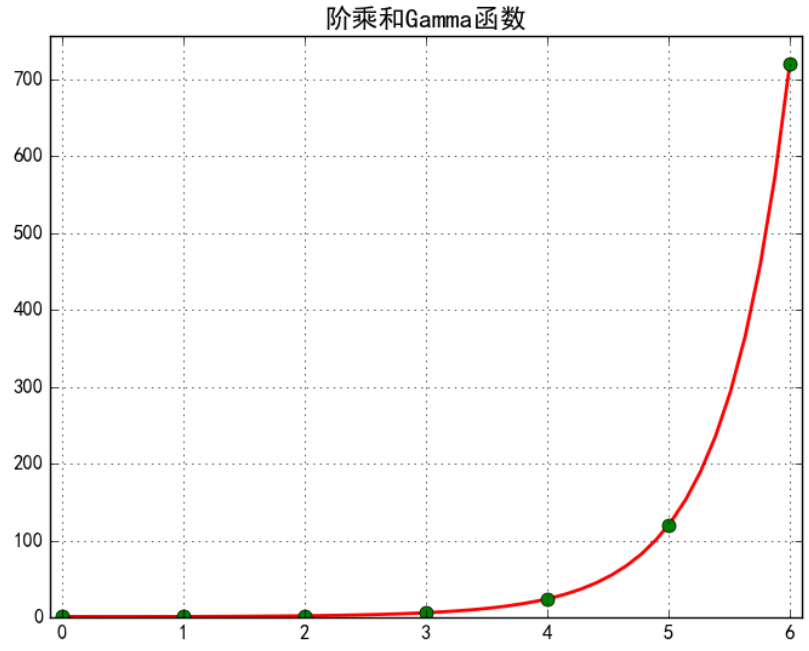

0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Z
X
Y

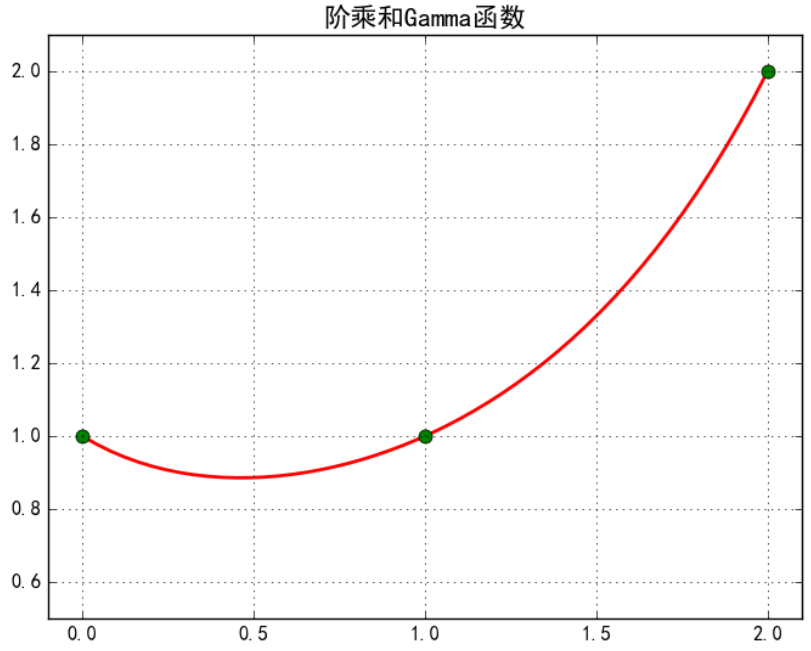
Γ函数

$$\Gamma(x)=(x-1)\cdot\Gamma(x-1)\Rightarrow\frac{\Gamma(x)}{\Gamma(x-1)}=x-1$$

- Γ函数是阶乘在实数上的推广

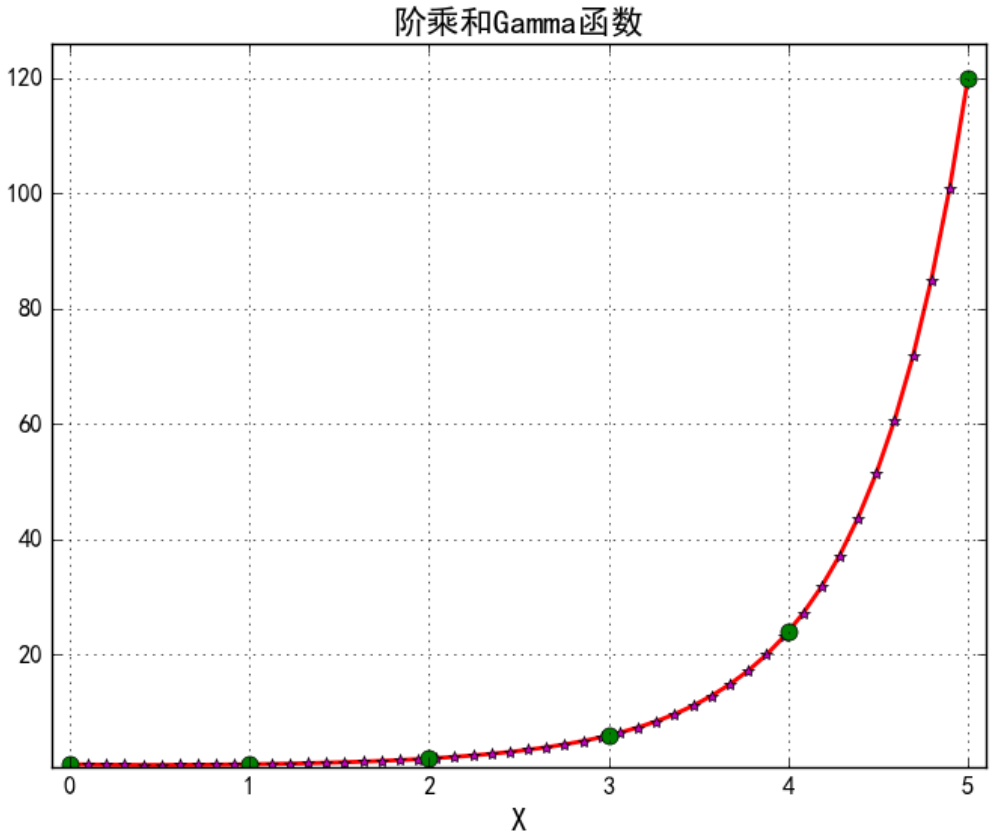
$$\Gamma(x)=\int_0^{+\infty}t^{x-1}e^{-t}dt=(x-1)!$$





Γ函数

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.special import gamma
from scipy.special import factorial

mpl.rcParams['axes.unicode_minus'] = False
mpl.rcParams['font.sans-serif'] = 'SimHei'


if __name__ == '__main__':
    N = 5
    x = np.linspace(0, N, 50)
    y = gamma(x+1)
    plt.figure(facecolor='w')
    plt.plot(x, y, 'r-', x, y, 'm*', lw=2)
    z = np.arange(0, N+1)
    f = factorial(z, exact=True)    # 阶乘
    print f
    plt.plot(z, f, 'go', markersize=8)
    plt.grid(b=True)
    plt.xlim(-0.1,N+0.1)
    plt.ylim(0.5, np.max(y)*1.05)
    plt.xlabel(u'X', fontsize=15)
    plt.ylabel(u'Gamma(X) - 阶乘', fontsize=15)
    plt.title(u'阶乘和Gamma函数', fontsize=16)
    plt.show()
```

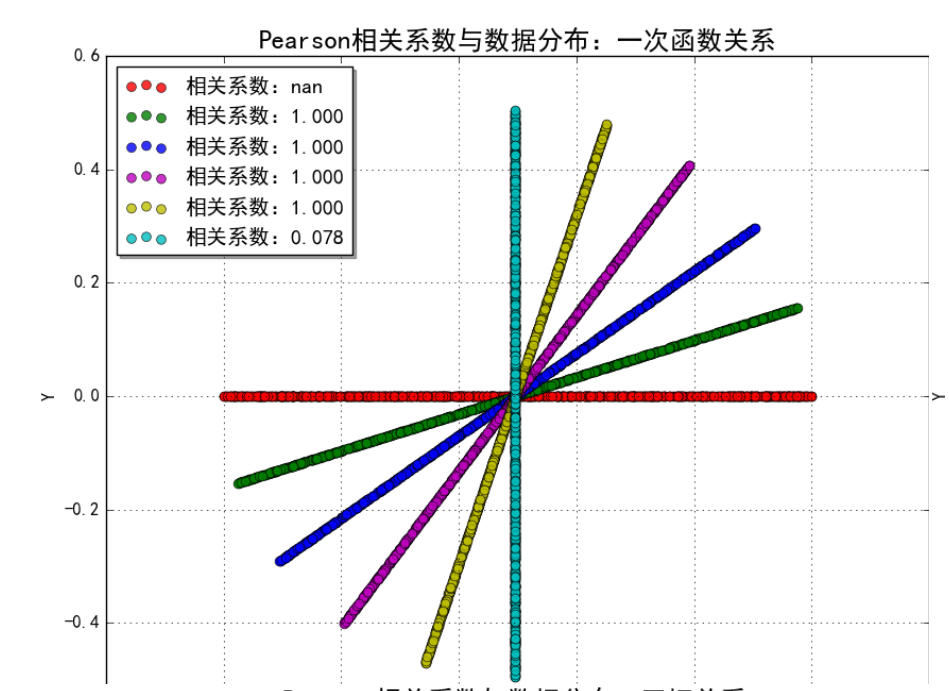


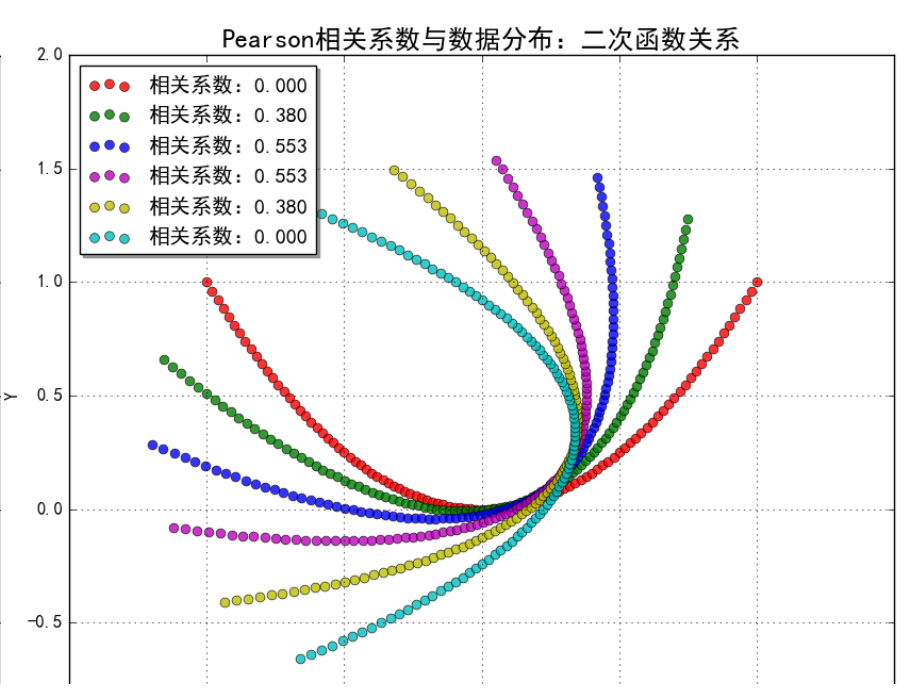
Pearson相关系数

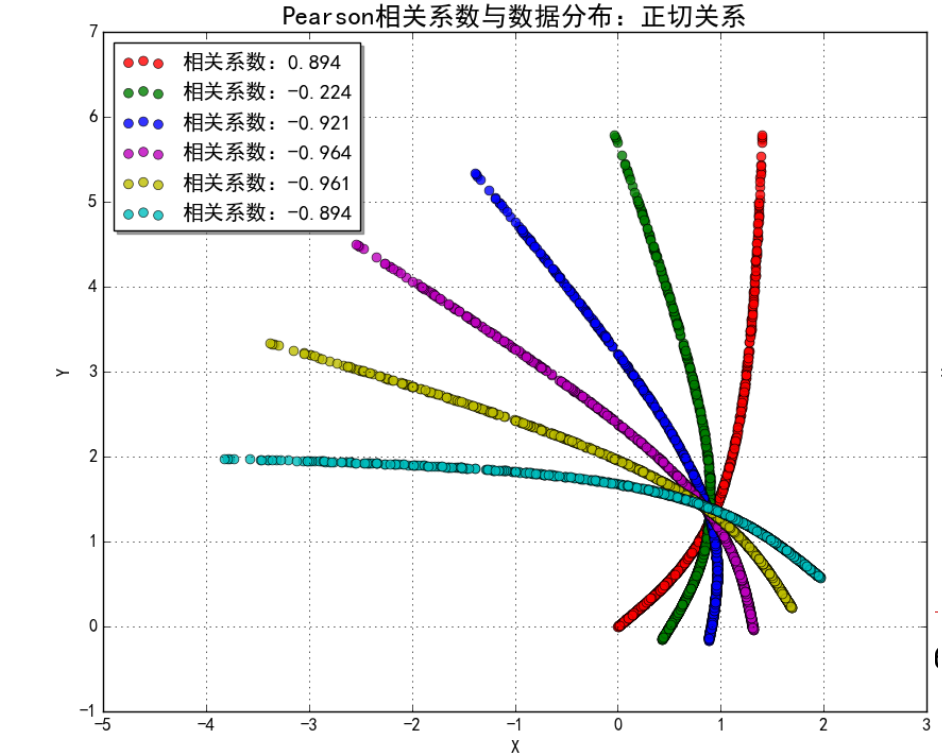
- 定义相关系数：

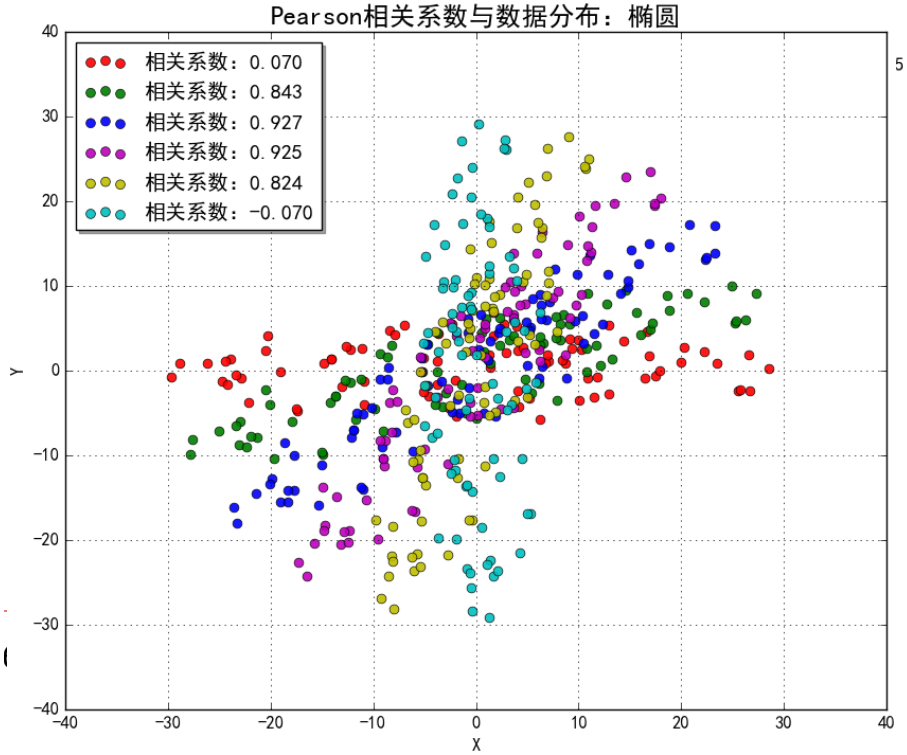
$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$$

- 由协方差上界定理可知，$|\rho| \leq 1$
  - 当且仅当X与Y有线性关系时，等号成立


Pearson相关系数与数据分布：一次函数关系
相关系数：nan
相关系数：1.000
相关系数：1.000
相关系数：1.000
相关系数：1.000
相关系数：0.078
Y


Pearson相关系数与数据分布：二次函数关系
相关系数：0.000
相关系数：0.380
相关系数：0.553
相关系数：0.553
相关系数：0.380
相关系数：0.000
Y

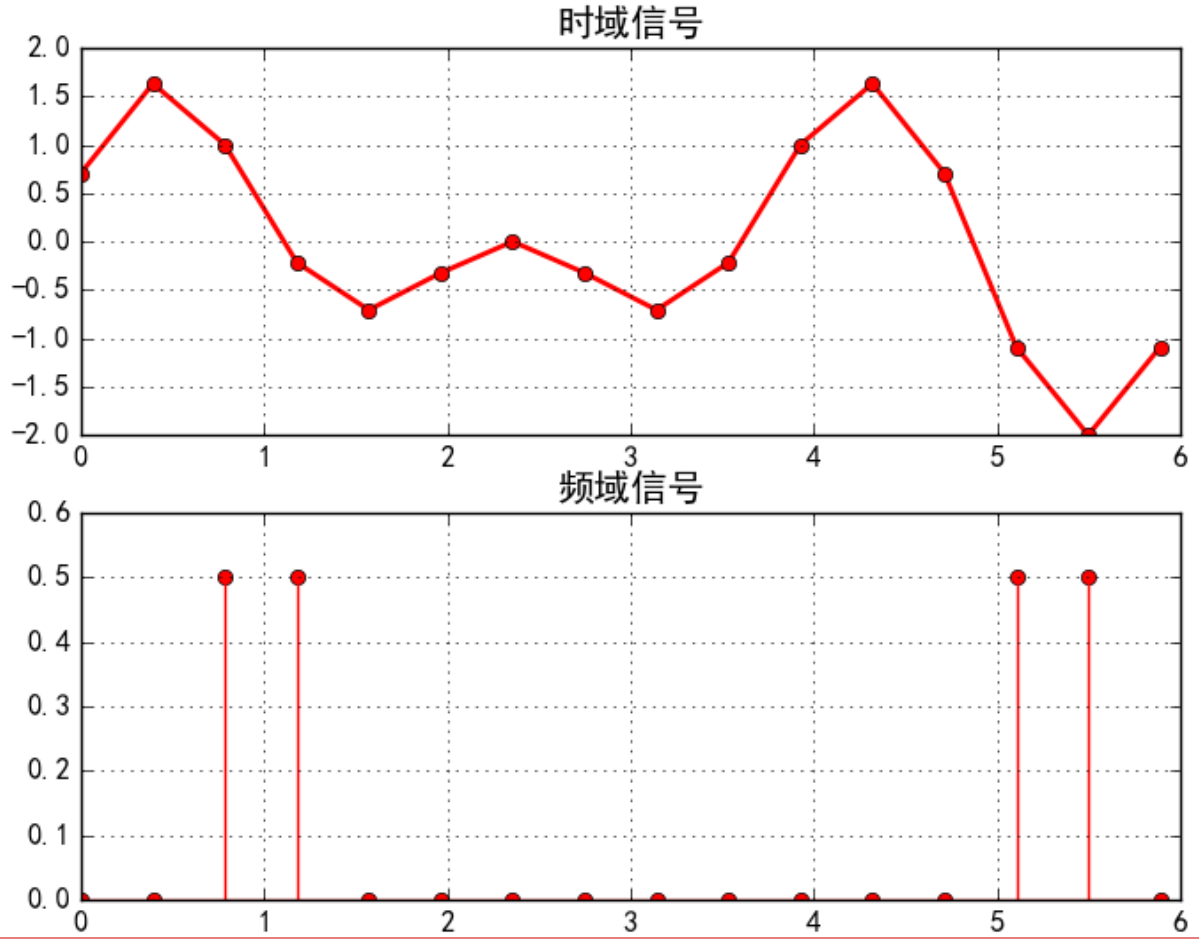

Pearson相关系数与数据分布：正切关系
相关系数：0.894
相关系数：-0.224
相关系数：-0.921
相关系数：-0.964
相关系数：-0.961
相关系数：-0.894
Y
X


Pearson相关系数与数据分布：椭圆
相关系数：0.070
相关系数：0.843
相关系数：0.927
相关系数：0.925
相关系数：0.824
相关系数：-0.070
Y
X

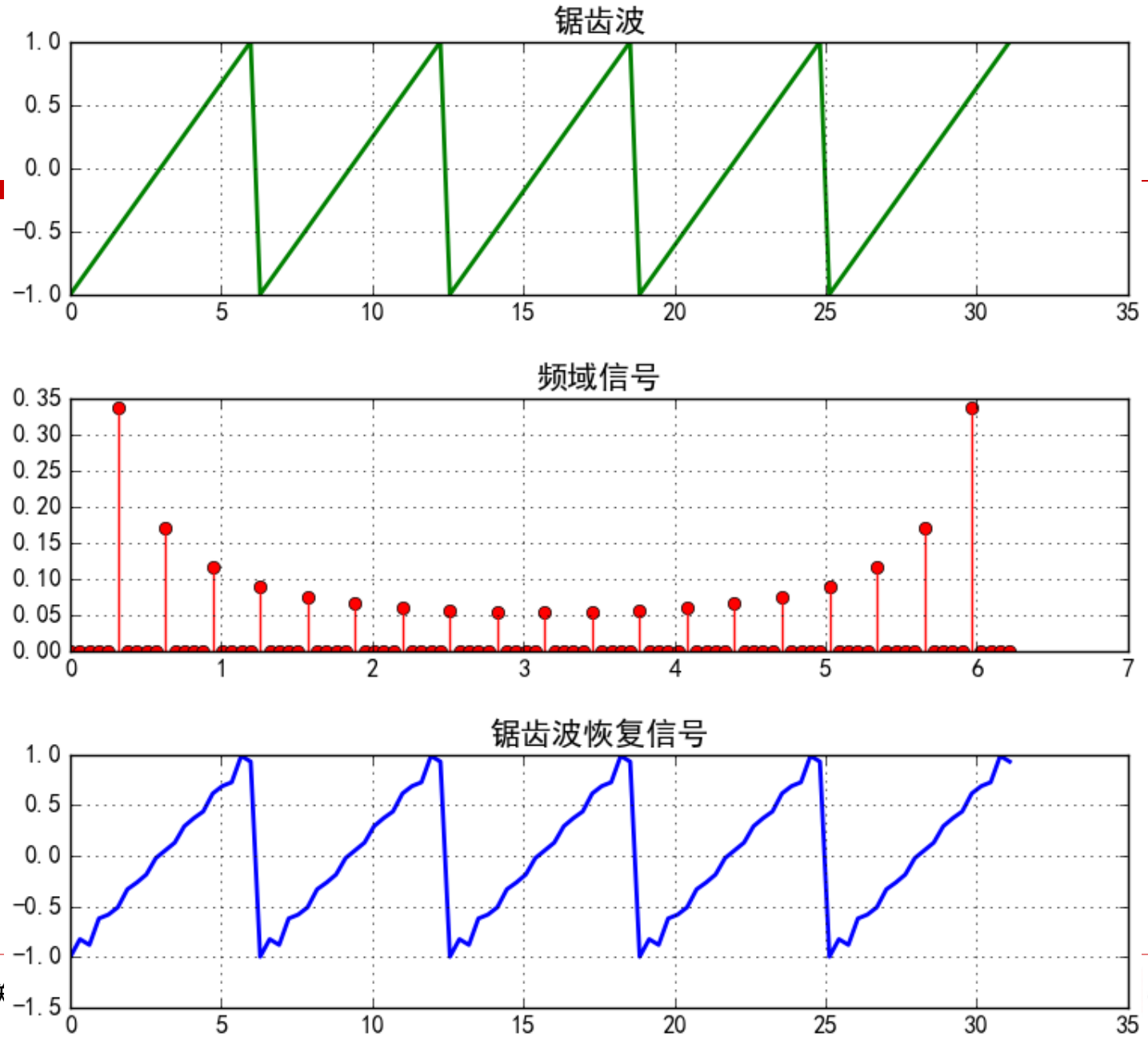
Code

```
def calc_pearson(x, y):
    std1 = np.std(x)
    # np.sqrt(np.mean(x**2) - np.mean(x)**2)
    std2 = np.std(y)
    cov = np.cov(x, y, bias=True)[0,1]
    return cov / (std1 * std2)


def pearson(x, y, tip):
    clrs = list('rgbmyc')
    plt.figure(figsize=(10, 8), facecolor='w')
    for i, theta in enumerate(np.linspace(0, 90, 6)):
        xr, yr = rotate(x, y, theta)
        p = stats.pearsonr(xr, yr)[0]
        # print calc_pearson(xr, yr)
        print '旋转角度：', theta, 'Pearson相关系数：', p
        str = u'相关系数：%.3f' % p
        plt.scatter(xr, yr, s=40, alpha=0.9, linewidths=0.5, c=clr
    plt.legend(loc='upper left', shadow=True)
    plt.xlabel(u'X')
    plt.ylabel(u'Y')
    plt.title(u'Pearson相关系数与数据分布：%s' % tip, fontsize=18)
    plt.grid(b=True)
    plt.show()
```

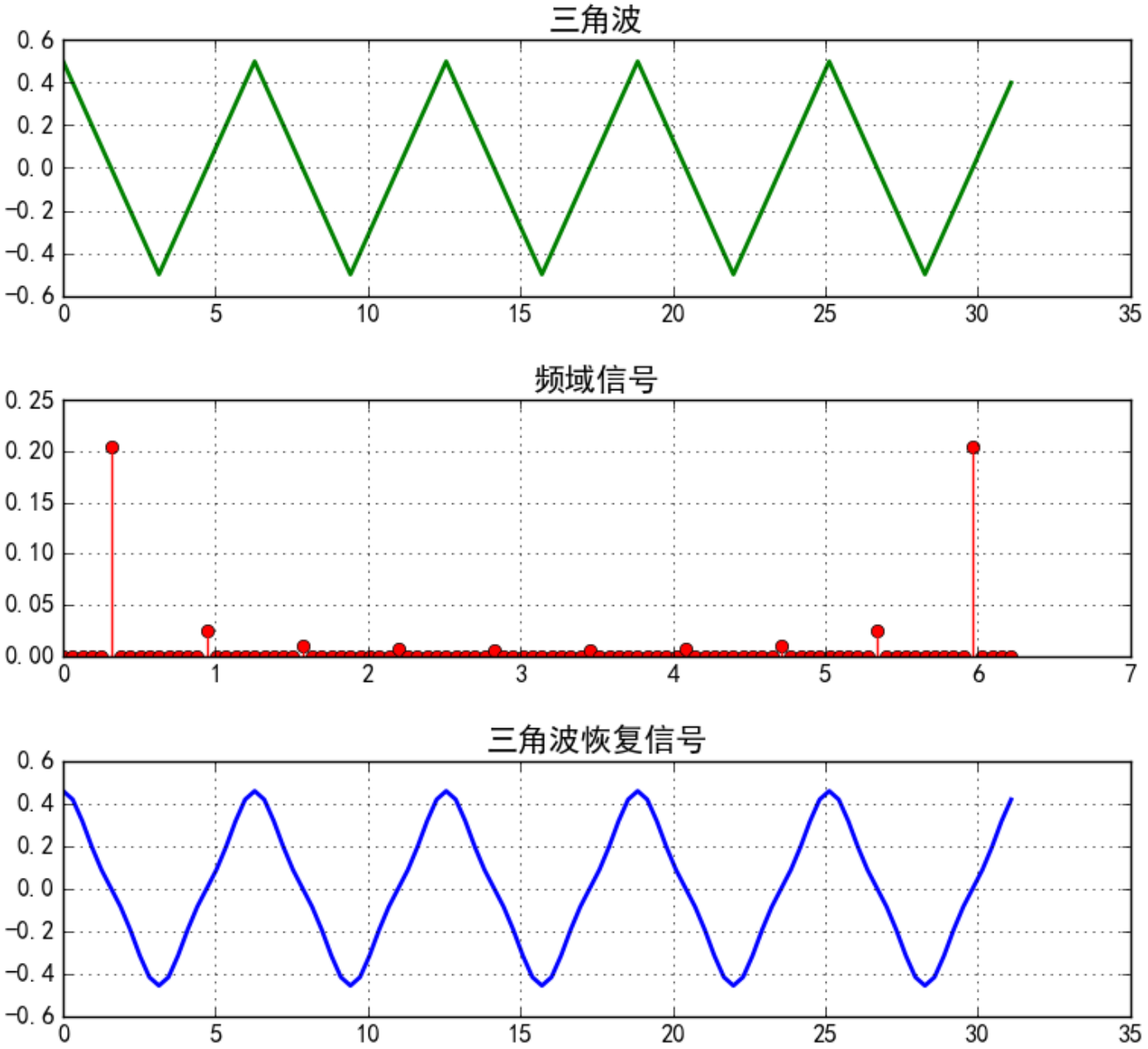
时域与频域信号


时域信号
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
1
2
3
4
5
6
频域信号
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
6

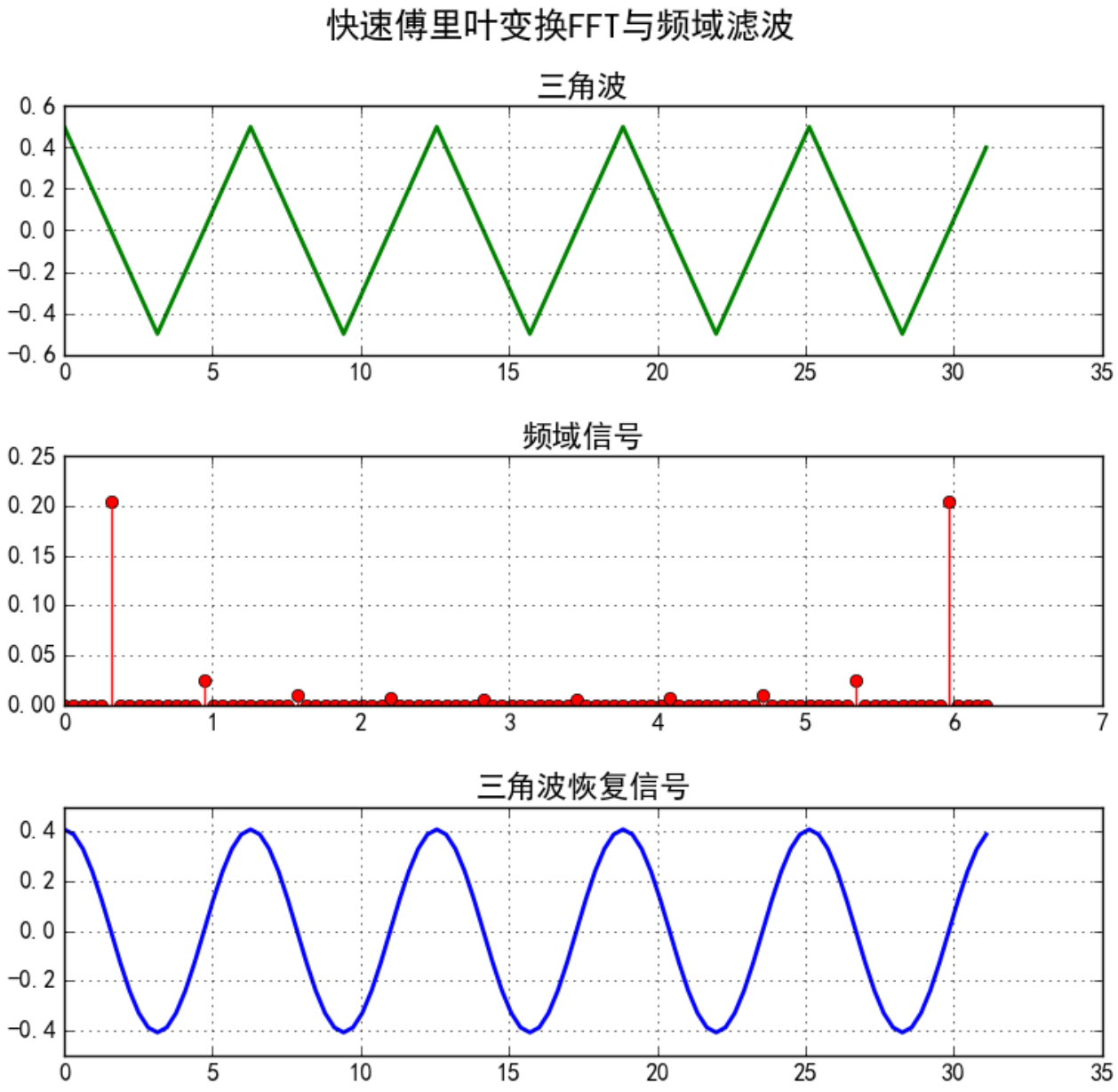
快速傅里叶变换FFT与频域滤波



快速傅里叶变换FFT与频域滤波



不同的阈值


快速傅里叶变换FFT与频域滤波
三角波
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
5
10
15
20
25
30
35
频域信号
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
7
三角波恢复信号
0.4
0.2
0.0
−0.2
−0.4
0
5
10
15
20
25
30
35

SVD的提法

$$\left(A^T \cdot A\right)v_i = \lambda_i v_i \Rightarrow \begin{cases} \sigma_i = \sqrt{\lambda_i} \\ u_i = \dfrac{1}{\sigma_i} A \cdot v_i \end{cases} \Rightarrow A = U\Sigma V^T$$

- 奇异值分解(Singular Value Decomposition)是一种重要的矩阵分解方法，可以看做对称方阵在任意矩阵上的推广。
  - Singular：突出的、奇特的、非凡的
  - 似乎更应该称之为“优值分解”
- 假设A是一个m×n阶实矩阵，则存在一个分解使得：

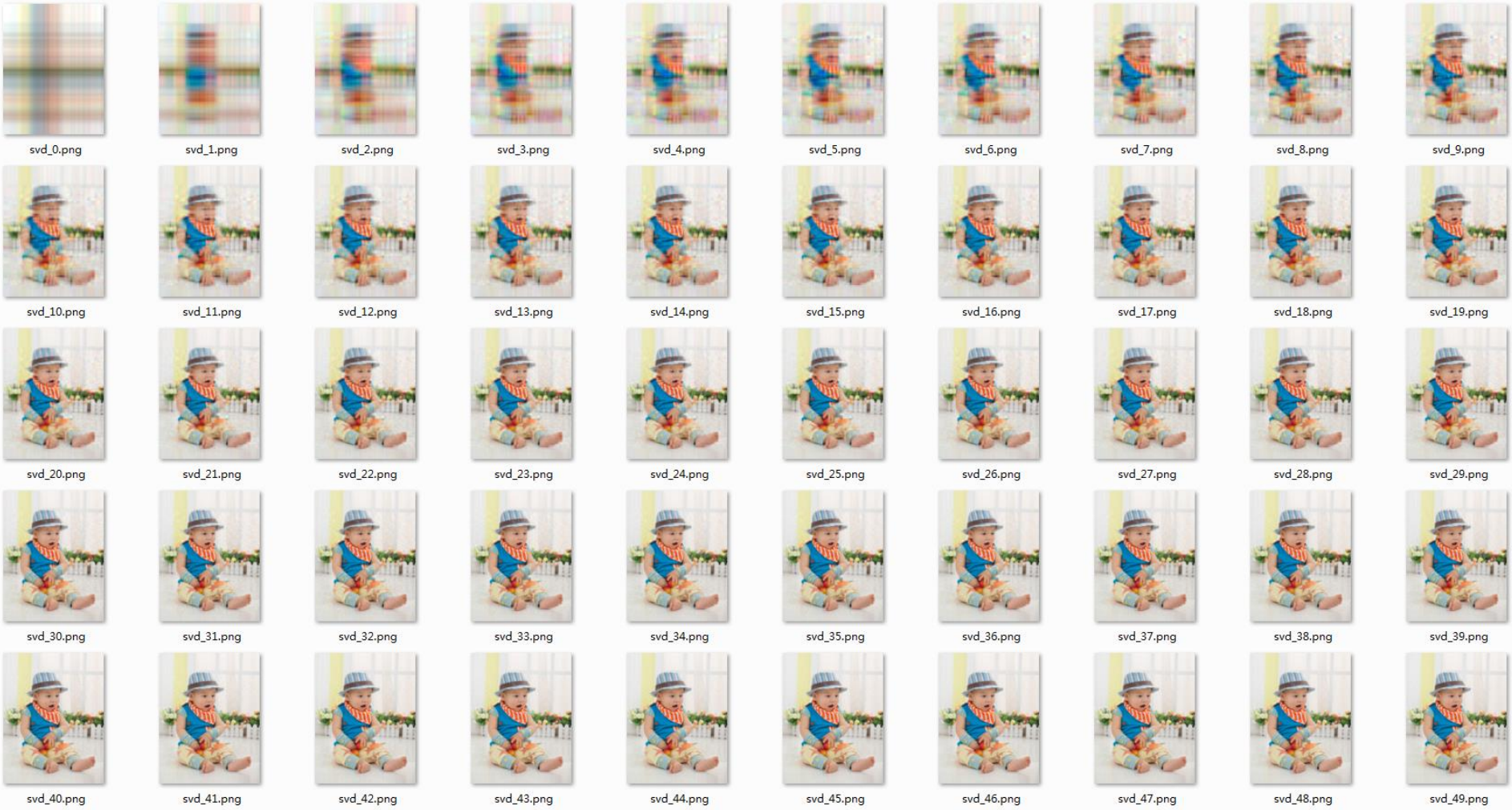
$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$$

  - 通常将奇异值由大而小排列。这样，Σ便能由A唯一确定了。
- 与特征值、特征向量的概念相对应：
  - Σ对角线上的元素称为矩阵A的奇异值；
  - U的第i列称为A的关于σ_i的左奇异向量；
  - V的第i列称为A的关于σ_i的右奇异向量。

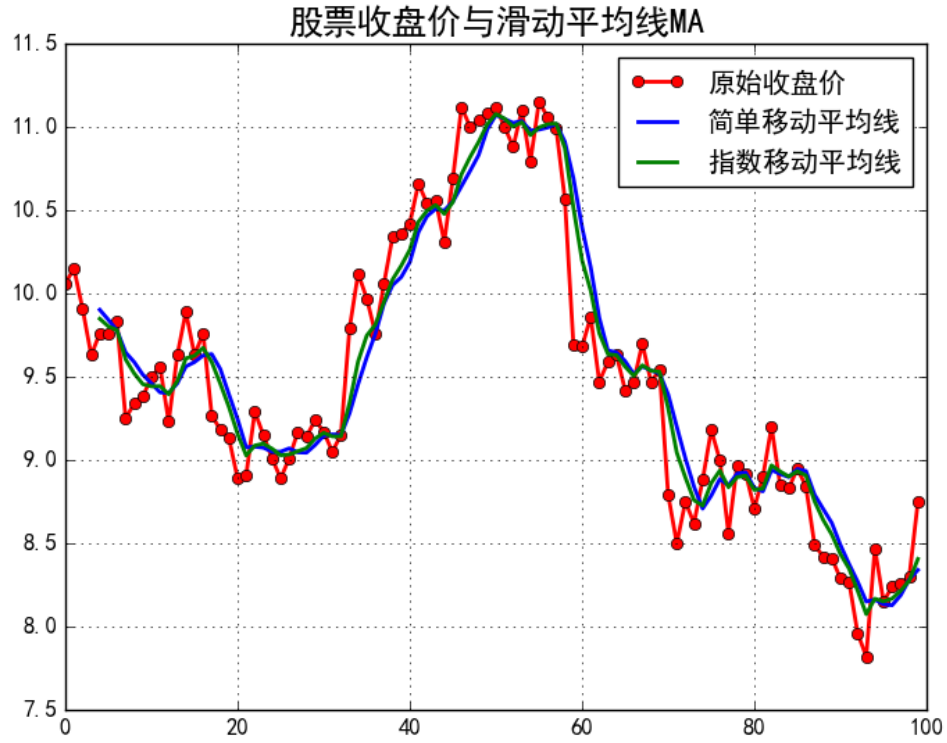
奇异值分解-效果

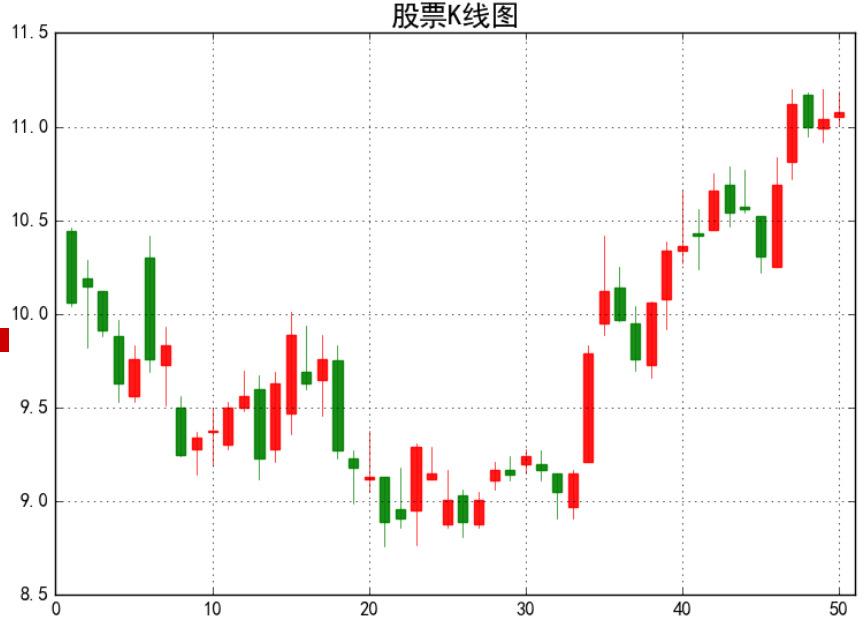
```
def restore(sigma, u, v, K): # 奇异值、左特征向量、右特征向量
    print K
    m = len(u)
    n = len(v[0])
    a = np.zeros((m, n))
    for k in range(K+1):
        for i in range(m):
            a[i] += sigma[k] * u[i][k] * v[k]
    b = a.astype('uint8')
    Image.fromarray(b).save("svd_" + str(K) + ".png")
```

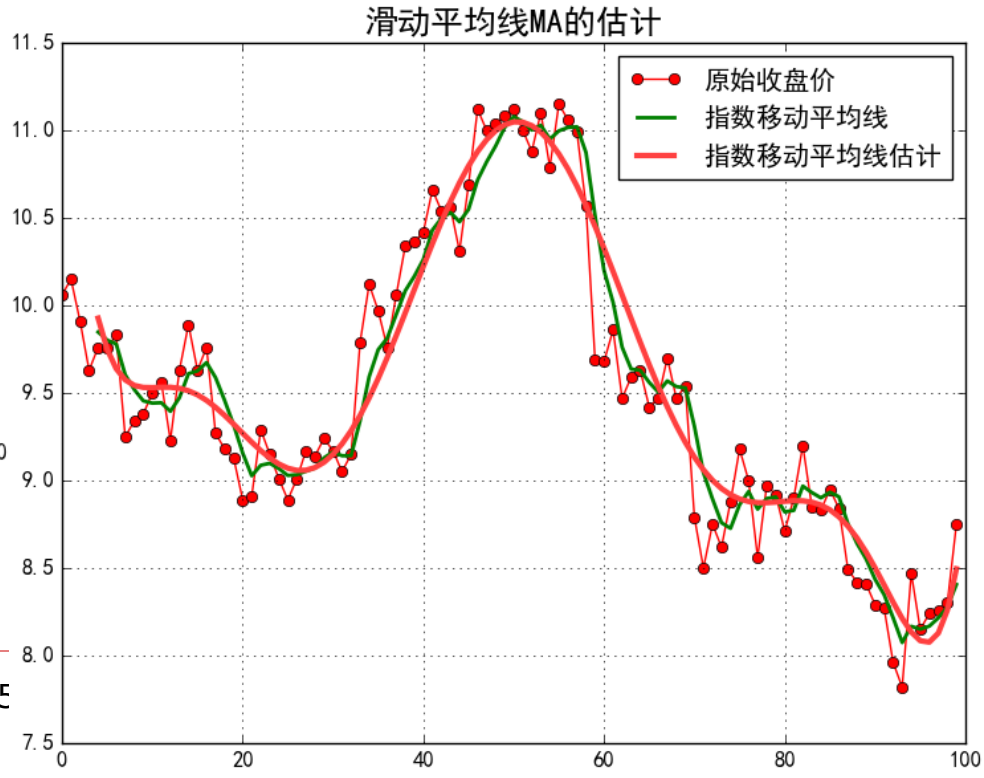
SVD


svd_0.png
svd_1.png
svd_2.png
svd_3.png
svd_4.png
svd_5.png
svd_6.png
svd_7.png
svd_8.png
svd_9.png
svd_10.png
svd_11.png
svd_12.png
svd_13.png
svd_14.png
svd_15.png
svd_16.png
svd_17.png
svd_18.png
svd_19.png
svd_20.png
svd_21.png
svd_22.png
svd_23.png
svd_24.png
svd_25.png
svd_26.png
svd_27.png
svd_28.png
svd_29.png
svd_30.png
svd_31.png
svd_32.png
svd_33.png
svd_34.png
svd_35.png
svd_36.png
svd_37.png
svd_38.png
svd_39.png
svd_40.png
svd_41.png
svd_42.png
svd_43.png
svd_44.png
svd_45.png
svd_46.png
svd_47.png
svd_48.png
svd_49.png

某股票收盘数据处理


股票收盘价与滑动平均线MA
原始收盘价
简单移动平均线
指数移动平均线


股票K线图


滑动平均线MA的估计
原始收盘价
指数移动平均线
指数移动平均线估计

图像的卷积

laplacian.png

laplacian2.png

prewitt.png

prewitt_x.png

prewitt_y.png

soble.png

soble_x.png

soble_y.png

图像的卷积

laplacian.png

laplacian2.png

prewitt.png

prewitt_x.png

prewitt_y.png

soble.png

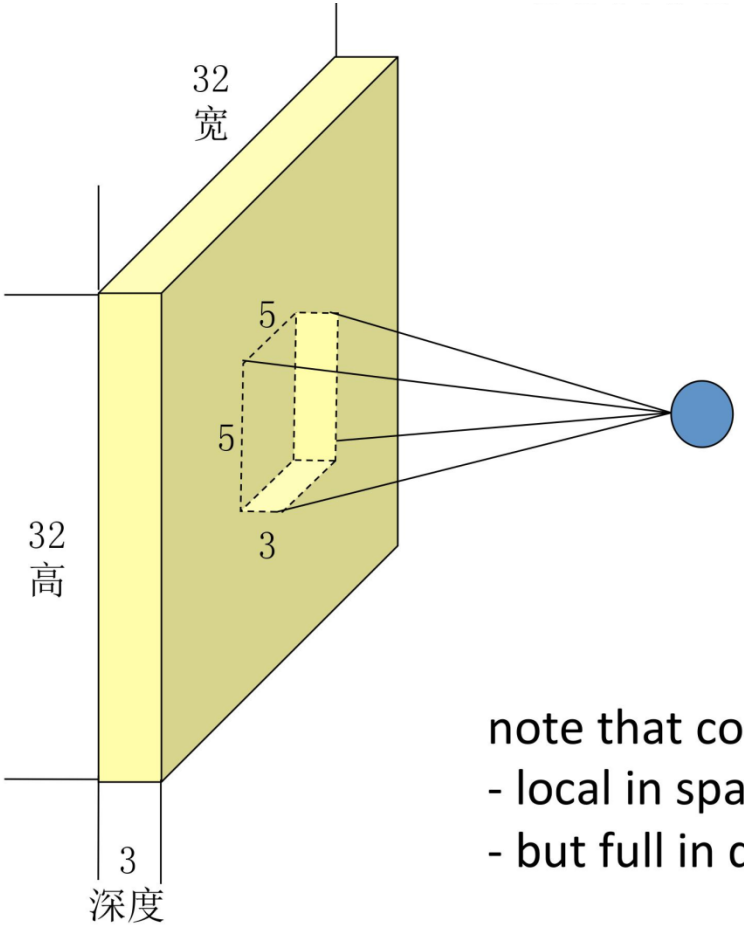
soble_x.png

soble_y.png

Code

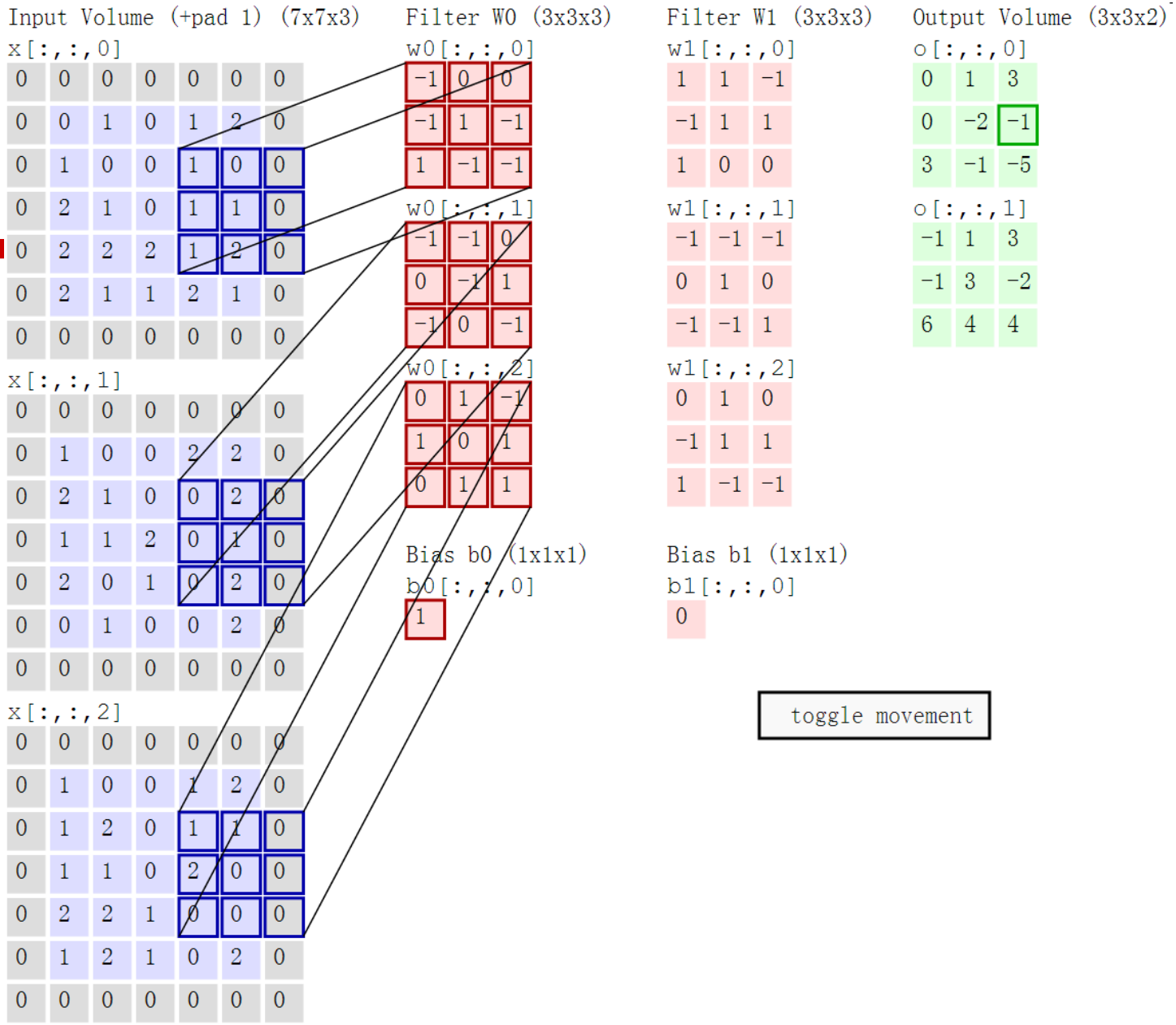
```
def convolve(image, weight):
    height, width = image.shape
    h, w = weight.shape
    height_new = height - h + 1
    width_new = width - w + 1
    image_new = np.zeros((height_new, width_new), dtype=np.float)
    for i in range(height_new):
        for j in range(width_new):
            image_new[i,j] = np.sum(image[i:i+h, j:j+w] * weight)
    image_new = image_new.clip(0, 255)
    image_new = np.rint(image_new).astype('uint8')
    return image_new
```

卷积网络

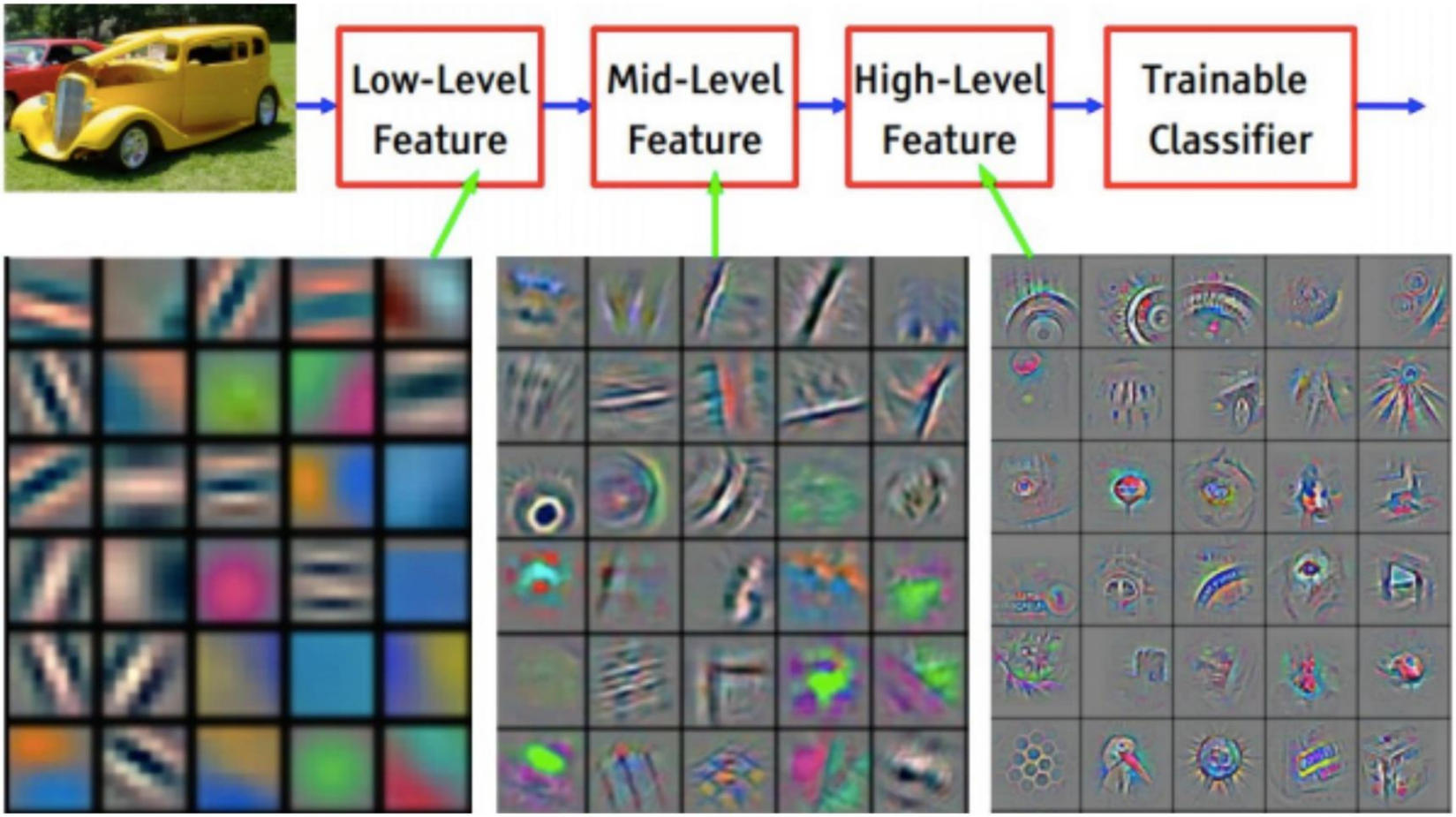


note that connectivity is:
- local in space (5x5 inside 32x32)
- but full in depth (all 3 depth channels)

卷积


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 0 1 2 0
0 1 0 0 1 0 0
0 2 1 0 1 1 0
0 2 2 2 1 2 0
0 2 1 1 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 2 2 0
0 2 1 0 0 2 0
0 1 1 2 0 1 0
0 2 0 1 0 2 0
0 0 1 0 0 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 0 0 1 2 0
0 1 2 0 1 1 0
0 1 1 0 2 0 0
0 2 2 1 0 0 0
0 1 2 1 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 0 0
-1 1 -1
1 -1 -1
w0[:,:,1]
-1 -1 0
0 -1 1
-1 0 -1
w0[:,:,2]
0 1 -1
1 0 1
0 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 -1
-1 1 1
1 0 0
w1[:,:,1]
-1 -1 -1
0 1 0
-1 -1 1
w1[:,:,2]
0 1 0
-1 1 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
0 1 3
0 -2 -1
3 -1 -5
o[:,:,1]
-1 1 3
-1 3 -2
6 4 4
toggle movement

深度网络


Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier

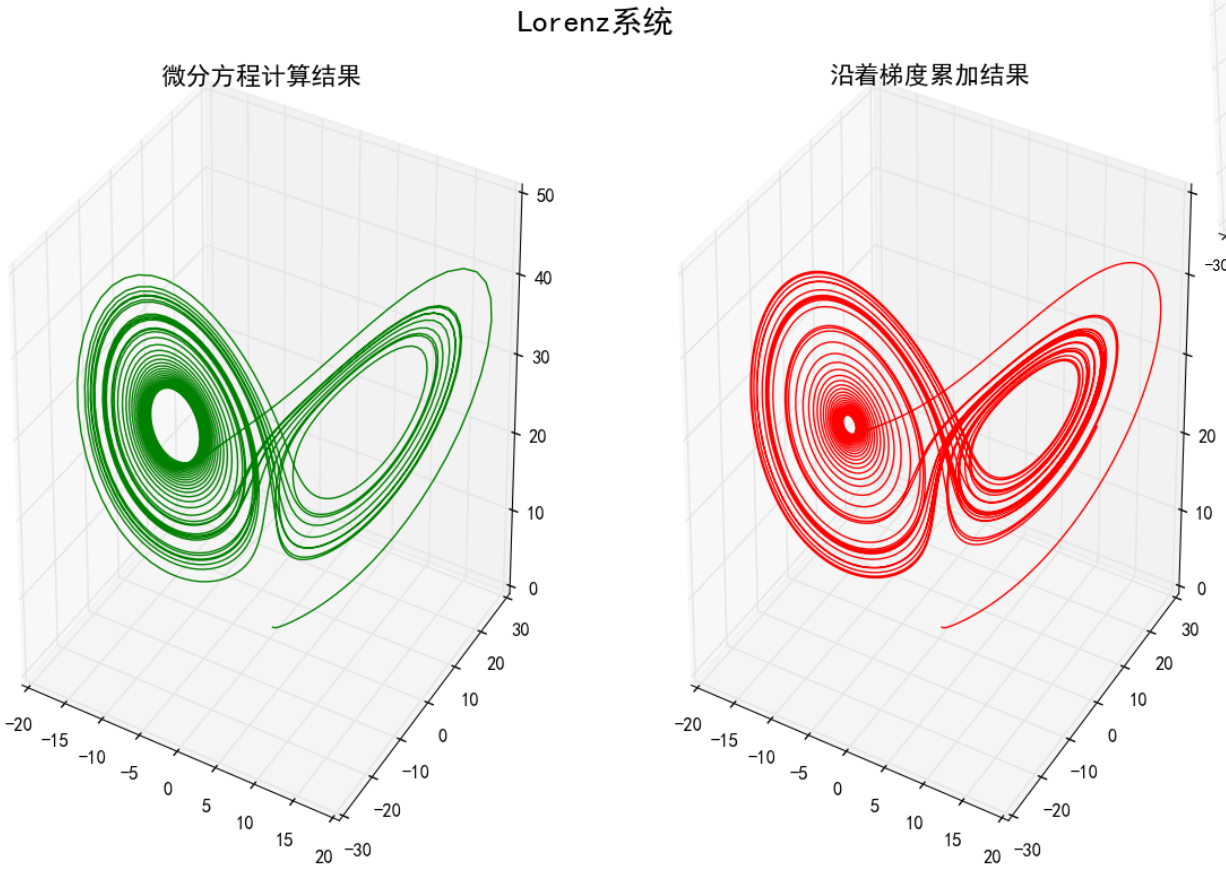
VGGNet

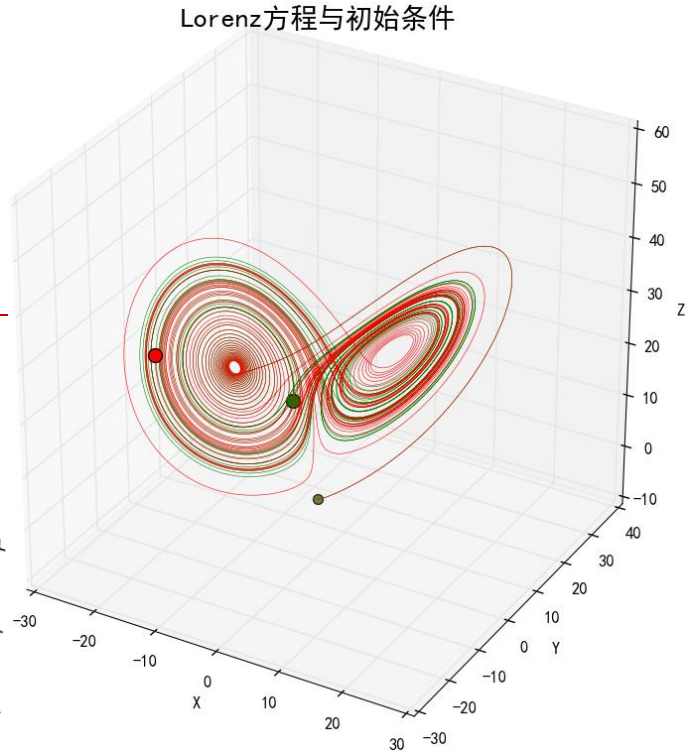
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

常微分方程

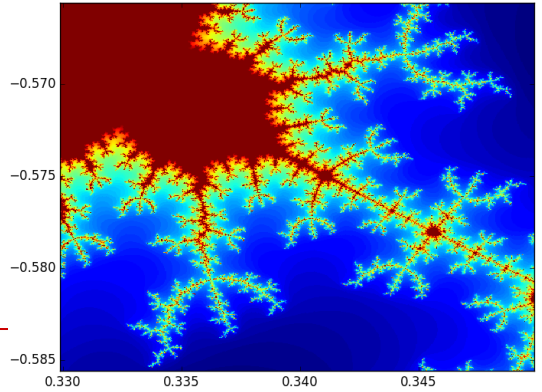




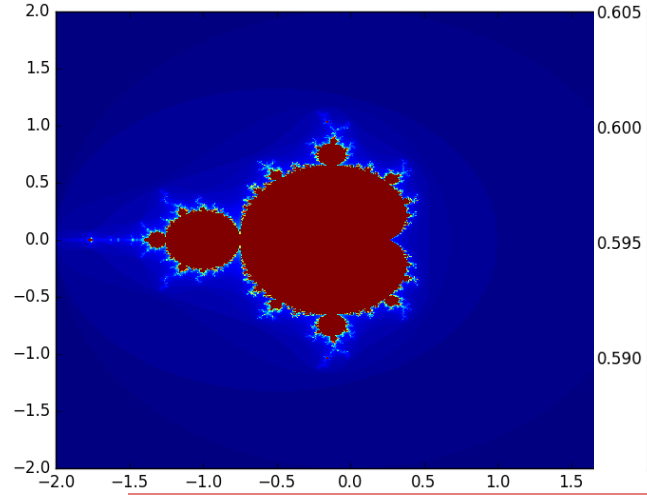
Code

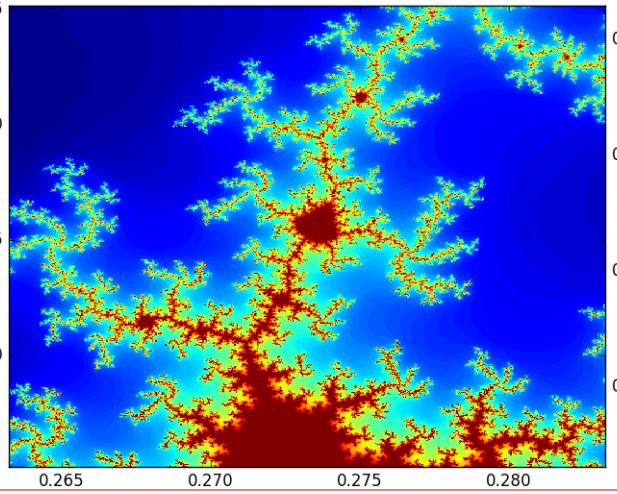
```
s0 = (0., 1., 0.)
t = np.arange(0, 30, 0.01)
s = odeint(lorenz, s0, t)
plt.figure(figsize=(12, 8), facecolor='w')
plt.subplot(121, projection='3d')
plt.plot(s[:, 0], s[:, 1], s[:, 2], c='g')
plt.title(u'微分方程计算结果', fontsize=16)

s = lorenz_trajectory(s0, 40000)
plt.subplot(122, projection='3d')
plt.plot(s[:, 0], s[:, 1], s[:, 2], c='r')
plt.title(u'沿着梯度累加结果', fontsize=16)

plt.tight_layout(1, rect=(0,0,1,0.98))
plt.suptitle(u'Lorenz系统', fontsize=20)
plt.show()

ax = Axes3D(plt.figure(figsize=(8, 8)))
s0 = (0., 1., 0.)
s1 = lorenz_trajectory(s0, 50000)
s0 = (0., 1.0001, 0.)
s2 = lorenz_trajectory(s0, 50000)
# 曲线
ax.plot(s1[:, 0], s1[:, 1], s1[:, 2], c='g', lw=0.4)
ax.plot(s2[:, 0], s2[:, 1], s2[:, 2], c='r', lw=0.4)
# 起点
ax.scatter(s1[0, 0], s1[0, 1], s1[0, 2], c='g', s=50, alpha=0.5)
ax.scatter(s2[0, 0], s2[0, 1], s2[0, 2], c='r', s=50, alpha=0.5)
# 终点
ax.scatter(s1[-1, 0], s1[-1, 1], s1[-1, 2], c='g', s=100)
ax.scatter(s2[-1, 0], s2[-1, 1], s2[-1, 2], c='r', s=100)
ax.set_title(u'Lorenz方程与初始条件', fontsize=20)
ax.set_xlabel(u'X')
ax.set_ylabel(u'Y')
ax.set_zlabel(u'Z')
plt.show()
```

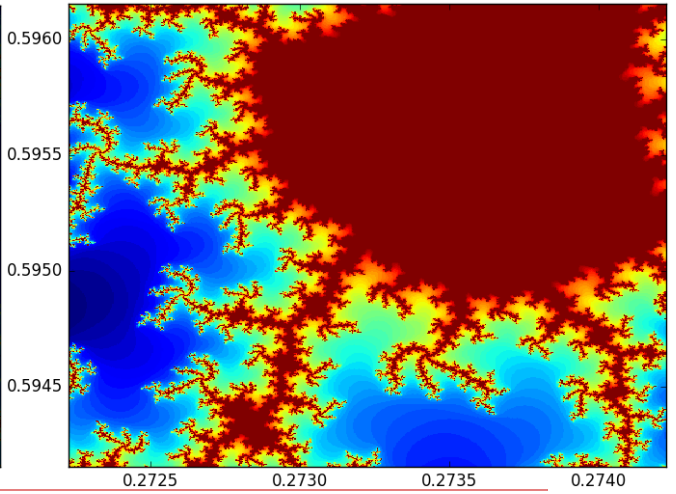
作业



- 实现任何一个函数曲线/曲面的Python显示。
  - 如Mandelbrot集
- 尝试使用SVD实现图像处理和特征提取。
- 熟悉Python的Numpy/Scipy数值计算数学库。







我们在这里

- http://wenda.ChinaHadoop.cn
  - 视频/课程/社区
- 微博
  - @ChinaHadoop
  - @邹博_机器学习
- 微信公众号
  - 小象
  - 大数据分析挖掘

感谢大家！

恳请大家批评指正！