一、Introduction

1. 安装Numpy 与SciPy

（1）使用 Python 发行包

常见的 Python 发行包：

Enthought Python Distribution (EPD)

ActivePython (AP)

Python(x,y)

（2）Python Windows

Step1：安装 Python

下载 https://www.python.org/downloads/release/python-2710/，64 位操作系统选择 Windows x86_64 MSI installer，32 位操作系统选择 Windows x86 MSI installer

双击安装（最好选择默认路径）

Step 2：安装包管理器 pip

a. 下载 https://pypi.python.org/pypi/pip/ ，选择 pip-8.1.1.tar.gz，解压。大命令行进入 dist\pip-8.1.1

b. 运行 python setup.py install

c. 修改环境变量 PATH，添加 C:\Python27\Scripts

Step 3：使用包管理器安装 numpy、scipy

NumPy 和 SciPy 在 windows 下需要手动安装,否则容易出现意外的错误。过程如下

a. 从 http://www.lfd.uci.edu/~gohlke/pythonlibs/ 下载相应 whl 包(下载 python2.7 版本 cp27，32 位下载 win32，64 位下载 win_amd64）手动安装，手动安装注意依赖关系

b. pip install wheel

c. pip install xxxxx.whl

Step 4：安装其他包

pip install pillow

pip install pandas

pip install -U scikit-learn

pip install matplotlib

pip install ipython

pip install pyreadline

（3）Python OS X

Step 1：安装 homebrew

/usr/bin/ruby -e "$(curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/master/install)"

Step 2：重新安装 python

brew install python

Step 3：安装 pip

sudo easy_install pip

Step 4：安装其他包

```
pip install numpy
pip install scipy
pip install pillow
pip install pandas
pip install -U scikit-learn
pip install matplotlib
pip install ipython
```

二、Numpy

1. numpy 数组与 python 列表效率对比

```
import numpy as np
# 创建大小为 10^7 的数组
arr = np.arange(1e7)
larr = arr.tolist()
def list_times(alist, scalar):
    for i, val in enumerate(alist):
        alist[i] = val * scalar
    return alist
# 利用 IPython 的魔术方法 timeit 计算运行时间
timeit arr * 1.1
>>> 1 loops, best of 3: 76.9 ms per loop
timeit list_times(larr, 1.1)
 >>> 1 loops, best of 3: 2.03 s per loop
```

2. 创建数组并设置数据类型

（1）从列表转换

```
alist = [1, 2, 3]
arr = np.array(alist)
```

（2）np.arange()

```
arr = np.arange(100)
arr = np.arange(10,100)
```

（3）np.zeros()

```
arr = np.zeros(5)
np.zeros((5,5))
cube = np.zeros((5,5,5)).astype(int) + 1
cube = np.ones((5, 5, 5)).astype(np.float16)
arr = np.zeros(2, dtype=int)
arr = np.zeros(2, dtype=np.float32)
```

（4）reshape()

```
arr1d = np.arange(1000)
arr3d = arr1d.reshape((10,10,10))
arr3d = np.reshape(arr1s, (10, 10, 10))
```

（5）revel()

作用与 reshape 相反

（6）shape

显示数据对象的形状

arr1d.shape

注意：对数据形状结构的改变只是改变了数据的显示形式，即只是改变了数据的引用，对一个数据的改变另一个也会被改变。

3. 记录数组

（1）创建记录数组并赋值

```
recarr = np.zeros((2,), dtype=('i4,f4,a10'))
#创建大小为 2 的记录数组，类型为 4 字节整数、4 字节浮点数和 10 字节字符
recarr[:] = [(1,2.,'Hello'),(2,3.,"World")]
```

（2）使用 zip()

```
recarr = np.zeros((2,), dtype=('i4,f4,a10'))
col1 = np.arange(2) + 1
col2 = np.arange(2, dtype=np.float32)
col3 = ['Hello', 'World']
recarr[:] = zip(col1, col2, col3)
```

（3）为每列数据命名

```
recarr.dtype.names = ('Integers' , 'Floats', 'Strings')
```

（4）使用列名访问数据

```
recarr('Integers')
```

4. 索引和切片

（1）numpy 提供了类似于 matlab 的索引和切片

```
alist=[[1,2],[3,4]]
alist[0][1]              #python 方式
arr = np.array(alist)
arr[0,1]                 #单个元素
arr[:,1]                 #第 1 列
arr[1,:]                 #第 1 行
```

（2）np.where()

根据条件获取索引号

```
index = np.where(arr>2)
new_arr = arr[index]
new_arr = np.delete(arr, index)
```

也可以这样操作：

```
index = arr > 2          #得到一个逻辑数组
new_arr = arr[index]
```

注意：第二种方法速度更快，而且可以用“~index”很容易的得到与 index 相反的逻辑数组。

5. NumPy 数组的布尔操作

NumPy 数组元素可以通过逻辑表达式方便的操作

例：

```
# 创建如 Plot A 所示的数组
img1 = np.zeros((20, 20)) + 3
img1[4:-4, 4:-4] = 6
img1[7:-7, 7:-7] = 9

# 获取数值大于 2 且小于 6 的元素索引
index1 = img1 > 2
index2 = img1 < 6
compound_index = index1 & index2

# 上式与下式结果相同
compound_index = (img1 > 3) & (img1 < 7)
img2 = np.copy(img1)
img2[compound_index] = 0
# 得到 Plot B.

# 更复杂的数组逻辑操作
index3 = img1 == 9
index4 = (index1 & index2) | index3
img3 = np.copy(img1)
img3[index4] = 0 # 得到 Plot C.
```

例：

```
import numpy.random as rand
a = rand.randn(100)
index = a > 0.2
b = a[index]
b = b ** 2 – 2
a[index] = b
```

6. 读写操作

（1）Python 读写文本文件

```
f = open('somefile.txt', 'r')        #以只读方式打开文件，'r'表示读
alist = f.readlines()                #将文件内容读入列表，每一行为一个列表元素
file f.close()                       #关闭文件
```

```
f = open('newtextfile.txt', 'w')   #以可写方式打开文件，'w'表示写
f.writelines(newdata)              #写入数据
f.close()                          #关闭文件
```

注意：读写完毕之后要将文件关闭

（2）Numpy 文件文件读写

Python 读写文件文件虽然方便且效率很好，但是不太适合处理极大的文件。当文件内容有结构，且为数字时用 NumPy 处理，存 numpy.ndarray 会更合适。

例：

```
import numpy as np
arr = np.loadtxt('somefile.txt')
np.savetxt('somenewfile.txt')
```

如果文件各列数据类型不一样，则需要指明数据类型，NumPy 用来保存数据的类型为 recarray，可以用处理 ndarray 同样的方法来对元素进行操作。recarray 数据类型不能直接保存为文本文件，如果需要的话可以使用 matplotlib.mlab 实现。

例：

文件 example.txt 内容如下

```
XR21 32.789 1
XR22 33.091 2
```

读入数据

```
table = np.loadtxt('example.txt',
                   dtype='names': ('ID', 'Result', 'Type'),
                   'formats': ('S4', 'f4', 'i2'))
```

提示：如果文本数据为 ASCII 格式的，使用 Asciitable 包读写会更加高效。

（3）二进制文件

文本文件处理简单方便，但是读写速度和文件大小都不能和二进制文件相比，因此大数据处理适合使用二进制文件。

例：

```
import numpy as np
data = np.empty((1000, 1000))     #创建一个较大的数组
np.save('test.npy', data)         #保存数据
np.savez('test.npz', data)        #压缩保存数据
newdata = np.load('test.npy')     #读入数据
```

注意：NumPy 使用 numpy.save 和 numpy.load 来读写二进制文件，但这种二进制文件只能在 NumPy 下读写，scipy.io 可以处理更通用的二进制文件

7. 数学运算

（1）线性代数

NumPy 数组间的运算只是相对应元素间的远算，不能用运算符进行矩阵运算，可以使用 numpy.dot 和 numpy.transpose 分别来进行矩阵乘法运算和矩阵转置。其优点在于常规操作时避免了对数据遍历。

NumPy 的 matrix 类型则可以直接用运算符号进行运算。

例：使用 matrix 解方程组

$$
\begin{aligned}
3x + 6y - 5z &= 12 \\
x - 3y + 2z &= -2 \\
5x - y + 4z &= 10
\end{aligned}
$$

$$
\begin{bmatrix} 3 & 6 & -5 \\ 1 & -3 & 2 \\ 5 & -1 & 4 \end{bmatrix}
\begin{bmatrix} x \\ y \\ z \end{bmatrix}
=
\begin{bmatrix} 12 \\ -2 \\ 10 \end{bmatrix}
$$

```
import numpy as np
A = np.matrix([[3, 6, -5], [1, -3, 2], [5, -1, 4]])     #定义矩阵
B = np.matrix([[12], [-2], [10]])
X = A ** (-1) * B                                        #求方程组
print(X)
```

例：使用数组解方程组

```
import numpy as np
a = np.array([[3, 6, -5], [1, -3, 2], [5, -1, 4]])
b = np.array([12, -2, 10])
x = np.linalg.inv(a).dot(b)
print(x)
```

注意：数组的运算速度更快，而且为了在使用中保持数据类型一致，建议使用数组。

三、SciPy

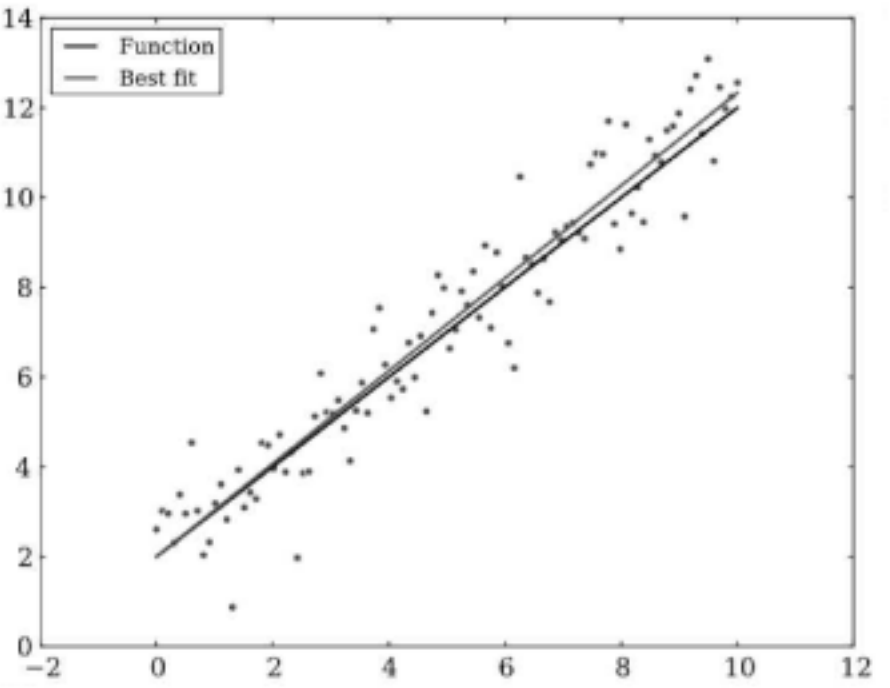
1. 最优化

（1）数据建模和拟合

SciPy 函数 curve_fit 使用基于卡方的方法进行线性回归分析。下面，首先使用 f(x)=ax+b 生成带有噪声的数据，然后使用用 curv_fit 来拟合。

例：线性回归

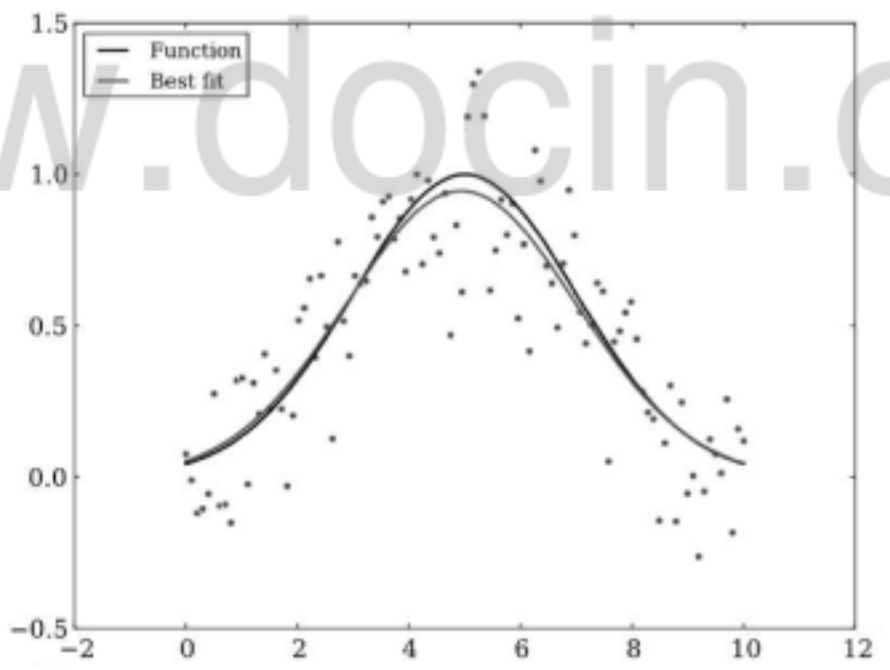
```
import numpy as np
from scipy.optimize import curve_fit
# 创建函数 f(x)=ax+b
def func(x, a, b):
    return a * x + b
# 创建干静数据
x = np.linspace(0, 10, 100)
y = func(x, 1, 2)
# 添加噪声
yn = y + 0.9 * np.random.normal(size=len(x))
# 拟合噪声数据
popt, pcov = curve_fit(func, x, yn)
# 输出最优参数
print(popt)
```



例：高斯分布拟合

$$a * \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

```
# 创建函数
def func(x, a, b, c):
    return a*np.exp(-(x-b)**2/(2*c**2))
# 生成干静数据
x = np.linspace(0, 10, 100)
y = func(x, 1, 5, 2)
# 添加噪声
yn = y + 0.2 * np.random.normal(size=len(x))
# 拟合
popt, pcov = curve_fit(func, x, yn)
```

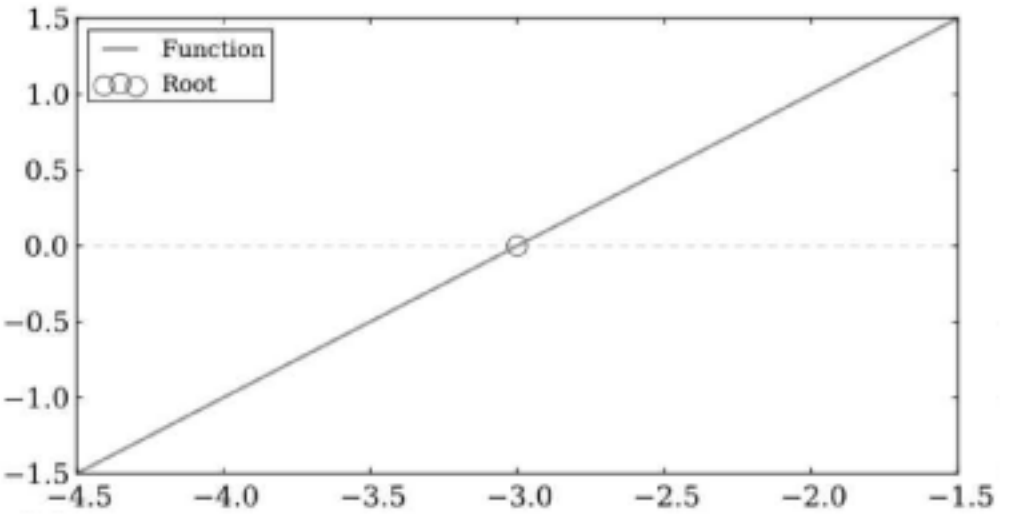


（2）函数求解

SciPy 的 optimize 模块中有大量的函数求解工具，fsolve 是其中最常用的。

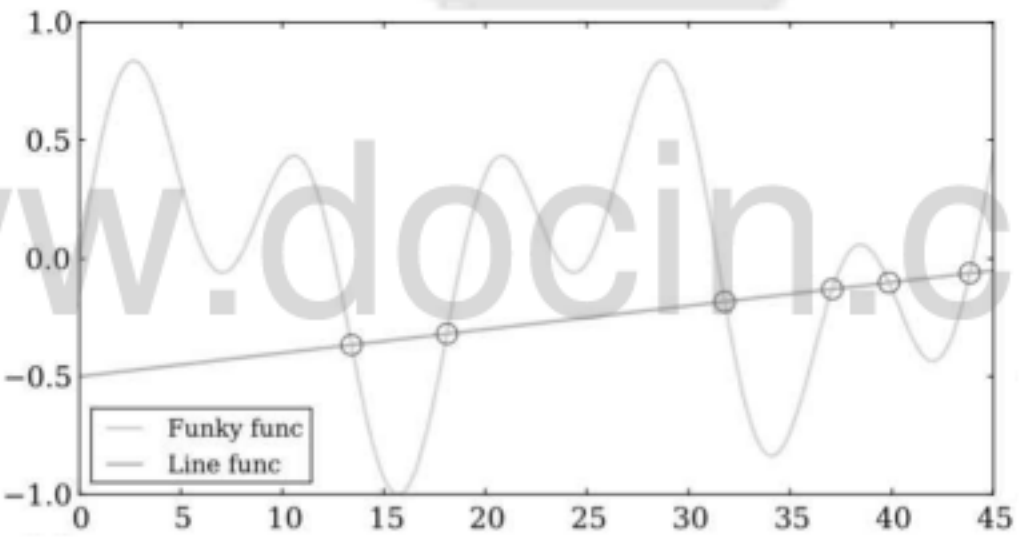
例：线性函数求解

```
from scipy.optimize import fsolve
import numpy as np
line = lambda x: x + 3
solution = fsolve(line, -2)
print solution
```



例：求函数交叉点

```
from scipy.optimize import fsolve
import numpy as np
# 用于求解的解函
def findIntersection(func1, func2, x0):
    return fsolve(lambda x : func1(x) - func2(x), x0)
# 两个函数
funky = lambda x : np.cos(x / 5) * np.sin(x / 2)
line = lambda x : 0.01 * x - 0.5

x = np.linspace(0,45,10000)
result = findIntersection(funky, line, [15, 20, 30, 35, 40, 45])
# 输出结果
print(result, line(result))
```



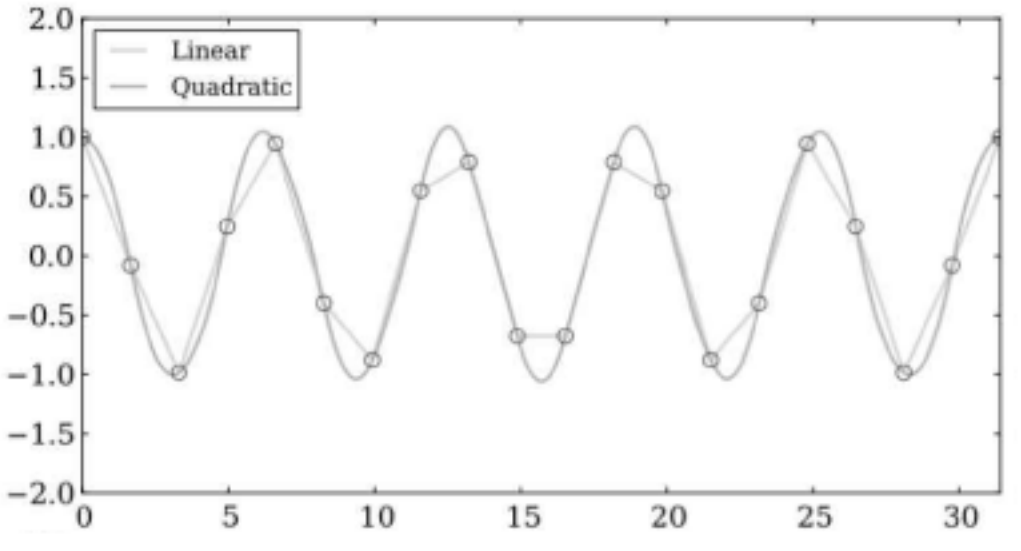
2. 插值

（1）interp1d

例：正弦函数插值

```
import numpy as np
from scipy.interpolate import interp1d
# 创建待插值的数据
x = np.linspace(0, 10 * np.pi, 20) y = np.cos(x)
# 分别用 linear 和 quadratic 插值
fl = interp1d(x, y, kind='linear')
fq = interp1d(x, y, kind='quadratic')
xint = np.linspace(x.min(), x.max(), 1000)
```

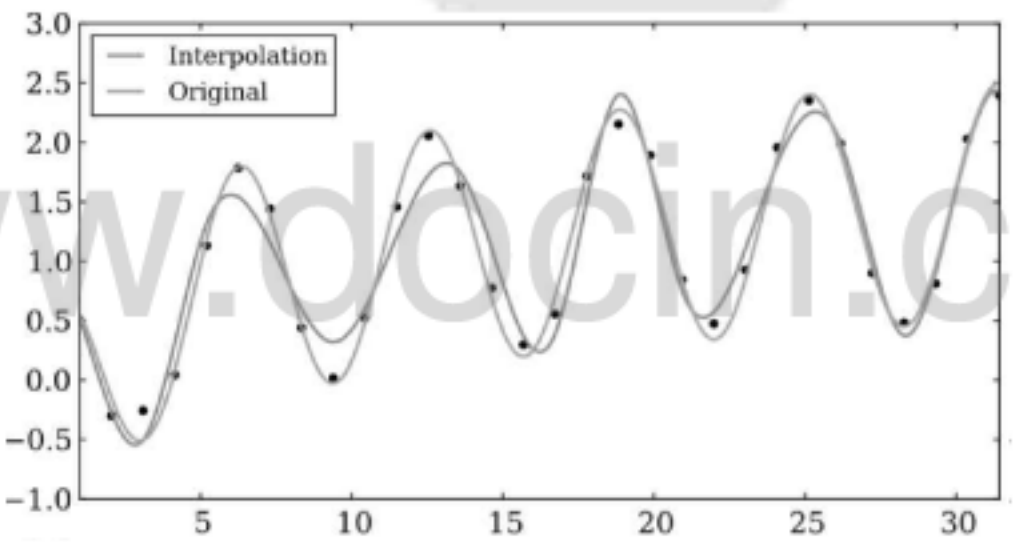
```
yintl = fl(xint)
yintq = fq(xint)
```



（2）UnivariateSpline

例：噪声数据插值

```
import numpy as np
from scipy.interpolate import UnivariateSpline
# 创建含噪声的待插值数据
sample = 30
x = np.linspace(1, 10 * np.pi, sample)
y = np.cos(x) + np.log10(x) + np.random.randn(sample) / 10
# 插值，参数 s 为 smoothing factor
f = UnivariateSpline(x, y, s=1)
xint = np.linspace(x.min(), x.max(), 1000)
yint = f(xint)
```

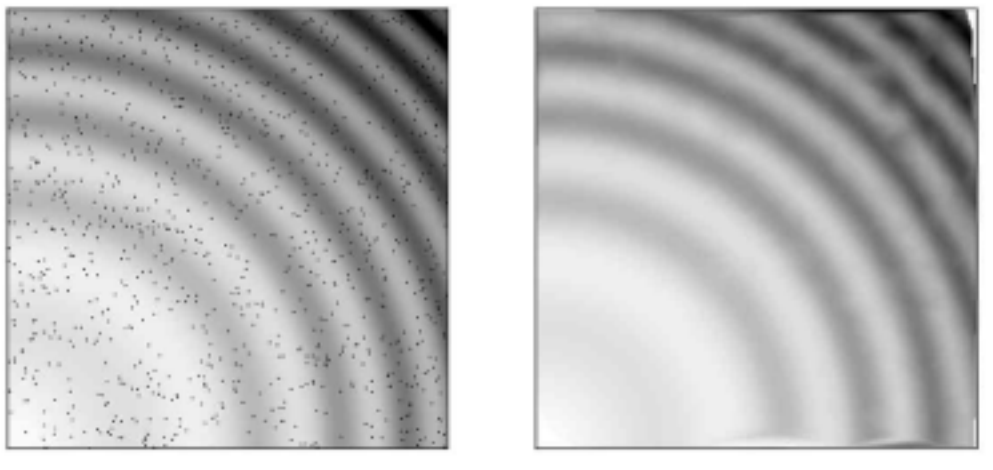


（3）griddata

例：利用插值重构图片

```
import numpy as np
from scipy.interpolate import griddata
# 定义一个函数
ripple = lambda x, y: np.sqrt(x**2 + y**2)+np.sin(x**2 + y**2)
# 生成 grid 数据，复数定义了生成 grid 数据的 step，若无该复数则 step 为 5
grid_x, grid_y = np.mgrid[0:5:1000j, 0:5:1000j]
# 生成待插值的样本数据
xy = np.random.rand(1000, 2)
sample = ripple(xy[:,0] * 5 , xy[:,1] * 5)
# 用 cubic 方法插值
```

```
grid_z0 = griddata(xy * 5, sample, (grid_x, grid_y), method='cubic')
```

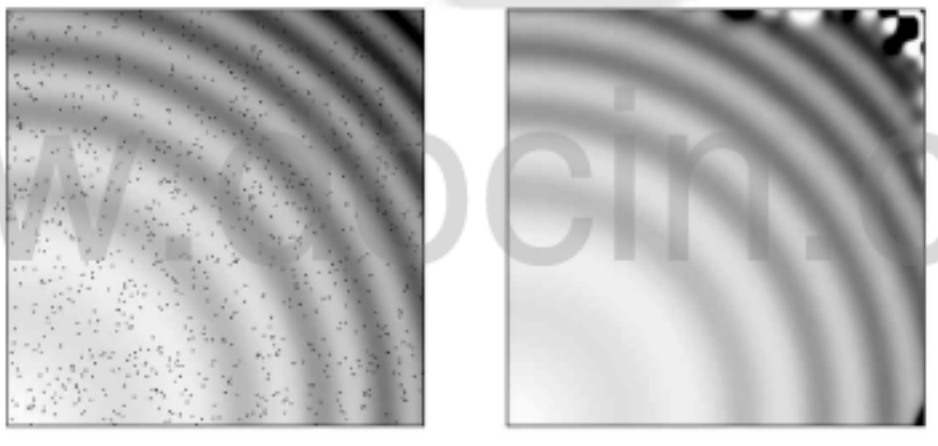


上图中，左侧为原始数据，其中的黑点是待插值的样本，右图为插值后的数据。要想提高质量，生成更大的样本数据即可。

（4）SmoothBivariateSpline

例：利用插值重构图片

```
import numpy as np
from scipy.interpolate import SmoothBivariateSpline as SBS
# 定义函数
ripple = lambda x, y: np.sqrt(x**2 + y**2)+np.sin(x**2 + y**2)
# 生成待插值样本
xy= np.random.rand(1000, 2)
x, y = xy[:,0], xy[:,1]
sample = ripple(xy[:,0] * 5 , xy[:,1] * 5)
# 插值
fit = SBS(x * 5, y * 5, sample, s=0.01, kx=4, ky=4)
interp = fit(np.linspace(0, 5, 1000), np.linspace(0, 5, 1000))
```



注意：SmoothBivariateSpline 有时候表现比 Spline 更好一些，但是它对样本数据更敏感一些，相对而言 Spline 更加健壮。

3. 积分

SicPy 中的积分是近似的数值积分，SymPy 是一个符号积分的工具包。

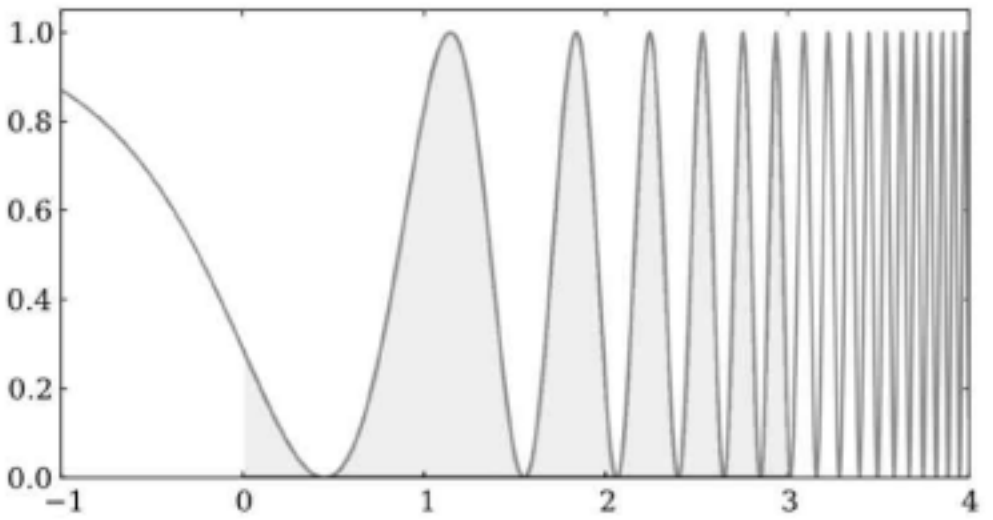
（1）解析积分

例：

$$\int_0^3 \cos^2(e^x)\,\mathrm{d}x$$

```
import numpy as np
from scipy.integrate import quad
# 定义被积函数
func = lambda x: np.cos(np.exp(x)) ** 2
```

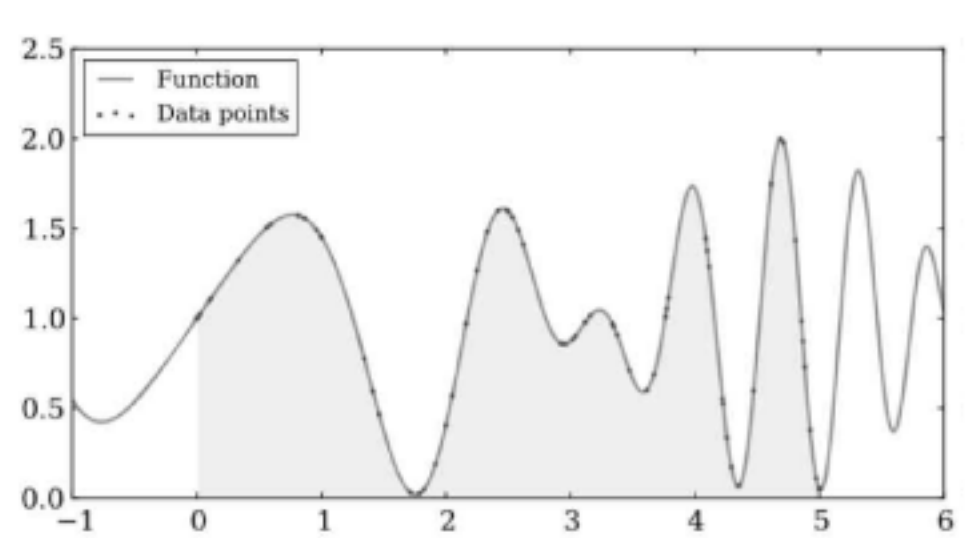
```
# 积分
solution = quad(func, 0, 3)
print solution
# 输出结果中第一个数字为积分值，第二个为误差
# (1.296467785724373, 1.397797186265988e-09)
```



（2）数值积分

例：

```
import numpy as np
from scipy.integrate import quad, trapz
# Setting up fake data
x = np.sort(np.random.randn(150) * 4 + 4).clip(0,5)
func = lambda x: np.sin(x) * np.cos(x ** 2) + 1
y = func(x)
# Integrating function with upper and lower # limits of 0 and 5, respectively
fsolution = quad(func, 0, 5)
dsolution = trapz(y, x=x)
print('fsolution = ' + str(fsolution[0]))
print('dsolution = ' + str(dsolution))
print('The difference is ' + str(np.abs(fsolution[0] - dsolution)))
# fsolution = 5.10034506754
# dsolution = 5.04201628314
# The difference is 0.0583287843989.
```



4. 统计

SciPy 中有包括 mean, std, median, argmax, 及 argmin 等在内的基本统计函数，而且 numpy.arrays 类型中内置了大部分统计函数，以便于使用。

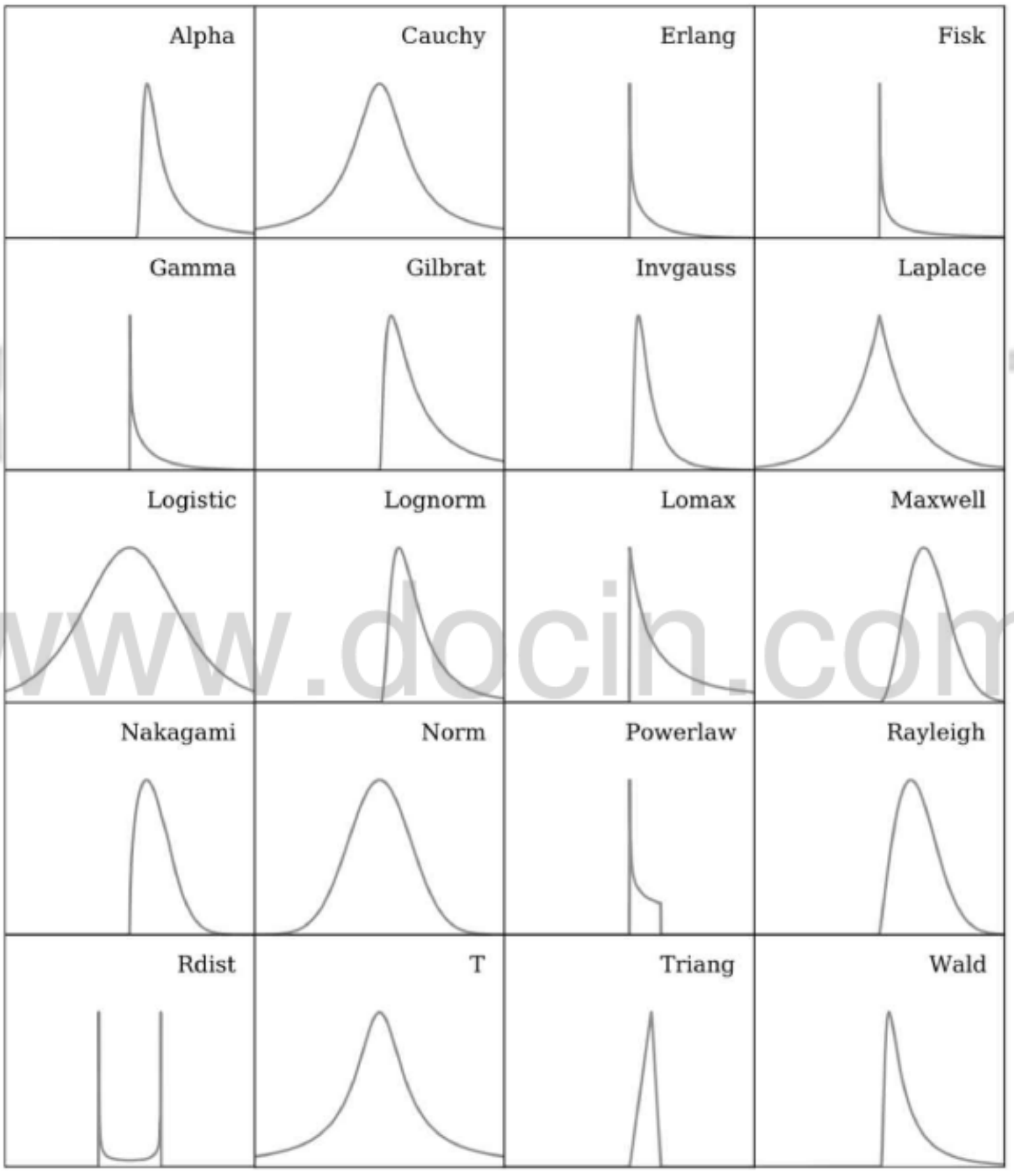
例：

```
import numpy as np
# 创建大小为 1000 的随机数组
elements x = np.random.randn(1000)
mean = x.mean()      #均值
std = x.std()        #标准差
var = x.var()        #方差
```

SciPy 中还包括了各种分布、函数等工具。

（1）连续和离散分布

SciPy 的 scipy.stats 包中包含了大概 80 种连续分布和 10 种离散分布。下图是其中的 20 种连续分布的概率密度函数。这些分布函数其实都依赖于 numpy.random 函数。



有几种方法来使用 scipy.stats 中的分布时：概率密度函数（PDFs）、累积分布函数（CDFs）、随机变量样本（RVSs）、百分比点函数（PPFs）等。下面基于标准正态分布函数，来演示如何使用这些分布。

$$\mathrm{PDF} = e^{(-x^2/2)/\sqrt{2\pi}}$$

例：

```
import numpy as np
import scipy.stats import norm
# 创建样本区间
x = np.linspace(-5,5,1000)
# 设置正态分布参数，loc 为均值，scale 为标准差
dist = norm(loc=0, scale=1)
# 得到正态分布的 PDF 和 CDF
pdf = dist.pdf(x)
cdf = dist.cdf(x)
# 根据分布生成 500 个随机数
sample = dist.rvs(500)
```

可以基于 SciPy.stats 中的任何连续分布生成随机数，有需要请查看文档。除此外，如泊松分布、二项分布、几何分布等离散分布的使用也很简单。下式为几何分布的概率分布函数（PMF）：

$$\mathrm{PMF} = (1-p)^{(k-1)}\,p$$

例：

```
import numpy as np
from scipy.stats import geom
# 设置几何分布的参数
p = 0.5
dist = geom(p)
# 设置样本区间
x = np.linspace(0, 5, 1000)
# 得到几何分布的 PMF 和 CDF
pmf = dist.pmf(x)
cdf = dist.cdf(x)
# 生成 500 个随机数
sample = dist.rvs(500)
```

（2）函数

SciPy 中有超过 60 种统计函数。stats 包中包括了诸如 kstest 和 normaltest 等的样本测试函数，用来检测样本是否服从某种分布。提示：在使用这些工具前，要对数据有较好的理解，否则可能会误读它们的结果。

例：样本分布检验

```
import numpy as np
from scipy import stats
# 生成包括 100 个服从正态分布的随机数样本
sample = np.random.randn(100)
# 用 normaltest 检验原假设
out = stats.normaltest(sample)
print('normaltest output')
print('Z-score = ' + str(out[0]))
print('P-value = ' + str(out[1]))
```

```
# kstest 是检验拟合度的 Kolmogorov-Smirnov 检验，这里针对正态分布进行检验，
# D 是 KS 统计量的值，越接近 0 越好
out = stats.kstest(sample, 'norm')
print('\nkstest output for the Normal distribution')
print('D = ' + str(out[0]))
print('P-value = ' + str(out[1]))
# 类似地可以针对其他分布进行检验，例如 Wald 分布
out = stats.kstest(sample, 'wald')
print('\nkstest output for the Wald distribution')
print('D = ' + str(out[0]))
print('P-value = ' + str(out[1]))
```

SciPy 的 stats 模块中还提供了一些描述函数，如几何平均（gmean）、偏度（skew）、样本频数（itemfreq）等。

例：

```
import numpy as np
from scipy import stats
#生成包括 100 个服从正态分布的随机数样本
sample = np.random.randn(100)
# 调和平均数，样本值须大于 0
out = stats.hmean(sample[sample > 0])
print('Harmonic mean = ' + str(out))
# 计算-1 到 1 之间样本的均值
out = stats.tmean(sample, limits=(-1, 1))
print('\nTrimmed mean = ' + str(out))
# 计算样本偏度
out = stats.skew(sample)
print('\nSkewness = ' + str(out))
# 函数 describe 可以一次给出样本的多种描述统计结果
out = stats.describe(sample)
print('\nSize = ' + str(out[0]))
print('Min = ' + str(out[1][0]))
print('Max = ' + str(out[1][1]))
print('Mean = ' + str(out[2]))
print('Variance = ' + str(out[3]))
print('Skewness = ' + str(out[4]))
print('Kurtosis = ' + str(out[5]))
```

SciPy 的 stats 模块中还有很多统计工具，可以满足绝大多数需要。还可以用 RPy，通过它能够在 Python 中调用 R 语言进行统计分析。此外，Pandas 是 python 的一个强大的工具包，它可以在大数据上进行快速的统计分析。

5. 空间和聚类分析

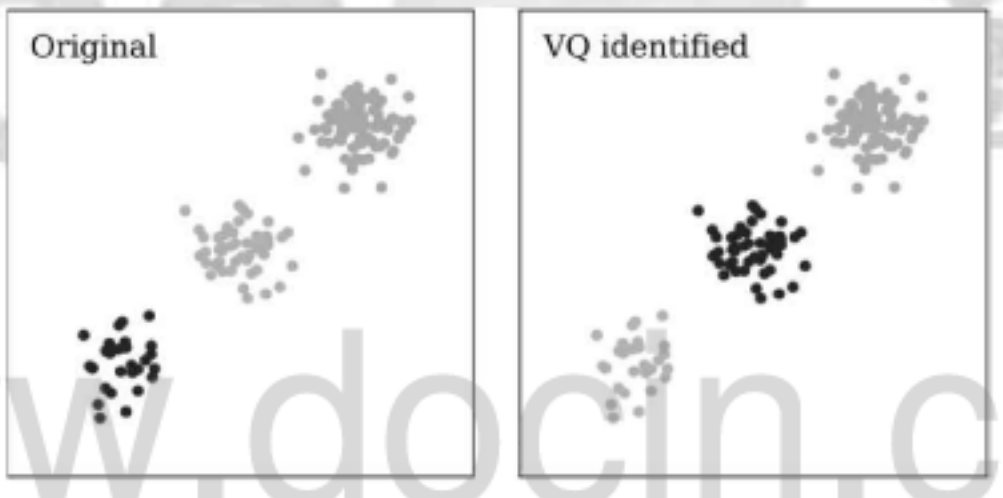
SciPy 包括 scipy.spatial 类和 scipy.cluster 类分别用于空间和聚类分析。前者用于分析数据点之间的距离，后者包括两个子类矢量量化（vq）和层次聚类（hierarchy）。

（1）矢量量化（Vector Quantization）

矢量量化是信号处理、数据压缩和聚类等领域通用的术语。这里仅关注其在聚类中的应用。

例：k 均值聚类

```
import numpy as np
from scipy.cluster import vq
# 生成数据
c1 = np.random.randn(100, 2) + 5
c2 = np.random.randn(30, 2) - 5
c3 = np.random.randn(50, 2)
# 将所有数据放入一个 180 x 2 的数组
data = np.vstack([c1, c2, c3])
# 利用 k 均值方法计算聚类的质心和方差
centroids, variance = vq.kmeans(data, 3)
# 变量 identified 中存放关于数据聚类的信息
identified, distance = vq.vq(data, centroids)
# 获得各类别的数据
vqc1 = data[identified == 0]
vqc2 = data[identified == 1]
vqc3 = data[identified == 2]
```



（2）层次聚类

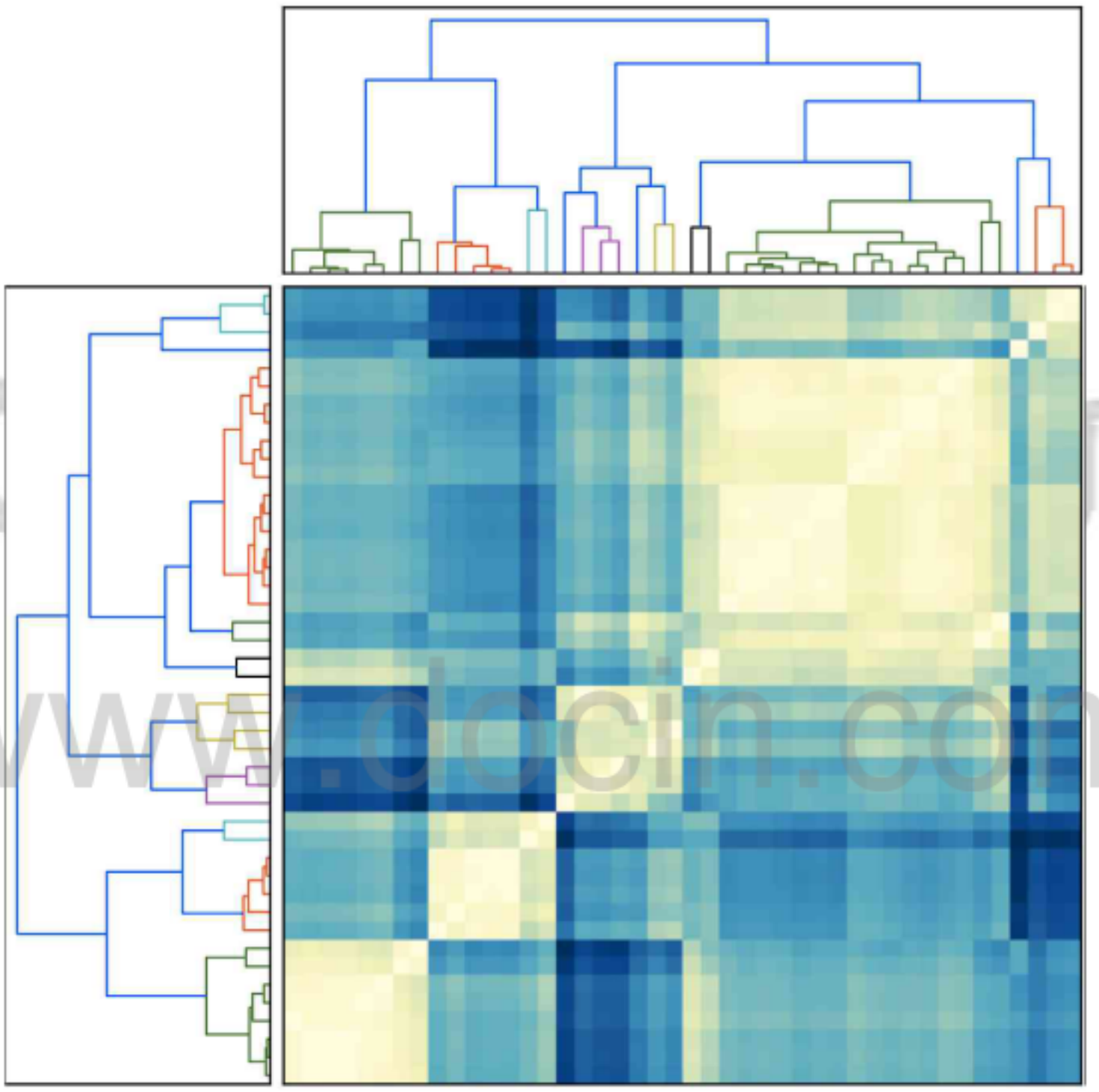
层次聚类是一种重要的聚类方法，但其输出结果比较复杂，不能像 k 均值那样给出清晰的聚类结果。下面是一个层次聚类的例子，输入一个距离矩阵，输出为一个树状图。

例：

```
# coding:utf-8
import numpy as np
import matplotlib.pyplot as mpl
from scipy.spatial.distance import pdist, squareform
import scipy.cluster.hierarchy as hy


# 用于生成聚类数据的函数
def clusters(number=20, cnumber=5, csize=10):
    # 聚类服从高斯分布
    rnum = np.random.rand(cnumber, 2)
    rn = rnum[:, 0] * number
```

```
    rn = rn.astype(int)
    rn[np.where(rn < 5)] = 5
    rn[np.where(rn > number / 2.)] = round(number / 2., 0)
    ra = rnum[:, 1] * 2.9
    ra[np.where(ra < 1.5)] = 1.5

    cls = np.random.randn(number, 3) * csize

    # Random multipliers for central point of cluster
    rxyz = np.random.randn(cnumber - 1, 3)
    for i in xrange(cnumber - 1):
        tmp = np.random.randn(rn[i + 1], 3)
        x = tmp[:, 0] + (rxyz[i, 0] * csize)
        y = tmp[:, 1] + (rxyz[i, 1] * csize)
        z = tmp[:, 2] + (rxyz[i, 2] * csize)
        tmp = np.column_stack([x, y, z])
        cls = np.vstack([cls, tmp])
    return cls


# 创建待聚类数据及矩离矩阵
cls = clusters()
D = pdist(cls[:, 0:2])
D = squareform(D)

# 绘制左侧树状图
fig = mpl.figure(figsize=(8, 8))
ax1 = fig.add_axes([0.09, 0.1, 0.2, 0.6])
Y1 = hy.linkage(D, method='complete')
cutoff = 0.3 * np.max(Y1[:, 2])
Z1 = hy.dendrogram(Y1, orientation='left', color_threshold=cutoff)
ax1.xaxis.set_visible(False)
ax1.yaxis.set_visible(False)

#绘制顶部树状图
ax2 = fig.add_axes([0.3, 0.71, 0.6, 0.2])
Y2 = hy.linkage(D, method='average')
cutoff = 0.3 * np.max(Y2[:, 2])
Z2 = hy.dendrogram(Y2, color_threshold=cutoff)
ax2.xaxis.set_visible(False)
ax2.yaxis.set_visible(False)

# 显示距离矩阵
ax3 = fig.add_axes([0.3, 0.1, 0.6, 0.6])
```

```
idx1 = Z1['leaves']
idx2 = Z2['leaves']
D = D[idx1, :]
D = D[:, idx2]
ax3.matshow(D, aspect='auto', origin='lower', cmap=mpl.cm.YlGnBu)
ax3.xaxis.set_visible(False)
ax3.yaxis.set_visible(False)

# 保存图片，显示图片
fig.savefig('cluster_hy_f01.pdf', bbox = 'tight')
mpl.show()
```



在上图虽然计算了数据点之间的距离，但是还是难以将各类区分开。函数 fcluster 可以根据阈值来区分各类，其输出结果依赖于 linkage 函数所采用的方法，如 complete 或 single 等，它的第二个参数即是阈值。dendrogram 函数中默认的阈值是 0.7 * np.max(Y[:, 2])，这里还使用 0.3。

例：

```
# 导入的包同上例一致，函数 cluster 同上例
# 获得不同类别数据点的坐标
def group(data, index):
    number = np.unique(index)
```

```
        groups = []
        for i in number:
            groups.append(data[index == i])
        return groups

# 创建数据
cls = clusters()
# 计算 kinkage 矩阵
Y = hy.linkage(cls[:,0:2], method='complete')
# 从层次数据结构中，用 fcluster 函数将层次结构的数据转为 flat clusters
cutoff = 0.3 * np.max(Y[:, 2])
index = hy.fcluster(Y, cutoff, 'distance')
# 使用 grooup 函数将数据划分类别
groups = group(cls, index)
# 绘制数据点
fig = mpl.figure(figsize=(6, 6))
ax = fig.add_subplot(111)
colors = ['r', 'c', 'b', 'g', 'orange', 'k', 'y', 'gray']
for i, g in enumerate(groups):
    i = np.mod(i, len(colors))
    ax.scatter(g[:,0], g[:,1], c=colors[i], edgecolor='none', s=50)
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
fig.savefig('cluster_hy_f02.pdf', bbox = 'tight')
mpl.show()
```


6. 信号和图像处理

SimPy 可以高效的处理 JPEG 和 PNG 等格式的图片，下面利用 SimPy 将多副图片叠加成一副图片。

本部分实例更多内容参见：

http://stackoverflow.com/questions/9251580/stacking-astronomy-images-with-python

例：

```
import numpy as np
from scipy.misc import imread, imsave
from glob import glob
# 获取文件夹中全部 JPG 图片
files = glob('space/*.JPG')
# 打开第一个图片
im1 = imread(files[0]).astype(np.float32)
# 叠加图片
for i in xrange(1, len(files)):
    print i
    im1 += imread(files[i]).astype(np.float32)
# 保存图片
imsave('stacked_image.jpg', im1)
```

例：

```
import numpy as np
from scipy.misc import imread, imsave
from glob import glob
# 该函数对比两副图片,选择两副图片中相同位置上较亮的点生成新的图片
def chop_lighter(image1, image2):
    s1 = np.sum(image1, axis=2)
    s2 = np.sum(image2, axis=2)
    index = s1 < s2
    image1[index, 0] = image2[index, 0]
    image1[index, 1] = image2[index, 1]
    image1[index, 2] = image2[index, 2]
    return image1
files = glob('space/*.JPG')
im1 = imread(files[0]).astype(np.float32)
im2 = np.copy(im1)

for i in xrange(1, len(files)):
    print i
```

```
        im = imread(files[i]).astype(np.float32)
        im1 += im
        im2 = chop_lighter(im2, im)
# 调整并保存图像
imsave('stacked_image.jpg', im1/im1.max() + im2/im2.max()*0.2)
```

注意：imread 和 imsave 需要安装 pillow 包，pip install pillow

Python 环境中的 JPG 图片格式为(426, 640, 3)的 NumPy 数组（RGB）

7. 稀疏矩阵

NumPy 处理 10^6 级别的数据没什么大问题，当数据量达到 10^7 级别时速度开始变慢，内存受到限制（具体情况取决于实际内存大小）。当处理超大规模数据集，比如 10^{10} 级别，且数据中包含大量的 0 时，可采用稀疏矩阵可显著的提高速度和效率。

提示：使用 data.nbytes 可查看数据所占空间大小

例：矩阵与稀疏矩阵运算对比

```
# coding:utf-8
import numpy as np
from scipy.sparse.linalg import eigsh
from scipy.linalg import eigh
import scipy.sparse
import time
N = 3000
# 创建随机稀疏矩阵
m = scipy.sparse.rand(N, N)
# 创建包含相同数据的数组
a = m.toarray()
print('The numpy array data size: ' + str(a.nbytes) + ' bytes')
print('The sparse matrix data size: ' + str(m.data.nbytes) + ' bytes')
# 数组求特征值
t0 = time.time()
res1 = eigh(a)
dt = str(np.round(time.time() - t0, 3)) + ' seconds'
print('Non-sparse operation takes ' + dt)
# 稀疏长阵求特征值
t0 = time.time()
```

```
res2 = eigsh(m)
dt = str(np.round(time.time() - t0, 3)) + ' seconds'
print('Sparse operation takes ' + dt)
```

非几何的稀疏矩阵可用于优化、经济建模、数学和统计,和网络/图等的运算。利用 scipy.io 模块可读写 Matrix Market、Harwell-Boeing 或 MatLab 格式的稀疏矩阵数据文件。

四、SciKit：SciPy 更进一步

SciKit 是 SciPy 的进阶和补充，有超过 20 个包（现在超过 50），更多信息参见 https://scikits.appspot.com/scikits ，本部分介绍其中两个使用最为广泛的： Scikit-image（比 scipy.ndimage 功能更强大的图片包）和 Scikit-learn（机器学习包）。

1. Scikit-Image

SciPy 的 ndimage 类包含许多有用的工具，可用于处理多维数据，例如基本过滤器（如高斯平滑）、傅里叶变换、morphology、插值、测量等。基于这些功能，我们可以处理更复杂的问题。Scikit-image 即是基于这些，提供了更丰富的功能，主要模块包括色彩空间转换、图像亮度调整算法、特征检测、锐化过滤器和去噪、读/写等。

（1）动态阈值转换

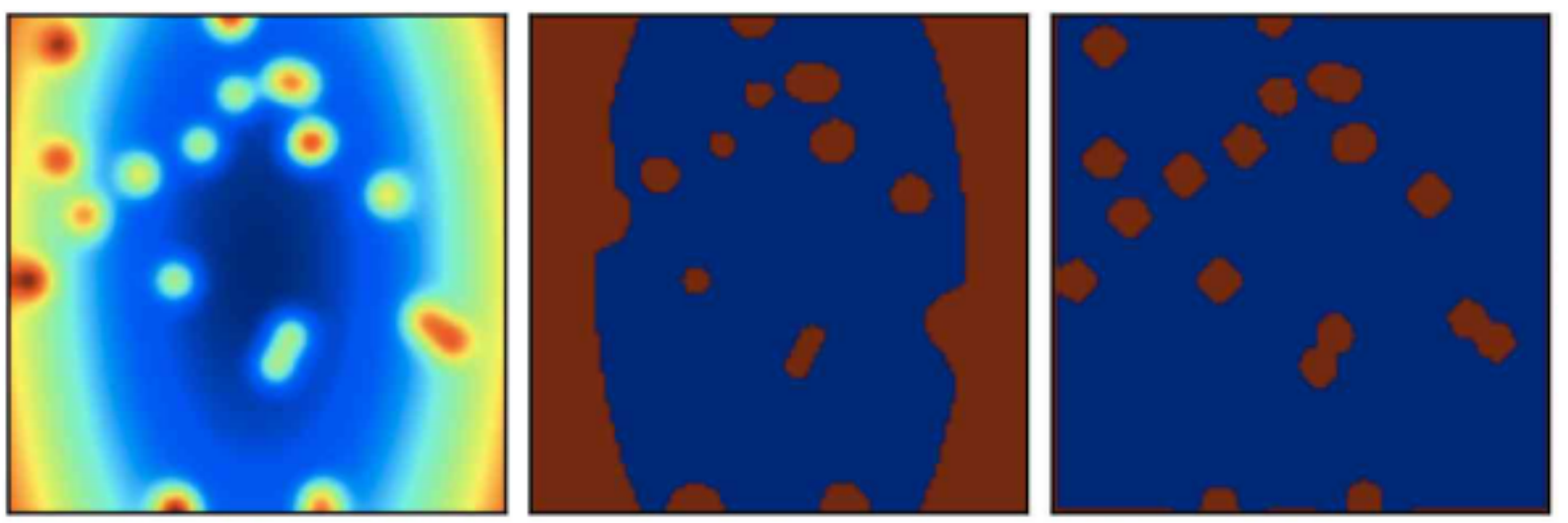
提取图像元素在图像处理中很常见，被称为域值转换（thresholding）。经典的技术处理均匀背景的图像很容易，但复杂背景的图片就较为困难了。Scikit-image 使用适应性阈值转换技术来解决这类问题，而且使用很简单。

下面用一个例子来说明。在该例子中，图片中包括一些随机的模糊点，而且背景非均匀，应用 Scikit-image 来提取其中的点。

例：图像元素识别——基本方法和自适应方法对比

```
import numpy as np
import matplotlib.pyplot as mpl
import scipy.ndimage as ndimage
import skimage.filter as skif
# 在非均匀背景中生成数据点
x = np.random.uniform(low=0, high=100, size=20).astype(int)
y = np.random.uniform(low=0, high=100, size=20).astype(int)
# 创建非均匀背景
func = lambda x, y: x**2 + y**2
grid_x, grid_y = np.mgrid[-1:1:100j, -2:2:100j]
bkg = func(grid_x, grid_y)
bkg = bkg / np.max(bkg)
# 创建数据点
clean = np.zeros((100,100))
clean[(x,y)] += 5
clean = ndimage.gaussian_filter(clean, 3)
clean = clean / np.max(clean)
```

```
# 将数据和背景相结合
fimg = bkg + clean
fimg = fimg / np.max(fimg)
# 定义待提取对像最小相邻距离
block_size = 3
# Adaptive threshold function which returns image map of structures that are
different relative to background
adaptive_cut = skif.threshold_adaptive(fimg, block_size, offset=0)

# Global threshold
global_thresh = skif.threshold_otsu(fimg)
global_cut = fimg > global_thresh

# Creating figure to highlight difference between adaptive and global threshold
methods
fig = mpl.figure(figsize=(8, 4))
fig.subplots_adjust(hspace=0.05, wspace=0.05)

ax1 = fig.add_subplot(131)
ax1.imshow(fimg)
ax1.xaxis.set_visible(False)
ax1.yaxis.set_visible(False)

ax2 = fig.add_subplot(132)
ax2.imshow(global_cut)
ax2.xaxis.set_visible(False)
ax2.yaxis.set_visible(False)

ax3 = fig.add_subplot(133)
ax3.imshow(adaptive_cut)
ax3.xaxis.set_visible(False)
ax3.yaxis.set_visible(False)

fig.savefig('scikit_image_f01.pdf', bbox_inches='tight')
mpl.show()
```



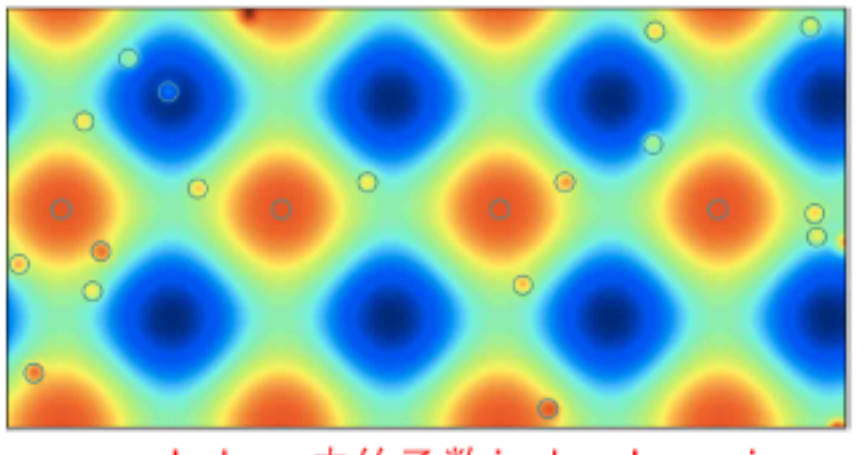
（2）局部极大值

与上一个问题类似，需要找出非均匀背景中元素的像素坐标。这里使用skimage.morphology.is_local_maximum。

例：

```
import numpy as np
import matplotlib.pyplot as mpl
import scipy.ndimage as ndimage
import skimage.morphology as morph
import skimage.feature as ft
# 生成非均匀背景数据
x = np.random.uniform(low=0, high=200, size=20).astype(int)
y = np.random.uniform(low=0, high=400, size=20).astype(int)
# 创建非均匀背景
func = lambda x, y: np.cos(x)+ np.sin(y)
grid_x, grid_y = np.mgrid[0:12:200j, 0:24:400j]
bkg = func(grid_x, grid_y)
bkg = bkg / np.max(bkg)

# 创建待识别对象点
clean = np.zeros((200,400))
clean[(x,y)] += 5
clean = ndimage.gaussian_filter(clean, 3)
clean = clean / np.max(clean)
# 将待识别对象点与背景结合
fimg = bkg + clean
fimg = fimg / np.max(fimg)

fig = mpl.figure(figsize=(8, 4))
ax = fig.add_subplot(111)
ax.imshow(fimg)

# 计算局部最大值（方法 1）
# lm1 = ft.peak_local_max(fimg, indices=False)
# x1, y1 = np.where(lm1.T == True)
# ax.scatter(x1, y1, s=100, facecolor='none', edgecolor='#009999')

#计算局部最大值（方法 2）
lm1 = ft.peak_local_max(fimg)
ax.scatter(lm1[:,1], lm1[:,0], s=100, facecolor='none', edgecolor='#009999')

ax.set_xlim(0,400)
ax.set_ylim(0,200)
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
```

```
fig.savefig('scikit_image_f02.pdf', bbox_inches='tight')
mpl.show()
```

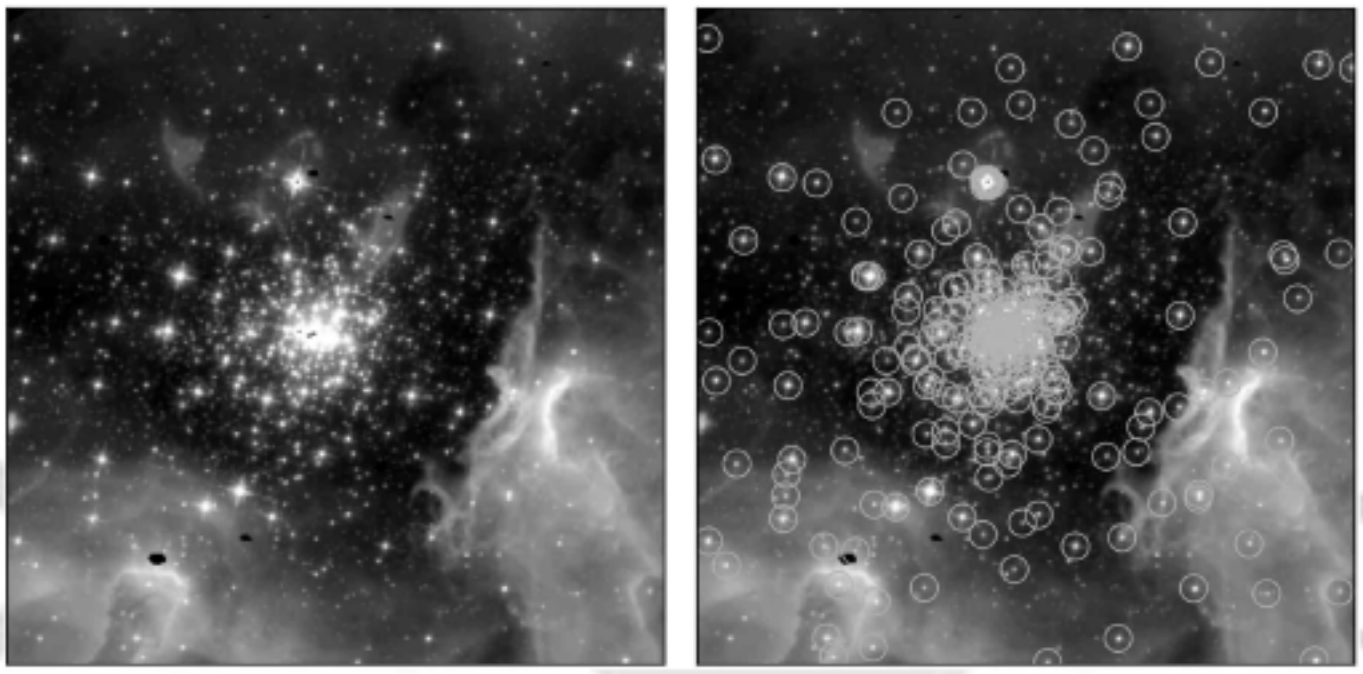


注意 :原文使用 skimage.morphology 中的函数 is_local_maximum 来计算局部最大值，该函数在 scikit-image 的 0.8 版本后放入 skimage.feature 模块中，改名为 peak_local_max 且功能有所变化。默认情况下，其返回值为局部最大值的坐标（参见方法 2），如果设置参数 indices=False 则返回值同 is_local_maximum 一样为布尔数组（参见方法 1），两种方法结果完全一样。

例：

```
import numpy as np
import pyfits
import matplotlib.pyplot as mpl
import skimage.exposure as skie
import skimage.feature as ft

# 载入 fits 格式的图片
img = pyfits.getdata('stellar_cluster.fits')[500:1500, 500:1500]
# 图片预处理
limg = np.arcsinh(img)
limg = limg / limg.max()
low = np.percentile(limg, 0.25)
high = np.percentile(limg, 99.5)
opt_img = skie.exposure.rescale_intensity(limg, in_range=(low, high))
# 过滤噪声并识别局部最大值
# lm = morph.is_local_maximum(limg)
lm = ft.peak_local_max(limg,indices=False)

x1, y1 = np.where(lm.T == True)
v = limg[(y1, x1)]
lim = 0.5        #除去像素值小于 0.5 的点
x2, y2 = x1[v > lim], y1[v > lim]
fig = mpl.figure(figsize=(8,4))
fig.subplots_adjust(hspace=0.05, wspace=0.05)
ax1 = fig.add_subplot(121)
ax1.imshow(opt_img)
ax1.set_xlim(0, img.shape[1])
ax1.set_ylim(0, img.shape[0])
```

```
ax1.xaxis.set_visible(False)
ax1.yaxis.set_visible(False)

ax2 = fig.add_subplot(122)
ax2.imshow(opt_img)
ax2.scatter(x2, y2, s=80, facecolor='none', edgecolor='#FF7400')
ax2.set_xlim(0, img.shape[1])
ax2.set_ylim(0, img.shape[0])
ax2.xaxis.set_visible(False)
ax2.yaxis.set_visible(False)
fig.savefig('scikit_image_f03.pdf', bbox_inches='tight')
mpl.show()
```



注意：运行上述程序前先安装包 PyFITS，用于读取 fits 格式的数据。
fits 格式的图片可在这里寻找下载 http://fits.gsfc.nasa.gov/fits_samples.html

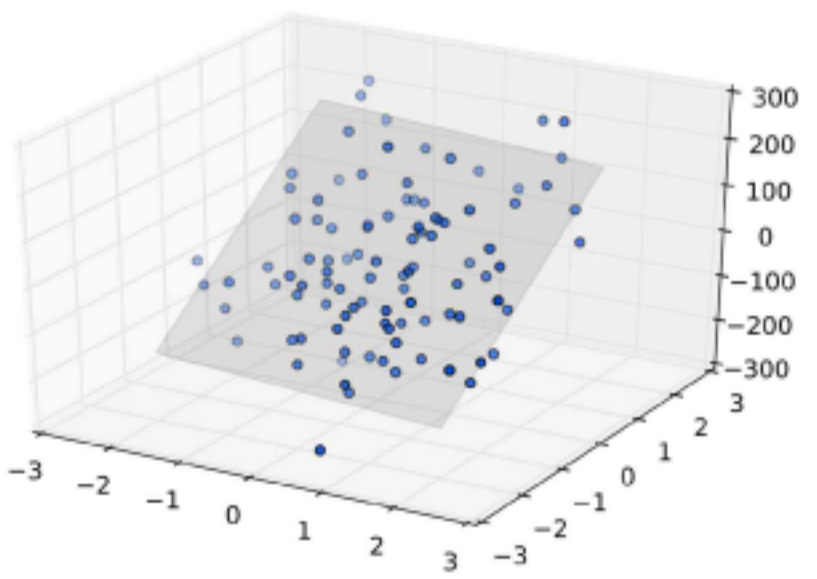
2. SciKit-Learn

（1）线性回归

例：三维数据最小两乘回归

```
import numpy as np
import matplotlib.pyplot as mpl
from mpl_toolkits.mplot3d import Axes3D
from sklearn import linear_model
from sklearn.datasets.samples_generator import make_regression
# 生成用于训练和测试的数据
X, y = make_regression(n_samples=100, n_features=2,
n_informative=1,random_state=0, noise=50)
# X 为二维数据,y 为值. 首先将其分布训练数据 X_train, y_train 和测试数据 X_test,
y_test
X_train, X_test = X[:80], X[-20:]
y_train, y_test = y[:80], y[-20:]

# 创建模型实例
regr = linear_model.LinearRegression()
```

```
# 训练模型
regr.fit(X_train, y_train)
# 系数
print(regr.coef_)
# [-10.25691752 90.5463984 ]

# 预测
X1 = np.array([1.2, 4])
print(regr.predict(X1))
# 350.860363861

# 测试，输出为预测的决定系数 R^2
print(regr.score(X_test, y_test))
# 0.949827492261

fig = mpl.figure(figsize=(8, 5))
ax = fig.add_subplot(111, projection='3d')
# ax = Axes3D(fig)

# 绘制数据点
ax.scatter(X_train[:,0], X_train[:,1], y_train, facecolor='#00CC00')
ax.scatter(X_test[:,0], X_test[:,1], y_test, facecolor='#FF7800')

# 绘制训练好的模型
coef = regr.coef_
line = lambda x1, x2: coef[0] * x1 + coef[1] * x2

grid_x1, grid_x2 = np.mgrid[-2:2:10j, -2:2:10j]
ax.plot_surface(grid_x1, grid_x2, line(grid_x1, grid_x2), alpha=0.1, color='k')
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
ax.zaxis.set_visible(False)
fig.savefig('scikit_learn_regression.pdf', bbox='tight')
mpl.show()
```

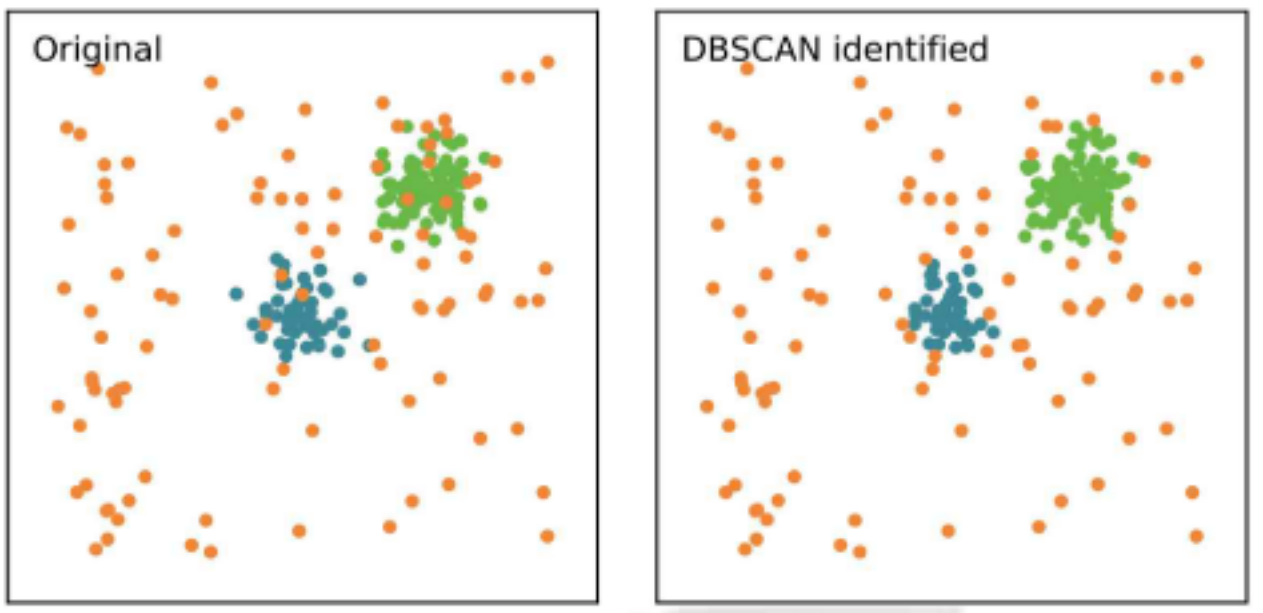


（2）聚类

例：DBSCAN 方法聚类

```
import numpy as np
import matplotlib.pyplot as mpl
from scipy.spatial import distance
from sklearn.cluster import DBSCAN
# 生成数据
c1 = np.random.randn(100, 2) + 5
c2 = np.random.randn(50, 2)
# 创建一个均匀分布的背景
u1 = np.random.uniform(low=-10, high=10, size=100)
u2 = np.random.uniform(low=-10, high=10, size=100)
c3 = np.column_stack([u1, u2])
# 将数据合并成 150 x 2 的数组
data = np.vstack([c1, c2, c3])
# 聚类,db.labels_ 是标识数据所属类别的数组
db = DBSCAN(eps=0.95, min_samples=10).fit(data)
labels = db.labels_

# 获取数据坐标.数据中有两个类别,分别标记为 0 和 1，标记为-1 的为噪声数据
dbc1 = data[labels == 0]
dbc2 = data[labels == 1]
noise = data[labels == -1]

x1, x2 = -12, 12
y1, y2 = -12, 12

fig = mpl.figure()
fig.subplots_adjust(hspace=0.1, wspace=0.1)

ax1 = fig.add_subplot(121, aspect='equal')
ax1.scatter(c1[:,0], c1[:,1], lw=0.5, color='#00CC00')
ax1.scatter(c2[:,0], c2[:,1], lw=0.5, color='#028E9B')
ax1.scatter(c3[:,0], c3[:,1], lw=0.5, color='#FF7800')
ax1.xaxis.set_visible(False)
ax1.yaxis.set_visible(False)
ax1.set_xlim(x1, x2)
ax1.set_ylim(y1, y2)
ax1.text(-11, 10, 'Original')

ax2 = fig.add_subplot(122, aspect='equal')
ax2.scatter(dbc1[:,0], dbc1[:,1], lw=0.5, color='#00CC00')
ax2.scatter(dbc2[:,0], dbc2[:,1], lw=0.5, color='#028E9B')
ax2.scatter(noise[:,0], noise[:,1], lw=0.5, color='#FF7800')
```

```
ax2.xaxis.set_visible(False)
ax2.yaxis.set_visible(False)
ax2.set_xlim(x1, x2)
ax2.set_ylim(y1, y2)
ax2.text(-11, 10, 'DBSCAN identified')

fig.savefig('scikit_learn_clusters.pdf', bbox_inches='tight')
mpl.show()
```



五、综结

略

==========OVER==========