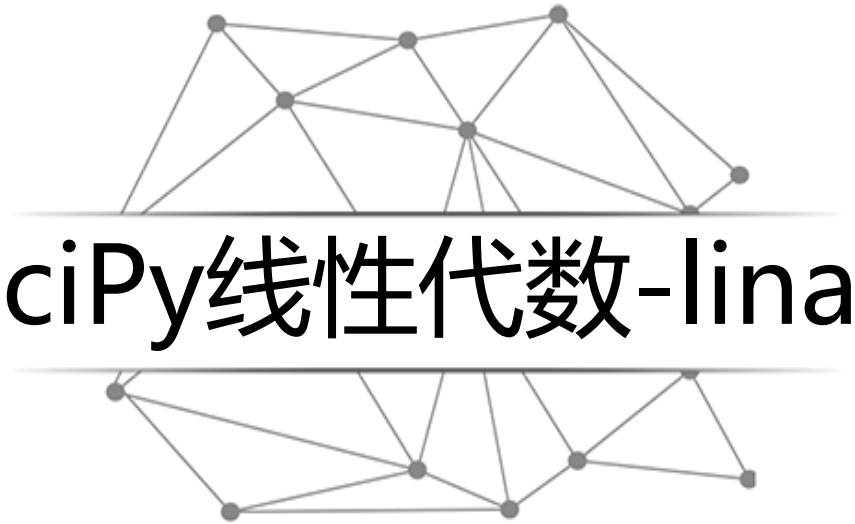
Scipy基础二

SV11

黄天羽

www.python123.org



SciPy线性代数-linalg

SciPy的linalg模块

Numpy和SciPy都提供了线性代数函数库linalg，SciPy更为全面：

- 解线性方程组
- 最小二乘解
- 特征值和特征向量
- 奇异值分解
- … …

解线性方程组

```
numpy.linalg.solve(A, b)
scipy.linalg.solve(A, b)
```

求解线性方程组$\mathrm{Ax} = b$ 即 $\mathrm{x} = \boldsymbol{A}^{-1}b$

A：m x n的方形矩阵

x：长为m的向量

b：长为m的向量

求解线性方程组$\mathrm{AX} = \mathrm{B}$ 即 $\mathrm{X} = \boldsymbol{A}^{-1}\mathrm{B}$

X：m x n的矩阵

B：m x n的矩阵

解线性方程组-举例

求解A^{-1}B：

```
import numpy as np
from scipy import linalg
import timeit

m, n = 50, 50
A = np.random.rand(m, n)
B = np.random.rand(m, n)
def my_func1():
    X1 = linalg.solve(A, B)
def my_func2():
    X2 = np.dot(linalg.inv(A), B)
t1 = timeit.Timer(stmt=my_func1).timeit(number=100)
t2 = timeit.Timer(stmt=my_func2).timeit(number=100)
print(t1, t2)
```

解线性方程组-举例

运行程序：

```
0.17003382103633652 0.02654141414682320 6
>>>
```

比较两种方法的运行时间！

特征值和特征向量

n x n的矩阵A可以看作n维空间中的线性变换。

- 如果x为n维空间中的一个向量，那么A与x的矩阵乘积是对x进行线性变换之后的向量。
- 如果x是线性变换的特征向量，那么经过这个线性变换后，得到新向量仍与原来的x保持在同乙方向上，长度可能发生改变。
- 特征向量的长度在该线性变换下缩放的比例称为其特征值。

$$Ax = \lambda x$$

特征值和特征向量-举例

以二维平面上的线性变换矩阵为例：

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1,-0.3],[-0.1,0.9]])
>>> evalues, evectors = linalg.eig(A)
>>> evalues  #特征值
array([ 1.13027756+0.j,  0.76972244+0.j])
>>> evectors #特征向量
array([[ 0.91724574,  0.79325185],
       [-0.3983218 ,  0.60889368]])
>>>
```

奇异值分解-SVD

奇异值分解是线性代数中一种重要的矩阵分解。

- 假设M是一个m x n阶矩阵，存在一个分解使得：$M = U\Sigma V^*$
- 其中U是m x m阶酉矩阵；Σ是半正定m x n阶矩阵；V^*是V的共轭转置，是n x n阶酉矩阵。
- Σ对角线上的元素为M的奇异值，并通常按照从大到小排列

$$M = U\Sigma V^*$$

奇异值分解-SVD

Linalg库中的svd函数：对矩阵进行奇异值分解

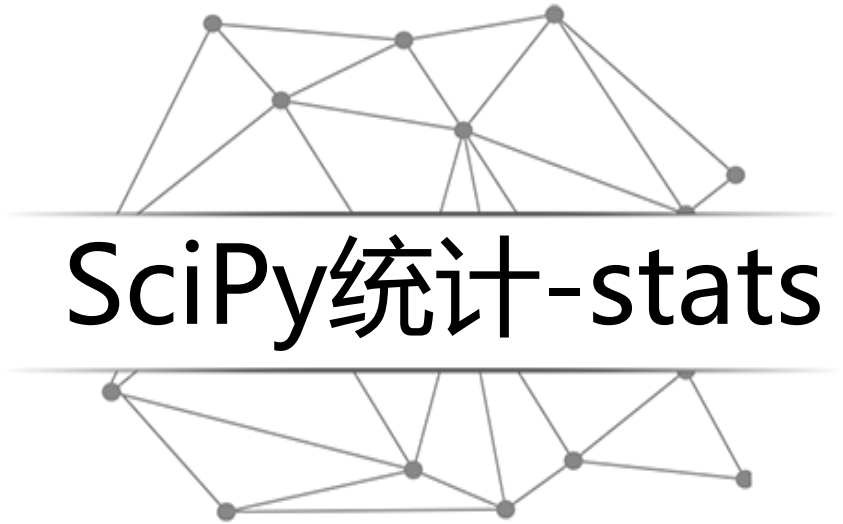
其调用形式为：

```
U,s,V=svd(M)
```

奇异值分解-举例

SVD分解为例：

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1,-0.3],[-0.1,0.9]])
>>> U,s,V = linalg.svd(A)
>>> U
array([[-0.81937847,  0.57325293],
       [ 0.57325293,  0.81937847]])
>>> s
array([ 1.16140394,  0.74909338])
>>> V
array([[-0.7548655 ,  0.65587962],
       [ 0.65587962,  0.7548655 ]])
>>>
```



SciPy统计-stats

SciPy的stats模块

Stats模块包含了多种概率分布的随机变量

连续随机变量是rv_continuous派生类的对象

离散随机变量是rv_discrete派生类的对象

- 连续概率分布
- 离散概率分布
- 核密度估计
- 二项分布、泊松分布、伽玛分布
- 学生t-分布与t检验
- 卡方分布和卡方检验

连续概率分布

获取stats模块中所有的连续随机变量：

```
>>> from scipy import stats
>>> from scipy import stats
>>> [k for k, v in stats.__dict__.items() if isinstance(v, stats.rv_continuous)]
['ksone', 'kstwobign', 'norm', 'alpha', 'anglit', 'arcsine', 'beta', 'betaprime',
'bradford', 'burr', 'burr12', 'fisk', 'cauchy', 'chi', 'chi2', 'cosine', 'dgamma',
'dweibull', 'expon', 'exponnorm', 'exponweib', 'exponpow', 'fatiguelife', 'foldcau
chy', 'f', 'foldnorm', 'frechet_r', 'weibull_min', 'frechet_l', 'weibull_max', 'ge
nlogistic', 'genpareto', 'genexpon', 'genextreme', 'gamma', 'erlang', 'gengamma',
'genhalflogistic', 'gompertz', 'gumbel_r', 'gumbel_l', 'halfcauchy', 'halflogistic
', 'halfnorm', 'hypsecant', 'gausshyper', 'invgamma', 'invgauss', 'invweibull', 'j
ohnsonsb', 'johnsonsu', 'laplace', 'levy', 'levy_l', 'levy_stable', 'logistic', 'l
oggamma', 'loglaplace', 'lognorm', 'gilbrat', 'maxwell', 'mielke', 'kappa4', 'kapp
a3', 'nakagami', 'ncx2', 'ncf', 't', 'nct', 'pareto', 'lomax', 'pearson3', 'powerl
aw', 'powerlognorm', 'powernorm', 'rdist', 'rayleigh', 'reciprocal', 'rice', 'reci
pinvgauss', 'semicircular', 'skewnorm', 'trapz', 'triang', 'truncexpon', 'truncnor
m', 'tukeylambda', 'uniform', 'vonmises', 'vonmises_line', 'wald', 'wrapcauchy', '
gennorm', 'halfgennorm', 'argus']
>>>
```

连续概率分布

连续随机变量对象有如下方法：

方法	说明
rvs	对随机变量进行随机取值，可以通过size参数指定输出的数组大小
pdf	随机变量的概率密度函数
cdf	随机变量的积累分布函数，它是概率密度函数的积分
sf	随机变量的生存函数，它的值是1-cdf(t)
ppf	累积分布函数的反函数
stat	计算随机变量的期望值和方差
fit	对一组随机取样进行拟合，找出最适合取样数据的概率密度函数的系数

连续概率分布

以正态分布为例，获取默认正态分布随机变量的期望值和方差：

```
>>> from scipy import stats
>>> stats.norm.stats()
(array(0.0), array(1.0))
>>>
```

连续概率分布

Norm可以像函数一样使用，通过loc和scale参数可以指定随机变量的偏移和缩放参数。

```
>>> X = stats.norm(loc=1.0, scale=2.0)
>>> X.stats()
(array(1.0), array(4.0))
>>>
```

连续概率分布

调用随机变量X的rvs()方法，得到包含一万次随机取样值的数组x：

```
>>> x = X.rvs(size = 10000)#对随机变量取1000个值
>>> import numpy as np
>>> np.mean(x)
0.9969997326048677
>>> np.var(x)
3.9504119033401905
>>>
```

使用 mean()、var()计算此数组的均值和方差，其结果符合随机变量X的特性。

离散概率分布

当分布函数的值域为离散时称之为离散概率分布。

例如：投掷有六个面的骰子时，获得1到6的整数，因此所得到的赶驴分布为离散的。

离散概率分布-举例

stats模块中离散分布随机变量都从rv_discrete类继承，也可以直接用rv_discrete类自定义离散概率分布。投掷骰子举例：

- 数组x保存骰子的所有可能值
- 数组p保存每个值出现的概率
- 创建表示这个骰子的随机变量dice

```
>>> from scipy import stats
>>> x = range(1,7)
>>> p = (1.0/6, 1.0/6, 1.0/6, 1.0/6, 1.0/6, 1.0/6)
>>> dice = stats.rv_discrete(values=(x, p))
>>>
```

离散概率分布-举例

投掷此骰子20次，获得符合概率p的随机数：

```
>>> from scipy import stats
>>> x = range(1,7)
>>> p = (1.0/6, 1.0/6, 1.0/6, 1.0/6, 1.0/6, 1.0/6)
>>> dice = stats.rv_discrete(values=(x, p))
>>> dice.rvs(size=20)
array([5, 6, 1, 1, 4, 5, 5, 2, 5, 4, 4, 2, 1, 6, 5, 3, 1, 5, 4, 1])
>>>
```

离散概率分布-举例

中心极限定律：大量相互独立的随机变量，其均值的分布以正态分布为极限。如何验证？

由于每一次投掷骰子可以看作一个独立的随机事件，投掷骰子50次的平均值可以看作“大量相互独立的随机变量”，其平均值的分布应该十分接近正态分布。

离散概率分布-举例

```
>>> import numpy as np
>>> samples = dice.rvs(size=(20000,50))
>>> samples_mean = np.mean(samples, axis=1)
>>> samples
array([[4, 4, 2, ..., 6, 5, 3],
       [4, 5, 2, ..., 6, 1, 3],
       [4, 2, 2, ..., 6, 3, 6],
       ...,
       [3, 4, 5, ..., 4, 2, 6],
       [5, 4, 3, ..., 1, 1, 2],
       [6, 3, 4, ..., 4, 5, 1]])
>>> samples_mean
array([ 3.3 ,  3.36,  3.34, ...,  3.6 ,  3.6 ,  3.8 ])
>>>
```

核密度估计-举例

- 前面例子中每个点是离散的，因此平均值也是离散的（直方图来显示）
- 更平滑的显示样本的概率，进行kde.gaussian_kde()进行核密度估计
- 核密度估计与拟合的正态分布十分相似

验证了中心极限定理！

核密度估计-举例

```
from scipy import stats
import numpy as np
import pylab as pl

x = range(1,7)
p = (1.0/6, 1.0/6, 1.0/6, 1.0/6, 1.0/6, 1.0/6)
dice = stats.rv_discrete(values=(x, p))
samples = dice.rvs(size=(20000,50))
samples_mean = np.mean(samples, axis=1)   #概率平均值
_,bins,step = pl.hist(
    samples_mean, bins=100, normed=True, histtype="step", label="Histogram")
kde = stats.kde.gaussian_kde(samples_mean)  #核密度估计
x = np.linspace(bins[0],bins[-1],100)
pl.plot(x, kde(x),label="kde")
mean, std = stats.norm.fit(samples_mean)    #拟合正态分布
pl.plot(x, stats.norm(mean,std).pdf(x), alpha=0.8, label="normal fitting")
pl.legend()
pl.show()
```

核密度估计-举例

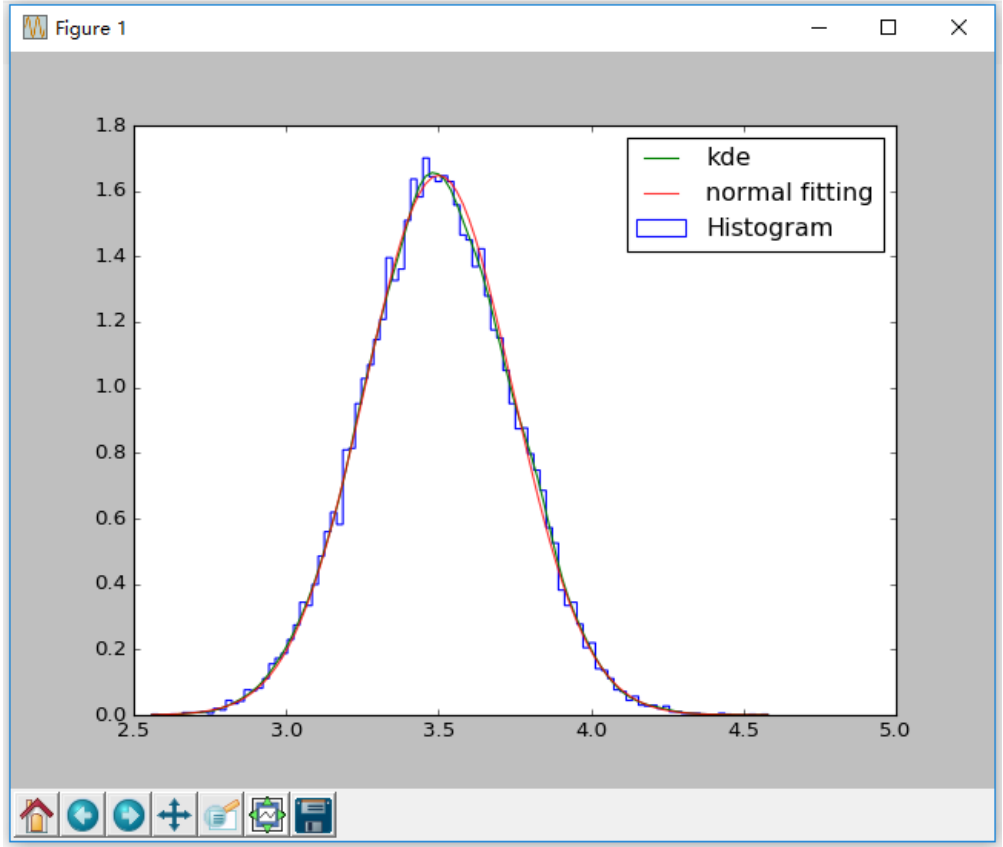
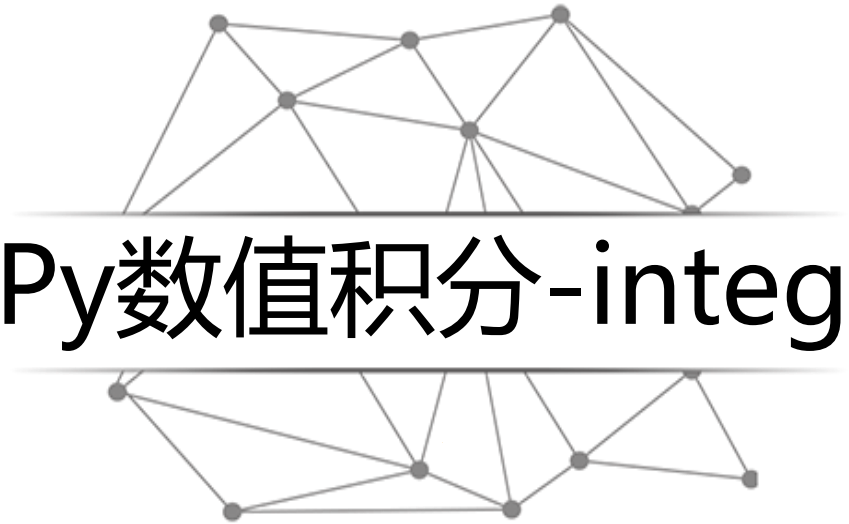


Figure 1
kde
normal fitting
Histogram
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
3.0
3.5
4.0
4.5
5.0



SciPy数值积分-integrate

SciPy的integrate模块

integrate模块提供了几种数值积分算法，包括对常微分方程组（ODE）的数值积分。

- 计算球体体积
- 解常微分方程

球的体积求解

数值积分是对定积分的数值求解，比如利用数值积分计算半圆的面积。

- 半径为1
- 单位半圆的曲线方程为$y = \sqrt{1 - x^2}$

```
def half_circle(x):
        return(1-x**2)**0.5
```

球的体积求解-面积求解

简单的方法：将半球分为许多小矩形，计算这些矩形之和。

```
>>> import numpy as np
>>> N = 10000
>>> x = np.linspace(-1, 1, N)
>>> dx = x[1]-x[0]
>>> y = half_circle(x)
>>> dx*np.sum(y)*2
3.141589326930737 3
>>>
```

球的体积求解-面积求解

使用integrate.quad()进行数值阶分：

```
>>> from scipy import integrate
>>> pi_half, err = integrate.quad(half_circle, -1, 1)
>>> pi_half*2
3.141592653589797
>>>
```

更为精确！

球的体积求解

多重定积分可以通过多次调用quad()实现。Integrage提供了

- 二重定积分 dblquad()
- 三重定积分 tplquad()

球的体积求解

二重积分dblquad函数调用形式为 ：

```
dblquad(func2d, a, b, gfun, hfun)
```

func2d：二重积分函数，假定x，y是func3d的两个参数

a，b：被积分函数的第一个变量x的积分区间

gfun，hfun：被积分函数第二个变量y的积分区间，通过变量x计算出变量y的积分区间

球的体积求解

单位半球面上的点(x,y,z)满足方程$x^2 + y^2 + z^2 = 1$

则z轴坐标值为：

```
def half_sphere(x,y):
    return(1-x**2-y**2)**0.5
```

球的体积求解

对于单位球,使用二重积分计算体积：

- 对于x轴从-1到1进行积分
- 对于y轴从-half_circle(x)到half_circle(x)进行积分

则半球体的二重积分公式为：

$$V = \int_{-1}^{1} \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \sqrt{1 - x^2 - y^2}\, dy\, dx$$

球的体积求解

```
from scipy import integrate

def half_circle(x):
    return(1-x**2)**0.5

def half_sphere(x,y):
    return(1-x**2-y**2)**0.5

volume, error = integrate.dblquad(half_sphere, -1, 1,
                                  lambda x: -half_circle(x),
                                  lambda x: half_circle(x))
print(volume, error)
```

球的体积求解

程序运行结果为：

```
2.094395102393199 1.0002356720661965e-09
>>>
```

通过球体积公式计算半球体积：

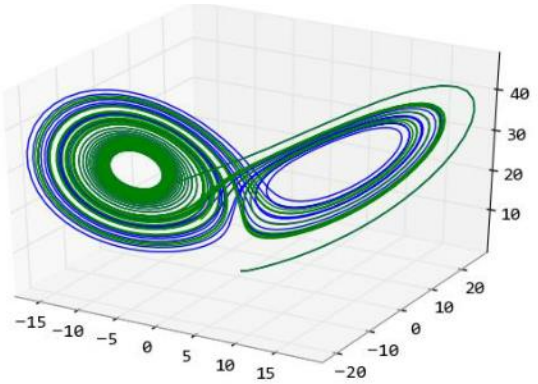
```
>>> np.pi*4/3/2
2.0943951023931953
>>>
```

解常微分方程

Integrage模块提供了对常微分方程组进行积分的函数：

odeint()

举例：如何计算洛伦茨吸引子的轨迹？



解常微分方程-洛伦茨吸引子轨迹

三个微分方程定义

$$\begin{cases} \frac{dx}{dt} = \sigma \cdot (y - x) \\ \frac{dy}{dt} = x \cdot (\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases}$$

洛伦茨吸引子的轨迹定义了三维空间中各个坐标点上的速度矢量，从某个坐标开始沿着速度矢量进行积分，计算出无质量点再此空间中的运动轨迹。

解常微分方程-洛伦茨吸引子轨迹

三个微分方程定义$\begin{cases} \frac{dx}{dt} = \sigma \cdot (y - x) \\ \frac{dy}{dt} = x \cdot (\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases}$

α、β、γ为三个常数，不同的参数可以计算出不同的运动轨迹：

x(t)、y(t)、z(t)。

解常微分方程-洛伦茨吸引子轨迹

定义函数lorenz（ ），计算出某个坐标点在各个方向上的微分值

```
import numpy as np

def lorenz(w, t, a, b, c):
    # 给出位置矢量w, 和三个参数a, b, c计算出
    # dx/dt, dy/dt, dz/dt的值
    x, y, z = w.tolist()
    # 直接与lorenz的计算公式对应
    return np.array([a * (y - x), x * (b - z) - y, x * y - c * z])
```

解常微分方程-洛伦茨吸引子轨迹

odeint（ ）参数：

lorenz：计算某个位置上各个方向的速度的函数

(0.0, 1.0, 0.0)：位置初始值，计算常微分方程所需的各个变量的初始值

t：表示时间的数组，odeint()对此数组中的每个时间点进行求解，得出所有时间点的位置

args：直接传递给lorenz，积分过程中为常量

解常微分方程-洛伦茨吸引子轨迹

调用odeint（ ），对微分方程求解。

```
t = np.arange(0, 30, 0.01)  # 创建时间点
# 调用ode对lorenz进行求解
track = odeint(lorenz, (0.0, 1.00, 0.0), t, args=(10.0, 28.0, 3.0))
```