python第三方库与统计分布

Python第三方库

- Numpy
- Scipy
- Pandas
- Matplotlib
- Statsmodels
- Sklearn

统计分析/数据可视化

- SciPy —基于 Python 的数学、科学、工程开源软件生态系统。
- NumPy—Python 科学计算基础包。
- Numba —Python 的低级虚拟机 JIT 编译器，Cython and NumPy 的开发者编写，供科学计算使用
- NetworkX —为复杂网络使用的高效软件。
- Pandas—这个库提供了高性能、易用的数据结构及数据分析工具。
- Open Mining—Python 中的商业智能工具（Pandas web 接口）。
- PyMC —MCMC 采样工具包。
- zipline—Python 的算法交易库。
- PyDy—全名 Python Dynamics，协助基于 NumPy， SciPy，IPython 以及 matplotlib 的动态建模工作流。
- SymPy —符号数学 Python 库。

统计分析/数据可视化

- statsmodels—Python 的统计建模及计量经济学库。
- astropy —Python 天文学程序库，社区协作编写
- matplotlib —Python 的 2D 绘图库。
- bokeh—Python 的交互式 Web 绘图库。
- plotly —Python and matplotlib 的协作 web 绘图库。
- vincent—将 Python 数据结构转换为 Vega 可视化语法。
- d3py—Python 的绘图库，基于 D3.js。
- ggplot —和R语言里的 ggplot2 提供同样的 API。
- Kartograph.py—Python 中渲染 SVG 图的库，效果漂亮。
- pygal—Python 下的 SVG 图表生成器。

机器学习

- MLlib in Apache Spark—Spark 下的分布式机器学习库。
- scikit-learn—基于 SciPy 的机器学习模块
- graphlab-create —包含多种机器学习模块的库（回归，聚类，推荐系统，图分析等），基于可以磁盘存储的 DataFrame。
- BigML—连接外部服务器的库。
- pattern—Python 的 web 挖掘模块
- NuPIC—Numenta 公司的智能计算平台。
- Pylearn2—基于 Theano 的机器学习库。
- hebel —Python 编写的使用 GPU 加速的深度学习库。
- gensim—主题建模工具。

机器学习

- PyBrain—另一个机器学习库。
- Crab —可扩展的、快速推荐引擎。
- python-recsys —Python 实现的推荐系统。
- Restricted Boltzmann Machines —Python 实现的受限波尔兹曼机。[深度学习]。
- Bolt —在线学习工具箱。
- CoverTree —cover tree 的 Python 实现，scipy.spatial.kdtree 便捷的替代。
- nilearn—Python 实现的神经影像学机器学习库。
- Shogun—机器学习工具箱。
- Pyevolve —遗传算法框架。
- Caffe —考虑了代码清洁、可读性及速度的深度学习框架
- breze—深度及递归神经网络的程序库，基于 Theano。

自然语言处理

- NLTK —一个领先的平台，用来编写处理人类语言数据的 Python 程序
- Pattern—Python 可用的 web 挖掘模块，包括自然语言处理、机器学习等工具。
- TextBlob—为普通自然语言处理任务提供一致的 API，以 NLTK 和 Pattern 为基础，并和两者都能很好兼容。
- jieba—中文断词工具。
- SnowNLP —中文文本处理库。
- loso—另一个中文断词库。
- genius —基于条件随机域的中文断词库。
- nut —自然语言理解工具包。

计算机视觉

- SimpleCV—开源的计算机视觉框架，可以访问如 OpenCV 等高性能计算机视觉库。使用 Python 编写，可以在 Mac、Windows 以及 Ubuntu 上运行。
- Opencv—高性能计算机视觉框架

深度学习

- sklearn-theano：早期的深度学习库
- Tensorflow—谷歌深度学习框架
- Pytorch—脸书深度学习框架
- Mxnet—亚马逊主导的深度学习框架
- Paddlepaddle—百度主导的深度学习框架
- 还有Caffe、Keras（并入tensorflow）

Numpy

- NumPy(Numerical Python) 是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。
- NumPy 的前身 Numeric 最早是由 Jim Hugunin 与其它协作者共同开发，2005 年，Travis Oliphant 在 Numeric 中结合了另一个同性质的程序库 Numarray 的特色，并加入了其它扩展而开发了 NumPy。NumPy 为开放源代码并且由许多协作者共同维护开发。
- NumPy 是一个运行速度非常快的数学库，主要用于数组计算，包含：
- 一个强大的N维数组对象 ndarray
- 广播功能函数
- 整合 C/C++/Fortran 代码的工具
- 线性代数、傅里叶变换、随机数生成等功能

Numpy

- NumPy 最重要的一个特点是其 N 维数组对象 ndarray，它是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。
- ndarray 对象是用于存放同类型元素的多维数组。
- ndarray 中的每个元素在内存中都有相同存储大小的区域。
- ndarray 内部由以下内容组成：
- 一个指向数据（内存或内存映射文件中的一块数据）的指针。
- 数据类型或 dtype，描述在数组中的固定大小值的格子。
- 一个表示数组形状（shape）的元组，表示各维度大小的元组。
- 一个跨度元组（stride），其中的整数指的是为了前进到当前维度下一个元素需要"跨过"的字节数。
- 跨度可以是负数，这样会使数组在内存中后向移动，切片中 **obj[::-1]** 或 **obj[:,::-1]** 就是如此。
- 创建一个 ndarray 只需调用 NumPy 的 array 函数即可：

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

Scipy

- SciPy 是一个开源的 Python 算法库和数学工具包。
- Scipy 是基于 Numpy 的科学计算库，用于数学、科学、工程学等领域，很多有一些高阶抽象和物理模型需要使用 Scipy。
- SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解和其他科学与工程中常用的计算。
- Scipy 是一个用于数学、科学、工程领域的常用软件包，可以处理最优化、线性代数、积分、插值、拟合、特殊函数、快速傅里叶变换、信号处理、图像处理、常微分方程求解器等。 。
- SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解和其他科学与工程中常用的计算。
- NumPy 和 SciPy 的协同工作可以高效解决很多问题，在天文学、生物学、气象学和气候科学，以及材料科学等多个学科得到了广泛应用。

SciPy 模块列表

SciPy 常量模块

SciPy 优化器

SciPy 稀疏矩阵

SciPy 图结构

SciPy 空间数据

SciPy Matlab 数组

SciPy 插值

Scipy 显著性检验

Scipy

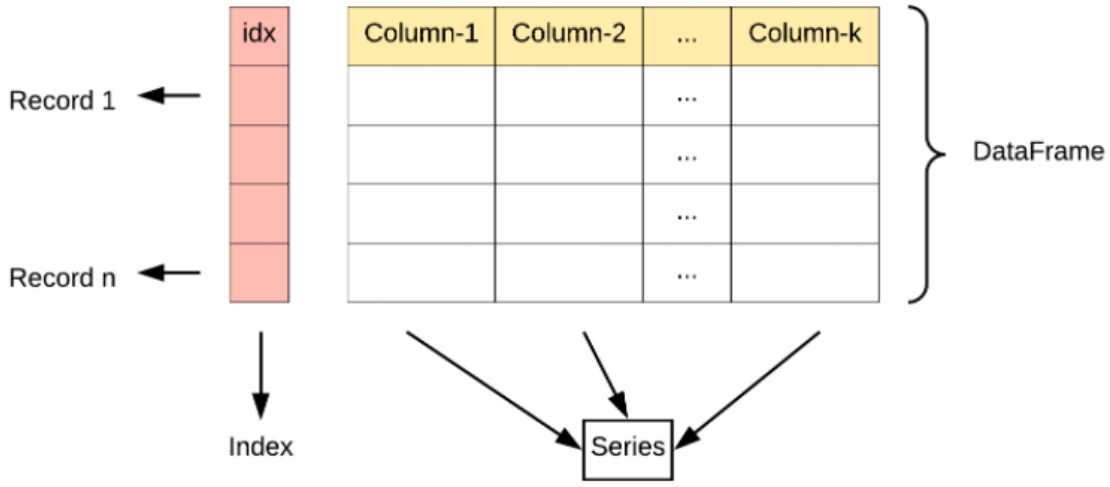
模块名	功能	参考文档
scipy.cluster	向量量化	cluster API
scipy.constants	数学常量	constants API
scipy.fft	快速傅里叶变换	fft API
scipy.integrate	积分	integrate API
scipy.interpolate	插值	interpolate API
scipy.io	数据输入输出	io API
scipy.linalg	线性代数	linalg API
scipy.misc	图像处理	misc API
scipy.ndimage	N 维图像	ndimage API
scipy.odr	正交距离回归	odr API
scipy.optimize	优化算法	optimize API
scipy.signal	信号处理	signal API
scipy.sparse	稀疏矩阵	sparse API
scipy.spatial	空间数据结构和算法	spatial API
scipy.special	特殊数学函数	special API
scipy/stats	统计函数	stats.mstats API

Pandas

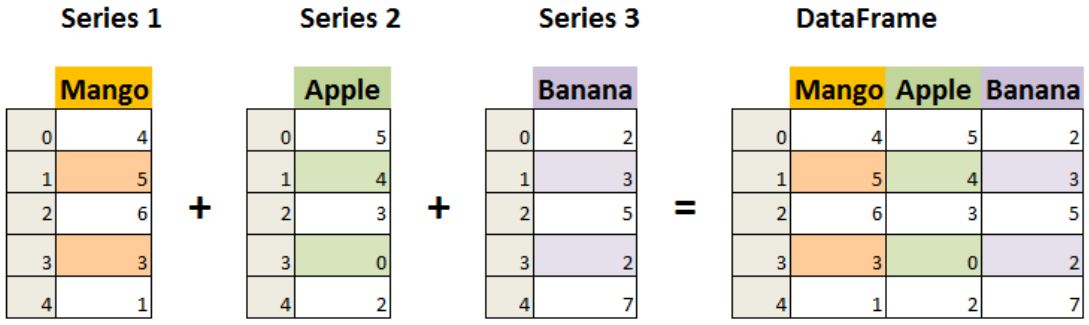
- **Pandas 是 Python 语言的一个扩展程序库，用于数据分析。**
- Pandas 是一个开放源码、BSD 许可的库，提供高性能、易于使用的数据结构和数据分析工具。
- Pandas 名字衍生自术语 "panel data"和 "Python data analysis"（Python 数据分析）。
- Pandas 一个强大的分析结构化数据的工具集，基础是 Numpy（提供高性能的矩阵运算）。
- Pandas 可以从各种文件格式比如 CSV、JSON、SQL、Microsoft Excel 导入数据。
- Pandas 可以对各种数据进行运算操作，比如归并、再成形、选择，还有数据清洗和数据加工特征。
- Pandas 广泛应用在学术、金融、统计学等各个数据分析领域。
- Pandas 的主要数据结构是 Series （一维数据）与 DataFrame（二维数据），这两种数据结构足以处理金融、统计、社会科学、工程等领域里的大多数典型用例。

Pandas

DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。DataFrame 既有行索引，它可以被看做由 Series 组成的字典（共同用一个索引）。



- DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。DataFrame 既有行索引也有列索引，它可以被看做由 Series 组成的字典（共同用一个索引）。
-



DataFrame 构造方法如下：

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

```
import pandas as pd

df = pd.read_csv('nba.csv')

print(df)
```

Matplotlib&Seaborn

- Matplotlib 是 Python 的绘图库，它能让使用者很轻松地将数据图形化，并且提供多样化的输出格式。
- Matplotlib 可以用来绘制各种静态，动态，交互式的图表。
- Matplotlib 是一个非常强大的 Python 画图工具，我们可以使用该工具将很多数据通过图表的形式更直观的呈现出来。
- Matplotlib 可以绘制线图、散点图、等高线图、条形图、柱状图、3D 图形、甚至是图形动画等等。
- Matplotlib 通常与 NumPy 和 SciPy（Scientific Python）一起使用， 这种组合广泛用于替代 MatLab，是一个强大的科学计算环境，有助于我们通过 Python 学习数据科学或者机器学习。
- Seaborn是一种开源的数据可视化工具，它在Matplotlib的基础上进行了更高级的API封装，因此可以进行更复杂的图形设计和输出。Seaborn是Matplotlib的重要补充，可以自主设置在Matplotlib中被默认的各种参数，而且它能高度兼容NumPy与Pandas数据结构以及Scipy与statsmodels等统计模式。

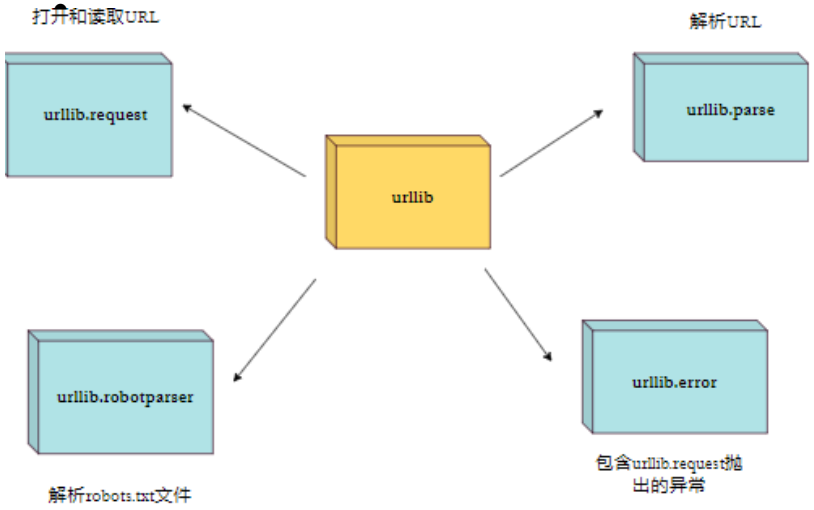
python爬虫

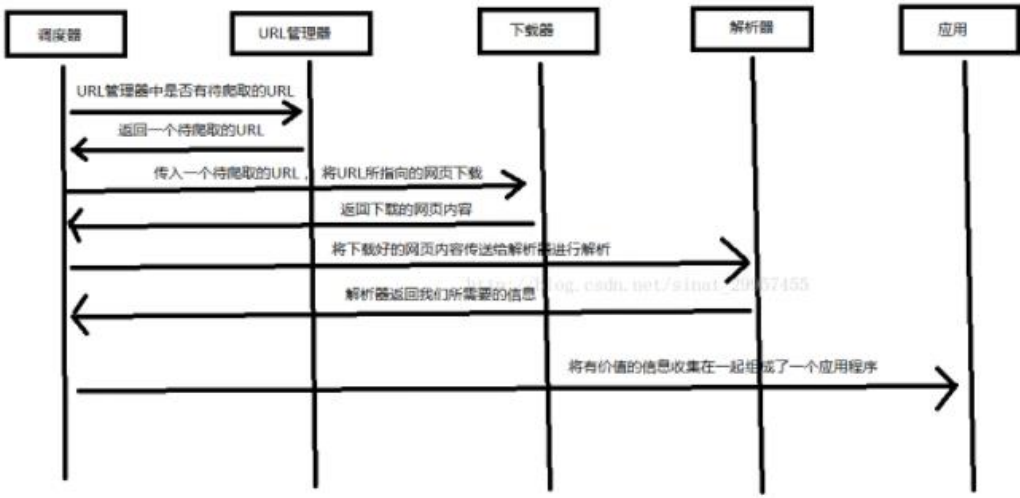
- **爬虫：**一段自动抓取互联网信息的程序，从互联网上抓取对于我们有价值的信息。
- Python 爬虫架构主要由五个部分组成，分别是调度器、URL管理器、网页下载器、网页解析器、应用程序（爬取的有价值数据）。
- **调度器：**相当于一台电脑的CPU，主要负责调度URL管理器、下载器、解析器之间的协调工作。
- **URL管理器：**包括待爬取的URL地址和已爬取的URL地址，防止重复抓取URL和循环抓取URL，实现URL管理器主要用三种方式，通过内存、数据库、缓存数据库来实现。
- **网页下载器：**通过传入一个URL地址来下载网页，将网页转换成一个字符串，网页下载器有urllib2（Python官方基础模块）包括需要登录、代理、和cookie，requests(第三方包)
- **网页解析器：**将一个网页字符串进行解析，可以按照我们的要求来提取出我们有用的信息，也可以根据DOM树的解析方式来解析。网页解析器有正则表达式（直观，将网页转成字符串通过模糊匹配的方式来提取有价值的信息，当文档比较复杂的时候，该方法提取数据的时候就会非常的困难）、html.parser（Python自带的）、beautifulsoup（第三方插件，可以使用Python自带的html.parser进行解析，也可以使用lxml进行解析，相对于其他几种来说要强大一些）、lxml（第三方插件，可以解析 xml 和 HTML），html.parser 和 beautifulsoup 以及 lxml 都是以 DOM 树的方式进行解析的。
- **应用程序：**就是从网页中提取的有用数据组成的一个应用。

爬虫工具：urllib、scrapy

Urllib

- Python urllib 库用于操作网页 URL，并对网页的内容进行抓取处理。
- **urllib.request** - 打开和读取 URL。
- **urllib.error** - 包含 urllib.request 抛出的异常。
- **urllib.parse** - 解析 URL。
- **urllib.robotparser** - 解析 robots.txt 文件。





```python
import urllib.request
import urllib.parse

url = 'https://www.runoob.com/try/py3/py3_urllib_test.php'  # 提交到表单页面
data = {'name':'RUNOOB', 'tag' : '菜鸟教程'}   # 提交数据
header = {
    'User-Agent':'Mozilla/5.0 (X11; Fedora; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.36'
}   #头部信息
data = urllib.parse.urlencode(data).encode('utf8')  # 对参数进行编码，解码使用 urllib.parse.urldecode
request=urllib.request.Request(url, data, header)   # 请求处理
reponse=urllib.request.urlopen(request).read()      # 读取结果

fh = open("./urllib_test_post_runoob.html","wb")    # 将文件写入到当前目录中
fh.write(reponse)
fh.close()
```

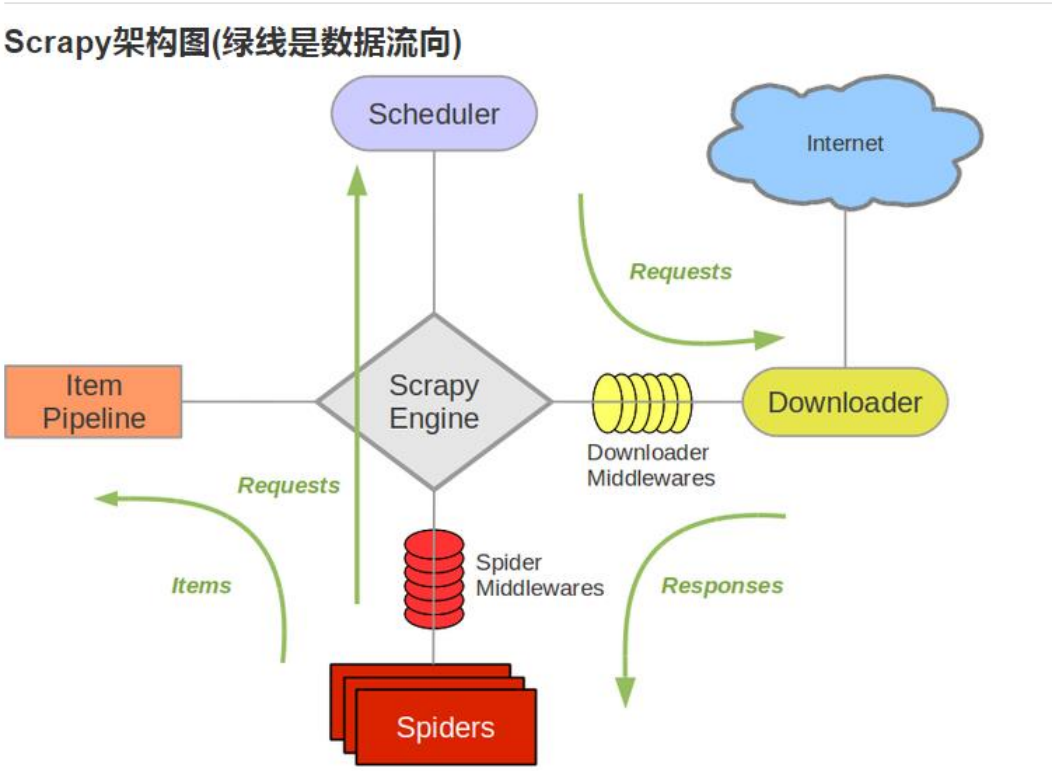
```python
import urllib.request
import urllib.error

myURL1 = urllib.request.urlopen("https://www.jd.com/")
print(myURL1.getcode())   # 200

try:
    myURL2 = urllib.request.urlopen("https://www.jd.com/no.html")
except urllib.error.HTTPError as e:
    if e.code == 404:
        print(404)   # 404
```

scrapy

- Scrapy 是用 Python 实现的一个为了爬取网站数据、提取结构性数据而编写的应用框架。
- Scrapy 常应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。
- 通常我们可以很简单的通过 Scrapy 框架实现一个爬虫，抓取指定网站的内容或图片。



scrapy

- **Scrapy Engine(引擎)**: 负责Spider、ItemPipeline、Downloader、Scheduler中间的通讯，信号、数据传递等。
- **Scheduler(调度器)**: 它负责接受引擎发送过来的Request请求，并按照一定的方式进行整理排列，入队，当引擎需要时，交还给引擎。
- **Downloader（下载器）**：负责下载Scrapy Engine(引擎)发送的所有Requests请求，并将其获取到的Responses交还给Scrapy Engine(引擎)，由引擎交给Spider来处理，
- **Spider（爬虫）**：它负责处理所有Responses,从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler(调度器).
- **Item Pipeline(管道)**：它负责处理Spider中获取到的Item，并进行进行后期处理（详细分析、过滤、存储等）的地方。
- **Downloader Middlewares（下载中间件）**：你可以当作是一个可以自定义扩展下载功能的组件。
- **Spider Middlewares（Spider中间件）**：你可以理解为是一个可以自定扩展和操作引擎和Spider中间通信的功能组件（比如进入Spider的Responses;和从Spider出去的Requests）

Statsmodels

- statsmodels为scipy提供了补充，以进行统计计算，包括描述性统计以及统计模型的估计和推断。
- statsmodels主要包括如下子模块：
- 回归模型：线性回归，广义线性模型，稳健的线性模型，线性混合效应模型等等。
- 方差分析（ANOVA）。
- 时间序列分析：AR，ARMA，ARIMA，VAR和其它模型。
- 非参数方法： 核密度估计，核回归。
- 统计模型结果可视化。

- 线性回归模型：
  - 普通最小二乘法
  - 广义最小二乘法
  - 加权最小二乘法
  - 具有自回归误差的最小二乘
  - 分位数回归
  - 递归最小二乘法

- 具有混合效应和方差成分的混合线性模型
- GLM：支持所有单参数指数族分布的广义线性模型
- 用于二项式和泊松的贝叶斯混合GLM
- GEE：单向聚类或纵向数据的广义估计方程
- 离散模型:
  - Logit 和 Probit
  - 多项 logit (MNLogit)
  - 泊松和广义泊松回归
  - 负二项式回归
  - 零膨胀计数模型

- RLM: 鲁棒的线性模型，支持多个 M 估计器。
- 时间序列分析：时间序列分析模型
  - 完整的StateSpace建模框架
    - 季节性ARIMA和ARIMAX模型
    - VARMA和VARMAX模型
    - 动态因子模型
    - 未观测到的组件模型

- 马尔可夫切换模型（MSAR），也称为隐马尔可夫模型（HMM）
- 单变量时间序列分析：AR，ARIMA
- 矢量自回归模型，VAR和结构VAR
- 矢量误差修正模型，VECM
- 指数平滑，Holt-Winters
- 时间序列的假设检验：单位根，协整和其他
- 用于时间序列分析的描述性统计数据和过程模型

- 生存分析:
  - 比例风险回归（Cox模型）
  - 生存者函数估计（Kaplan-Meier）
  - 累积发生率函数估计

- 多变量:
  - 缺失数据的主成分分析
  - 旋转因子分析
  - MANOVA
  - 典型相关

- 非参数统计：单变量和多变量核密度估计
- 数据集：用于示例和测试的数据集
- 统计：广泛的统计检验
  - 诊断和规格检验
  - 拟合优度和正态性检验
  - 多元测试函数
  - 各种其他统计检验

- 使用MICE进行插补，秩序统计回归和高斯插补
- 调解分析
- 图形包括用于数据和模型结果的可视化分析的绘图功能
- 输入/输出
  - 用于读取Stata .dta文件的工具，但pandas具有较新的版本
  - 表输出到ascii，latex和html

- 其他模型
- Sandbox：statsmodels包含一个 sandbox 文件夹，其中包含处于开发和测试各个阶段的代码，因此不被视为“生产就绪”。其中包括：
  - 广义矩法（GMM）估计器
  - 核回归
  - scipy.stats.distributions的各种扩展
  - 面板数据模型
  - 信息理论测度

主要过程

- Liner regression models：线性回归模型

Examples

```
# Load modules and data
import numpy as np
import statsmodels.api as sm
spector_data = sm.datasets.spector.load()
spector_data.exog = sm.add_constant(spector_data.exog, prepend=False)

# Fit and summarize OLS model
mod = sm.OLS(spector_data.endog, spector_data.exog)
res = mod.fit()
print res.summary()
```

主要过程

- General linear models：一般线型模型，主要用于各种设计的方差分析

Examples

```
# Load modules and data
import statsmodels.api as sm
data = sm.datasets.scotland.load()
data.exog = sm.add_constant(data.exog)

# Instantiate a gamma family model with the default link function.
gamma_model = sm.GLM(data.endog, data.exog, family=sm.families.Gamma())
gamma_results = gamma_model.fit()
```

主要过程

- robust linear models：稳健回归模型

Examples

```
# Load modules and data
import statsmodels.api as sm
data = sm.datasets.stackloss.load()
data.exog = sm.add_constant(data.exog)

# Fit model and print summary
rlm_model = sm.RLM(data.endog, data.exog, M=sm.robust.norms.HuberT())
rlm_results = rlm_model.fit()
print rlm_results.params
```

主要过程

- Discrete choice models：离散选择模型，logit模型属于离散选择模型

Examples

```
# Load the data from Spector and Mazzeo (1980)
spector_data = sm.datasets.spector.load()
spector_data.exog = sm.add_constant(spector_data.exog)

# Logit Model
logit_mod = sm.Logit(spector_data.endog, spector_data.exog)
logit_res = logit_mod.fit()
print logit_res.summary()
```

主要过程

- ANOVA：方差分析模型

Examples

```
In [1]: import statsmodels.api as sm

In [2]: from statsmodels.formula.api import ols

In [3]: moore = sm.datasets.get_rdataset("Moore", "car",
   ...:                                  cache=True) # load data
   ...:

In [4]: data = moore.data

In [5]: data = data.rename(columns={"partner.status" :
   ...:                            "partner_status"}) # make name pythonic
   ...:

In [6]: moore_lm = ols('conformity ~ C(fcategory, Sum)*C(partner_status, Sum)',
   ...:                data=data).fit()
   ...:

In [7]: table = sm.stats.anova_lm(moore_lm, typ=2) # Type 2 ANOVA DataFrame

In [8]: print table
                                    sum_sq  df          F    PR(>F)
C(fcategory, Sum)                11.614700   2   0.276958  0.759564
C(partner_status, Sum)          212.213778   1  10.120692  0.002874
C(fcategory, Sum):C(partner_status, Sum)  175.488928   2   4.184623  0.023572
Residual                        817.763961  39        NaN       NaN
```

主要过程

- Time series analysis：时间序列分析

Descriptive Statistics and Tests

stattools.acovf(x[, unbiased, demean, fft])	Autocovariance for 1D
stattools.acf(x[, unbiased, nlags, confint, ...])	Autocorrelation function for 1d arrays.
stattools.pacf(x[, nlags, method, alpha])	Partial autocorrelation estimated
stattools.pacf_yw(x[, nlags, method])	Partial autocorrelation estimated with non-recursive yule_walker
stattools.pacf_ols(x[, nlags])	Calculate partial autocorrelations
stattools.ccovf(x, y[, unbiased, demean])	crosscovariance for 1D
stattools.ccf(x, y[, unbiased])	cross-correlation function for 1d
stattools.periodogram(X)	Returns the periodogram for the natural frequency of X
stattools.adfuller(x[, maxlag, regression, ...])	Augmented Dickey-Fuller unit root test
stattools.q_stat(x, nobs[, type])	Return's Ljung-Box Q Statistic
stattools.grangercausalitytests(x, maxlag[, ...])	four tests for granger non causality of 2 timeseries
stattools.levinson_durbin(s[, nlags, isacov])	Levinson-Durbin recursion for autoregressive processes

主要过程

- Nonparametric estimators：非参估计

Module Reference

The public functions and classes are

smoothers_lowess.lowess(endog, exog[, frac, ...])	LOWESS (Locally Weighted Scatterplot Smoothing)
kde.KDEUnivariate(endog)	Univariate Kernel Density Estimator.
kernel_density.KDEMultivariate(data, var_type)	Multivariate kernel density estimator.
kernel_density.KDEMultivariateConditional(...)	Conditional multivariate kernel density estimator.
kernel_density.EstimatorSettings([...])	Object to specify settings for density estimation or regression.
kernel_regression.KernelReg(endog, exog, ...)	Nonparametric kernel regression class.
kernel_regression.KernelCensoredReg(endog, ...)	Nonparametric censored regression.

主要过程

- a wide range of statistical tests:各种统计检验

durbin_watson(resids)	Calculates the Durbin-Watson statistic
jarque_bera(resids)	Calculate residual skewness, kurtosis, and do the JB test for normality
omni_normtest(resids[, axis])	Omnibus test for normality
acorr_ljungbox(x[, lags, boxpierce])	Ljung-Box test for no autocorrelation
acorr_breush_godfrey(results[, nlags, store])	Breush Godfrey Lagrange Multiplier tests for residual autocorrelation
HetGoldfeldQuandt	test whether variance is the same in 2 subsamples
het_goldfeldquandt	see class docstring
het_breushpagan(resid, exog_het)	Breush-Pagan Lagrange Multiplier test for heteroscedasticity
het_white(resid, exog[, retres])	White's Lagrange Multiplier Test for Heteroscedasticity
het_arch(resid[, maxlag, autolag, store, ...])	Enlge's Test for Autoregressive Conditional Heteroscedasticity (ARCH)
linear_harvey_collier(res)	Harvey Collier test for linearity
linear_rainbow(res[, frac])	Rainbow test for linearity
linear_lm(resid, exog[, func])	Lagrange multiplier test for linearity against functional alternative
breaks_cusumolsresid(olsresidual[, ddof])	cusum test for parameter stability based on ols residuals
breaks_hansen(olsresults)	test for model stability, breaks in parameters for ols, Hansen 1992
recursive_olsresiduals(olsresults[, skip, ...])	calculate recursive ols with residuals and cusum test statistic
CompareCox	Cox Test for non-nested models
compare_cox	Cox Test for non-nested models
CompareJ	J-Test for comparing non-nested models
compare_j	J-Test for comparing non-nested models
unitroot_adf(x[, maxlag, trendorder, ...])	
normal_ad(x[, axis])	Anderson-Darling test for normal distribution unknown mean and variance
kstest_normal(x[, pvalmethod])	Lillifors test for normality,
lillifors(x[, pvalmethod])	Lillifors test for normality,

Baidu 经验
jingyan.baidu.com

主要过程

- 以各种方式输出表格：text，latex，html；读取各种格式的数据

Module Reference

foreign.StataReader(fname[, missing_values, ...])	Stata .dta file reader.
foreign.StataWriter(fname, data[, ...])	A class for writing Stata binary dta files from array-like objects
foreign.genfromdta(fname[, missing_flt, ...])	Returns an ndarray or DataFrame from a Stata .dta file.
foreign.savetxt(fname, X[, names, fmt, ...])	Save an array to a text file.
table.SimpleTable(data[, headers, stubs, ...])	Produce a simple ASCII, CSV, HTML, or LaTeX table from a
table.csv2st(csvfile[, headers, stubs, title])	Return SimpleTable instance,
smpickle.save_pickle(obj, fname)	Save the object to file via pickling.
smpickle.load_pickle(fname)	Load a previously saved object from file

主要过程

- 绘图功能

Goodness of Fit Plots

gofplots.qqplot(data[, dist, distargs, a, ...])	Q-Q plot of the quantiles of x versus the quantiles/ppf of a d
gofplots.qqline(ax, line[, x, y, dist, fmt])	Plot a reference line for a qqplot.
gofplots.qqplot_2samples(data1, data2[, ...])	Q-Q Plot of two samples' quantiles.
gofplots.ProbPlot(data[, dist, fit, ...])	Class for convenient construction of Q-Q, P-P, and probab

Boxplots

boxplots.violinplot(data[, ax, labels, ...])	Make a violin plot of each dataset in the *data* sequence.
boxplots.beanplot(data[, ax, labels, ...])	Make a bean plot of each dataset in the *data* sequence.

Correlation Plots

correlation.plot_corr(dcorr[, xnames, ...])	Plot correlation of many variables in a tight color grid.
correlation.plot_corr_grid(dcorrs[, titles, ...])	Create a grid of correlation plots.
plot_grids.scatter_ellipse(data[, level, ...])	Create a grid of scatter plots with confidence ellipses.

Functional Plots

Baidu 经验
jingyan.baidu.com

回归模型结果

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
Intercept	38.6517	9.456	4.087	0.000	19.826	57.478
Region[T.E]	-15.4278	9.727	-1.586	0.117	-34.793	3.938
Region[T.N]	-10.0170	9.260	-1.082	0.283	-28.453	8.419
Region[T.S]	-4.5483	7.279	-0.625	0.534	-19.039	9.943
Region[T.W]	-10.0913	7.196	-1.402	0.165	-24.418	4.235
Literacy	-0.1858	0.210	-0.886	0.378	-0.603	0.232
Wealth	0.4515	0.103	4.390	0.000	0.247	0.656

Omnibus:	3.049	Durbin-Watson:	1.785
Prob(Omnibus):	0.218	Jarque-Bera (JB):	2.694
Skew:	-0.340	Prob(JB):	0.260
Kurtosis:	2.454	Cond. No.	371.

随机变量的分类

离散型随机变量

非离散型随机变量 — 其中一种重要的类型为

连续性随机变量

统计计算　统计分布

exp：指数，
gamma：伽玛，
lnorm：对数正态，
cauchy：柯西，
geom：几何分布，
nbinom：负二项，
signrank：符号秩，
tukey：学生化极差

weibull：威布尔，
beta：贝塔
logis：逻辑分布，
binom：二项分布，
hyper：超几何，
pois：泊松
wilcox：秩和，

例4： $Y \sim B(20,0.2)$

求服从二项分布的随机变量Y分布率的值

binom(k，n，p)密度函数/质量函数
（Probability mass function）

```
>>> from scipy.stats import binom
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Calculate the first four moments:

```
>>> n, p = 5, 0.4
>>> mean, var, skew, kurt = binom.stats(n, p, moments='mvsk')
```

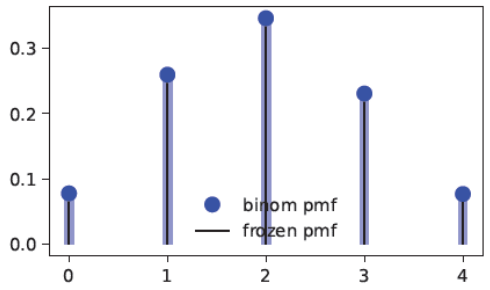
Display the probability mass function (pmf):

```
>>> x = np.arange(binom.ppf(0.01, n, p),
...               binom.ppf(0.99, n, p))
>>> ax.plot(x, binom.pmf(x, n, p), 'bo', ms=8, label='binom pmf')
>>> ax.vlines(x, 0, binom.pmf(x, n, p), colors='b', lw=5, alpha=0.5)
```

Alternatively, the distribution object can be called (as a function) to fix the shape and location. This returns a "frozen" RV object holding the given parameters fixed.

Freeze the distribution and display the frozen pmf:

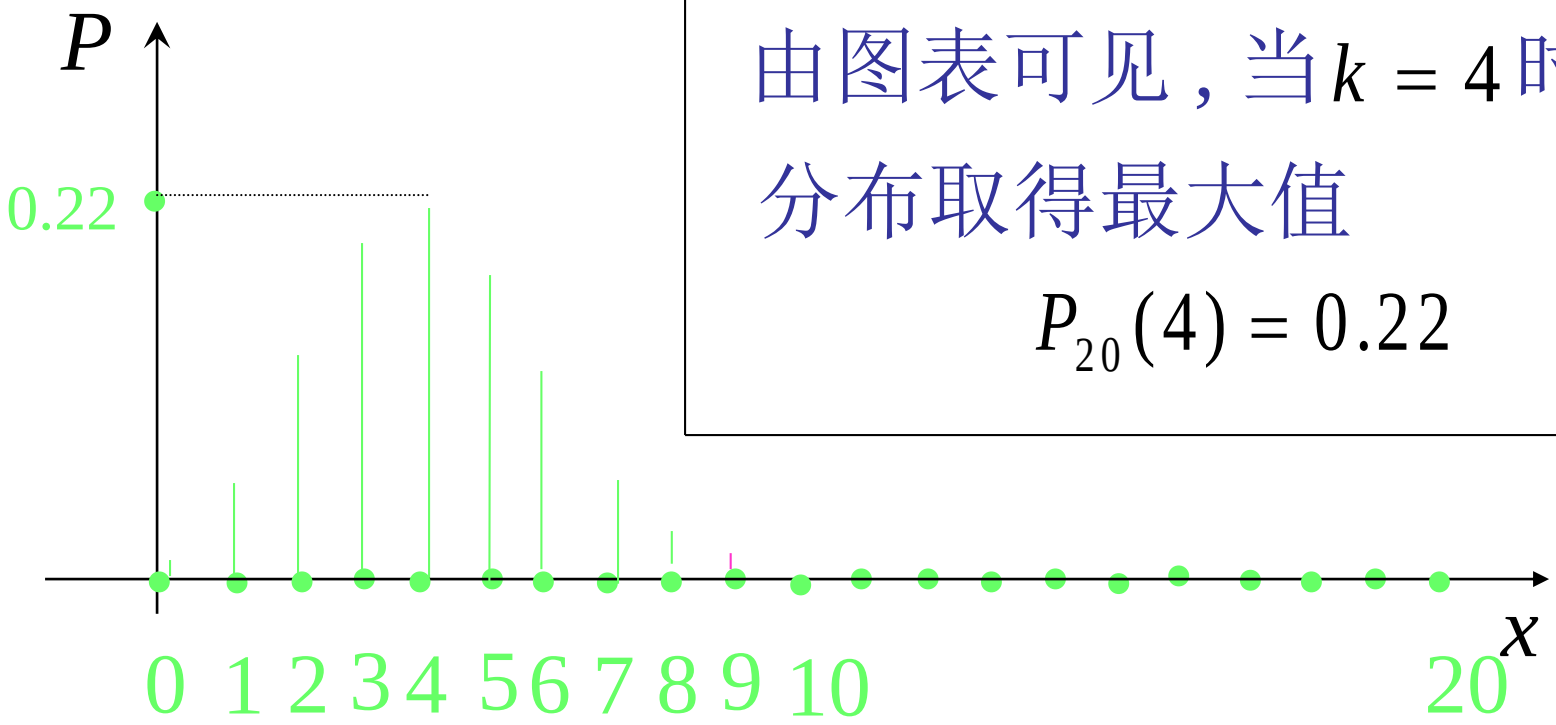
```
>>> rv = binom(n, p)
>>> ax.vlines(x, 0, rv.pmf(x), colors='k', linestyles='-', lw=1,
...         label='frozen pmf')
>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```



rvs(n, p, loc=0, size=1, random_state=None)	Random variates.
pmf(k, n, p, loc=0)	Probability mass function.
logpmf(k, n, p, loc=0)	Log of the probability mass function.
cdf(k, n, p, loc=0)	Cumulative distribution function.
logcdf(k, n, p, loc=0)	Log of the cumulative distribution function.
sf(k, n, p, loc=0)	Survival function (also defined as `1 - cdf`, but *sf* is sometimes more accurate).
logsf(k, n, p, loc=0)	Log of the survival function.
ppf(q, n, p, loc=0)	Percent point function (inverse of `cdf` — percentiles).
isf(q, n, p, loc=0)	Inverse survival function (inverse of `sf`).
stats(n, p, loc=0, moments='mv')	Mean('m'), variance('v'), skew('s'), and/or kurtosis('k').
entropy(n, p, loc=0)	(Differential) entropy of the RV.
expect(func, args=(n, p), loc=0, lb=None, ub=None, conditional=False)	Expected value of a function (of one argument) with respect to the distribution.
median(n, p, loc=0)	Median of the distribution.
mean(n, p, loc=0)	Mean of the distribution.
var(n, p, loc=0)	Variance of the distribution.
std(n, p, loc=0)	Standard deviation of the distribution.
interval(alpha, n, p, loc=0)	Endpoints of the range that contains fraction alpha [0, 1] of the distribution

设$X \sim B(20,0.2)$

0	1	2	3	4	5	6	7	8	9	10	11 ~ 20
.01	.06	.14	.21	.22	.18	.11	.06	.02	.01	.002	< .001



由图表可见，当$k = 4$时，分布取得最大值

$$P_{20}(4) = 0.22$$

例5： 设 $X \sim B(8, 1/3)$

求服从二项分布的随机变量X分布率的值

```
binom(0,8,1/3)
x=0:8;
```

结果：

```
ans =    0.0390
ans =    0.1561
y =    0.0390    0.1561    0.2731    0.2731    0.1707    0.0683
       0.0171    0.0024    0.0002
```

(3) **Poisson 分布** $\pi(\lambda)$ 或 $P(\lambda)$

若 $P(X=k)=e^{-\lambda}\dfrac{\lambda^k}{k!}$，$k=0,1,2,\cdots$

其中 $\lambda>0$ 是常数，则称 X 服从参数为 λ 的Poisson 分布，记作 $\pi(\lambda)$ 或 $P(\lambda)$

应用场合

在一定时间间隔内：

电话总机接到的电话次数；

一匹布上的疵点个数；

大卖场的顾客数；

市级医院急诊病人数；

一个容器中的细菌数；

某一地区发生的交通事故的次数

放射性物质发出的粒子数；

一本书中每页印刷错误的个数；

等等

都可以看作是源源不断出现的随机质点流，若它们满足一定的条件，则称为Poisson流，在长为 t 的时间内出现的质点数 $X_t \sim P(\lambda t)$

```
>>> from scipy.stats import poisson
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Calculate the first four moments:

```
>>> mu = 0.6
>>> mean, var, skew, kurt = poisson.stats(mu, moments='mvsk')
```

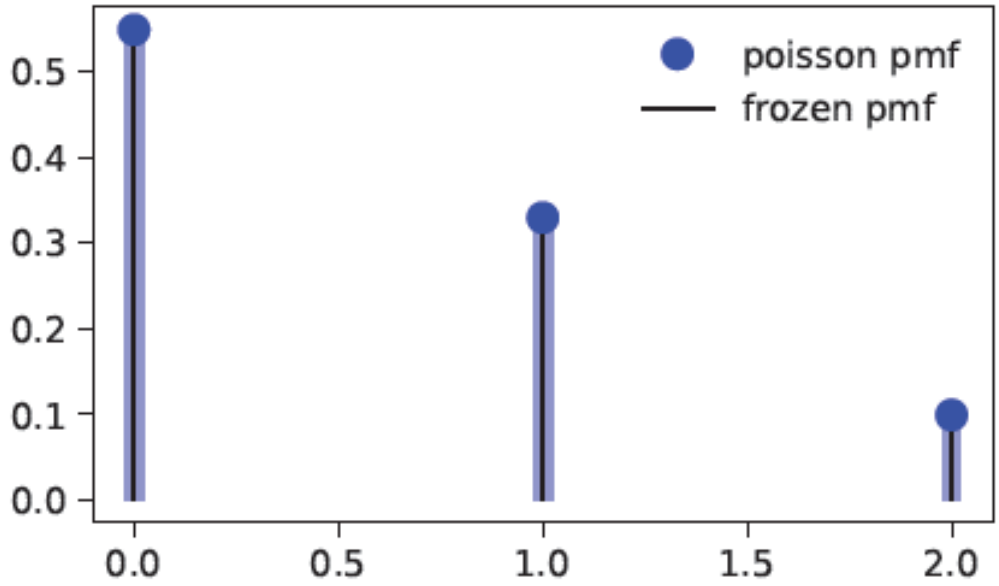
Display the probability mass function (`pmf`):

```
>>> x = np.arange(poisson.ppf(0.01, mu),
...               poisson.ppf(0.99, mu))
>>> ax.plot(x, poisson.pmf(x, mu), 'bo', ms=8, label='poisson pmf')
>>> ax.vlines(x, 0, poisson.pmf(x, mu), colors='b', lw=5, alpha=0.5)
```

Alternatively, the distribution object can be called (as a function) to fix the shape and location. This returns a "frozen" RV object holding the given parameters fixed.

Freeze the distribution and display the frozen `pmf`:

```
>>> rv = poisson(mu)
>>> ax.vlines(x, 0, rv.pmf(x), colors='k', linestyles='-', lw=1,
...           label='frozen pmf')
>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```




```
>>> prob = poisson.cdf(x, mu)
>>> np.allclose(x, poisson.ppf(prob, mu))
True
```

Generate random numbers:

```
>>> r = poisson.rvs(mu, size=1000)
```

例7三个人共同负责90台设备发生故障不能及时维修的概率为

$$P(X>3)\approx\sum_{k=4}^{90}e^{-0.9}\frac{0.9^k}{k!}=1-\sum_{k=0}^{3}e^{-0.9}\frac{0.9^k}{k!}$$

$$=\sum_{k=4}^{\infty}e^{-0.9}\frac{0.9^k}{k!}-\sum_{k=91}^{\infty}e^{-0.9}\frac{0.9^k}{k!}$$

$$\approx\sum_{k=4}^{\infty}e^{-0.9}\frac{0.9^k}{k!}=0.013459$$

poisson. pmf(0,0.9)+poisson. pmf (1,0.9)+ poisson. pmf (2,0.9)+poisson. pmf(3,0.9) = 0.9865

$$1-\sum_{k=0}^{3}e^{-0.9}\frac{0.9^k}{k!}=0.0135$$

离散型随机变量的分布函数

$$F(x)=P(X\le x)=P\Big(\bigcup_{x_k\le x}(X=x_k)\Big)$$

$$=\sum_{x_k\le x}P(X=x_k)=\sum_{x_k\le x}p_k$$

$$p_k=P(X=x_k)=F(x_k)-F(x_{k-1})$$

$F(x)$ 是分段阶梯函数，在 X 的可能取值 x_k 处发生间断，间断点为第一类跳跃间断点，在间断点处有跃度 p_k

例8： $Y \sim B(20,0.2)$

求服从二项分布的随机变量X分布函数的值

结果：

```
ans =    0.9994
y =    0.0115    0.0692    0.2061    0.4114    0.6296    0.8042    0.9133
0.9679    0.9900    0.9974    0.9994    0.9999    1.0000  1.0000
1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

§2.3 连续型随机变量

连续型随机变量的概念

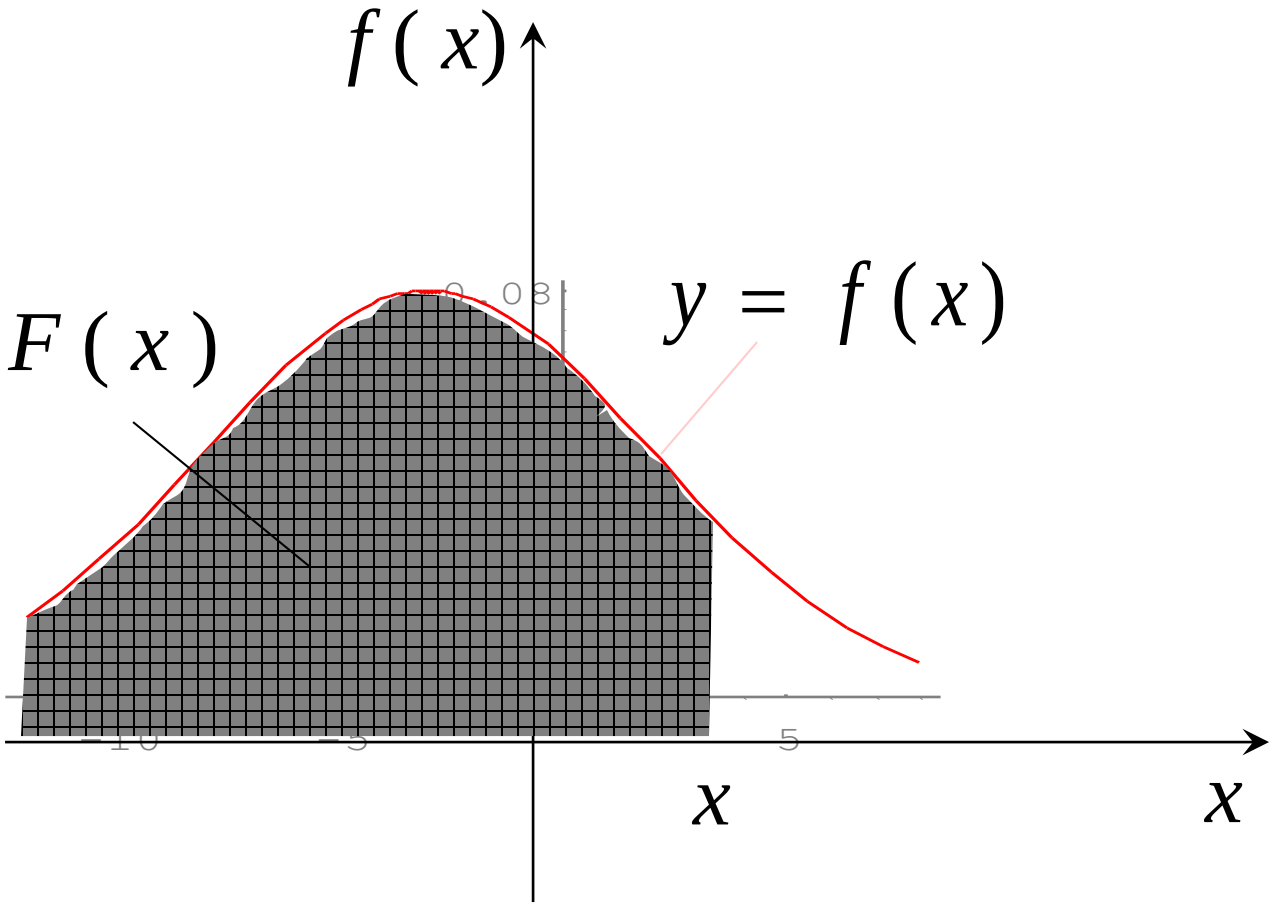
定义 设 X 是一随机变量，若存在一个非负可积函数 $f(x)$，使得

$$F(x)=\int_{-\infty}^{x} f(t)\mathrm{d}t \qquad -\infty < x < +\infty$$

其中$F(x)$是它的分布函数

则称 X 是连续型随机变量，$f(x)$是它的概率密度函数(**p.d.f.**)，简称为密度函数或概率密度

分布函数$F(x)$与密度函数$f(x)$的几何意义



p.d.f. $f(x)$的性质

- $f(x) \geq 0$
- $\int_{-\infty}^{+\infty} f(x)\mathrm{d}x = F(+\infty) = 1$

常利用这两个性质检验一个函数能否作为连续性随机变量的密度函数，或求其中的未知参数

- 在 $f(x)$ 的连续点处，

$$f(x) = F'(x)$$

$f(x)$ 描述了 X 在 x 附近单位长度的区间内取值的概率

注意: 对于连续型随机变量X, $P(X=a)=0$

这里 a 可以是随机变量 X 的一个可能的取值

$$0 \le P(X=a) = F(a) - F(a-0)$$

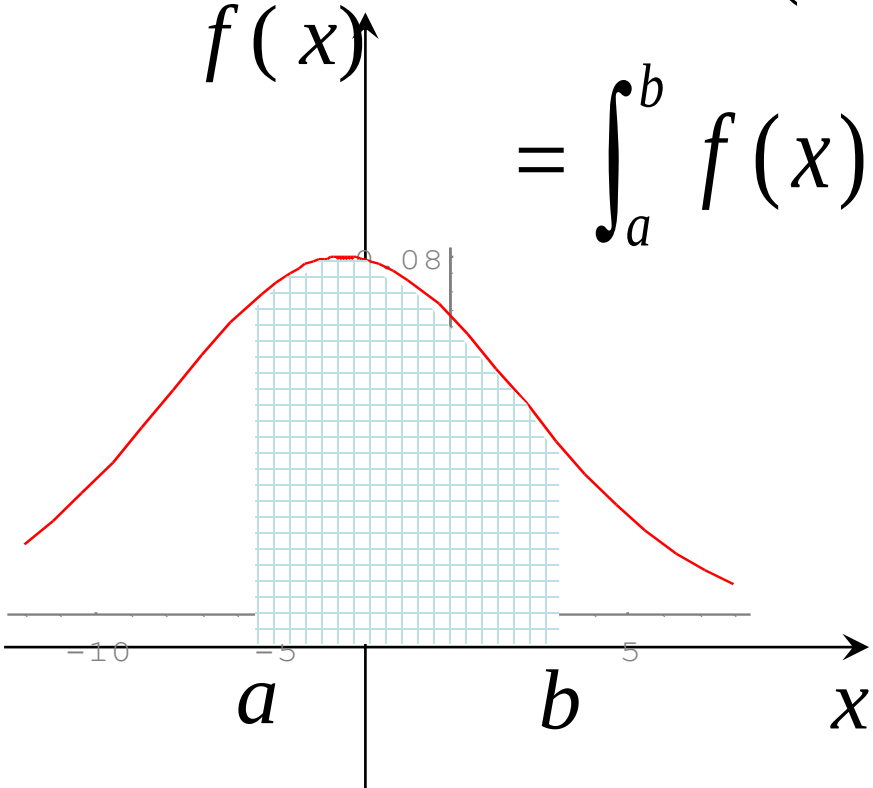
命题 连续型随机变量取任一常数的概率为零

对于连续型随机变量X

$$P(a < X \le b) = P(a \le X \le b)$$

$$= P(a < X < b)$$

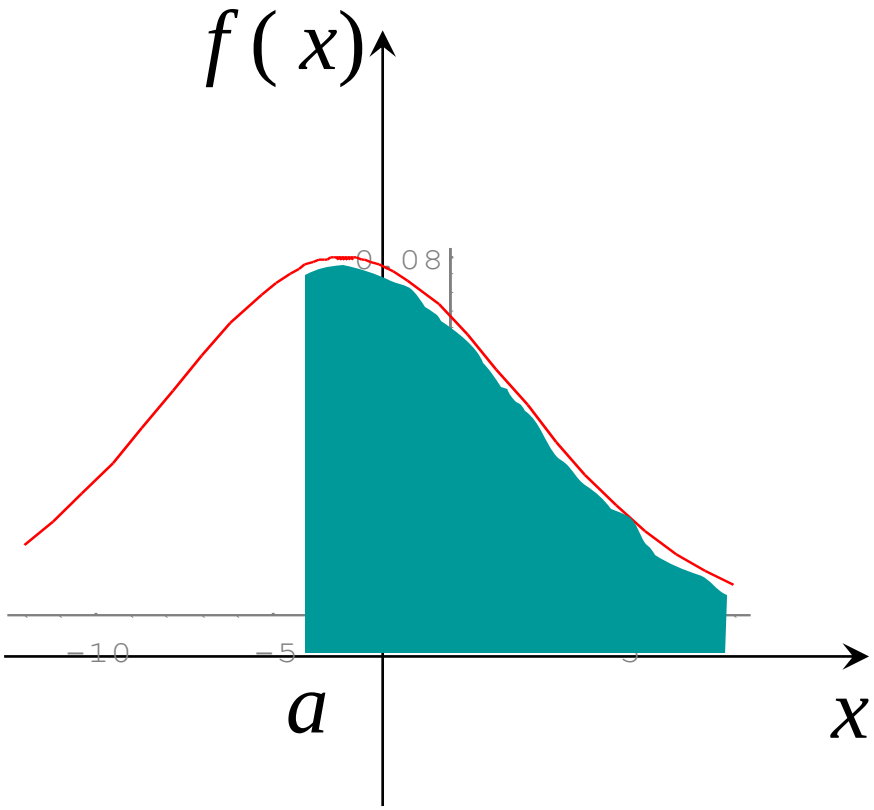
$$= P(a \le X < b)$$



$$= \int_a^b f(x)\mathrm{d}x = F(b) - F(a)$$

$$P(X \le b) = P(X < b) = F(b)$$

$$P(X > a) = P(X \ge a) = 1 - F(a)$$



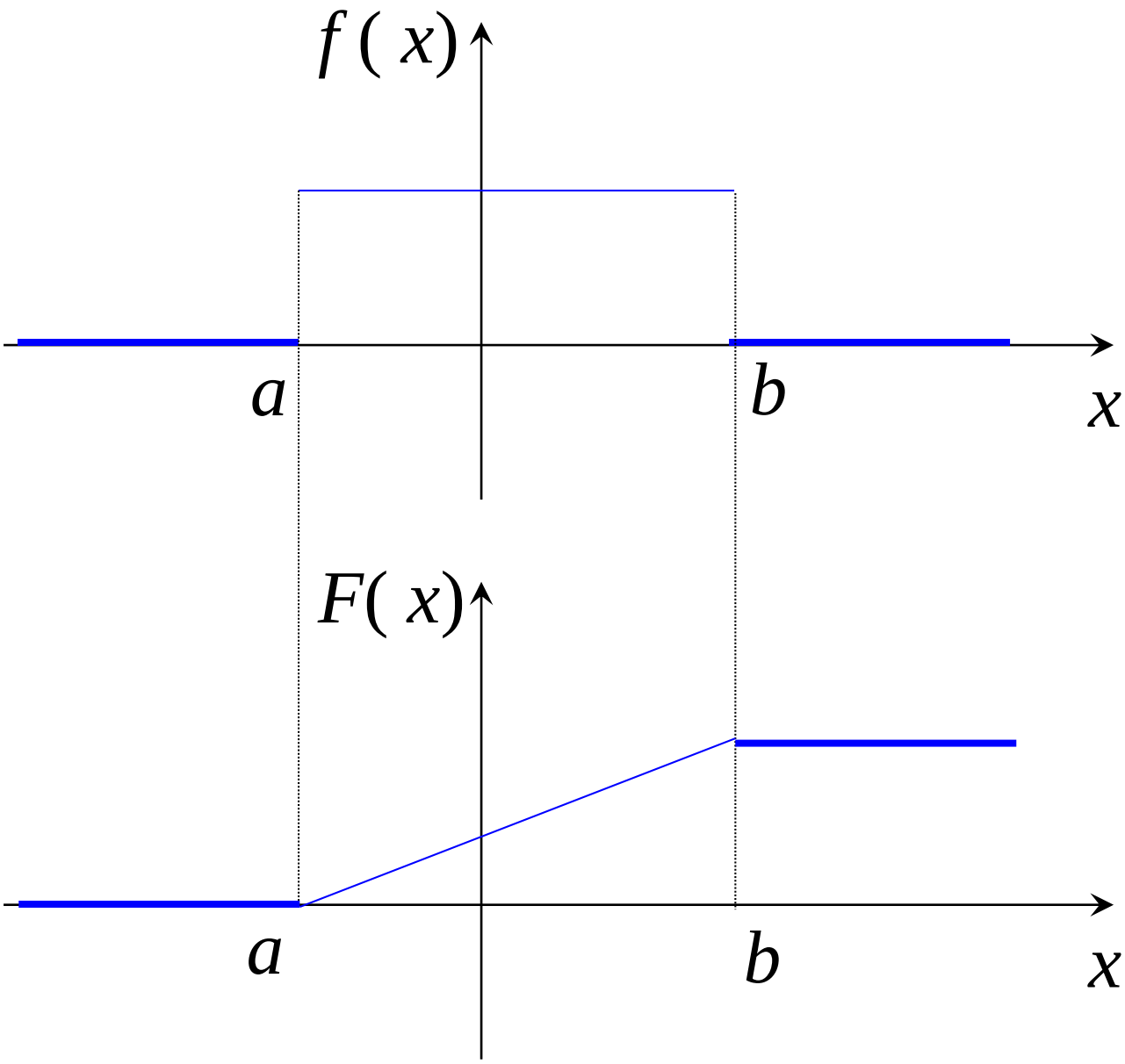
常见的连续性随机变量的分布

(1) 均匀分布

若 X 的密度函数为 $f(x)$，则称 X 服从区间 (a,b)上的均匀分布 记作 $X \sim U(a,b)$

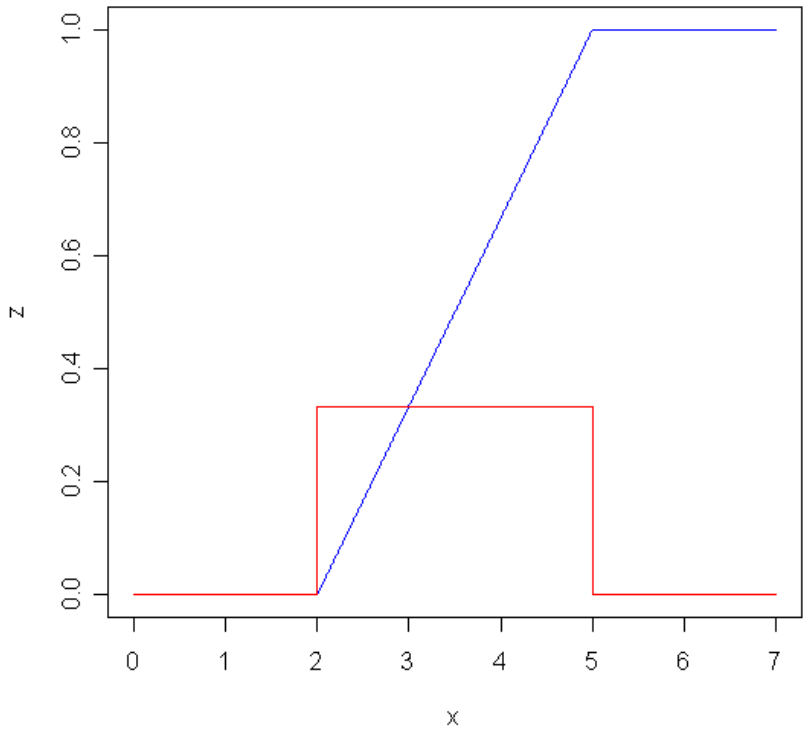
其中 $$f(x)=\begin{cases}\dfrac{1}{b-a}, & a<x<b\\ 0, & \text{其他}\end{cases}$$

X 的分布函数为 $$F(x)=\begin{cases}0, & x<a,\\ \dfrac{x-a}{b-a}, & a\le x<b,\\ 1 & x\ge b\end{cases}$$


f (x)
a
b
x
F(x)
a
b
x

$$X \sim U[a,b]$$

例 10: 画出均匀分布 U(2,5)的概率密度函数和分布函数的图形.



```
>>> from scipy.stats import uniform
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Calculate the first four moments:

```
>>> mean, var, skew, kurt = uniform.stats(moments='mvsk')
```

Display the probability density function (`pdf`):

```
>>> x = np.linspace(uniform.ppf(0.01),
...                 uniform.ppf(0.99), 100)
>>> ax.plot(x, uniform.pdf(x),
...         'r-', lw=5, alpha=0.6, label='uniform pdf')
```

Alternatively, the distribution object can be called (as a function) to fix the shape, location and scale parameters. This returns a "frozen" RV object holding the given parameters fixed.

Freeze the distribution and display the frozen `pdf`:

```
>>> rv = uniform()
>>> ax.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')
```

Check accuracy of `cdf` and `ppf`:

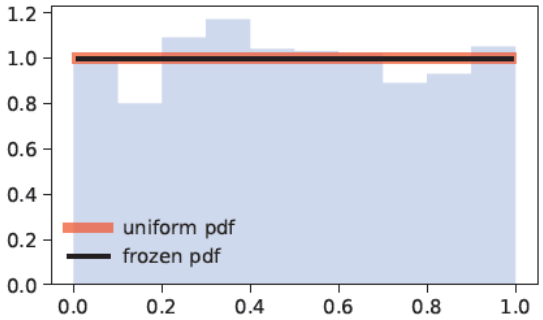
```
>>> vals = uniform.ppf([0.001, 0.5, 0.999])
>>> np.allclose([0.001, 0.5, 0.999], uniform.cdf(vals))
True
```

Generate random numbers:

```
>>> r = uniform.rvs(size=1000)
```

And compare the histogram:

```
>>> ax.hist(r, density=True, histtype='stepfilled', alpha=0.2)
>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```



Methods

rvs(loc=0, scale=1, size=1, random_state=None)	Random variates.
pdf(x, loc=0, scale=1)	Probability density function.
logpdf(x, loc=0, scale=1)	Log of the probability density function.
cdf(x, loc=0, scale=1)	Cumulative distribution function.
logcdf(x, loc=0, scale=1)	Log of the cumulative distribution function.
sf(x, loc=0, scale=1)	Survival function (also defined as `1 - cdf`, but *sf* is sometimes more accurate).
logsf(x, loc=0, scale=1)	Log of the survival function.
ppf(q, loc=0, scale=1)	Percent point function (inverse of `cdf` — percentiles).
isf(q, loc=0, scale=1)	Inverse survival function (inverse of `sf`).
moment(n, loc=0, scale=1)	Non-central moment of order n
stats(loc=0, scale=1, moments='mv')	Mean('m'), variance('v'), skew('s'), and/or kurtosis('k').
entropy(loc=0, scale=1)	(Differential) entropy of the RV.
fit(data)	Parameter estimates for generic data. See scipy.stats.rv_continuous.fit for detailed documentation of the keyword arguments.
expect(func, args=(), loc=0, scale=1, lb=None, ub=None, conditional=False, **kwds)	Expected value of a function (of one argument) with respect to the distribution.
median(loc=0, scale=1)	Median of the distribution.
mean(loc=0, scale=1)	Mean of the distribution.
var(loc=0, scale=1)	Variance of the distribution.
std(loc=0, scale=1)	Standard deviation of the distribution.
interval(alpha, loc=0, scale=1)	Endpoints of the range that contains fraction alpha [0, 1] of the distribution

(2) 指数分布

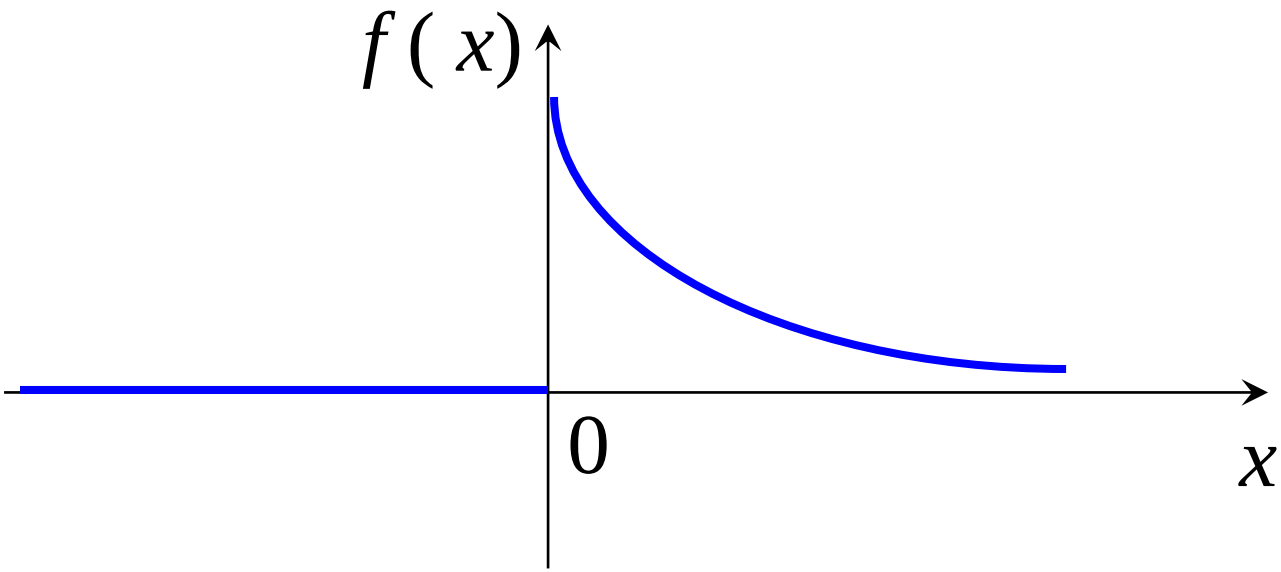
若 X 的密度函数为

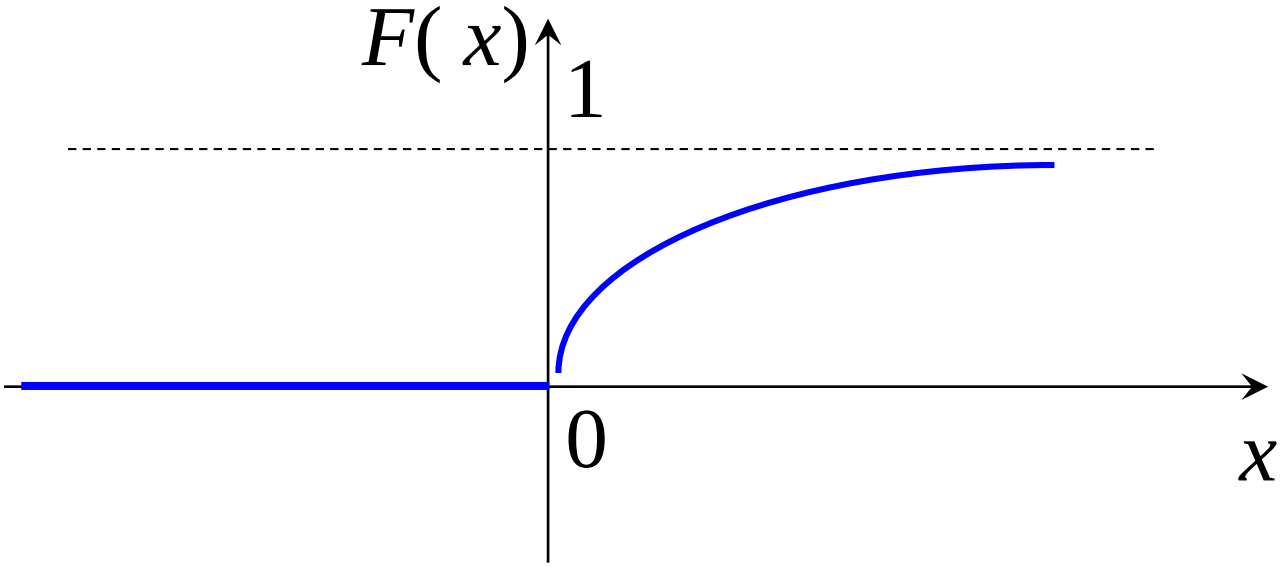
$$f(x)=\begin{cases}\lambda e^{-\lambda x}, & x>0\\ 0, & \text{其他}\end{cases}\qquad \lambda>0 \text{ 为常数}$$

则称 X 服从 参数为λ的指数分布

记作 $X\sim E(\lambda)$

X 的分布函数为 $F(x)=\begin{cases}0, & x<0\\ 1-e^{-\lambda x}, & x\geq 0\end{cases}$


f (x)
0
x


F(x)
1
0
x

```
>>> from scipy.stats import expon
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Calculate the first four moments:

```
>>> mean, var, skew, kurt = expon.stats(moments='mvsk')
```

Display the probability density function (`pdf`):

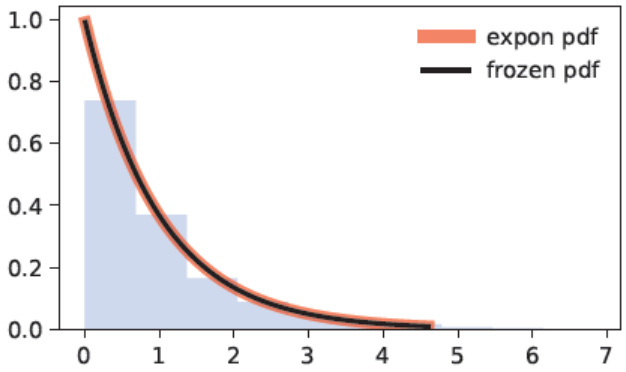
```
>>> x = np.linspace(expon.ppf(0.01),
...                 expon.ppf(0.99), 100)
>>> ax.plot(x, expon.pdf(x),
...         'r-', lw=5, alpha=0.6, label='expon pdf')
```

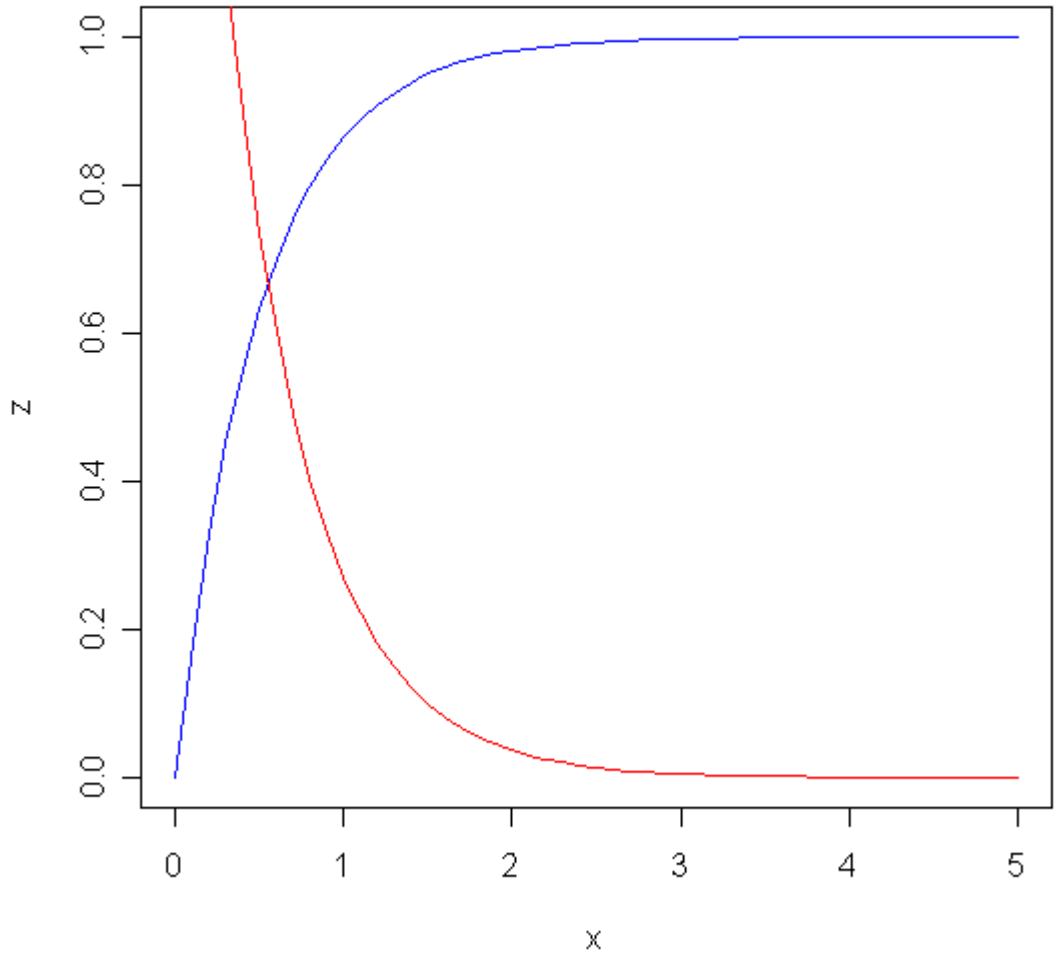
Alternatively, the distribution object can be called (as a function) to fix the shape, location and scale parameters. This returns a "frozen" RV object holding the given parameters fixed.

Freeze the distribution and display the frozen `pdf`:

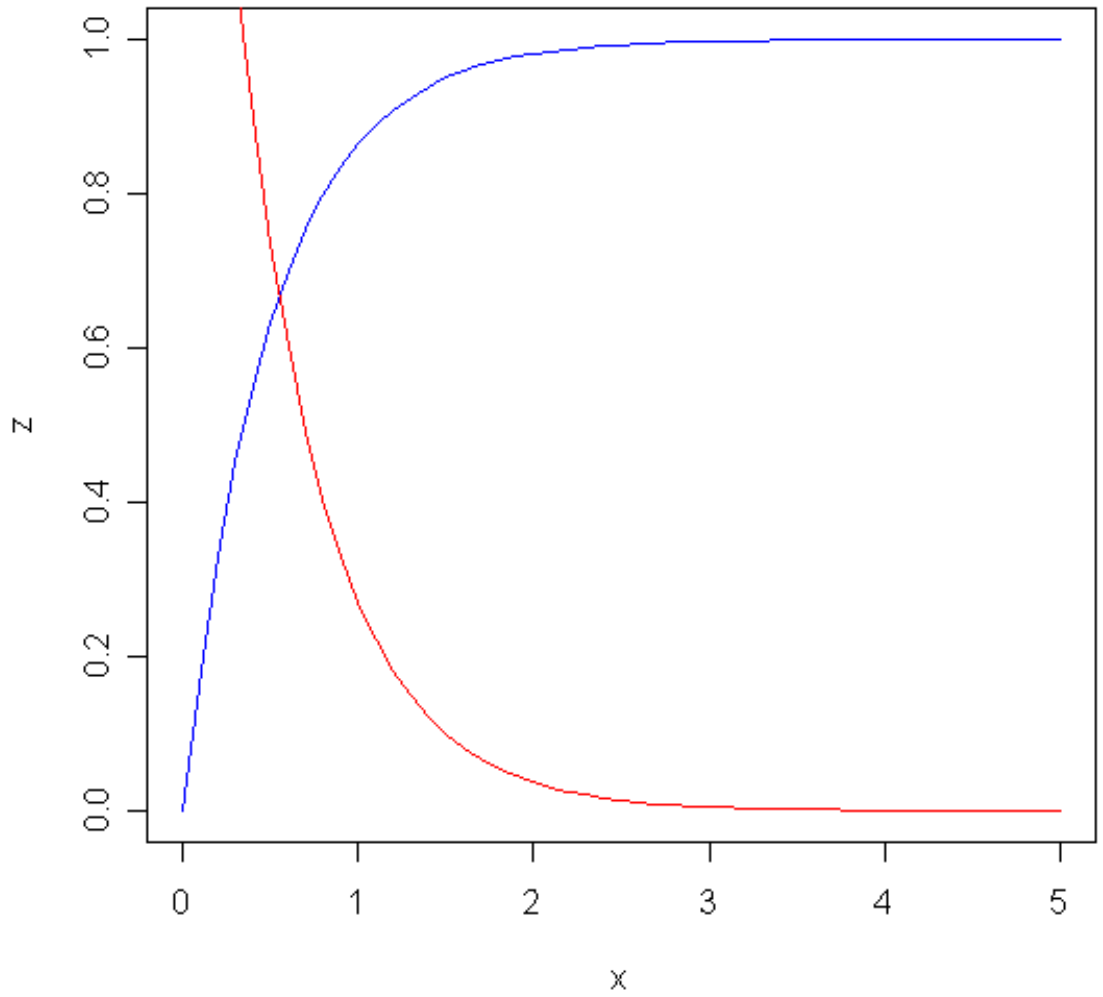
```
>>> rv = expon()
>>> ax.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')
```

```
>>> ax.hist(r, density=True, histtype='stepfilled', alpha=0.2)
>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```





结果： **Result = 0.1353291**



结果： **result1 = 0.993262**

result2 = 1.0000

(3) 正态分布

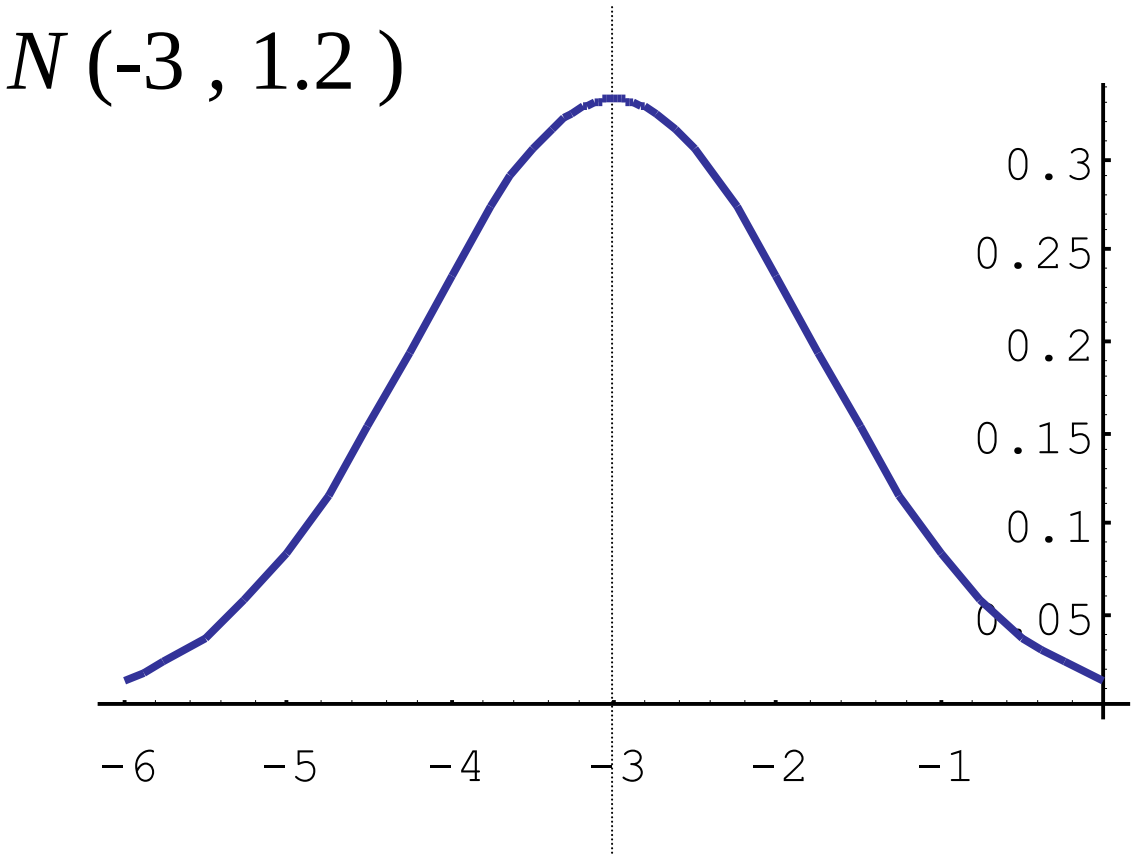
若X 的密度函数为

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad -\infty<x<+\infty$$

μ,σ 为常数, $\sigma>0$

则称 X 服从参数为 μ , σ^2 的正态分布

记作 $X\sim N(\mu,\sigma^2)$



$\mu = -3$

$f(x)$ 的性质：

- 图形关于直线 $x = \mu$ 对称： $f(\mu + x) = f(\mu - x)$

  在 $x = \mu$ 时, $f(x)$ 取得最大值

$$\frac{1}{\sqrt{2\pi}\sigma}$$

  在 $x = \mu \pm \sigma$ 时, 曲线 $y = f(x)$ 在对应的点处有拐点

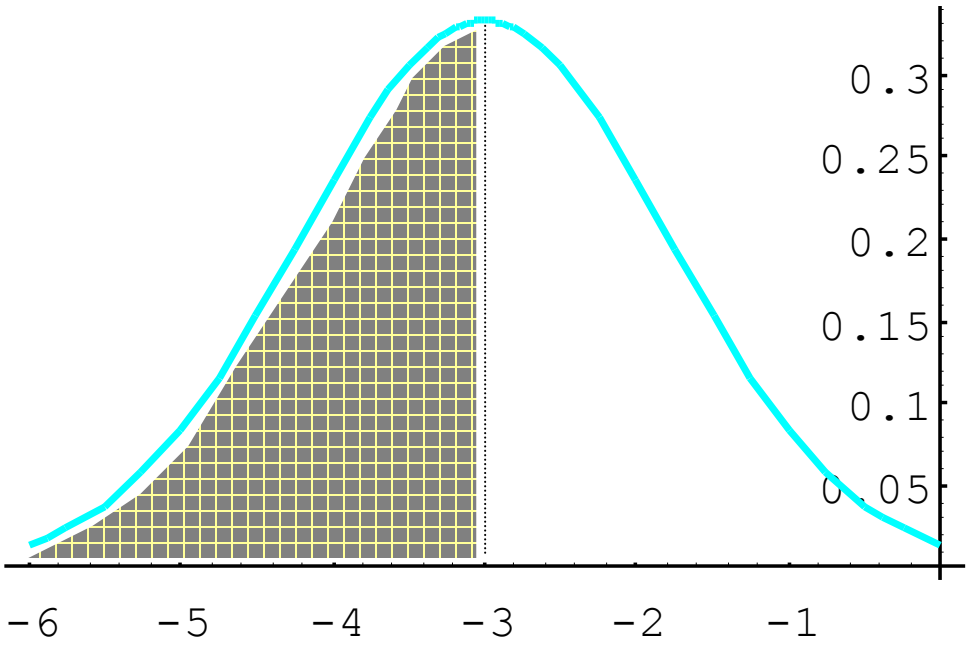
  曲线 $y = f(x)$ 以x轴为渐近线

  曲线 $y = f(x)$ 的图形呈单峰状

$$P(X \le \mu) = F(\mu)$$

$$= 1 - F(\mu) = P(X > \mu)$$

$$= \frac{1}{2}$$



- $f(x)$ 的两个参数：

μ — 位置参数

即固定 σ, 对于不同的 μ, 对应的 $f(x)$ 的形状不变化，只是位置不同

σ — 尺度参数

固定 μ，对于不同的 σ，$f(x)$ 的形状不同.

若 $\sigma_1 < \sigma_2$ 则 $\dfrac{1}{\sqrt{2\pi}\sigma_1} > \dfrac{1}{\sqrt{2\pi}\sigma_2}$ 前者取 μ 附近值的概率更大. $x = \mu \pm \sigma_1$ 所对应的拐点比 $x = \mu \pm \sigma_2$ 所对应的拐点更靠近直线 $x = \mu$

例13 设 $X \sim N(1,4)$, 求 $P(0 \le X \le 1.6)$

解

$$P(0 \le X \le 1.6) = \Phi\left(\frac{1.6-1}{2}\right) - \Phi\left(\frac{0-1}{2}\right)$$

$$= \Phi(0.3) - \Phi(-0.5)$$

$$= \Phi(0.3) - [1 - \Phi(0.5)]$$

$$= 0.6179 - [1 - 0.6915]$$

$$= 0.3094$$

```
>>> from scipy.stats import norm
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Calculate the first four moments:

```
>>> mean, var, skew, kurt = norm.stats(moments='mvsk')
```

Display the probability density function (`pdf`):

```
>>> x = np.linspace(norm.ppf(0.01),
...                 norm.ppf(0.99), 100)
>>> ax.plot(x, norm.pdf(x),
...         'r-', lw=5, alpha=0.6, label='norm pdf')
```

Alternatively, the distribution object can be called (as a function) to fix the shape, location and scale parameters. This returns a "frozen" RV object holding the given parameters fixed.

Freeze the distribution and display the frozen `pdf`:

```
>>> rv = norm()
>>> ax.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')
```

Check accuracy of `cdf` and `ppf`:

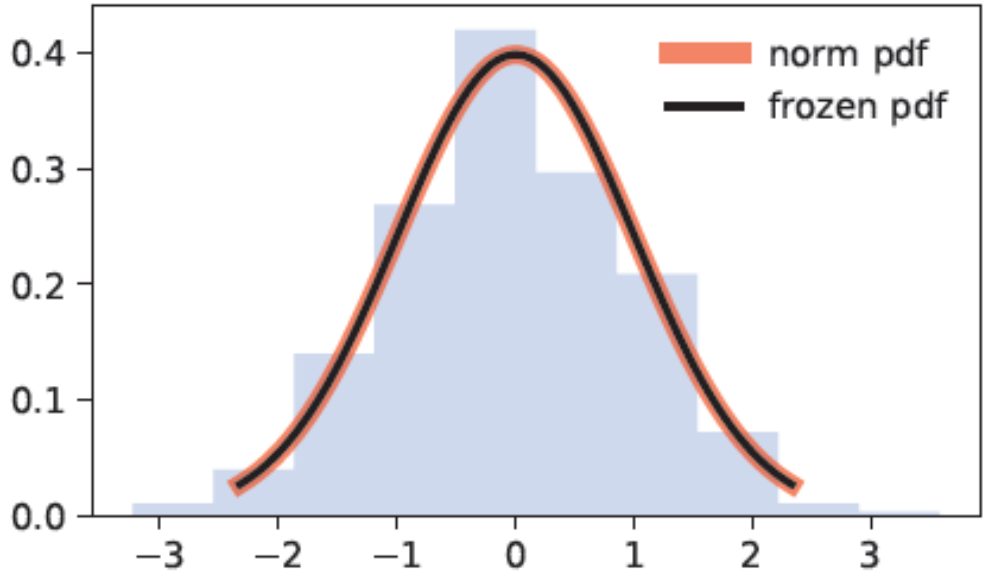
```
>>> vals = norm.ppf([0.001, 0.5, 0.999])
>>> np.allclose([0.001, 0.5, 0.999], norm.cdf(vals))
True
```

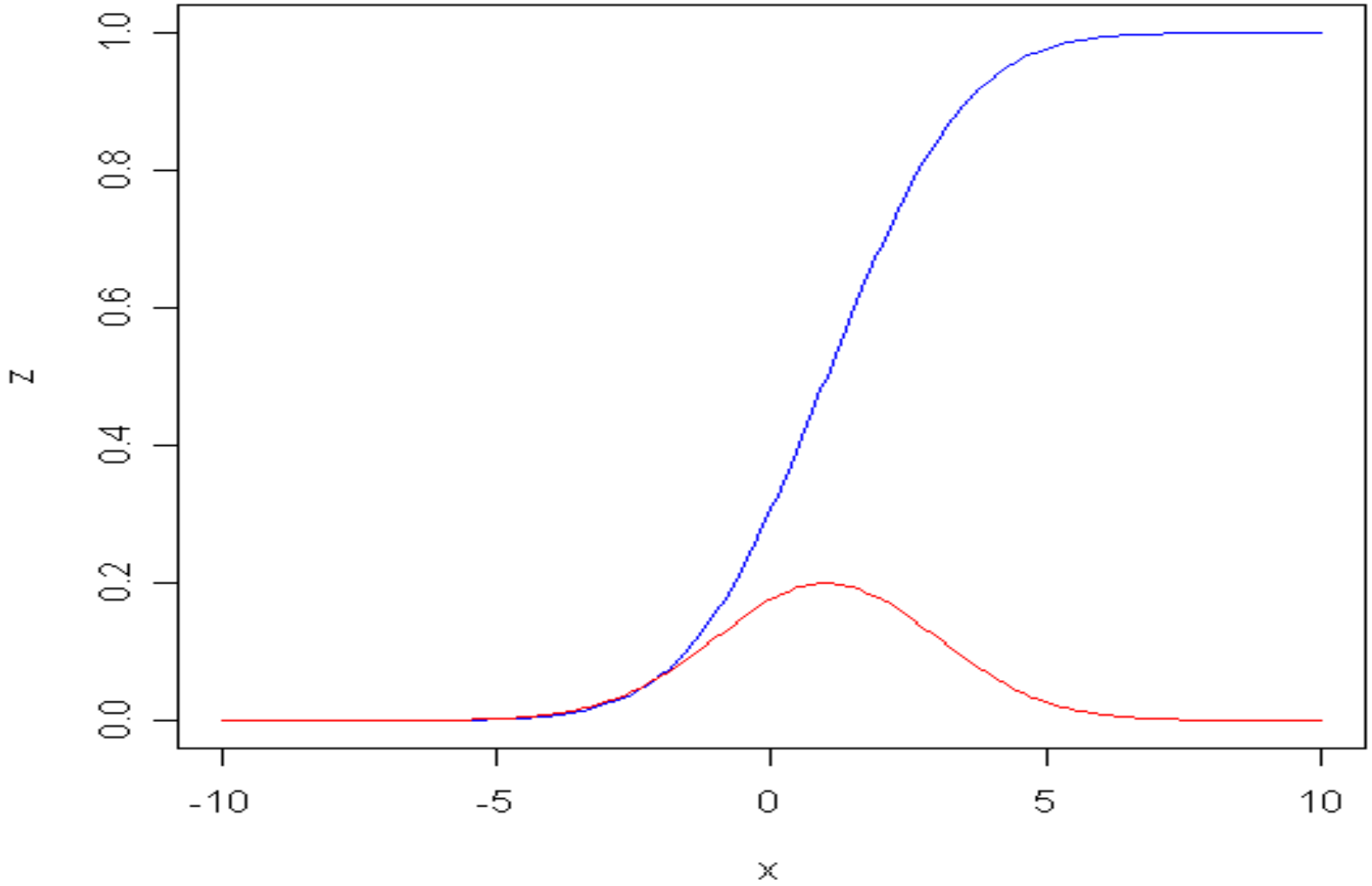
Generate random numbers:

```
>>> r = norm.rvs(size=1000)
```

And compare the histogram:

```
>>> ax.hist(r, density=True, histtype='stepfilled', alpha=0.2)
>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```





结果： **Result =** 0.4937903

密度估计（分布估计）

概率密度估计（Probabilistic Density Estimation），简称密度估计（Density Estimation），是基于一些观测样本来估计一个随机变量的密度函数（Probabilistic Density Function，PDF）。密度估计在数据建模、机器学习中使用广泛。密度估计方法可以分为两类：参数密度估计和非参数密度估计。

参数密度估计（Parametric Density Estimation）

根据先验知识假设随机变量服从某种分布，然后通过训练样本来估计分布的参数.

估计方法：极大似然估计

$$\log p(\mathcal{D};\theta) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)};\theta).$$

非参数密度估计（Nonparametric Density Estimation）

不假设数据服从某种分布，通过将样本空间划分为不同的区域并估计每个区域的概率来近似数据的概率密度函数。

参数密度估计

假设样本 $\boldsymbol{x} \in \mathbb{R}^D$ 服从正态分布

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\Big),$$

其中 $\boldsymbol{\mu}$ 和 $\boldsymbol{\Sigma}$ 分别为正态分布的均值和方差.

数据集 $\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^N$ 的对数似然函数为

$$\log p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{N}{2}\log\Big((2\pi)^D|\boldsymbol{\Sigma}|\Big) - \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}).$$

分别求上式关于 $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ 的偏导数，并令其等于 0. 可得，

$$\boldsymbol{\mu}^{ML} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}^{(n)},$$

$$\boldsymbol{\Sigma}^{ML} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}^{ML})(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}^{ML})^{\top}.$$

数据集 $\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^N$ 的对数似然函数为

$$\log p(\mathcal{D}|\boldsymbol{\mu}) = \sum_{n=1}^{N}\sum_{k=1}^{K} x_k^{(n)}\log(\mu_k). \tag{9.34}$$

多项分布的参数估计为约束优化问题. 引入拉格朗日乘子 λ，将原问题转换为无约束优化问题.

$$\max_{\mu,\lambda} \quad \sum_{n=1}^{N}\sum_{k=1}^{K} x_k^{(n)}\log(\mu_k) + \lambda\Big(\sum_{k=1}^{K}\mu_k - 1\Big). \tag{9.35}$$

分别求上式关于 μ_k, λ 的偏导数，并令其等于 0. 可得，

$$\mu_k^{ML} = \frac{m_k}{N}, \qquad 1 \le k \le K \tag{9.36}$$

其中 $m_k = \sum_{n=1}^{N} x_k^{(n)}$ 为数据集中取值为第 k 个状态的样本数量.

有一大批糖果,现从中随机地取16袋,称得重量(克)如下，设袋装糖果的重量服从正态分布,试求总体均值μ和总体方差σ^2的点估计。

506	508	499	503	504	510	497	512
514	505	493	496	506	502	509	496

参数密度估计一般存在以下问题

- 模型（分布）选择问题
  - 如何选择数据分布的密度函数？
  - 实际数据的分布往往是非常复杂的，而不是简单的正态分布或多项分布。

$$X \sim f(x) = \begin{cases} \frac{1}{\theta} e^{-(x-\mu)/\theta}, & x \geq \mu \\ 0, & \text{其它} \end{cases} \quad \theta, \mu \text{为未知参数}$$

- 不可观测变量问题
  - 即我们用来训练的样本只包含部分的可观测变量，还有一些非常关键的变量是无法观测的，这导致我们很难准确估计数据的真实分布。
- 维度灾难问题
  - 高维数据的参数估计十分困难
  - 随着维度的增加，估计参数所需要的样本数量指数增加。在样本不足时会出现过拟合。

非参密度估计

非参数密度估计(直方图方法除外)需要保留整个训练集。
而参数密度估计不需要保留整个训练集，因此在存储和计算上更加高效。

非参数密度估计（Nonparametric Density Estimation）是不假设数据服从某种分布，通过将样本空间划分为不同的区域并估计每个区域的概率来近似数据的概率密度函数。

对于高维空间中的一个随机向量$\boldsymbol{x}$，假设其服从一个未知分布$p(\boldsymbol{x})$，则$\boldsymbol{x}$落入空间中的小区域$\mathcal{R}$的概率为

$$P = \int_{\mathcal{R}} p(\boldsymbol{x}) d\boldsymbol{x}.$$

给定N 个训练样本$D = \left\{\boldsymbol{x}^{(n)}\right\}_{n=1}^{N}$,落入区域R的样本数量K服从二项分布

$$P_K = \binom{N}{K} P^K (1-P)^{1-K},$$

当N 非常大时，我们可以近似认为

$$P \approx \frac{K}{N}$$

假设区域R足够小，其内部的概率密度是相同的，V是区域的面体积，则有

$$P \approx p(\boldsymbol{x})V$$

结合上述两个公式，得到

$$p(\boldsymbol{x}) \approx \frac{K}{NV}$$

直方图方法（Histogram Method）

以一元为例，假定有数据 $x_1, x_2, \cdots, x_n \in [a,b)$. 对区间 $[a,b)$ 做如下划分，即 $a = a_0 < a_1 < a_2 < \cdots < a_k = b$, $I_i = [a_{i-1}, a_i)$, $i = 1, 2, \cdots, k$. 我们有 $\bigcup_{i=1}^{k} I_i = [a,b)$, $I_i \cap I_j = \varnothing$, $i \neq j$. 令 $n_i = \#\{x_i \in I_i\}$ 为落在 I_i 中数据的个数.

我们如下定义直方图密度估计,

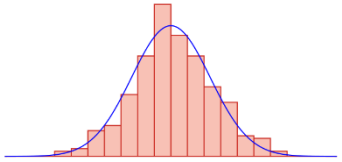
$$\hat{p}(x) = \begin{cases} \dfrac{n_i}{n(a_i - a_{i-1})}, & \text{当} x \in I_i; \\ 0, & \text{当} x \notin [a,b), \end{cases}$$

在实际操作中，我们经常取相同的区间，即 I_i $(i = 1, 2, \cdots, k)$ 的宽度均为 h, 在此情况下，有

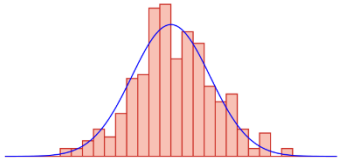
$$\hat{p}(x) = \begin{cases} \dfrac{n_i}{nh}, & \text{当} x \in I_i; \\ 0, & \text{当} x \notin [a,b). \end{cases}$$

上式中, h 既是归一化参数, 又表示每一组的组距, 称为带宽或窗宽. 另外，我们可以看到

$$\int_a^b \hat{p}(x)\mathrm{d}x = \sum_{i=1}^{k} \int_{I_i} n_i/(nh)\mathrm{d}x = \sum_{i=1}^{k} n_i/n = 1.$$



(a) 10 个区间（bin）



(b) 30 个区间（bin）

由上面的例子，我们可以看出，选择不同的带宽，我们会得到不同的结果. 选择合适的带宽，对于得到好的密度估计是很重要的. 在计算最优带宽前，我们先定义 $\hat{p}$ 的平方损失风险 $R(\hat{p}, p) = \int (\hat{p}(x) - p(x))^2 \, \mathrm{d}x$.

定理 7.2 $\int p'(x) \, \mathrm{d}x < +\infty$, 则在平方损失风险下, 有

$$R(\hat{p}, p) \approx \frac{h^2}{12} \int (p'(u))^2 \, \mathrm{d}u + \frac{1}{nh}.$$

极小化上式, 得到理想带宽为

$$h^* = \frac{1}{n^{1/3}} \left(\frac{6}{\int p'(x)^2 \, \mathrm{d}x} \right)^{1/3}.$$

于是理想的带宽为 $h = Cn^{-1/3}$.

直方图的密度定义公式很容易扩展到任意维空间. 设有 n 个观测点 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$, 将空间分成若干小区域 R, V 是区域 R 所包含的体积. 如果有 k 个点落入 R, 则可以得到如下密度公式：$p(\boldsymbol{x})$ 的估计为

$$p(\boldsymbol{x}) \approx \frac{k/n}{V}. \tag{7.1}$$

核密度估计（Kernel Density Estimation）

假设$\mathcal{R}$为d维空间中的一个以点$\boldsymbol{x}$为中心的“超立方体”，并定义核函数来表示一个样本$\boldsymbol{z}$是否落入该超立方体中

$$\phi\left(\frac{z-x}{H}\right)=\begin{cases}1, if|z-x|<\frac{H}{2}\\0, \quad other\end{cases}$$

这个核函数称为Parzen窗核函数。
点 $\boldsymbol{x}$ 的密度估计为

$$p(\boldsymbol{x})=\frac{K}{NH^D}=\frac{1}{NH^D}\sum_{n=1}^{N}\phi\left(\frac{\boldsymbol{x}^{(n)}-\boldsymbol{x}}{H}\right)$$

$$\phi\left(\frac{\boldsymbol{z}-\boldsymbol{x}}{H}\right)=\frac{1}{(2\pi)^{1/2}H}\exp\left(-\frac{\|\boldsymbol{z}-\boldsymbol{x}\|^2}{2H^2}\right),$$

其中h^2是高斯核函数的方差. 这样，点$\boldsymbol{x}$的密度估计为

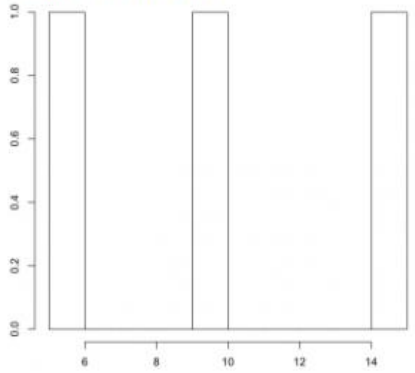
$$p(\boldsymbol{x})=\frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi)^{1/2}H}\exp\left(-\frac{\|\boldsymbol{z}-\boldsymbol{x}\|^2}{2H^2}\right).$$

这里令$u=\frac{z-x}{H}$

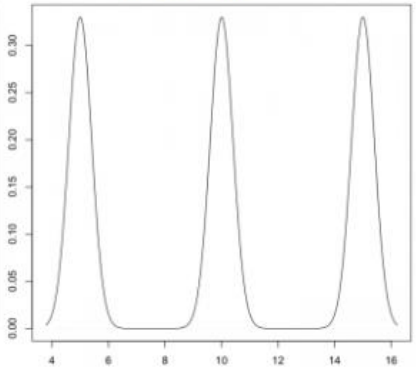
核函数名称	核函数 $K(u)$
Parzen 窗 (Uniform)	$\frac{1}{2}I(\lvert u\rvert\leqslant 1)$
三角 (Triangle)	$(1-\lvert u\rvert)I(\lvert u\rvert\leqslant 1)$
Epanechikov	$\frac{3}{4}(1-u^2)I(\lvert u\rvert\leqslant 1)$
四次 (Quartic)	$\frac{15}{16}(1-u^2)I(\lvert u\rvert\leqslant 1)$
三权 (Triweight)	$\frac{35}{32}(1-u^2)^3I(\lvert u\rvert\leqslant 1)$
高斯 (Gauss)	$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}u^2\right)$
余弦 (Cosinus)	$\frac{\pi}{4}\cos\left(\frac{\pi}{2}u\right)I(\lvert u\rvert\leqslant 1)$
指数 (Exponent)	$\exp\{\lvert u\rvert\}$

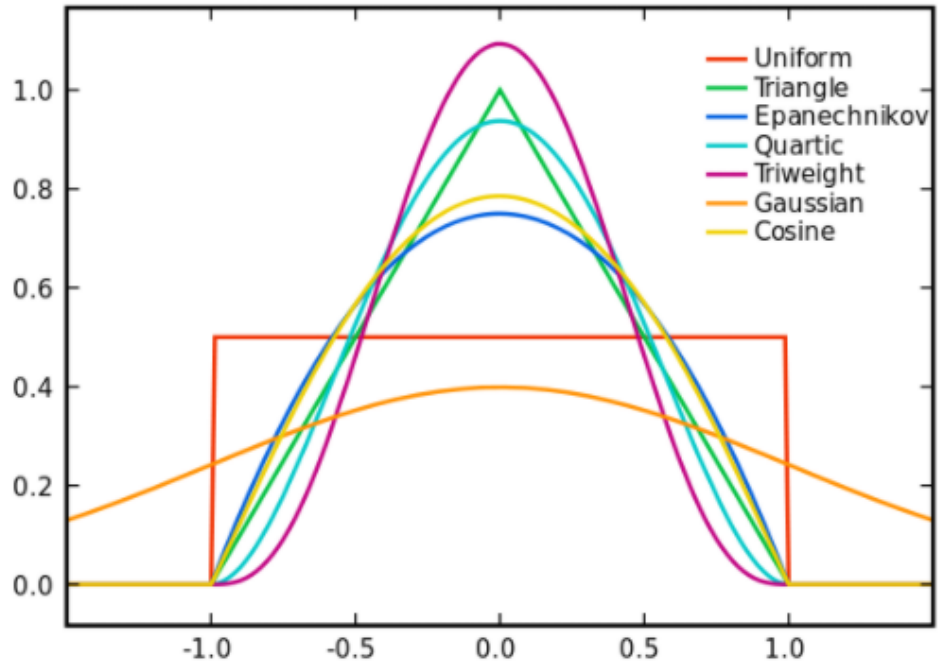
以下面3个数据点的一维数据集为例：5, 10, 15

绘制成直方图是这样的：



而使用KDE则是：





Epanechnikov 内核在均方误差意义下是最优的，效率损失也很小。由于高斯内核方便的数学性质，也经常使用 $K(x) = \phi(x)$，$\phi(x)$为标准正态概率密度函数。核密度估计与直方图很类似，但相比于直方图还有光滑连续的性质。

均匀核函数 k(x)=1/2,-1≤x≤1 加入带宽h后： kh(x)=1/(2h),-h≤x≤h

三角核函数 k(x)=1-|x|,-1≤x≤1 加入带宽h后： kh(x)=(h-|x|)/h^2,-h≤x≤h

伽马核函数 kxi(x)=[x^(α-1)exp{-xα/xi}]/[(xi/α)^α.Γ(α)]

高斯核函数K(x,xc)=exp(-||x-xc||^2/(2*σ)^2)其中xc为核函数中心,σ为函数的宽度参数

K近邻估计

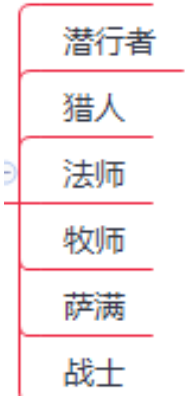
核密度估计方法中的核宽度是固定的，因此同一个宽度可能对高密度的区域过大，而对低密度区域过小。一种更灵活的方式是设置一种可变宽度的区域，并使得落入每个区域中样本数量为固定的K。要估计点x的密度，首先找到一个以x为中心的球体，使得落入球体的样本数量为K，就可以计算出点x的密度。所以，这种方法称为K 近邻方法（K-Nearest Neighbor Method）。

$$\tilde{p}_n(\boldsymbol{x}) = \frac{k_n/n}{V_n}$$

在K 近邻方法中，K 的选择也十分关键。如果K 太小，无法有效地估计密度函数；而K 太大也会使得局部的密度不准确，并且增加计算开销。

$$k = k_0 n^{4/(d+4)}$$

K 近邻方法也经常用于分类问题，称为K 近邻分类器（K-Nearest Neighbor Classifier）。



聚类（Clustering），告诉你属于哪个类别。

密度估计就是分布式表示（Distributed Representation）
告诉你属于这个类别的概率。

class sklearn.neighbors.KernelDensity(*, *bandwidth=1.0*, *algorithm='auto'*, *kernel='gaussian'*, *metric='euclidean'*, *atol=0*, *rtol=0*, *breadth_first=True*, *leaf_size=40*, *metric_params=None*)

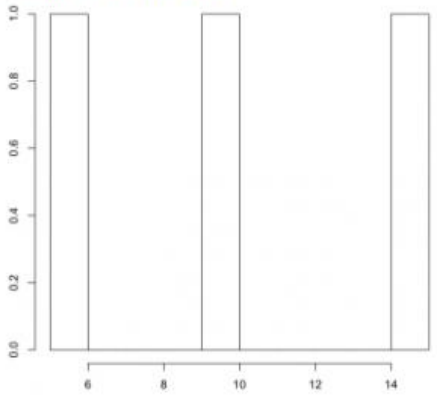
```
from sklearn.neighbors import KernelDensity
import numpy as np
rng = np.random.RandomState(42)
X = rng.random_sample((100, 3))
kde = KernelDensity(kernel='gaussian', bandwidth=0.5).fit(X)
log_density = kde.score_samples(X[:3])
log_density
```

KDE带宽h

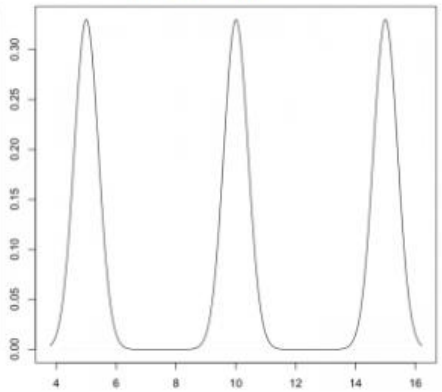
- 如何选定核函数的“方差”呢？这其实是由带宽h来决定，不同的带宽下的核函数估计结果差异很大。
- 带宽反映了KDE曲线整体的平坦程度，也即观察到的数据点在KDE曲线形成过程中所占的比重。带宽越大，观察到的数据点在最终形成的曲线形状中所占比重越小，KDE整体曲线就越平坦；带宽越小，观察到的数据点在最终形成的曲线形状中所占比重越大，KDE整体曲线就越陡峭。

以下面3个数据点的一维数据集为例：5, 10, 15

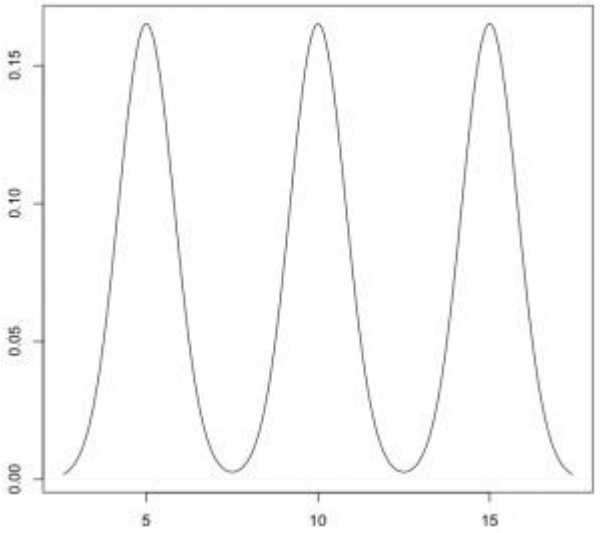
绘制成直方图是这样的：



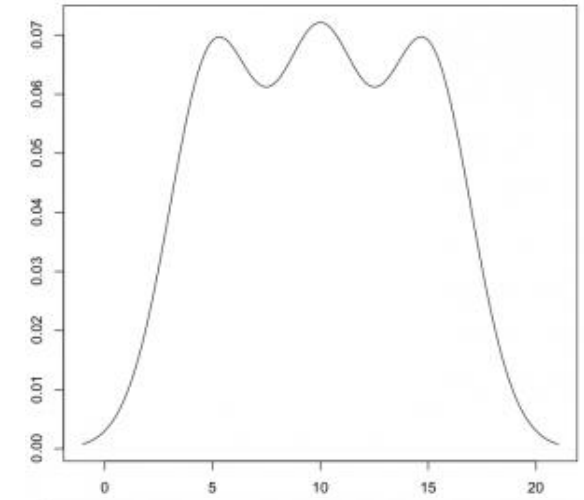
而使用KDE则是：



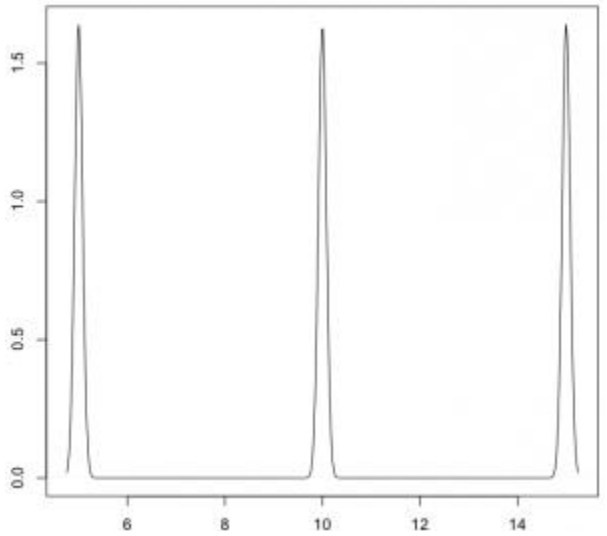
- 还是以上面3个数据点的一维数据集为例，如果增加带宽，那么生成的KDE曲线就会变平坦：



如果进一步增加带宽，那么KDE曲线在变平坦的同时，还会发生波形合成：



相反，如果减少带宽，那么KDE曲线就会变得更加陡峭：

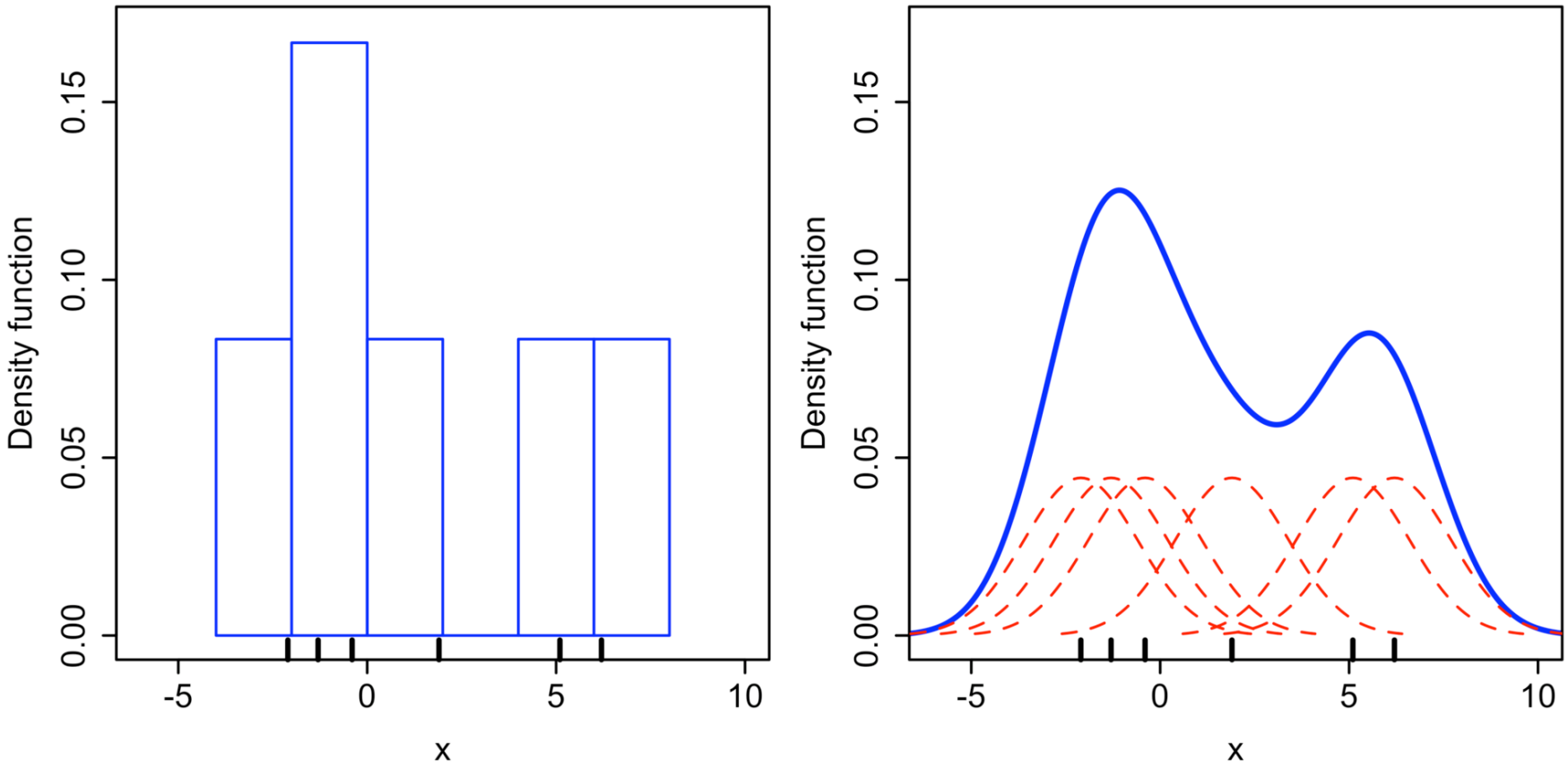


从数学上来说，对于数据点Xi，如果带宽为h，那么在Xi处所形成的曲线函数为(其中K为核函数)：

$$\frac{1}{h}K\left(\frac{x-x_i}{h}\right)$$

在上面的函数中，K函数内部的h分母用于调整KDE曲线的宽幅，而K函数外部的h分母则用于保证曲线下方的面积符合KDE的规则(KDE曲线下方面积和为1)。

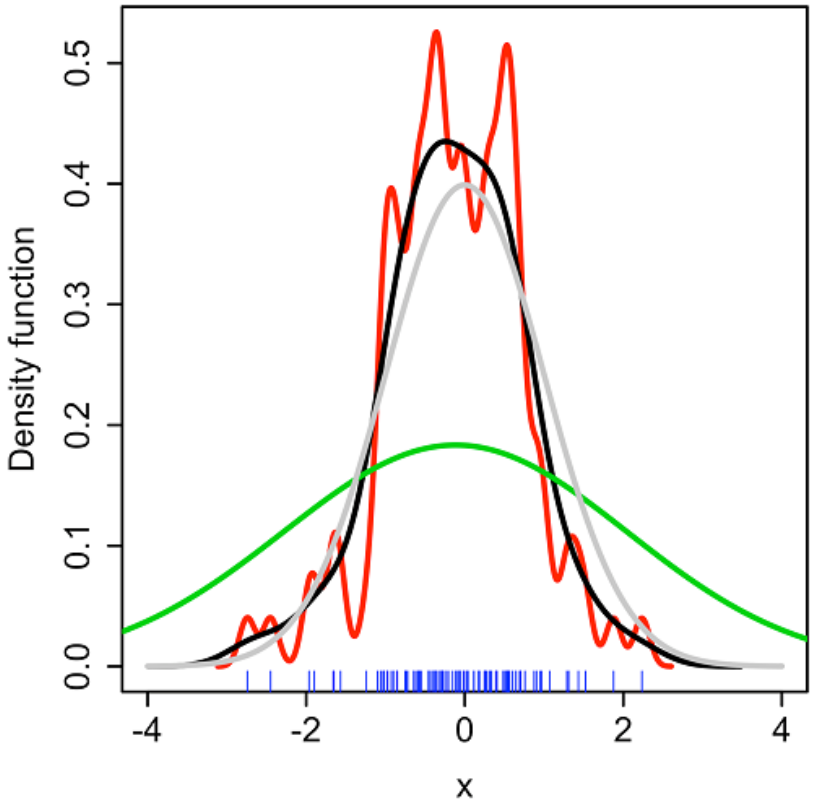
- 下图为直方图与核函数估计对 $x_1 = -2.1, x_2 = -1.3, x_3 = -0.4, x_4 = 1.9, x_5 = 5.1, x_6 = 6.2$ 六个点的“拟合”结果。



在直方图中，横轴间隔为2，数据落到某个区间，此区间y轴增加1/12。在核密度估计中，不放令正态分布方差为2.25，红色的虚线表示由每一个数据得到的正态分布，叠加一起得到核密度估计的结果，用蓝色表示。

带宽的选择

- 那么问题就来了，如何选定核函数的“方差”呢？这其实是由h来决定，不同的带宽下的核函数估计结果差异很大，如下图：



带宽的选择

带宽的选择很大程度上取决于主观判断：如果认为真实的概率分布曲线是比较平坦的，那么就选择较大的带宽；相反，如果认为真实的概率分布曲线是比较陡峭的，那么就选择较小的带宽。

带宽计算好像也有相应的方法，如R语言中计算带宽时，默认采用”nrd0″方法。

如何选择h？显然是选择可以使误差最小的。下面用平均积分平方误差（mean intergrated squared error）的大小来衡量h的优劣。

$$\mathrm{MISE}(h) = E\int (\hat{f}_h(x) - f(x))^2\,dx.$$

在weak assumptions下，MISE (h) =AMISE(h) + o(1/(nh) + h4) ，其中AMISE为渐进的MISE。而AMISE有，

$$\mathrm{AMISE}(h) = \frac{R(K)}{nh} + \frac{1}{4}m_2(K)^2h^4R(f'')$$

其中，

$$R(g) = \int g(x)^2\,dx \quad m_2(K) = \int x^2K(x)\,dx$$

为了使MISE(h)最小，则转化为求极点问题，

$$\frac{\partial}{\partial h}\mathrm{AMISE}(h) = -\frac{R(K)}{nh^2} + m_2(K)^2h^3R(f'') = 0$$

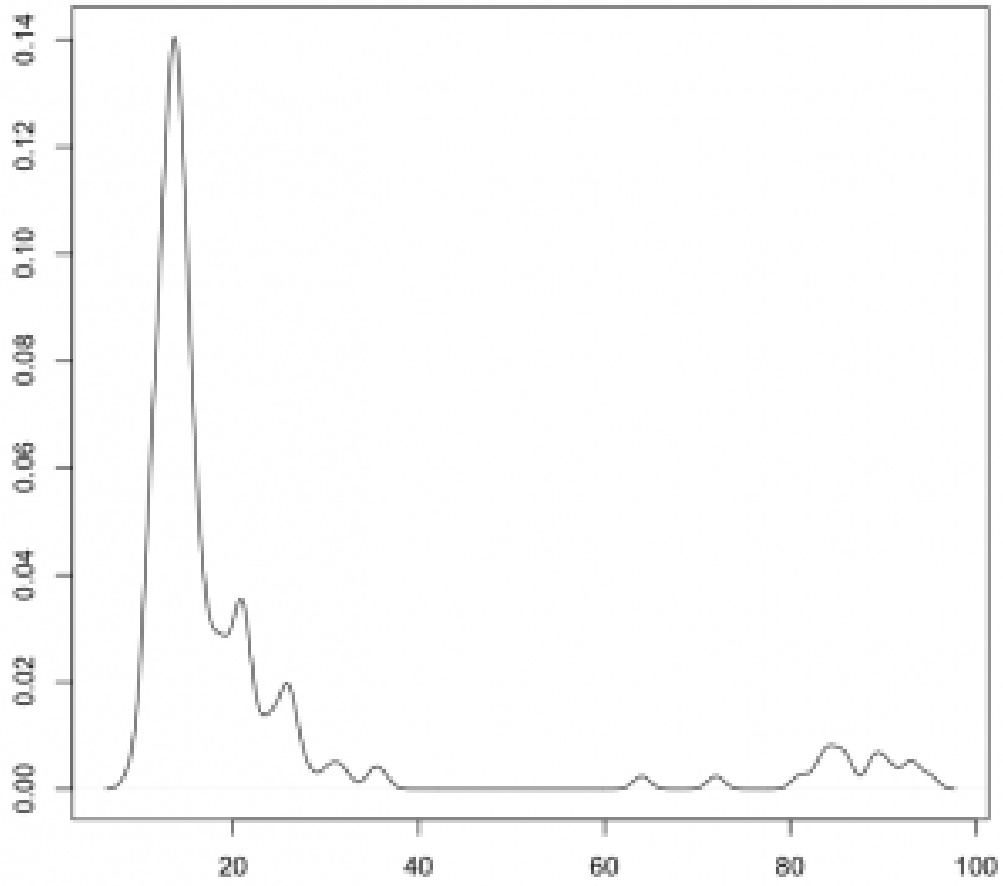
$$h_{\mathrm{AMISE}} = \frac{R(K)^{1/5}}{m_2(K)^{2/5}R(f'')^{1/5}n^{1/5}}.$$

当核函数确定之后，h公式里的R、m、f''都可以确定下来，有（hAMISE ~ n-1/5），AMISE(h) = O(n-4/5)。

如果带宽不是固定的，其变化取决于估计的位置（balloon estimator）或样本点（逐点估计 pointwise estimator），由此可以产产生一个非常强大的方法称为自适应或可变带宽核密度估计。

带宽的选择

- 在选择合适的核函数及带宽后，KDE可以模拟真实的概率分布曲线，并得到平滑而漂亮的结果。以近200个点的CPU使用率为例，使用KDE绘制的结果为：



作业：

1. 随机变量$X \sim \mathrm{B}(10, 0.4)$, 求密度函数和分布函数, 求$P(X = 3)$;
2. 随机变量$X \sim \mathrm{U}(2, 4)$, 画出 密度函数和分布函数, 求$P(2.5 < X < 3.5)$;
3. 随机变量$X \sim \mathrm{E}(4)$, 画出 密度函数和分布函数, 求$P(1 < X < 5)$;
4. 随机变量$X \sim N(1, 4)$, 画出 密度函数和分布函数, 求$P(-2 < X < 3)$.
5. 随便找一组/生成数据（样本量大于1000），利用不同的核函数进行核密度估计。