# Python 计算库教程

# 1 计算库简介

Python 是一种面向对象的、动态的程序设计语言。具有非常简洁而清晰的语法，适合于完成各种高层任务。它既可以用来快速开发程序脚本，也可以用来开发大规模的软件。

随着 NumPy，Pandas，SciPy，Matplotlib 等众多科学计算程序库的开发，Python 越来越适合于做科学计算、绘制高质量的 2D 和 3D 图像。和科学计算领域最流行的商业软件 Matlab 相比，Python 是一门通用的程序设计语言，比 Matlab 所采用的脚本语言的应用范围更广泛，有更多的程序库的支持。虽然 Matlab 中的许多高级功能和 toolbox 目前还是无法替代的，不过在日常的科研开发之中仍然有很多的工作是可以用 Python 代劳的。

# 2 Numpy 库使用

## 2.1 Numpy 快速入门

Numpy 是 Python 第一个矩阵类型库，提供了大量矩阵处理的函数。非正式地来说，它是一个使运算更简便，执行速度更快的库，因为它的内部运算是通过 C 语言实现的。numpy 包含了两种基本的数据类型：矩阵和数组。二者在处理上稍有不同，如果之前熟悉 MATLAB，矩阵的处理不是难事。

### 2.1.1 基本概念

Numpy 支持比 Python 本身更为丰富的数值类型，可用.astype()方法在不同的数值类型之间相互转换，可以使用 .dtype 来查看 dtype 属性。

Numpy 最核心且最重要的一个特性就是 ndarray 多维数组对象，它区别于 python 的标准类，拥有对高维数组的处理能力，这也是数值计算过程中缺一不可的重要特性。主要通过以下途径创建数组：

1) 从 Python 数组结构列表，元组等转换。
2) 使用 np.arange、np.ones、np.zeros 等 numpy 原生方法。
3) 从存储空间读取数组。
4) 通过使用字符串或缓冲区从原始字节创建数组。
5) 使用特殊函数，如 random。

### 2.1.2 简单使用数组 Array

简单示例：

```
from numpy import array
```

```
mm = array((1,1,1))
nn = array((1,2,3))
sum = mm + nn
print(sum)   # [2 3 4]
#对每个元素平方
sum2 = sum**2
print(sum2[0],sum2[1],sum2[2],sum2) # 4 9 16 [4 9 16]
#多维数组：
MulArr = array([[1,2,3],[1,1,1]])
print(MulArr) # [ [1 2 3] [1 1 1] ]
print(MulArr[0],MulArr[1]) #  [1 2 3] [1 1 1]
#当把两个数组乘起来的时候，两个数组的元素将对应相乘
a1 = array([1,2,3])
a2 = array([3,4,5])
a12 = a1 * a2
print(a12)  # [ 3  8 15]
```

### 2.1.3 简单使用矩阵 matrix

```
from numpy import mat ,matrix,shape
ss = mat([1,2,3])
mm = matrix([1,2,3])
print(ss,mm)    #[[1 2 3]] [[1 2 3]]
print(mm[0,1])  # 2
#把 Python 列表转成 Numpy 矩阵
pyList = [5,11,13]
mList = mat(pyList)
print(mList)    # [[ 5 11 13]]
#矩阵相乘必须行数等于列数
#矩阵 SS 转置
MulMat = mm * ss.T
```

```
print(MulMat) # [[14]]
print(shape(mm)) # (1, 3) 表示一行三列
mrand = mat([4,5,1,3])
print(mrand)  # [[4 5 1 3]]
mrand.sort()
print(mrand,mrand.mean()) # [[1 3 4 5]] 3.25
```

# 2.2 Numpy 随机抽样库

Numpy 的随机抽样功能非常强大，主要由 numpy.random 模块完成。主要包括以下方法：

### 2.2.1 随机纯小数函数 rand

numpy.random.rand(d0, d1, ..., dn) 方法的作用为：指定一个数组，并使用 [0, 1) 区间随机数据填充，这些数据均匀分布。示例用法：

random.rand(2,5)

### 2.2.2 随机正太分布纯小数函数 randn

numpy.random.randn(d0, d1, ..., dn) 与 numpy.random.rand(d0, d1, ..., dn) 的区别在于，返回的随机数据符合标准正太分布。示例用法：

random.randn(1,10)

### 2.2.3 随机整数函数 randint

randint(low, high, size, dtype) 方法将会生成 [low, high) 的随机整数。注意这是一个半开半闭区间。

### 2.2.4 随机整数函数 random_integers

random_integers(low, high, size) 方法将会生成 [low, high] 的 np.int 类型随机整数。注意这是一个闭区间。

### 2.2.5 随机浮点数 random_sample

random_sample(size) 方法将会在 [0, 1) 区间内生成指定 size 的随机浮点数。

与 numpy.random.random_sample 类似的方法还有：

- `numpy.random.random([size])`
- `numpy.random.ranf([size])`
- `numpy.random.sample([size])`

这四个函数的效果类似。

### 2.2.6 随机数函数 choice

choice(a, size, replace, p) 方法将会给定的 1 维数组里生成随机数。

### 2.2.7 概率密度分布

除了上面介绍的 6 中随机数生成方法，numpy 还提供了大量的满足特定概率密度分布的样本生成方法。它们的使用方法和上面非常相似，这里就不再一一介绍了。列举如下：

1） random.beta(a，b，size)：从 Beta 分布中生成随机数。
2） random.binomial(n, p, size)：从二项分布中生成随机数。
3） random.chisquare(df，size)：从卡方分布中生成随机数。
4） random.dirichlet(alpha，size)：从 Dirichlet 分布中生成随机数。
5） random.exponential(scale，size)：从指数分布中生成随机数。
6） random.f(dfnum，dfden，size)：从 F 分布中生成随机数。
7） random.gamma(shape，scale，size)：从 Gamma 分布中生成随机数。
8） random.geometric(p，size)：从几何分布中生成随机数。
9） random.gumbel(loc，scale，size)：从 Gumbel 分布中生成随机数。
10）random.hypergeometric(ngood, nbad, nsample, size)：从超几何分布中生成随机数。
11）random.laplace(loc，scale，size)：从拉普拉斯双指数分布中生成随机数。
12）random.logistic(loc，scale，size)：从逻辑分布中生成随机数。
13）random.lognormal(mean，sigma，size)：从对数正态分布中生成随机数。
14）random.logseries(p，size)：从对数系列分布中生成随机数。
15）random.multinomial(n，pvals，size)：从多项分布中生成随机数。
16）random.multivariate_normal(mean, cov, size)：从多变量正态分布绘制随机样本。
17）random.negative_binomial(n, p, size)：从负二项分布中生成随机数。
18）random.noncentral_chisquare(df，nonc，size)：从非中心卡方分布中生成随机数。
19）random.noncentral_f(dfnum, dfden, nonc, size)：从非中心 F 分布中抽取样本。
20）random.normal(loc，scale，size)：从正态分布绘制随机样本。
21） random.pareto(a，size)：从具有指定形状的 Pareto II 或 Lomax 分布中生成随机数。
22）random.poisson(lam，size)：从泊松分布中生成随机数。
23）random.power(a，size)：从具有正指数 a-1 的功率分布中在 0，1 中生成随机数。
24）random.rayleigh(scale，size)：从瑞利分布中生成随机数。
25）random.standard_cauchy(size)：从标准 Cauchy 分布中生成随机数。
26）random.standard_exponential(size)：从标准指数分布中生成随机数。
27）random.standard_gamma(shape，size)：从标准 Gamma 分布中生成随机数。
28）random.standard_normal(size)：从标准正态分布中生成随机数。
29）random.standard_t(df，size)：从具有 df 自由度的标准学生 t 分布中生成随机数。
30）random.triangular(left，mode，right，size)：从三角分布中生成随机数。
31）random.uniform(low，high，size)：从均匀分布中生成随机数。
32）random.vonmises(mu，kappa，size)：从 von Mises 分布中生成随机数。
33）random.wald(mean，scale，size)：从 Wald 或反高斯分布中生成随机数。
34）random.weibull(a，size)：从威布尔分布中生成随机数。
35）random.zipf(a，size)：从 Zipf 分布中生成随机数。

### 2.2.8 示例程序

```
import numpy.random as ra
print(ra.rand(2,5))
# [[ 0.05013455  0.55763131  0.08014356  0.62709848  0.31415141]
print(ra.randn(1,10))
# [[ 1.19805691 0.73918176 0.42227802-1.39853262  0.61980295 -0.05620518
#  -0.11752008  0.88871697 -1.21298003 -0.80712286]]
print(ra.randint(2,5,10))
# [3 4 2 3 2 4 2 2 4 4]
print(ra.random_integers(2,5,10))
# [2 3 3 2 3 3 3 5 4 5]
print(ra.random_sample([10]))
# [ 0.06092396 0.79561814  0.0985014   0.72389196  0.09962334  0.54996594
# 0.95786768  0.28105025  0.07484735  0.8273271 ]
print(ra.choice(10,5))
# [3 4 5 2 0]
```

# 2.3 Numpy 数学函数

使用 python 自带的运算符，你可以完成数学中的加减乘除，以及取余、取整，幂次计算等。导入自带的 math 模块之后，里面又包含绝对值、阶乘、开平方等一些常用的数学函数。不过，这些函数仍然相对基础。如果要完成更加复杂一些的数学计算，就会显得捉襟见肘了。

numpy 为我们提供了更多的数学函数，以帮助我们更好地完成一些数值计算。下面就依次来看一看。

### 2.3.1 三角函数

首先，看一看 numpy 提供的三角函数功能。这些方法有：

1) numpy.sin(x)：三角正弦。
2) numpy.cos(x)：三角余弦。
3) numpy.tan(x)：三角正切。
4) numpy.arcsin(x)：三角反正弦。
5) numpy.arccos(x)：三角反余弦。
6) numpy.arctan(x)：三角反正切。

7) numpy.hypot(x1,x2)：直角三角形求斜边。
8) numpy.degrees(x)：弧度转换为度。
9) numpy.radians(x)：度转换为弧度。
10) numpy.deg2rad(x)：度转换为弧度。
11) numpy.rad2deg(x)：弧度转换为度。

这些函数用法非常简单，就不再一一举例了。

### 2.3.2 双曲函数

在数学中，双曲函数是一类与常见的三角函数类似的函数。双曲函数经常出现于某些重要的线性微分方程的解中，使用 numpy 计算它们的方法为：

1) numpy.sinh(x)：双曲正弦。
2) numpy.cosh(x)：双曲余弦。
3) numpy.tanh(x)：双曲正切。
4) numpy.arcsinh(x)：反双曲正弦。
5) numpy.arccosh(x)：反双曲余弦。
6) numpy.arctanh(x)：反双曲正切。

### 2.3.3 数值修约

数值修约，又称数字修约，是指在进行具体的数字运算前，按照一定的规则确定一致的位数，然后舍去某些数字后面多余的尾数的过程[via. 维基百科]。比如，我们常听到的「4 舍 5 入」就属于数值修约中的一种。

1) numpy.around(a)：平均到给定的小数位数。
2) numpy.round_(a)：将数组舍入到给定的小数位数。
3) numpy.rint(x)：修约到最接近的整数。
4) numpy.fix(x, y)：向 0 舍入到最接近的整数。
5) numpy.floor(x)：返回输入的底部(标量 x 的底部是最大的整数 i)。
6) numpy.ceil(x)：返回输入的上限(标量 x 的底部是最小的整数 i).
7) numpy.trunc(x)：返回输入的截断值。

### 2.3.4 求和、求积、差分

下面这些方法用于数组内元素或数组间进行求和、求积以及进行差分。

1) numpy.prod(a, axis, dtype, keepdims)：返回指定轴上的数组元素的乘积。
2) numpy.sum(a, axis, dtype, keepdims)：返回指定轴上的数组元素的总和。
3) numpy.nanprod(a, axis, dtype, keepdims)：返回指定轴上的数组元素的乘积，将 NaN 视作 1。
4) numpy.nansum(a, axis, dtype, keepdims)：返回指定轴上的数组元素的总和，将 NaN 视作 0。
5) numpy.cumprod(a, axis, dtype)：返回沿给定轴的元素的累积乘积。
6) numpy.cumsum(a, axis, dtype)：返回沿给定轴的元素的累积总和。
7) numpy.nancumprod(a, axis, dtype)：返回沿给定轴的元素的累积乘积，将 NaN 视作 1。
8) numpy.nancumsum(a, axis, dtype)：返回沿给定轴的元素的累积总和，将 NaN 视作 0。

9) numpy.diff(a, n, axis)：计算沿指定轴的第 n 个离散差分。
10) numpy.ediff1d(ary, to_end, to_begin)：数组的连续元素之间的差异。
11) numpy.gradient(f)：返回 N 维数组的梯度。
12) numpy.cross(a, b, axisa, axisb, axisc, axis)：返回两个(数组）向量的叉积。
13) numpy.trapz(y, x, dx, axis)：使用复合梯形规则沿给定轴积分。

### 2.3.5 指数和对数

如果你需要进行指数或者对数求解，可以用到以下这些方法。

1) numpy.exp(x)：计算输入数组中所有元素的指数。
2) numpy.expm1(x)：对数组中的所有元素计算 exp(x） - 1.
3) numpy.exp2(x)：对于输入数组中的所有 p，计算 2 ** p。
4) numpy.log(x)：计算自然对数。
5) numpy.log10(x)：计算常用对数。
6) numpy.log2(x)：计算二进制对数。
7) numpy.log1p(x)：log(1 + x)。
8) numpy.logaddexp(x1, x2)：log2(2**x1 + 2**x2)。
9) numpy.logaddexp2(x1, x2)：log(exp(x1) + exp(x2))。

### 2.3.6 算术运算

当然，numpy 也提供了一些用于算术运算的方法，使用起来会比 python 提供的运算符灵活一些，主要是可以直接针对数组。

1) numpy.add(x1, x2)：对应元素相加。
2) numpy.reciprocal(x)：求倒数 1/x。
3) numpy.negative(x)：求对应负数。
4) numpy.multiply(x1, x2)：求解乘法。
5) numpy.divide(x1, x2)：相除 x1/x2。
6) numpy.power(x1, x2)：类似于 x1^x2。
7) numpy.subtract(x1, x2)：减法。
8) numpy.fmod(x1, x2)：返回除法的元素余项。
9) numpy.mod(x1, x2)：返回余项。
10) numpy.modf(x1)：返回数组的小数和整数部分。
11) numpy.remainder(x1, x2)：返回除法余数。

### 2.3.7 矩阵和向量积

求解向量、矩阵、张量的点积等同样是 numpy 非常强大的地方。

1) numpy.dot(a,b)：求解两个数组的点积。
2) numpy.vdot(a,b)：求解两个向量的点积。
3) numpy.inner(a,b)：求解两个数组的内积。
4) numpy.outer(a,b)：求解两个向量的外积。
5) numpy.matmul(a,b)：求解两个数组的矩阵乘积。
6) numpy.tensordot(a,b)：求解张量点积。
7) numpy.kron(a,b)：计算 Kronecker 乘积。

### 2.3.8 其他

除了上面这些归好类别的方法，numpy 中还有一些用于数学运算的方法，归纳如下：

1) numpy.angle(z, deg)：返回复参数的角度。
2) numpy.real(val)：返回数组元素的实部。
3) numpy.imag(val)：返回数组元素的虚部。
4) numpy.conj(x)：按元素方式返回共轭复数。
5) numpy.convolve(a, v, mode)：返回线性卷积。
6) numpy.sqrt(x)：平方根。
7) numpy.cbrt(x)：立方根。
8) numpy.square(x)：平方。
9) numpy.absolute(x)：绝对值，可求解复数。
10) numpy.fabs(x)：绝对值。
11) numpy.sign(x)：符号函数。
12) numpy.maximum(x1, x2)：最大值。
13) numpy.minimum(x1, x2)：最小值。
14) numpy.nan_to_num(x)：用 0 替换 NaN。
15) numpy.interp(x, xp, fp, left, right, period)：线性插值。

### 2.3.9 示例程序

```
import numpy as np
a = np.array([1.21, 2.53, 3.86])
print(a)                          # [ 1.21    2.53    3.86]
print(np.around(a),np.round(a))   # [ 1.    3.    4.] [ 1.    3.    4.]
print(np.fix(a),np.floor(a))      # [ 1.    2.    3.] [ 1.    2.    3.]
print(np.ceil(a),np.trunc(a))     # [ 2.    3.    4.] [ 1.    2.    3.]

b=np.arange(5)
print(b)                          # [0 1 2 3 4]
print(np.prod(b))                 #  所有元素乘积       0
print(np.sum(b))                  #  所有元素和         10
print(np.nanprod(b))              # 默认轴上所有元素乘积       0
print(np.nansum(b))               #  默认轴上所有元素和          10
print(np.cumprod(b))              #  默认轴上元素的累积乘积    [0 0 0 0 0]
print(np.diff(b))                 # 默认轴上元素差分             [1 1 1 1]

a1 = np.random.randint(1, 10, 5)
a2 = np.random.randint(1, 10, 5)
print(a1,a2)                      # [9 4 9 7 4] [9 7 2 7 5]
suma = np.add(a1, a2)
print(suma)                       # [18 11 11 14    9]
print(np.reciprocal(a1))          # [0 0 0 0 0]
print(np.negative(a1))            #  # [-9 -4 -9 -7 -4]
print(np.multiply(a1,a2))         # [81 28 18 49 20]
```

```
print(np.divide(a1,a2))          # [ 1. 0.57142857 4.5 1. 0.8 ]
print(np.power(a1,a2))           # [387420489 16384 81 823543 1024]
print(np.subtract(a1,a2))        # [ 0 -3   7   0 -1]
print(np.fmod(a1,a2))            # [0 4 1 0 4]
print(np.mod(a1,a2))             # [0 4 1 0 4]
print(np.remainder(a1,a2))       # [0 4 1 0 4]
```

# 2.4 Numpy 代数运算

### 2.4.1 代数运算函数

numpy 中还包含一些代数运算的方法，尤其是涉及到矩阵的计算方法，求解特征值、特征向量、逆矩阵等，非常方便。

1) numpy.linalg.cholesky(a)：Cholesky 分解。
2) numpy.linalg.qr(a ,mode)：计算矩阵的 QR 因式分解。
3) numpy.linalg.svd(a ,full_matrices,compute_uv)：奇异值分解。
4) numpy.linalg.eig(a)：计算正方形数组的特征值和右特征向量。
5) numpy.linalg.eigh(a, UPLO)：返回 Hermitian 或对称矩阵的特征值和特征向量。
6) numpy.linalg.eigvals(a)：计算矩阵的特征值。
7) numpy.linalg.eigvalsh(a, UPLO)：计算 Hermitian 或真实对称矩阵的特征值。
8) numpy.linalg.norm(x ,ord,axis,keepdims)：计算矩阵或向量范数。
9) numpy.linalg.cond(x ,p)：计算矩阵的条件数。
10) numpy.linalg.det(a)：计算数组的行列式。
11) numpy.linalg.matrix_rank(M ,tol)：使用奇异值分解方法返回秩。
12) numpy.linalg.slogdet(a)：计算数组的行列式的符号和自然对数。
13) numpy.trace(a ,offset,axis1,axis2,dtype,out)：沿数组的对角线返回总和。
14) numpy.linalg.solve(a,b)：求解线性矩阵方程或线性标量方程组。
15) numpy.linalg.tensorsolve(a,b ,axes)：为 x 解出张量方程 a x = b
16) numpy.linalg.lstsq(a,b ,rcond)：将最小二乘解返回到线性矩阵方程。
17) numpy.linalg.inv(a)：计算逆矩阵。
18) numpy.linalg.pinv(a ,rcond)：计算矩阵的（Moore-Penrose）伪逆。
19) numpy.linalg.tensorinv(a ,ind)：计算 N 维数组的逆。

### 2.4.2 示例程序

```
import numpy as np

a = np.array([3, 4])
print(np.linalg.norm(a))    # 5.0

b = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])
c = np.array([1, 0, 1])

# 矩阵和向量之间的乘法
```

```
print(np.dot(b, c))                # array([ 4, 10, 16])
print(np.dot(c, b.T))              # array([ 4, 10, 16])

print(np.trace(b))                 #  求矩阵的迹，15
print(np.linalg.det(b))            #  求矩阵的行列式值，0
print(np.linalg.matrix_rank(b))    #  求矩阵的秩，2，不满秩，因为行与行之间等差

d = np.array([ [2, 1], [1, 2]])
'''
对正定矩阵求本征值和本征向量
本征值为 u，array([ 3.,  1.])
本征向量构成的二维 array 为 v，
array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
是沿着 45° 方向
eig()是一般情况的本征值分解，对于更常见的对称实数矩阵，
eigh()更快且更稳定，不过输出的值的顺序和 eig()是相反的
'''
u, v = np.linalg.eig(d)
print(u,v)                         # [ 3.  1.] [[ 0.70710678 -0.70710678]
# Cholesky 分解并重建
l = np.linalg.cholesky(d)
print(l)                           #  [ 0.70710678  0.70710678]]
print(np.dot(l, l.T))              # [[ 1.41421356  0.] [ 0.70710678  1.22474487]]
                                   # [[ 2.  1.] [ 1.  2.]]
e = np.array([ [1, 2], [3, 4]])

#  对不镇定矩阵，进行 SVD 分解并重建
U, s, V = np.linalg.svd(e)
print(U,s,V)                       #[[-0.40455358 -0.9145143 ] [-0.9145143   0.40455358]]
                                   # [ 5.4649857   0.36596619]
                                   # [[-0.57604844 -0.81741556] [ 0.81741556 -0.57604844]]
S = np.array([ [s[0], 0], [0, s[1]]])
print(np.dot(U, np.dot(S, V)))     # [[ 1.  2.] [ 3.  4.]]
```

**建议**：深入学习可浏览官方文档 http://docs.scipy.org/doc/

# 3 Pandas 库使用

Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。

# 3.1 基础知识

### 1）Pandas 的主要功能

1. 具备对齐功能的数据结构 DataFrame、Series
2. 集成时间序列功能
3. 提供丰富的数学运算和操作
4. 灵活处理缺失数据

### 2）数据类型

Pandas 基于两种数据类型：series 与 dataframe。

一个 series 是一个一维的数据类型，其中每一个元素都有一个标签。如果你阅读过这个系列的关于 Numpy 的文章，你就可以发现 series 类似于 Numpy 中元素带标签的数组。其中，标签可以是数字或者字符串。

s = pd.Series([1,3,5,np.nan,6,8])

一个 dataframe 是一个二维的表结构。Pandas 的 dataframe 可以存储许多种不同的数据类型，并且每一个坐标轴都有自己的标签。你可以把它想象成一个 series 的字典项。

```
# 通过字典和 Series 对象创建: 取 Series 对象中标签的并集作为索引，缺失值为 NaN
pd.DataFrame({'one':pd.Series([1,2,3],index=['a','b','c']),
'two':pd.Series([1,2,3,4],index=['b','a','c','d'])})
```

DataFrame 的类似与数据库中的表，一般很少手动创建 DataFrame，通常是从文件中读取，后续例程有相关展示。

# 3.2 数据处理例程

### 1) 生成数据表

```
#常见数据处理程序
import numpy as np
import pandas as pd
#导入 CSV 或者 xlsx 文件
df = pd.DataFrame(pd.read_csv('name.csv',header=1))
df = pd.DataFrame(pd.read_excel('name.xlsx'))
#用 pandas 创建数据表
df = pd.DataFrame({"id":[1001,1002,1003,1004,1005,1006],
  "date":pd.date_range('20130102', periods=6),
    "city":['Beijing ', 'SH', ' guangzhou ', 'Shenzhen', 'shanghai', 'BEIJING '],
  "age":[23,44,54,32,34,32],
  "category":['100-A','100-B','110-A','110-C','210-A','130-F'],
    "price":[1200,np.nan,2133,5433,np.nan,4432]},
    columns =['id','date','city','category','age','price'])
```

## 2）数据表信息查看

```
#维度查看
df.shape
#数据表基本信息（维度、列名称、数据格式、所占空间等）
df.info()
#每一列数据的格式
df.dtypes
#某一列格式
df['B'].dtype
#空值
df.isnull()
#查看某一列空值
df.isnull()
#查看某一列的唯一值
df['B'].unique()
#查看数据表的值
df.values
#查看列名称
df.columns
#查看前 10 行数据、后 10 行数据
df.head()        #默认前 10 行数据
df.tail()        #默认后 10  行数据
```

## 3）数据表清洗

```
#用数字 0 填充空值
df.fillna(value=0)
#使用列 prince 的均值对 NA 进行填充
df['prince'].fillna(df['prince'].mean())
#清楚 city 字段的字符空格
df['city']=df['city'].map(str.strip)
#大小写转换
df['city']=df['city'].str.lower()
#更改数据格式
df['price'].astype('int')
#更改列名称
df.rename(columns={'category': 'category-size'})
#删除后出现的重复值
df['city'].drop_duplicates()
#删除先出现的重复值
df['city'].drop_duplicates(keep='last')
#数据替换
df['city'].replace('sh', 'shanghai')
```

### 4）数据预处理

```
#数据类型示例程序 1—简单数据类型
df1=pd.DataFrame({"id":[1001,1002,1003,1004,1005,1006,1007,1008],
"gender":['male','female','male','female','male','female','male','female'],
"pay":['Y','N','Y','Y','N','Y','N','Y',],
"m-point":[10,12,20,40,40,40,30,20]})
#数据表合并
df_inner=pd.merge(df,df1,how='inner')  # 匹配合并，交集
df_left=pd.merge(df,df1,how='left')          #
df_right=pd.merge(df,df1,how='right')
df_outer=pd.merge(df,df1,how='outer')  #并集
#设置索引列
df_inner.set_index('id')
#按照特定列的值排序：
df_inner.sort_values(by=['age'])
#按照索引列排序：
df_inner.sort_index()
#如果 prince 列的值>3000，group 列显示 high，否则显示 low：
df_inner['group'] = np.where(df_inner['price'] > 3000,'high','low')
#对复合多个条件的数据进行分组标记
df_inner.loc[(df_inner['city'] == 'beijing') & (df_inner['price'] >= 4000), 'sign']=1
#对 category 字段的值依次进行分列，并创建数据表，索引值为 df_inner 的索引列，列名称为 category 和 size
pd.DataFrame((x.split('-')
   for x in df_inner['category']),index=df_inner.index,columns=['category','size']))
#将完成分裂后的数据表和原 df_inner 数据表进行匹配
df_inner=pd.merge(df_inner,split,right_index=True, left_index=True)
```

### 5）数据提取

主要用到的三个函数：loc,iloc 和 ix，loc 函数按标签值进行提取，iloc 按位置进行提取，ix 可以同时按标签和位置进行提取。

```
#按索引提取单行的数值
df_inner.loc[3]
#按索引提取区域行数值
df_inner.iloc[0:5]
#重设索引
df_inner.reset_index()
#设置日期为索引
df_inner=df_inner.set_index('date')
#提取 4 日之前的所有数据
df_inner[:'2013-01-04']
#使用 iloc 按位置区域提取数据
df_inner.iloc[:3,:2]
```

```
#冒号前后的数字不再是索引的标签名称，而是数据所在的位置，从 0 开始，三行两列。
#适应 iloc 按位置单独提起数据
df_inner.iloc[[0,2,5],[4,5]] #提取第 0、2、5 行，4、5 列
#使用 ix 按索引标签和位置混合提取数据
df_inner.ix[:'2013-01-03',:4] #2013-01-03 号之前，前四列数据
#判断 city 列的值是否为北京
df_inner['city'].isin(['beijing'])
#判断 city 列里是否包含 beijing 和 shanghai，然后将符合条件的数据提取出来
df_inner.loc[df_inner['city'].isin(['beijing','shanghai'])]
#提取前三个字符，并生成数据表
pd.DataFrame(category.str[:3])
```

### 6）数据筛选

使用与、或、非三个条件配合大于、小于、等于对数据进行筛选，并进行计数和求和。

```
#使用“与”进行筛选
df_inner.loc[(df_inner['age'] > 25) & (df_inner['city'] == 'beijing'), ['id','city','age','category','gender']]
#使用“或”进行筛选
df_inner.loc[(df_inner['age'] > 25) | (df_inner['city'] == 'beijing'), ['id','city','age','category','gender']].sort(['age'])
#使用“非”条件进行筛选
df_inner.loc[(df_inner['city'] != 'beijing'), ['id','city','age','category','gender']].sort(['id'])
#对筛选后的数据按 city 列进行计数
df_inner.loc[(df_inner['city'] != 'beijing'), ['id','city','age','category','gender']].sort(['id']).city.count()
#使用 query 函数进行筛选
df_inner.query('city == ["beijing", "shanghai"]')
#对筛选后的结果按 prince 进行求和
df_inner.query('city == ["beijing", "shanghai"]').price.sum()
```

### 7）数据汇总

主要函数是 groupby 和 pivote_table

```
#对所有的列进行计数汇总
df_inner.groupby('city').count()
#按城市对 id 字段进行计数
df_inner.groupby('city')['id'].count()
#对两个字段进行汇总计数
df_inner.groupby(['city','size'])['id'].count()
#对 city 字段进行汇总，并分别计算 prince 的合计和均值
df_inner.groupby('city')['price'].agg([len,np.sum, np.mean])
```

### 8）数据统计

数据采样，计算标准差，协方差和相关系数

```
#简单的数据采样
```

```
df_inner.sample(n=3)
#手动设置采样权重
weights = [0, 0, 0, 0, 0.5, 0.5]
df_inner.sample(n=2, weights=weights)
#采样后不放回
df_inner.sample(n=6, replace=False)
#采样后放回
df_inner.sample(n=6, replace=True)
#数据表描述性统计
df_inner.describe().round(2).T #round 函数设置显示小数位，T 表示转置
#计算列的标准差
df_inner['price'].std()
#计算两个字段间的协方差
df_inner['price'].cov(df_inner['m-point'])
#数据表中所有字段间的协方差
df_inner.cov()
#两个字段的相关性分析
df_inner['price'].corr(df_inner['m-point'])
#相关系数在-1 到 1 之间，接近 1 为正相关，接近-1 为负相关，0 为不相关
#数据表的相关性分析
df_inner.corr()
```

### 9）数据输出

分析后的数据可以输出为 xlsx 格式和 csv 格式

```
#写入 Excel
df_inner.to_excel('excel_to_python.xlsx', sheet_name='bluewhale_cc')
#写入到 CSV
df_inner.to_csv('excel_to_python.csv')
```

# 4 Scipy 库使用

Scipy 是一个高级的科学计算库，它和 Numpy 联系很密切，Scipy 一般都是操控 Numpy 数组来进行科学计算，所以可以说是基于 Numpy 之上了。Scipy 有很多子模块可以应对不同的应用，例如插值运算，优化算法、图像处理、数学统计等。

| 模块名 | 功能 |
| --- | --- |
| scipy.cluster | 向量量化 |
| scipy.constants | 数学常量 |

| | |
|---|---|
| scipy.fftpack | 快速傅里叶变换 |
| scipy.integrate | 积分 |
| scipy.interpolate | 插值 |
| scipy.io | 数据输入输出 |
| scipy.linalg | 线性代数 |
| scipy.ndimage | N 维图像 |
| scipy.odr | 正交距离回归 |
| scipy.optimize | 优化算法 |
| scipy.signal | 信号处理 |
| scipy.sparse | 稀疏矩阵 |
| scipy.spatial | 空间数据结构和算法 |
| scipy.special | 特殊数学函数 |
| scipy.stats | 统计函数 |

SciPy 函数库在 NumPy 库的基础上增加了众多的数学、科学以及工程计算中常用的库函数。例如线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等等。

# 4.1 快速使用

Scipy 是一个高级的科学计算库，它和 Numpy 联系很密切，Scipy 一般都是操控 Numpy 数组来进行科学计算，所以可以说是基于 Numpy 之上了。Scipy 有很多子模块可以应对不同的应用，例如插值运算，优化算法、图像处理、数学统计等。

以下列出 Scipy 的子模块：

### 1）组成模块

### 2）文件输入和输出：scipy.io

这个模块可以加载和保存 matlab 文件：

```
from scipy import io as spio
a = np.ones((3, 3))
spio.savemat('file.mat', {'a': a}) #  保存字典到 file.mat
data = spio.loadmat('file.mat', struct_as_record=True)
print(data['a'])
```

打印结果为：

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

.关于这个模块的文档：https://docs.scipy.org/doc/scipy/reference/io.html#module-scipy.io

### 3）线性代数操作：scipy.linalg

假如我们要计算一个方阵的行列式，我们需要调用 det()函数：

```
from scipy import linalg
arr = np.array([[1, 2], [3, 4]])
linalg.det(arr)
arr = np.array([[3, 2], [6, 4]])
linalg.det(arr)
```

比如求一个矩阵的转置：

```
arr = np.array([[1, 2], [3, 4]])
iarr = linalg.inv(arr)
print(iarr)
```

结果为：array([[-2. ,  1. ],[ 1.5, -0.5]])

### 4）快速傅里叶变换：scipy.fftpack

首先我们用 numpy 初始化正弦信号：

```
import numpy as np
time_step = 0.02
period = 5.
time_vec = np.arange(0, 20, time_step)
sig = np.sin(2 * np.pi / period * time_vec) + \0.5 * np.random.randn(time_vec.size)
```

如果我们要计算该信号的采样频率，可以用 scipy.fftpack.fftfreq()函数，计算它的快速傅里叶变换使用 scipy.fftpack.fft():

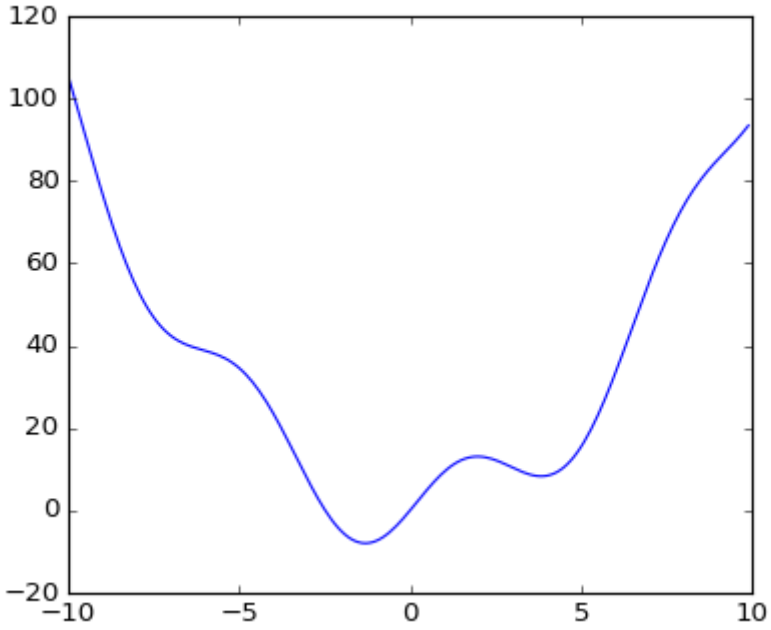
```
from scipy import fftpack
sample_freq = fftpack.fftfreq(sig.size, d=time_step)
sig_fft = fftpack.fft(sig)
```

Numpy 中也有用于计算快速傅里叶变换的模块：numpy.fft
但是 scipy.fftpack 是我们的首选，因为应用了更多底层的工具，工作效率要高一些。

### 5）优化器：scipy.optimize

scipy.optimize 通常用来最小化一个函数值，我们举个栗子：
构建一个函数并绘制函数图：

```
def f(x):
    return x**2 + 10*np.sin(x)
x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
plt.show()
```




如果我们要找出这个函数的最小值，也就是曲线的最低点。就可以用到 BFGS 优化算法(Broyden–Fletcher–Goldfarb–Shanno algorithm)：

```
optimize.fmin_bfgs(f, 0)
Optimization terminated successfully.
        Current function value: -7.945823
        Iterations: 5
        Function evaluations: 24
        Gradient evaluations: 8
array([-1.30644003])
```

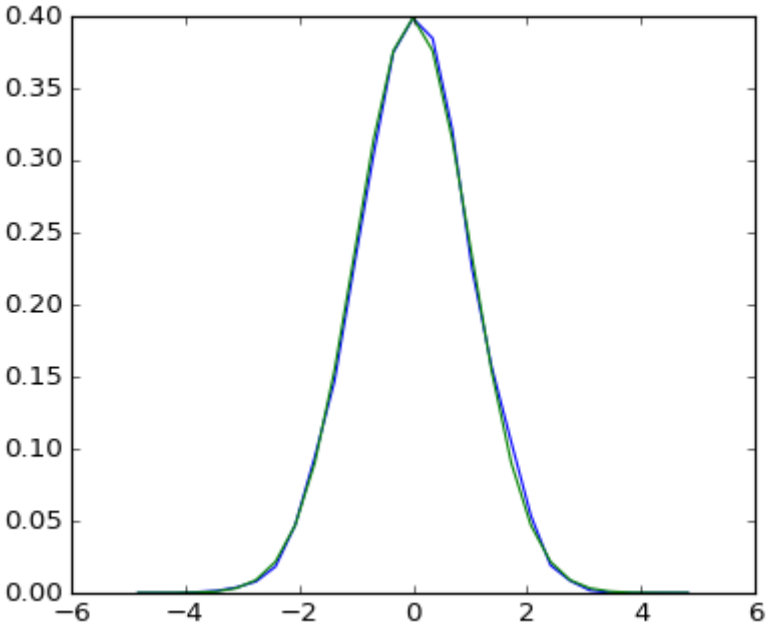
可以得到最低点的值为-1.30644003，optimize.fmin_bfgs(f, 0)第二个参数 0 表示从 0 点的位置最小化，找到最低点（该点刚好为全局最低点）。假如我从 3 点的位置开始梯度下降，那么得到的将会是局部最低点 3.83746663：

```
>>> optimize.fmin_bfgs(f, 3, disp=0)
array([ 3.83746663])
```

假如你无法选出 the global minimum 的邻近点作为初始点的话可以使用 scipy.optimize.basinhopping()，具体就不展开描述。关于这个模块的其他功能，参考 scipy.optimize 。

### 6）统计工具：scipy.stats

首先我们随机生成 1000 个服从正态分布的数：




```
a = np.random.normal(size=1000)
#用 stats 模块计算该分布的均值和标准差。
loc, std = stats.norm.fit(a)
print(loc)
print(std)
#中位数
print(np.median(a))
```

# 4.2 最小二乘拟合

假设有一组实验数据(x[i], y[i])，我们知道它们之间的函数关系:y = f(x)，通过这些已知信息，需要确定函数中的一些参数项。例如，如果 f 是一个线型函数 f(x) = k*x+b，那么参数 k 和 b 就是我们需要确定的值。如果将这些参数用 p 表示的话，那么我们就是要找到一组 p 值使得如下公式中的 S 函数最小：

$$S(\mathbf{p}) = \sum_{i=1}^{m} [y_i - f(x_i, \mathbf{p})]^2$$

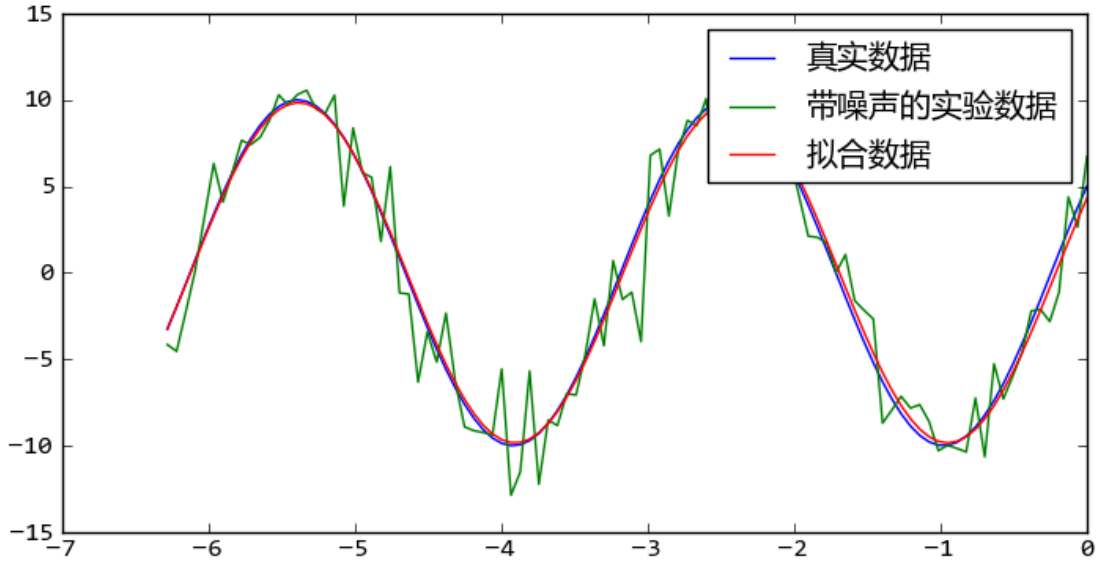
这种算法被称之为最小二乘拟合(Least-square fitting)。

scipy 中的子函数库 optimize 已经提供了实现最小二乘拟合算法的函数 leastsq。下面是用 leastsq 进行数据拟合的一个例子：

```
# -*- coding: utf-8 -*-
import numpy as np
from scipy.optimize import leastsq
```

```
import pylab as pl

def func(x, p):
    """
    数据拟合所用的函数: A*sin(2*pi*k*x + theta)
    """
    A, k, theta = p
    return A*np.sin(2*np.pi*k*x+theta)

def residuals(p, y, x):
    """
    实验数据 x, y 和拟合函数之间的差，p 为拟合需要找到的系数
    """
    return y - func(x, p)

x = np.linspace(0, -2*np.pi, 100)
A, k, theta = 10, 0.34, np.pi/6 # 真实数据的函数参数
y0 = func(x, [A, k, theta]) # 真实数据
y1 = y0 + 2 * np.random.randn(len(x)) # 加入噪声之后的实验数据

p0 = [7, 0.2, 0] # 第一次猜测的函数拟合参数

# 调用 leastsq 进行数据拟合
# residuals 为计算误差的函数
# p0 为拟合参数的初始值
# args 为需要拟合的实验数据
plsq = leastsq(residuals, p0, args=(y1, x))

print u"真实参数:", [A, k, theta]
print u"拟合参数", plsq[0] # 实验数据拟合后的参数

pl.plot(x, y0, label=u"真实数据")
pl.plot(x, y1, label=u"带噪声的实验数据")
pl.plot(x, func(x, plsq[0]), label=u"拟合数据")
pl.legend()
pl.show()
```

这个例子中我们要拟合的函数是一个正弦波函数，它有三个参数 **A, k, theta** ，分别对应振幅、频率、相角。假设我们的实验数据是一组包含噪声的数据 x, y1，其中 y1 是在真实数据 y0 的基础上加入噪声的到了。




通过 leastsq 函数对带噪声的实验数据 x, y1 进行数据拟合，可以找到 x 和真实数据 y0 之间的正弦关系的三个参数： A, k, theta。下面是程序的输出：

**真实参数**[10, 0.34000000000000002, 0.52359877559829882]

**拟合参数**[-9.84152775 0.33829767 -2.68899335]

调用 leastsq 函数对噪声正弦波数据进行曲线拟合

我们看到拟合参数虽然和真实参数完全不同，但是由于正弦函数具有周期性，实际上拟合参数得到的函数和真实参数对应的函数是一致的。

# 4.2 函数最小值

optimize 库提供了几个求函数最小值的算法：fmin, fmin_powell, fmin_cg, fmin_bfgs。下面的程序通过求解卷积的逆运算演示 fmin 的功能。

对于一个离散的线性时不变系统 h, 如果它的输入是 x，那么其输出 y 可以用 x 和 h 的卷积表示：

y=x*h

现在的问题是如果已知系统的输入 x 和输出 y，如何计算系统的传递函数 h；或者如果已知系统的传递函数 h 和系统的输出 y，如何计算系统的输入 x。这种运算被称为反卷积运算，是十分困难的，特别是在实际的运用中，测量系统的输出总是存在误差的。

下面用 fmin 计算反卷积，这种方法只能用在很小规模的数列之上，因此没有很大的实用价值，不过用来评价 fmin 函数的性能还是不错的。

```
# -*- coding: utf-8 -*-
# 本程序用各种 fmin 函数求卷积的逆运算

import scipy.optimize as opt
import numpy as np

def test_fmin_convolve(fminfunc, x, h, y, yn, x0):
    """
```

```
    x (*) h = y, (*)表示卷积
    yn 为在 y 的基础上添加一些干扰噪声的结果
    x0 为求解 x 的初始值
    """
    def convolve_func(h):
        """
        计算 yn - x (*) h 的 power
        fmin 将通过计算使得此 power 最小
        """
        return np.sum((yn - np.convolve(x, h))**2)

    # 调用 fmin 函数，以 x0 为初始值
    h0 = fminfunc(convolve_func, x0)

    print fminfunc.__name__
    print "---------------------"
    # 输出 x (*) h0 和 y 之间的相对误差
    print "error of y:", np.sum((np.convolve(x, h0)-y)**2)/np.sum(y**2)
    # 输出 h0 和 h 之间的相对误差
    print "error of h:", np.sum((h0-h)**2)/np.sum(h**2)
    print

def test_n(m, n, nscale):
    """
    随机产生 x, h, y, yn, x0 等数列，调用各种 fmin 函数求解 b
    m 为 x 的长度, n 为 h 的长度, nscale 为干扰的强度
    """
    x = np.random.rand(m)
    h = np.random.rand(n)
    y = np.convolve(x, h)
    yn = y + np.random.rand(len(y)) * nscale
    x0 = np.random.rand(n)

    test_fmin_convolve(opt.fmin, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_powell, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_cg, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_bfgs, x, h, y, yn, x0)

if __name__ == "__main__":
    test_n(200, 20, 0.1)
```

下面是程序的输出：

```
fmin
---------------------
error of y: 0.00568756699607
```

```
error of h: 0.354083287918

fmin_powell
----------------
error of y: 0.000116114709857
error of h: 0.000258897894009

fmin_cg
----------------
error of y: 0.000111220299615
error of h: 0.000211404733439

fmin_bfgs
----------------
error of y: 0.000111220251551
error of h: 0.000211405138529
```

# 4.3 非线性方程求解

optimize 库中的 fsolve 函数可以用来对非线性方程组进行求解。它的基本调用形式如下：

```
fsolve(func, x0)
```

func(x)是计算方程组误差的函数，它的参数 x 是一个矢量，表示方程组的各个未知数的一组可能解，func 返回将 x 代入方程组之后得到的误差；x0 为未知数矢量的初始值。如果要对如下方程组进行求解的话：

- f1(u1,u2,u3) = 0
- f2(u1,u2,u3) = 0
- f3(u1,u2,u3) = 0

那么 func 可以如下定义：

```
def func(x):
    u1,u2,u3 = x
    return [f1(u1,u2,u3), f2(u1,u2,u3), f3(u1,u2,u3)]
```

下面是一个实际的例子，求解如下方程组的解：

- 5*x1 + 3 = 0
- 4*x0*x0 - 2*sin(x1*x2) = 0
- x1*x2 - 1.5 = 0

程序如下：

```
from scipy.optimize import fsolve
from math import sin,cos

def f(x):
    x0 = float(x[0])
    x1 = float(x[1])
```

```
    x2 = float(x[2])
    return [
        5*x1+3,
        4*x0*x0 - 2*sin(x1*x2),
        x1*x2 - 1.5
    ]

result = fsolve(f, [1,1,1])

print result
print f(result)
```

输出为：

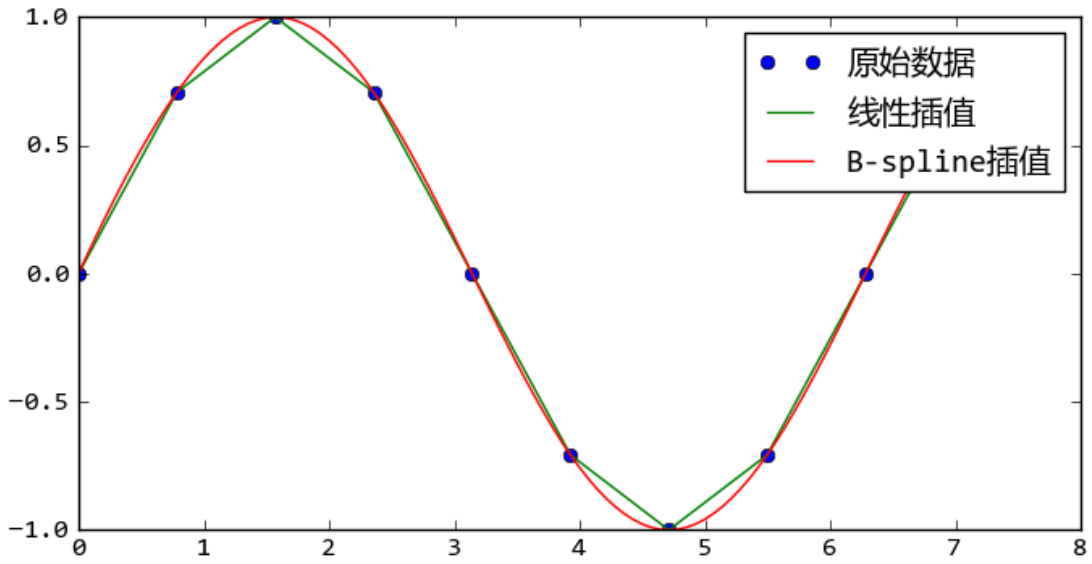
```
[-0.70622057 -0.6         -2.5        ]
[0.0, -9.1260332624187868e-14, 5.3290705182007514e-15]
```

由于 fsolve 函数在调用函数 f 时，传递的参数为数组，因此如果直接使用数组中的元素计算的话，计算速度将会有所降低，因此这里先用 float 函数将数组中的元素转换为 Python 中的标准浮点数，然后调用标准 math 库中的函数进行运算。

# 4.4 B-Spline 样条曲线

interpolate 库提供了许多对数据进行插值运算的函数。下面是使用直线和 B-Spline 对正弦波上的点进行插值的例子。

```
# -*- coding: utf-8 -*-
import numpy as np
import pylab as pl
from scipy import interpolate

x = np.linspace(0, 2*np.pi+np.pi/4, 10)
y = np.sin(x)

x_new = np.linspace(0, 2*np.pi+np.pi/4, 100)
f_linear = interpolate.interp1d(x, y)
tck = interpolate.splrep(x, y)
y_bspline = interpolate.splev(x_new, tck)

pl.plot(x, y, "o",  label=u"原始数据")
pl.plot(x_new, f_linear(x_new), label=u"线性插值")
pl.plot(x_new, y_bspline, label=u"B-spline 插值")
pl.legend()
pl.show()
```




在这段程序中，通过 interp1d 函数直接得到一个新的线性插值函数。而 B-Spline 插值运算需要先使用 splrep 函数计算出 B-Spline 曲线的参数，然后将参数传递给 splev 函数计算出各个取样点的插值结果。

# 5 Matplot 库使用

matplotlib 是 python 最著名的绘图库，它提供了一整套和 matlab 相似的命令 API，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入 GUI 应用程序中。

它的文档相当完备，并且 Gallery 页面 中有上百幅缩略图，打开之后都有源程序。因此如果你需要绘制某种类型的图，只需要在这个页面中浏览/复制/粘贴一下，基本上都能搞定。

本章节作为 matplotlib 的入门介绍，将较为深入地挖掘几个例子，从中理解和学习 matplotlib 绘图的一些基本概念。

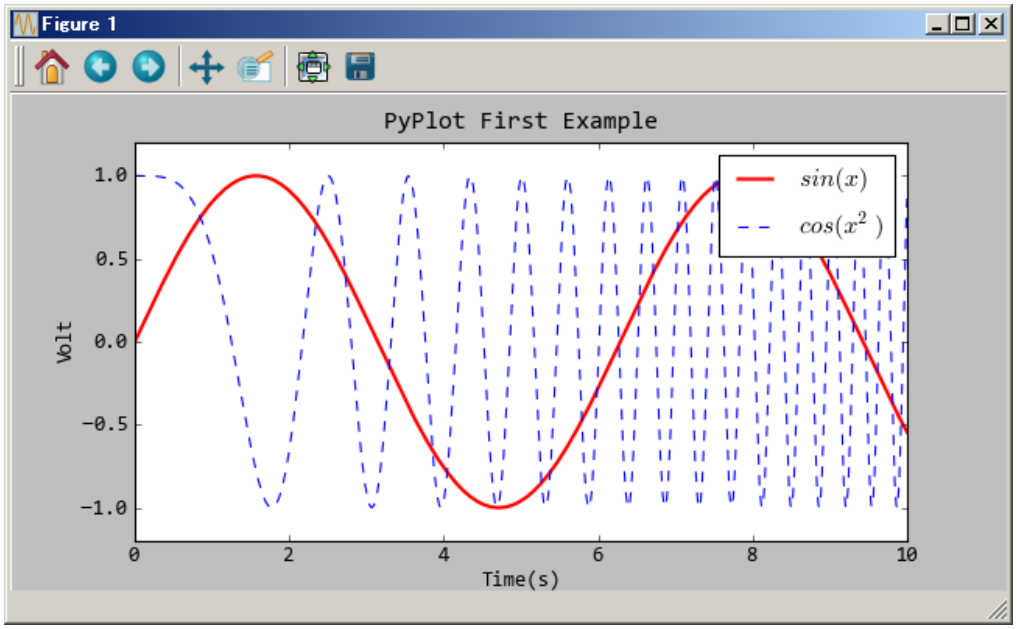
## 5.1 快速绘图

matplotlib 的 pyplot 子库提供了和 matlab 类似的绘图 API，方便用户快速绘制 2D 图表。

### 1）示例程序

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 1000)
y = np.sin(x)
z = np.cos(x**2)

plt.figure(figsize=(8,4))
```

```
plt.plot(x,y,label="$sin(x)$",color="red",linewidth=2)
plt.plot(x,z,"b--",label="$cos(x^2)$")
plt.xlabel("Time(s)")
plt.ylabel("Volt")
plt.title("PyPlot First Example")
plt.ylim(-1.2,1.2)
plt.legend()
plt.show()
```




**2）基本属性**

matplotlib 还提供了名为 pylab 的模块，其中包括了许多 numpy 和 pyplot 中常用的函数，方便用户快速进行计算和绘图。基本的属性有：

- **label**：给所绘制的曲线一个名字，此名字在图示(legend)中显示。只要在字符串前后添加"$"符号，matplotlib 就会使用其内嵌的 latex 引擎绘制的数学公式。
- **color**：指定曲线的颜色
- **linewidth**：指定曲线的宽度
- **xlabel**：设置 X 轴的文字
- **ylabel**：设置 Y 轴的文字
- **title**：设置图表的标题
- **ylim**：设置 Y 轴的范围

# 5.2 绘制多轴图

一个绘图对象(figure)可以包含多个轴(axis)，在 Matplotlib 中用轴表示一个绘图区域，可以将其理解为子图。上面的第一个例子中，绘图对象只包括一个轴，因此只显示了一个轴(子图)。我们可以使用 subplot 函数快速绘制有多个轴的图表。subplot 函数的调用形式如下：

```
subplot(numRows, numCols, plotNum)
```

subplot 将整个绘图区域等分为 numRows 行 * numCols 列个子区域，然后按照从左到右，从上到下的顺序对每个子区域进行编号，左上的子区域的编号为 1。如果 numRows，numCols 和 plotNum 这三个数都小于 10 的话，可以把它们缩写为一个整数，例如 subplot(323)和 subplot(3,2,3)是相同的。subplot 在 plotNum 指定的区域中创建一个轴对象。如果新创建的轴和之前创建的轴重叠的话，之前的轴将被删除。

下面的程序创建 3 行 2 列共 6 个轴，通过 axisbg 参数给每个轴设置不同的背景颜色。

```
for idx, color in enumerate("rgbyck"):
    plt.subplot(320+idx+1, axisbg=color)
plt.show()
```

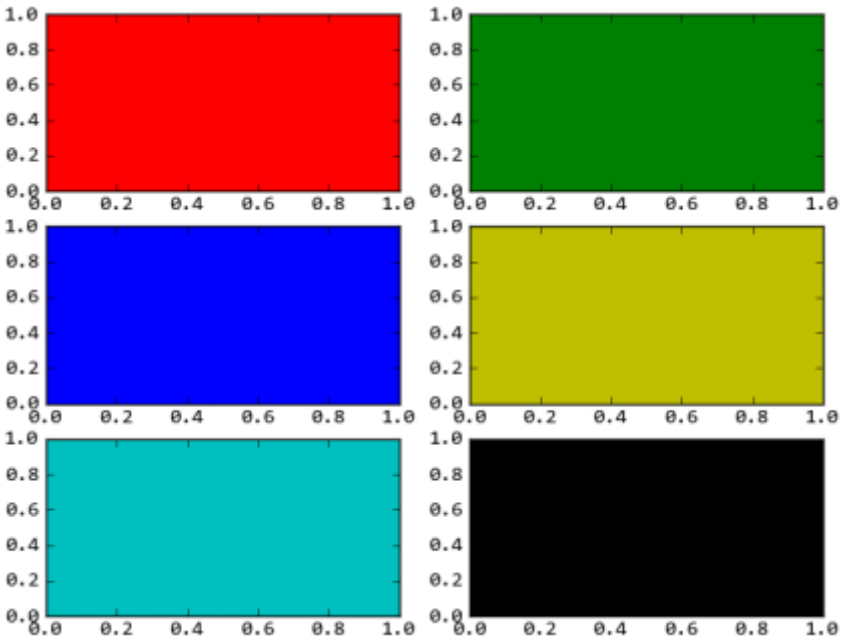


**图 5.2** 用 subplot 函数将 Figure 分为六个子图区域

如果希望某个轴占据整个行或者列的话，可以如下调用 subplot：

```
plt.subplot(221) # 第一行的左图
plt.subplot(222) # 第一行的右图
plt.subplot(212) # 第二整行
plt.show()
```

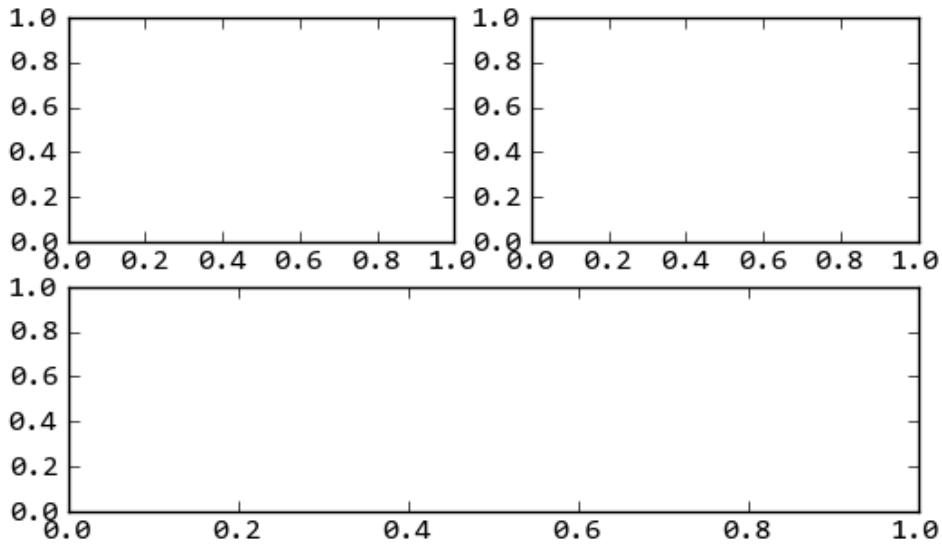



**图 5.3** 将 Figure 分为三个子图区域

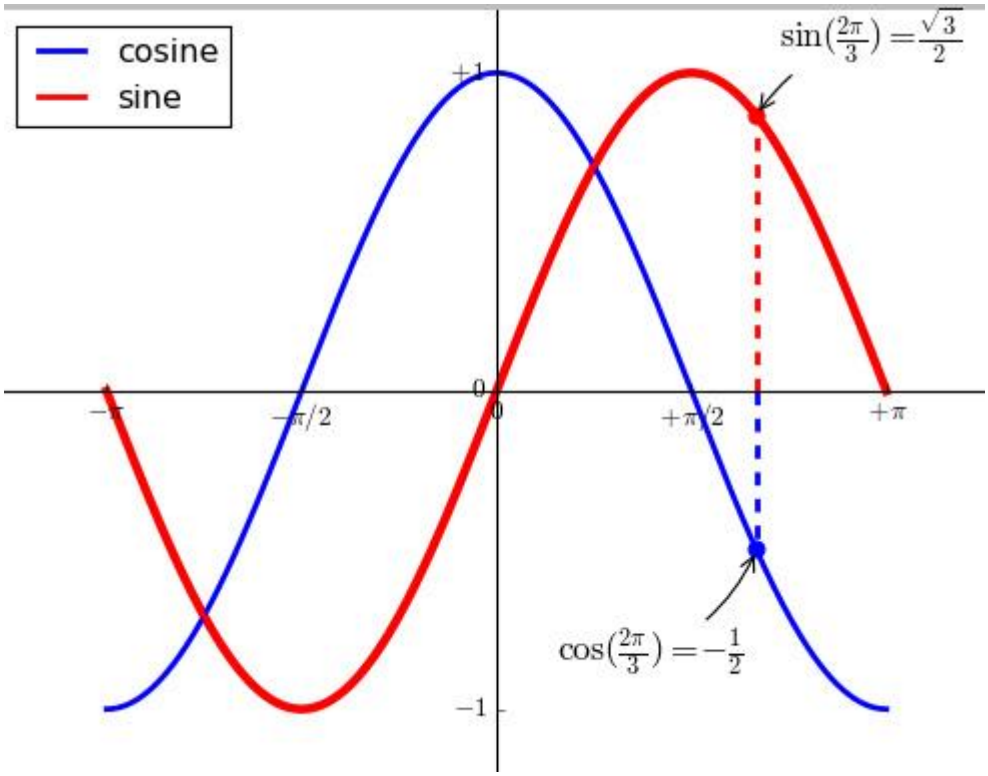
当绘图对象中有多个轴的时候，可以通过工具栏中的 Configure Subplots 按钮，交互式地调节轴之间的间距和轴与边框之间的距离。如果希望在程序中调节的话，可以调用 subplots_adjust 函数，它有 left, right, bottom, top, wspace, hspace 等几个关键字参数，这些参数的值都是 0 到 1 之间的小数，它们是以绘图区域的宽高为 1 进行正规化之后的坐标或者长度。

# 5.3 绘制复杂图形

### 1）带注释图

```
import numpy as np
import matplotlib.pyplot as plt
t = 2*np.pi/3
# 作一条垂直于 x 轴的线段，由数学知识可知，横坐标一致的两个点就在垂直于坐标轴的直线上了。这两个点是起始点。
plot([t,t],[0,np.cos(t)], color ='blue', linewidth=2.5, linestyle="--")
scatter([t,],[np.cos(t),], 50, color ='blue')

annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
         xy=(t, np.sin(t)), xycoords='data',
         xytext=(+10, +30), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plot([t,t],[0,np.sin(t)], color ='red', linewidth=2.5, linestyle="--")
scatter([t,],[np.sin(t),], 50, color ='red')
```

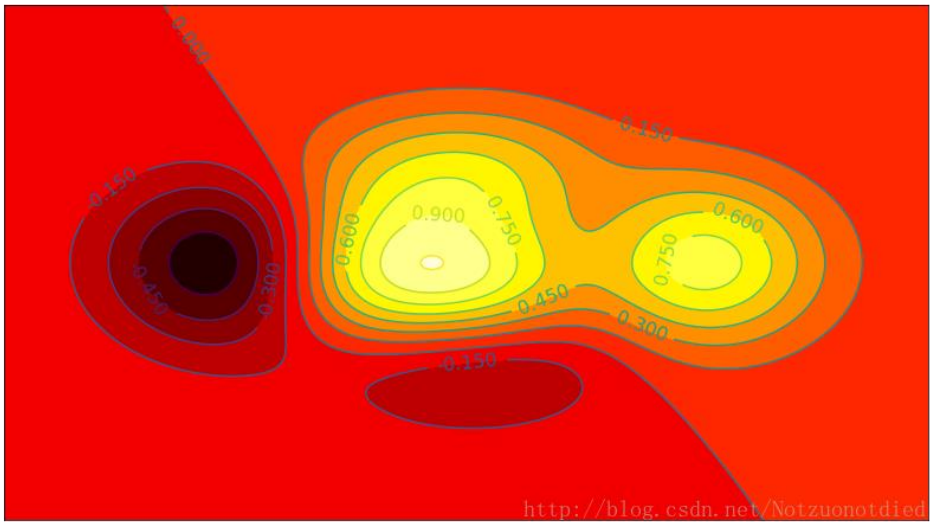
```
annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
         xy=(t, np.cos(t)), xycoords='data',
         xytext=(-90, -50), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```




**2）等高图**

```
import matplotlib.pyplot as plt
import numpy as np

def get_height(x, y):
    # the height function
    return (1-x/2+x**5+y**3)*np.exp(-x**2-y**2)
n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X, Y = np.meshgrid(x, y)

plt.figure(figsize=(14, 8))
# use plt.contourf to filling contours
# X, Y and value for (X, Y) point

# 横坐标、纵坐标、高度、 、透明度、cmap 是颜色对应表
# 等高线的填充颜色
plt.contourf(X, Y, get_height(X, Y), 16, alpah=0.7, cmap=plt.cm.hot)
# use plt.contour to add contour lines
```

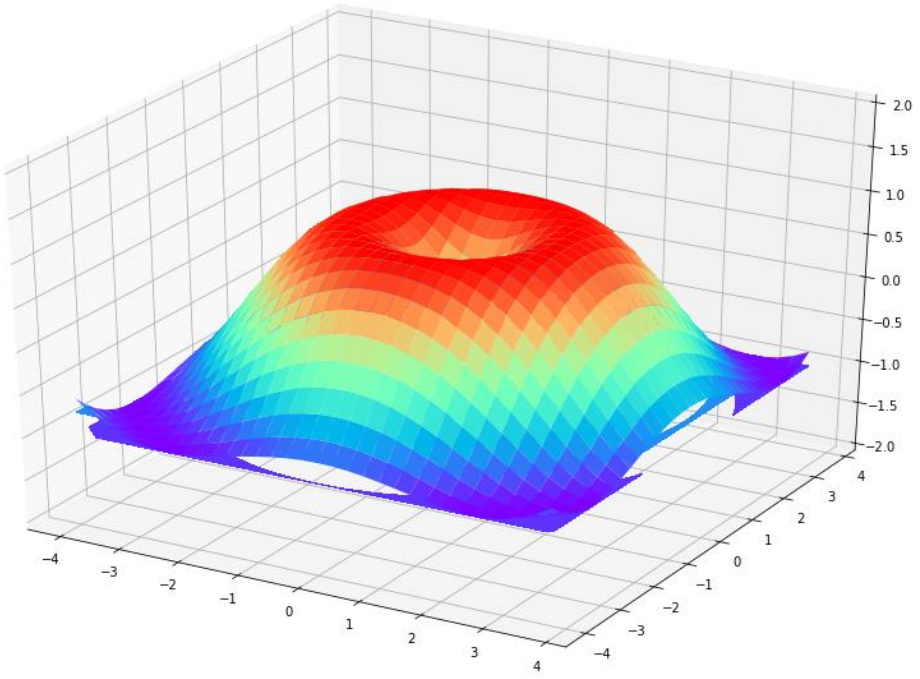
```
# 这里是等高线的线
C = plt.contour(X, Y, get_height(X, Y), 16, color='black', linewidth=.5)

# adding label
plt.clabel(C, inline=True, fontsize=16)

plt.xticks(())
plt.yticks(())
plt.show()
```




**3）3D 数据图**

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 8))
ax = Axes3D(fig)

# 生成 X，Y
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X,Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)

# height value
Z = np.sin(R)

# 绘图
# rstride（row）和 cstride(column)表示的是行列的跨度
ax.plot_surface(X, Y, Z,
```

```
                rstride=1,  # 行的跨度
                cstride=1,  # 列的跨度
                cmap=plt.get_cmap('rainbow')  # 颜色映射样式设置
            )

# offset 表示距离 zdir 的轴距离
ax.contourf(X, Y, Z, zdir='z', offest=-2, cmap='rainbow')
ax.set_zlim(-2, 2)

plt.show()
```




## 5.4 程序练习

1）可视化展示文件中各单词（区分大小写）的出现频率等信息
2）实时股价：可以查询股票当前价格。用户可以设定数据刷新频率，用色彩箭头、折线表示股价走势。

# 6 参考资料


1 Numpy 官方文档：http://www.numpy.org/
2 Pandas 官方文档：http://pandas.pydata.org/pandas-docs/stable/
3 matplotlib 官方文档：https://matplotlib.org/users/index.html
4 Scipy 官方文档 https://docs.scipy.org/doc/scipy/reference/index.html