numpy

```
from numpy import *
a = array([[1,2],[3,4],[5,6]]) #create a standard array, shape (3, 2)
v1 = array([1,2,3]) #create a standard vector
                    #shape (3,), v1.T = v1,
                    #可用matrix将向量映射为矩阵, v2 = matrix(v1), v1 = flatten(v2)
                    #也可用newaxis, v2=v1[newaxis,:], v3=v1[:,nweaxis]
v2 = array([[1,2,3]]) #create a row matrix, shape (3,1), v2.T = v3
v3 = array([[1],[2],[3]]) #create a column matrix, shape (1,3)
```

integer array indexing (fancy indexing)

a[row_index_vector,column_index_vector]

```
a[[0,0],[1,1]] #An example of integer array indexing. Prints "[2 2]"
```

[0,0]确定了引用a的行列表，[1,1]确定了列列表。表示第一个元素引用a的Row 0,Column 1，第二个元素同样引用a的Row 0,Column 1，值为2，等价于：

```
array([a[0,1],a[0,1])
a[2,1]   #基本的方框号引用，Print "6"
a[[2,1]] #缺省列列表的整数索引，引用整行
                    #Prints "array([[5,6],[3,4]])"
                    #也可写成 a.take([2,1])
```

再看一例：

```
a[[0,1,2],[0,1,0]] #Prints "[1,4,5]", it is equivalent to this:
array([a[0,0],a[1,1],a[2,0]])
```

Boolean array indexing.

```
(a>2).any()  #if there is an element more than 2, prints "True"
(a>2).all()  #if all of the elements are more than 2,prints "True"
a > 2        #An example of Boolean array indexing.
             #Prints "array([False,False],[True,True,],[True,True])"
```

slice indexing

格式为：M[lower:upper:step]

```
b = array([1,2,3,4,5])
b[1:3]  #default step is 1,prints "array([2,3])"
b[::2]  #lower and upper default to be beginning and end of the array
                  #Prints "array([1,3,5])"
b[3:]   #Prints "array([4,5])"
b[-1]   #负数索引从数组尾开始计算，第一个数为-1， Prints "5"
b[-3,:] #the last three elements,Prints "array([3,4,5])"
```

多维数组中也是一样的，如

```
a[::,:1] #Prints "array[[1],[3],[5]])"
```

Numpy also provides many functions to create arrays:

```
zeros((2,2)) #create an array of all zeros
ones((2,2)) #create an array of all ones
full((2,2),7) #create a constant array
eye(2) #create a 2X2 identity matrix
diag([1,2,3]) #create a diagonal matrix
              #but diag(M) also get diagonal elements of matrix M
diag([1,2,3],1) #create a diagonal matrix with offset from the main diagonal
                #Prints "array([[0,1,0,0],[0,0,2,0],[0,0,0,3],[0,0,0,0]])
          #but diag(M,-1) also get diagonal elements of matrix M  with offset -1
random.rand(2,2)  #create an array filled with random values within [0,1]
                  #random.random((2,2))
random.randn() #default 1,normal Gaussian distribution of mean 0 and variance 1.
#sigma*random.randn(d0,d1,...,dn)+mu #Generate d0 X d1 X... X dn dimensions
random samples from N(mu,sigma^2)
2.5*random.randn(2,4)+3 #Two-by-four array of samples from N(3,6.25)
```

arange(start,stop,step),the default start is 0 and default step is 1

```
arange(4)  #等价于arange(0,4,1) ,Prints "array([0,1,2,3])"
b = array([[0,1,1]])
```

```
a[arange(3),b] #Prints “array([1,4,6])”
```
linspace(start,stop,number) 创建等差数列
```
linspace(0,10,3)  #Prints “array([0,5,10])”
```
logspace(start,stop,number，base=ratio) 创建等比数列，默认 base=10,始末元素都是 10 的幂。
```
logspace(0,4,3) #Prints “array([1.0e+00,1.0e+02,1.0e+04])”
logspace(0,8,3,base=2) #Prints “array([1,16,256])”
```

Datatypes and other information
```
type(a) #Prints “<type 'numpy.ndarray'>”
a.dtype #Prints “dtype('int64')”
a.astype(float)  #change the datatype to float
array([[1,2],[3,4]],dtype=complex) #force a particular datatype
a.itemsize #bytes per element,Prints “8”
a.nbytes   #number of bytes, Prints “48”
a.ndim #number of dimensions, Prints “2”
a.size or size(a) #the amount of elements, Prints “6”
a.shape or shape(a) #Prints “(3,2)”
```

Array math
```
x = array([[1,2],[3,4]],dtype=float64)
y = array([[5,6],[7,8]],dtype=float64)
sum(x) #Compute sum of all elements,Prints “10”
sum(x,axis=0) #Compute sum of each column,Prints “array([4,6])”
                                  #在 Numpy 中，axis=0 限定每列
sum(x,axis=1) #Compute sum of each row,Prints “array([3,7])”
                                  #在 Numpy 中，axis=1 限定每行
x+y or add(x,y) #Prints “array([[6,8],[10,12]])”
x-y or subtract(x,y) #Prints “array([[-4,-4],[-4,-4]])”
x*y or multiply(x,y) #Prints “array([[5,12],[21,32]])”
x/y or divide(x,y) #Prints “array([[0.2,0.333],[0.429,0.5]])”
sqrt(x) #Prints “array([[1,1.414],1.732,2]])”
```
Note:与 Matlab 不同，所做的这些运算都是元素间的计算，而非矩阵运算,矩阵运算如下方法:
```
x.dot(y) or dot(x,y) # Matrix/vector product.Prints “array([[19,22],[43,50]])”
x.T #Transpose a matrix,Prints “array([[,3],[2,4]])”
array([1,2]).T #Note that taking the tranpose of a rank 1 array dose nothing
                                   #Prints “array([1,2])”
trace(x)   #same as: diag(A).sum()

mean(x) #所有元素的均值
std(x), var(x)  #标准差与方差
x.min(), x.max() #最小、最大的元素
sum(x), prod(x)  #总和，总乘
cumsum(x), cumprod(x)  #累加和，累乘积
```
加 axis=0 表示按列， axis=1 表示按行，如
```
mean(x,axis=0) 每列均值； mean(x,axis=1) 每行均值;
x.min(axis=0) 每列最小值; x.min(axis=1) 每行最小值
```

复数矩阵
```
conjugate(c) #求共轭
real(c)
imag(c)    #求矩阵实部和虚部
angle(c)
abs(c)     #求矩阵的幅角和绝对值
```

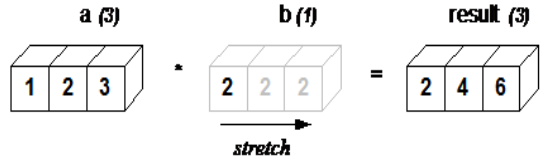
叠加与重复数组
函数 repeat, tile, vstack, hstack, 与 concatenate 能帮助我们以已有的矩阵为基础创建规模更大的矩阵。
```
a = array([[1,2],[3,4]])
repeat each element 3 times:
repeat(a, 3)  #Prints “array([1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4])”
tile the matrix 3 times:
tile(a, 3)  #在列方向上重复 3 次，默认行 1 次,相当于 tile(a,[1,3])
```

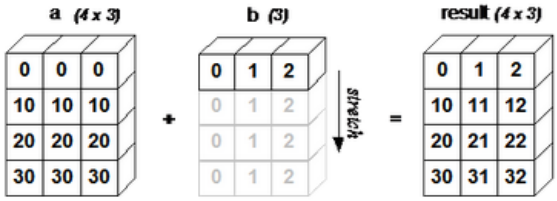
```
            #Prints “array([[1, 2, 1, 2, 1, 2],[3, 4, 3, 4, 3, 4]])”
```
concatenate 连接两个数组
```
b = array([[5, 6]])
concatenate((a, b), axis=0)   #Prints “array([[1, 2],[3, 4],[5, 6]])”
                              #也可用 vstack((a,b))
concatenate((a, b.T), axis=1) #Prints “array([[1, 2, 5],[3, 4, 6]])”
                              #也可用 hstack((a,b.T))
```

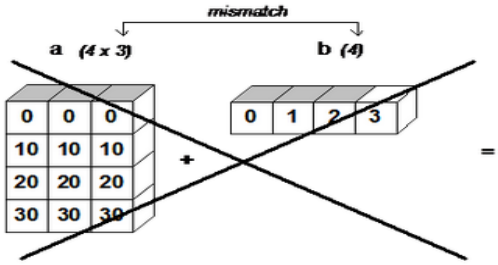
Broadcasting

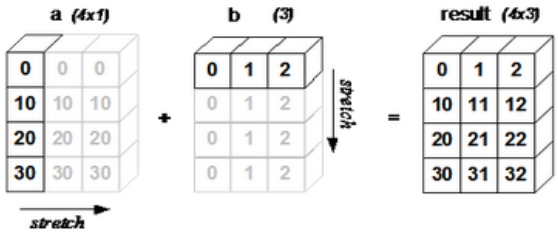
Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations.

```
v = array([1,3])
tile(v,(2,1))  #在行方向上重复 2 次,在列方向上重复 1 次
               #Prints “array([[1,3],[1,3]])”
x+tile(v,(2,1)) #Prints “array([[2,5],[4,7]])”,you can directly calculate like:
x+v  #Add v to each row of x using broadcasting,some explanation shown as
following:
```









详解： Array Broadcasting in numpy
Broadcasting

再看一例：

```
v = array([1,2,3])
w = array([4,5])
```
直接计算 v*w 报错，可以先将 v(shape(3,)) 改为 shape(3,1)。然后可根据 Broadcasting 计算：
```
p = v.reshape(3,1) #reshape v to be a column vector of shape (3,1)
                   #v.resize(3,1)
p*w #print “[[4,5],[8,10],[12,15]]”
```

浅拷贝与深拷贝

Python 中，与列表 list、词典 dict 相同，数组也为**可变数据对象(mutable object)**，即可以通过引用其元素重新赋值，改变对象自身(in-place change)。

```
b = a  // b = a.view()
       //浅拷贝，拷贝量改变原值跟着改变
b[0,0] = 10
a  //Prints “array[[10,2],[3,4]])”
```
可以通过 copy 函数改变这种情况，称为深拷贝
```
b = copy(a) // b = a.copy()
            //深拷贝,拷贝量改变原值不改变
b[0,0] = 5
a //Prints “array[[10,2],[3,4]])”
```

向量化函数:
比如定义下边函数:
```
def  fcn(x):
  if x>=0 :
    return 1
  else:
    return 0
```
当给此函数传入参数为向量 v = array([1,2,-1,-2])时，函数并不能有效执行；此时可以通过 vectorize 命令向量化函数:
```
Theta_vec  = vectorize(fcn)
Theta_vec(v)
```
显示结果 array([1,1,0,0])

数组 I/O 接口
将数组保存为 npy 格式，可以直接使用 numpy.save 和 numpy.load 来保存和读取:
```
M = random.rand(3,3)
save(“random-matrix.npy”,M)
load(“random-matrix.npy”)
```
```
array([[ 0.7077157 ,  0.18134295,  0.84455274],
       [ 0.65154244,  0.92134305,  0.8085764 ],
       [ 0.53212168,  0.02160698,  0.45667373]])
```

也可以使用 savetxt 和 loadtxt 命令，保存为更为通用的 txt 或 csv(Comma-Separated Values，字符分隔值)格式。csv 文件以纯文本形式存储表格数据（数字和文本），是一种常用的数据格式化文件类型
```
M = random.rand(3,3)
savetxt(“random-matrix.txt”,M,fmt='%.5f')  #save as txt file
                                           #fmt specifies the format
savetxt(“random-matrix.csv”,M,fmt='%.5f')  #save as csv file
```

查看数据:
```
loadtxt(“random-matrix.txt”)
genfromtxt(“random-matrix.csv”)
```
或 CTR+D 回到 Linux Shell
```
cat  random-matrix.csv
```
```
0.54155 0.98549 0.91746
0.48577 0.18127 0.97825
0.65033 0.60847 0.72435
```

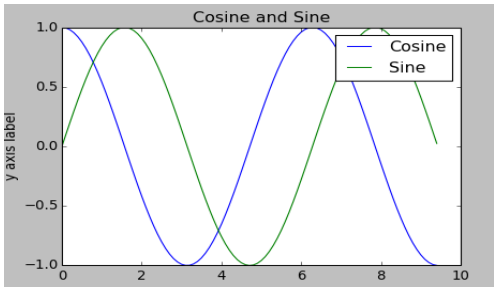
SciPy
The SciPy framework builds on top of the low-level NumPy framework for multidimensional arrays, and provides a large number of higher-level scientific algorithms. Some of the topics that SciPy covers are:

Special functions (scipy.special)
Integration (scipy.integrate)
Optimization (scipy.optimize)
Interpolation (scipy.interpolate)
Fourier Transforms (scipy.fftpack)
Signal Processing (scipy.signal)
Linear Algebra (scipy.linalg)
Sparse Eigenvalue Problems (scipy.sparse)
Statistics (scipy.stats)

Multi-dimensional image processing (scipy.ndimage)
File IO (scipy.io)

Linear Algebra
矩阵求逆
```
from scipy.linalg import *
inv(M)  or  M.I    #求逆
M.I*M      #结果为单位阵（存在可忽略误差）
           #导入 scipy 后运算符*又会重载?
```
求秩
```
linalg.matrix_rank(M)
```
行列式
```
linalg.det(M)
linalg.det(C.I)
```
求解线性问题
```
linalg.solve(A,b)  #solver for dense matrices
linalg.solve(F,E)  #least-squares solution to linear matrix equation
```
求矩阵均方根
```
linalg.sqrtm(A)
```

Integration
```
from scipy.integrate import quad, dblquad, tplquad
```
积分函数 quad
```
def f(x):
  return x**2
x_lower = 0
x_upper =1
val, abserr = quad(f, x_lower, x_upper)
```
带参数积分，可以使用 args 关键字
```
def f(x, n):
  return n*x**2
val, abserr = quad(f, x_lower, x_upper, args=(3,))
```
双重积分函数 dblquad
```
y_lower = 0
y_upper = 10
val, abserr = dblquad(lambda x,y :-x**2-y**2, x_lower, x_upper, lambda
x:y_lower, lambda x:y_upper)
```

Matplotlib
Matplotlib is a plotting library, which provides a plotting system similar to that of MATLAB.
Plotting
The most important function in matplotlib is plot, which allows you to plot 2D data. Here is a simple example:
```
import numpy
import matplotlib.pyplot as plt
x = arange(0,3*pi,0.1)
y_cos = cos(x)
y_sin = sin(x)  #pi,sin 都是 numpy 中定义的
plt.plot(x, y_sin)
plt.plot(x, y_cos)  #绘图
plt.xlabel('x axis label')
plt.ylabel('y axis label')  #标注 x,y 轴标签
plt.title('Sine and Cosine')  #标注标题
plt.legend(['Sine', 'Cosine']) #标注图例
plt.show() #显示
```


Subplots

You can plot different things in the same figure using the subplot function. Here is an example:

```
plt.subplot(2,1,1)
plt.plot(x,y_cos)
plt.title('Cosine')
plt.subplot(2,1,2)
plt.plot(x,y_sin)
plt.title('Sine')
plt.show()
```