Numpy

Many of the examples are taken from: http://www.scipy.org/Cookbook

Building Arrays

```
10 import numpy
```

Arrays can be created from the usual python lists and tuples using the array function.

```
13 a = numpy.array([1,2,3])
14 print a.shape
```

returns a one dimensional array of integers. The array instance a has a large set of methods and properties attached to it. For example, a.shape is the dimension of the array. In this case, it would simply be (3,).

```
(3,)
```

Arrays can also be created using numpy functions, e.g.: array of n-dim zeros

```
19 c = numpy.zeros((2,1,3))
20 print c
```

```
[[[ 0.  0.  0.]]

 [[ 0.  0.  0.]]]
```

```
22 print c.shape
```

```
(2, 1, 3)
```

Array of n-dim ones

```
24 d = numpy.ones((10))
25 print d
```

```
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

```
27 print d.shape
```

```
(10,)
```

Empty arrays of proper dim

```
29 d = numpy.empty((2,1))
30 print d
```

```
[[  3.41909780e-317]
 [  1.39069242e-309]]
```

Regular spaced numbers

```
33 e = numpy.arange(10,20,2)
34 print e
```

```
[10 12 14 16 18]
```

Evenly spaced samples from start to stop.

```
37 f = numpy.linspace(10,20,7)
38 print f
```

```
[ 10.          11.66666667  13.33333333  15.          16.66666667
  18.33333333  20.        ]
```

```
40 print f.shape
```

```
(7,)
```

Evenly spaced samples from start to stop, on log-scale.

```
42 f = numpy.logspace(10,20.,num=7,base=10.)
43 print f
```

```
[  1.00000000e+10   4.64158883e+11   2.15443469e+13   1.00000000e+15
   4.64158883e+16   2.15443469e+18   1.00000000e+20]
```

```
44 print f.shape
```

```
(7,)
```

```
46 print
```

Meshgrid

```
48 x,y = numpy.meshgrid([1,2,3], [4,5,6,7])
49 print x
```

```
[[1 2 3]
 [1 2 3]
 [1 2 3]
 [1 2 3]]
```

```
51 print y
```

Simple operations

```
[[4 4 4]
```

```
[5 5 5]
[6 6 6]
[7 7 7]]
```

Addition of two arrays adds the arrays elements-wise:

```
56 b = numpy.array((10,11,12))
57 print a
```

```
[1 2 3]
```

```
58 print b
```

```
[10 11 12]
```

```
60 print a + b
```

```
[11 13 15]
```

Subtraction, multiplication and division are defined similarly.

```
63 print a-b
```

```
[-9 -9 -9]
```

min, max, sorting, clip ...

```
65 c = numpy.array([2,45,1,9])
66 print c.min()
```

```
1
```

```
67 print c.max()
```

```
45
```

```
68 d = c.copy()
69 d.sort()
70 print c,d
```

```
[ 2 45  1  9] [ 1  2  9 45]
```

```
72 print c.clip(2,9)
```

```
[2 9 2 9]
```

argmin, argmax, argsort

```
74 indices = c.argsort()
75 print c[indices]
```

```
[ 1  2  9 45]
```

```
76 print c.argmin()
```

```
2
```

```
78 print c.argmax()
```

```
1
```

var, std, mean

```
80 print c.var()
```

```
324.6875
```

```
81 print c.std()
```

```
18.0190871023
```

```
83 print c.mean()
```

```
14.25
```

unique

```
85 d = numpy.array([1,2,3,2,3,1,5,6,4])
86 print numpy.unique(d)
```

```
[1 2 3 4 5 6]
```

```
88 print
```

Data type of the array.

The array construct uses the type of its argument. Since a was created from a list of integers, it is defined as an integer array:

```
92 a = numpy.array([1,2,3])
93 print a.dtype
```

```
int32
```

mathematical operations such as division will operate as usual in python, that is, will return an integer answer:

```
97 print numpy.divide(a,3)
```

```
[0 0 1]
```

If divided by a float the results will be a float value:

```
100 print numpy.divide(a,3.)
```

```
[ 0.33333333  0.66666667  1.        ]
```

Similarly we can define the type at initialization time:

```
102 a = numpy.array([1,2,3],dtype=float)
103 print numpy.divide(a,3)
```

```
[ 0.33333333  0.66666667  1.        ]
```

Casting is another possibility:

```
106 a = numpy.array([1,2,3],dtype=int)
107 b = a.astype('float')
108 print numpy.divide(a,3)
```

```
[0 0 1]
```

```
109 print numpy.divide(b,3)
```

```
[ 0.33333333  0.66666667  1.        ]
```

```
111 print
```

Indexing & Slicing

The elements of an array are accessed using the bracket notation a[i] where i is an integer index starting at 0. Sub-arrays can be accessed by using general indexes of the form start:stop:step. a[start:stop:step] will return a reference to a sub-array of array a starting with (including) the element at index start going up to (but not including) the element at index stop in steps of step:

```
116 a = numpy.array([1,2,3,4,5,6,7,8,9,10], float)
```

get all values

```
119 print a[:]
```

```
[  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
```

get the first 3 values

```
122 print a[0:3]
```

```
[ 1.  2.  3.]
```

get the last 3 values

```
125 print a[-3::]
```

```
[  8.   9.  10.]
```

get the sub-array from index 2 till 8

```
128 print a[2:8]
```

```
[ 3.  4.  5.  6.  7.  8.]
```

get the sub-array from index 2 till 8 in steps of 2

```
131 print a[2:8:2]
```

```
[ 3.  5.  7.]
```

This works of course in n-dim

```
133 a = numpy.ones((2,3,4,5,6))
134 b = a[0,1:3,2:3,:,0]
135 print b.shape
```

```
(2, 1, 5)
```

Indexing using bool-arrays

```
138 a = numpy.array([1,2,3,4,5,6,7,8,9,10], float)
139 i = a>=4
140 print i
```

```
[False False False  True  True  True  True  True  True  True]
```

```
142 print a[i]
```

```
[  4.   5.   6.   7.   8.   9.  10.]
```

multiple bool-arrays

```
144 j = a<7
145 print a[i&j]
```

```
[ 4.  5.  6.]
```

Removing all length-1 dimensions

```
148 a = numpy.ones((10,1))
149 print a.shape
```

```
(10, 1)
```

```
150 print a
```

```
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
```

```
   [ 1.]
   [ 1.]
   [ 1.]
   [ 1.]
   [ 1.]]
```

```
151 b = a.squeeze()
152 print b.shape
```

```
   (10,)
```

```
153 print b
```

```
   [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

```
155 print
```

Shaping arrays

Reshaping

```
161 a = 8*numpy.ones((2,3,4),float)
162 print a
```

```
   [[[ 8.  8.  8.  8.]
     [ 8.  8.  8.  8.]
     [ 8.  8.  8.  8.]]

    [[ 8.  8.  8.  8.]
     [ 8.  8.  8.  8.]
     [ 8.  8.  8.  8.]]]
```

```
163 b = a.reshape(6,4)
164 print b
```

```
   [[ 8.  8.  8.  8.]
    [ 8.  8.  8.  8.]
    [ 8.  8.  8.  8.]
    [ 8.  8.  8.  8.]
    [ 8.  8.  8.  8.]
    [ 8.  8.  8.  8.]]
```

Resizing

```
167 a = numpy.array([1,2])
168 print a
```

```
[1 2]
```

```
169 b = numpy.resize(a,(5))
170 print b
```

```
[1 2 1 2 1]
```

Flatten

```
173 a = numpy.ones((1,4,5,6))
174 print a.shape
```

```
(1, 4, 5, 6)
```

```
176 print a.flatten().shape
```

```
(120,)
```

Appending

```
180 a = numpy.arange(5)
181 print a
```

```
[0 1 2 3 4]
```

```
182 b = numpy.arange(3)
183 print b
```

```
[0 1 2]
```

```
184 c = numpy.append(a,b)
185 print c
```

```
[0 1 2 3 4 0 1 2]
```

Appending on the proper axis

```
188 a = numpy.array([[10,20,30],[40,50,60],[70,80,90]])
189 print numpy.append(a,[[15,15,15]],axis=0)
```

```
[[10 20 30]
 [40 50 60]
 [70 80 90]
```

```
 [15 15 15]]
```

```
191 print numpy.append(a,[[15],[15],[15]],axis=1)
```

```
[[10 20 30 15]
 [40 50 60 15]
 [70 80 90 15]]
```

Random numbers

```
195 a = numpy.random.uniform(low=0.,high=10.,size=(2,3))
196 print a
```

```
[[ 2.42552792  2.44301738  8.1649649 ]
 [ 2.20076233  8.2119898   5.10266528]]
```

```
197 b = numpy.random.normal(loc=10.,scale=2.,size=(100,100))
198 print b.shape,b.size
```

```
(100, 100) 10000
```

```
199 print b.mean()
```

```
9.96286597021
```

```
200 print b.std()
```

```
1.98620932767
```

```
202 print
```

Seeds

```
205 print numpy.random.uniform(low=0.1,high=2.),numpy.random.uniform(low=0.1,high=2.)
```

```
1.35181233302 1.92920766922
```

```
206 numpy.random.seed(67)
207 print numpy.random.uniform(low=0.1,high=2.),numpy.random.uniform(low=0.1,high=2.)
```

```
1.13711771688 1.73182756012
```

```
208 numpy.random.seed(67)
```

```
209 print numpy.random.uniform(low=0.1,high=2.),numpy.random.uniform(low=0.1,high=2.)
```

```
1.13711771688 1.73182756012
```

```
211
212 print dir(numpy.random)
```

```
['RandomState', '__RandomState_ctor', '__all__', '__builtins__', '__doc__', '__file__', '__name__', '__path__', '
    beta', 'binomial', 'bytes', 'chisquare', 'dirichlet', 'exponential', 'f', 'gamma', 'geometric', 'get_state', '
    gumbel', 'hypergeometric', 'info', 'laplace', 'logistic', 'lognormal', 'logseries', 'mtrand', 'multinomial', '
    multivariate_normal', 'negative_binomial', 'noncentral_chisquare', 'noncentral_f', 'normal', 'pareto', '
    permutation', 'poisson', 'power', 'rand', 'randint', 'randn', 'random', 'random_integers', 'random_sample', '
    ranf', 'rayleigh', 'sample', 'seed', 'set_state', 'shuffle', 'standard_cauchy', 'standard_exponential', '
    standard_gamma', 'standard_normal', 'standard_t', 'test', 'triangular', 'uniform', 'vonmises', 'wald', '
    weibull', 'zipf']
```

```
213 print
```

Histograms

1D

```
218 b = numpy.random.normal(loc=10.,scale=2.,size=(100))
```

```
220 hs = numpy.histogram(b)
```

2D

```
223 numpy.histogram2d
```

nD

```
226 numpy.histogramdd
```

Masked array: numpy.ma

```
229 # import numpy as np
230 # import numpy.ma as ma
231 # x = np.array([1, 2, 3, -1, 5])
232 # mx = ma.masked_array(x, mask=[0, 0, 0, 1, 0])
233 # mx.mean()
234 # 2.75
```

Input & Output

Saving and loading from text files

```
242 data = numpy.random.uniform(size=(5,5))
243 numpy.savetxt('myfile.txt', data)
244 loaded_data = numpy.loadtxt('myfile.txt')
245 print numpy.alltrue(data == loaded_data)
```

```
True
```

Automaticall zipping

```
248 data = numpy.random.uniform(size=(5,5))
249 numpy.savetxt('myfile.gz', data)
250 loaded_data = numpy.loadtxt('myfile.gz')
251 print numpy.alltrue(data == loaded_data)
```

Binary Files, save one array in one file

```
253 # numpy.save('test.npy', data)
254 # data2 = numpy.load('test.npy')
```

```
True
```

Save mulpiple arrays in one file

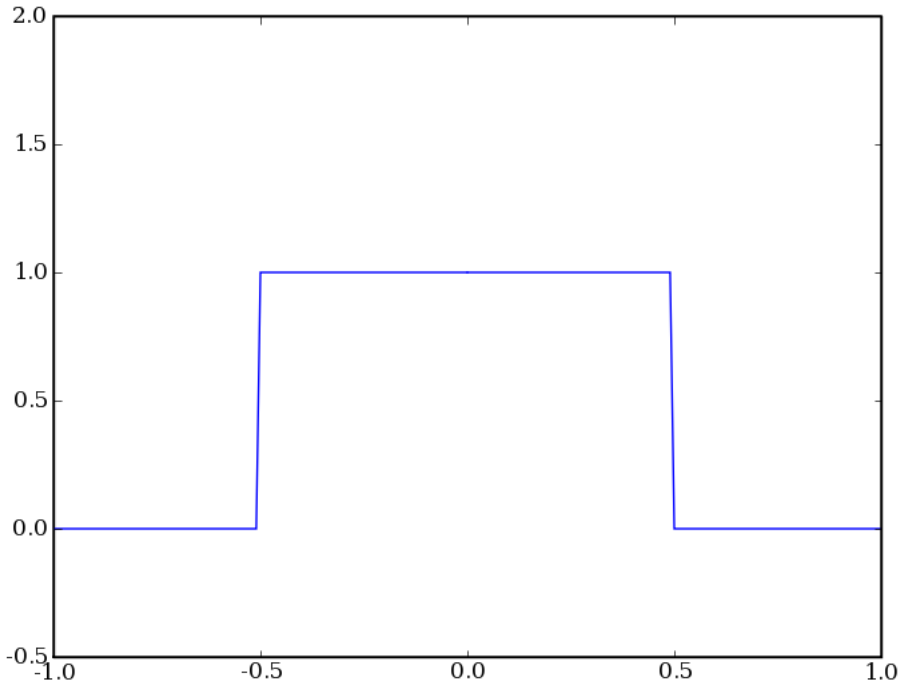
```
257 # a = numpy.random.uniform(size=(5,5))
258 # b = numpy.random.uniform(size=(5,5))
259 # numpy.savez('foo.npz', a=a,b=b)
260 # foo = numpy.load('foo.npz')
261 # print foo.files
262 # ['a', 'b']
263 # a2 = foo['a']
264 # b2 = foo['b']
265 print
```

FFT

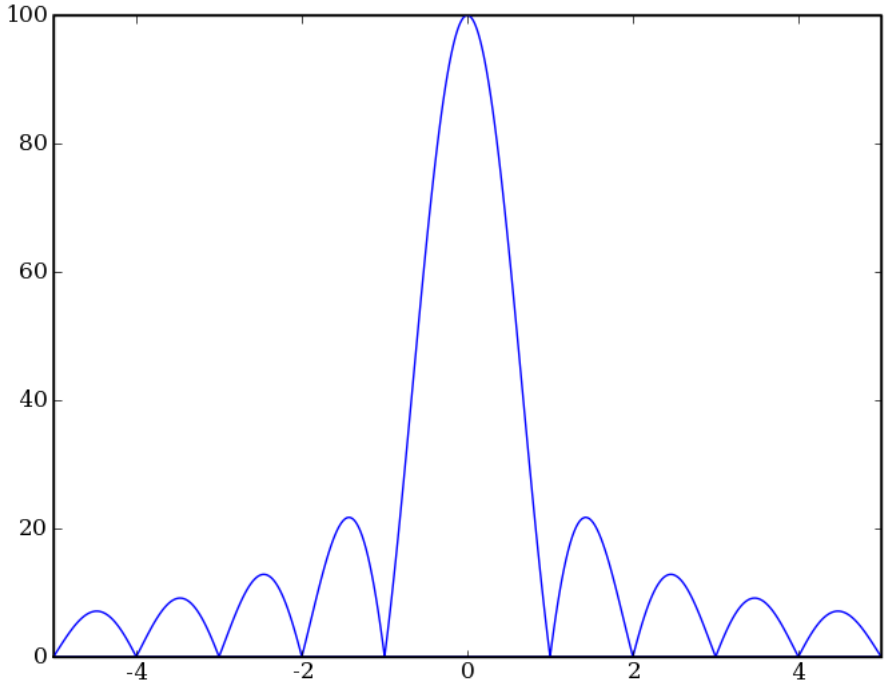
```
271 x = numpy.arange(-100,100,0.01)
272 y = numpy.zeros_like(x)
273 # A box of height 1
274 y[(x>-0.5)&(x<0.5)] = 1.0
```

Plotting

```
277 import pylab
278 pylab.figure()
279 pylab.plot(x,y)
280 pylab.axis([-1,1,-0.5,2])
281 pylab.show()
```




```
283
284 z = numpy.fft.fft(y)
285 f = numpy.fft.fftfreq(len(y),d=0.01)
286
287 pylab.figure()
288 pylab.plot(f,numpy.abs(z))
289 pylab.axis([-5,5,0,100])
290 pylab.show()
```


100
80
60
40
20
0
-4
-2
0
2
4