Python Crash Course Numpy, Scipy, Matplotlib

That is what learning is. You suddenly understand something you've understood all your life, but in a new way.

Doris Lessing

Steffen Brinkmann

Max-Planck-Institut für Astronomie, Heidelberg

IMPRESS, 2016

The ingredients

- A working Python installation
- Internet connection
- Passion for Python

If anything of the above is missing, please say so now!

Get help

- http://docs.scipy.org/doc/
- http://www.numpy.org/

Outline

Outline

What do you want to do?

- Create data or read data from disc
- Manipulate data
- Visualise data
- Write data back to disc

How do you want to do it?

- Create data or read data from disc (Python, numpy)
- Manipulate data (numpy, scipy)
- Visualise data (matplotlib)
- Write data back to disc (Python, numpy)

Outline

import numpy

```python
import numpy as np
```

ndarray

The basic data structure

```
li = [1,2,3]
a = np.array(li)
```

- An `ndarray` is a (usually fixed-size) multidimensional container of items of the same type and size.
- It is created (among others) by the function `np.array`

http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

ndarray

The basic data structure

```
>>> a.shape
(3,)
>>> a.dtype
dtype('int64')
```

- The number of dimensions and items in an array is defined by its shape, which is a tuple of N positive integers
- The type of items in the array is specified by a separate data-type object (dtype), one of which is associated with each ndarray.

http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

ndarray

The basic data structure

```
>>> a.sum()
6
>>> a[1:].sum()
5
```

- The contents of an ndarray can be accessed and modified by indexing or slicing the array, and via the methods and attributes of the ndarray and the `np` namespace.
- slicing generates a *view* on an array, not a new array.

http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

ndarray

Empty ndarrays make no sense

```
>>> python -m timeit -s\
        "import numpy as np; a = np.array([])"\
        "[np.append(a,x) for x in range(int(1e3))]"
100 loops, best of 3: 12.8 msec per loop
>>> python -m timeit -s\
        "a = []"\
        "[a.append(x) for x in range(int(1e3))]"
10000 loops, best of 3: 186 usec per loop
```

- A python list is a C array of pointers to values. Therefore, appending a value is fast, but operating on every value is slow.
- A numpy array is a C array of values. Therefore, appending a value is slow (reallocating), but operating on every value is fast.

Outline

Creation of ndarrays

From Python list

```
arr_1d = np.array([1,2,3])
arr_2d = np.array([[1,2,3],
                   [11,22,33]])
```

- 1D arrays are created from simple Python lists
- nD arrays are created from lists of lists [of lists...]

Creation of ndarrays

Filling with 0's or 1's

```
>>> np.zeros(8)
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> np.ones((2,3,5))
array([[[ 1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.]],
       [[ 1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.]]])
```

- `np.zeros(<shape>)` and `np.ones(<shape>)` return an array of shape `<shape>` filled with 0s (1s).

Creation of ndarrays

Ranges

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(3.5,10,2)
array([ 3.5,  5.5,  7.5,  9.5])
>>> np.linspace(1,3,5)
array([ 1. ,  1.5,  2. ,  2.5,  3. ])
>>> np.logspace(1,3,5)
array([10., 31.6227766, 100., 316.22776602, 1000.])
```

- `np.arange()` works just like Python's `range` but takes floats and returns a np.array.
- `np.linspace(<from>, <to>, <n>)` and `np.logspace(<from>, <to>, <n>)` return ranges of `<n>` numbers including the boundaries.

Reading from and Saving to Files

```
np.save('filename.npy',arr_1)
arr_1_reborn = np.load('filename.npy')
arr_2 = np.loadtxt(<table.dat>, usecols=(2,5))
```

- `np.save(<filename>, <array>)` and `np.load(<filename>)` are the preferred ways to save and load single arrays.
- Use `np.savez(<filename>, <array1>, <array2>,...)` to save multiple arrays
- Use `loadtxt(<filename>, <options>)` to conveniently load data from text tables

Outline

Working with ndarrays

Operators

```
>>> a,b = np.array([1,2]), np.array([11,22])
>>> a+b
array([12, 24])
>>> a*b
array([11, 44])
>>> a@b
55
```

- Operators work element-wise
- Except `@`, which in Python 3.x abbreviates `a.dot(b)`

Working with ndarrays

Comparison

```
>>> a = np.linspace(0,5,10)
>>> a < 2
array([ True,  True,  True,  True, False,
False, False, False, False, False], dtype=bool)
>>> np.all(a<2)
False
>>> np.any(a<2)
True
```

- Also comparison operators work element-wise.
- To get a scalar boolean, use `np.all` or `np.any`.

Working with ndarrays

Selection

```
>>> a = np.linspace(0,5,11)
>>> np.where(a<2)
(array([0, 1, 2, 3]),)
>>> a[np.where(a<2)]
array([ 0. ,  0.5,  1. ,  1.5])
>>> np.where(a<2,a,a*100)
array([   0. ,    0.5,    1. ,    1.5,  200. , 250. ,
        300. ,  350. , 400. ,  450. ,  500. ])
```

- To choose some values from an array based on a boolean array, use `np.where`

Working with ndarrays

Functions

```
>>> np.sin(a)
array([ 0.84147098,  0.90929743])
>>> np.exp(a)
array([ 2.71828183,  7.3890561 ])
```

- All functions and constants from `math` are present in `numpy` (or scipy).
- numpy functions work element-wise

From pure Python to NumPy

```
import math

def func(x):
    return math.sin(x) * math.exp(-0.5*x)

x = [0.1*i for i in xrange(10000001)]
y = [ func(ix) for ix in x]
```

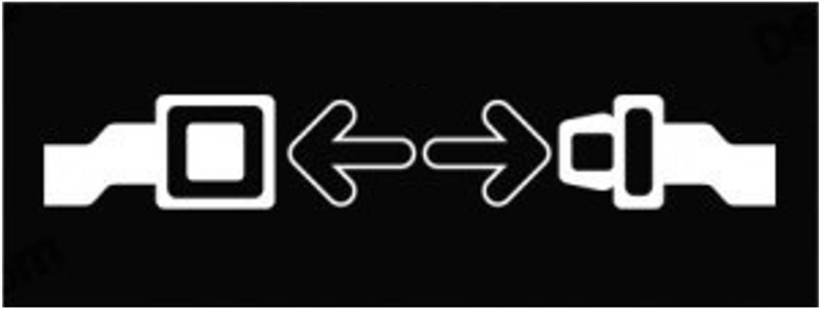
```
import numpy as np

def func(x):
    return np.sin(x) * np.exp(-0.5*x)

x = np.linspace(0,10,10000001)
y = func(x)
```

- convert loops to vectorised functions.

Exercises and Break



Outline

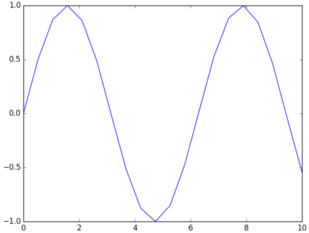
Basic plotting

plt.plot

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

x = np.linspace(0, 10, 20)
y = np.sin(x)

plt.plot(x, y)
plt.show()
```



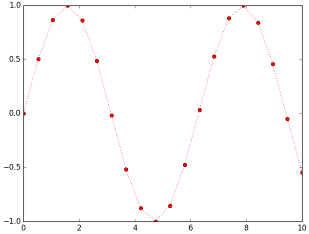
- Use `linspace` to create x coordinates
- Use arithmetic expression to calculate y coordinates
- Plot with `plt.plot(x,y)`
- Show plot with `plt.show()`

Basic plotting

Colours, markers, line styles

```
x = np.linspace(0, 10, 20)
y = np.sin(x)

plt.plot(x, y, 'ro:')
plt.show()
```



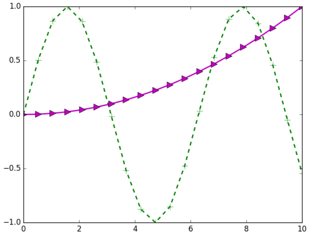
- Specify colours, markers and line style as third argument of `plt.plot`
- colours: r, g, b, c, m, y, k, w
- markers: `.` `,` `o` `v` `^` `<` `>` `*` `+` `x` and more
- line style: `-` `--` `-.` `:`
- line width (`lw`) and marker size (`ms`) as keyword arguments

Basic plotting

Multiple lines

```
x = np.linspace(0, 10, 20)
y = np.sin(x)

plt.plot(x, y, 'g+--',
         x, 1e-2*x**2, 'm>-',
         lw=2, ms=10)
plt.show()
```



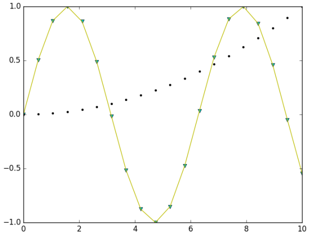
- Repeat positional arguments for multiple lines per plot or...

Basic plotting

Multiple lines

```
x = np.linspace(0, 10, 20)
y = np.sin(x)

plt.plot(x, y, 'cv')
plt.plot(x, y, 'y-')
plt.plot(x, 1e-2*x**2, 'k.')
plt.show()
```

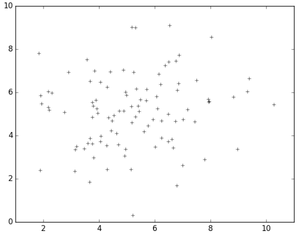


- Repeat calls to `plt.plot` for multiple lines per plot.

Outline

Scatter plots with plot

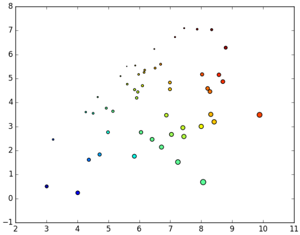
```
x,y=np.random.normal(5,2,(2,100))
plt.plot(x,y,'ko')
plt.show()
```



- Use np.random module to obtain all kind of randomly distributed data.
- Plot using plot without a line style.

Scatter plots with scatter

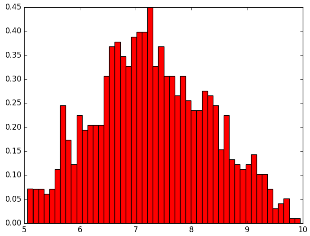
```
x,y=np.random.normal(5,2,(2,100))
plt.scatter(x,y,c=x+y, s=10*x-10*y
plt.show()
```



- c sets the colour (like z value in colour maps)
- s sets the size of the marker

Histograms

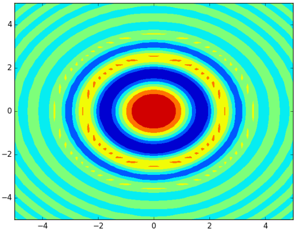
```
x,y=np.random.triangular(5,7,10,(2,1000))
n, bins, patches = plt.hist(x, 50,
                normed=1, facecolor='r')
plt.show()
```



- Use `np.random` module to obtain all kind of randomly distributed data.
- Generate and plot histogram by using `plt.hist`.
- The count and location of the bins are returned

Contour lines

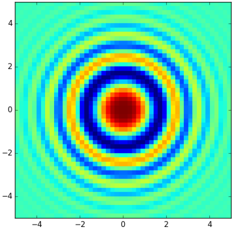
```
x = y = np.linspace(-5,5)
xx,yy = np.meshgrid(x,y)
zz = np.fromfunction(
        lambda i,j: np.cos(x[i]**2+y[j]**2)*
            np.exp(-(x[i]**2+y[j]**2)/10)
        ,(50,50), dtype=int)
plt.contourf(xx,yy,zz)
plt.show()
```



- Use `np.meshgrid` to generate grid coordinates for 2d plots.
- Use `np.fromfunction` to generate an array evaluating a function.
- Plot (filled) contour lines using `plt.contour` (`plt.contourf`)

Colour maps

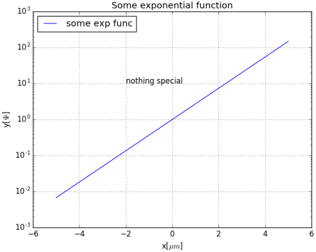
```
x = y = np.linspace(-5,5)
xx,yy = np.meshgrid(x,y)
zz = np.fromfunction(
        lambda i,j: np.cos(x[i]**2+y[j]**2)*
            np.exp(-(x[i]**2+y[j]**2)/10)
        ,(50,50), dtype=int)
plt.imshow(zz, interpolation='nearest',
        origin='lower', extent=(-5,5,-5,5))
plt.show()
```



- Use `plt.imshow` to generate grid coordinates for 2d plots.
- Choose from a variety of interpolation methods
- set `origin` to `'lower'` for math plots
- set `extent` to limits of `x` and `y`

Labels

```
x = np.linspace(-5,5)
y = np.exp(x)
p1 = plt.plot(x,y)
plt.yscale('log')
plt.legend((p1), (r'some exp func',),
        loc='upper left')
plt.xlabel('x[$\mu m$]')
plt.ylabel('y[$\Psi$]')
plt.title('Some exponential function')
plt.text(-2, 10, 'nothing special')
plt.grid(True)
plt.show()
```



- Use `plt.yscale` to set log or lin scale for `x` or `y`.
- Use `plt.legend`, `plt.xlabel`, `plt.ylabel`, `plt.title`, `plt.text` to set labels
- Use `plt.grid` to switch grid on an off

Outline

Overview

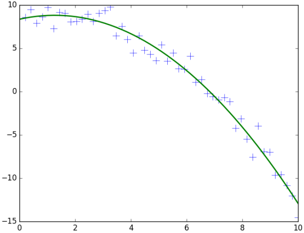
- Many submodules for constants, FFT, integration, linear algebra, interpolation, Multi-dimensional image processing, Optimization and root finding, statistics, signal processing ...
- There are many algorithms for each problems, make sure, you choose the right one for your problem.
- Usually you will need few functions. Import them directly

```
from scipy.special import j0
from scipy.interpolate import interp1d
```

Outline

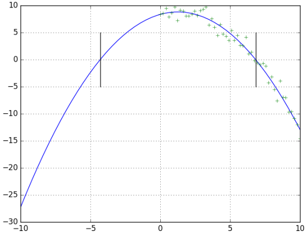
Fitting model to data

```
def model(x, a, b, c):
    return a*x**2+b*x + c
xdata = linspace(0,10)
ydata = model(xdata, -0.3, 0.90, 8.0) + randn(50)
popt, pcov = scipy.optimize.curve_fit(func, xdata, ydata)
yfit = func(xdata, *popt)
plot (xdata, ydata,'+', xdata, yfit, lw=2, ms=12)
```



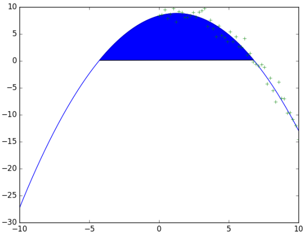
Finding roots

```
def model(x, a, b, c):
    return a*x**2+b*x + c
sol = scipy.optimize.fsolve(model, [-1, 1],
        args=tuple(popt))
print(sol)
# prints: array([-4.30211097,  6.83009459])
```



Integration

```
def model(x, a, b, c):
    return a*x**2+b*x + c
result = integrate.quad(lambda x: model(x,*popt),
          -4.3,  6.83)
print(result)
# prints: (65.43660782846362, 7.264922866452708e-13)
plt.plot(xdata,ytest,linspace(0,10), ydata, '+')
plt.fill(xdata[np.where(ytest>=0)],ytest[np.where(ytest>=0)])
```



For Further Reading I

E. Bressert.
SciPy and NumPy.
O'Reilly, 2012.

Ivan Idris.
NumPy Cookbook.
O'Reilly, 2012.