Introductory Scientific Computing with Python

More plotting, lists and numpy arrays

FOSSEE

Department of Aerospace Engineering
IIT Bombay

Mumbai, India

Outline

Outline

Why would I plot f(x)?

Do we plot analytical functions or experimental data?

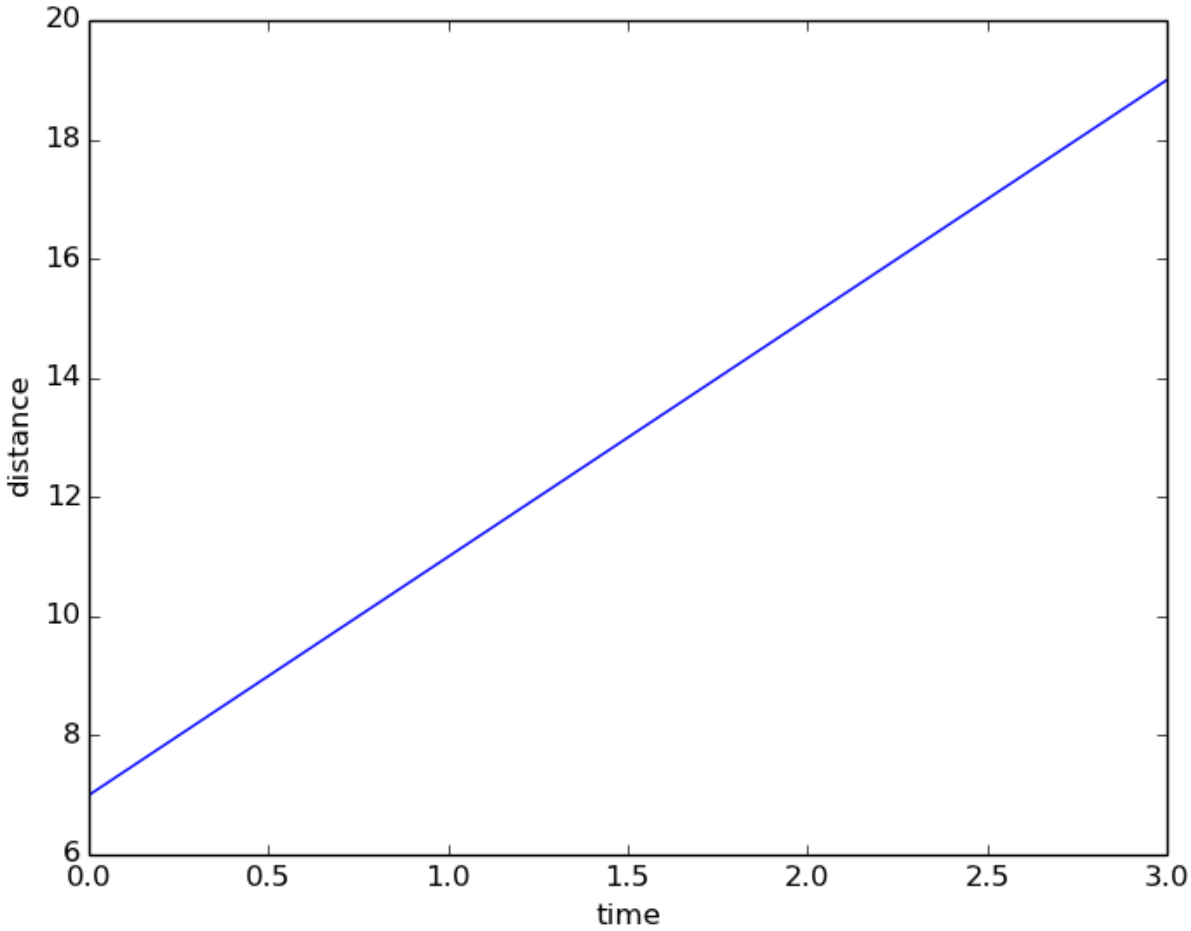
```
In [ ]: time = [0., 1., 2, 3]

In [ ]: distance = [7., 11, 15, 19]

In [ ]: plot(time,distance)
Out[ ]: [<matplotlib.lines.Line2D object at 0xa73a

In [ ]: xlabel('time')
Out[ ]: <matplotlib.text.Text object at 0x986e9ac>

In [ ]: ylabel('distance')
Out[ ]: <matplotlib.text.Text object at 0x98746ec>
```



Is this what you have?

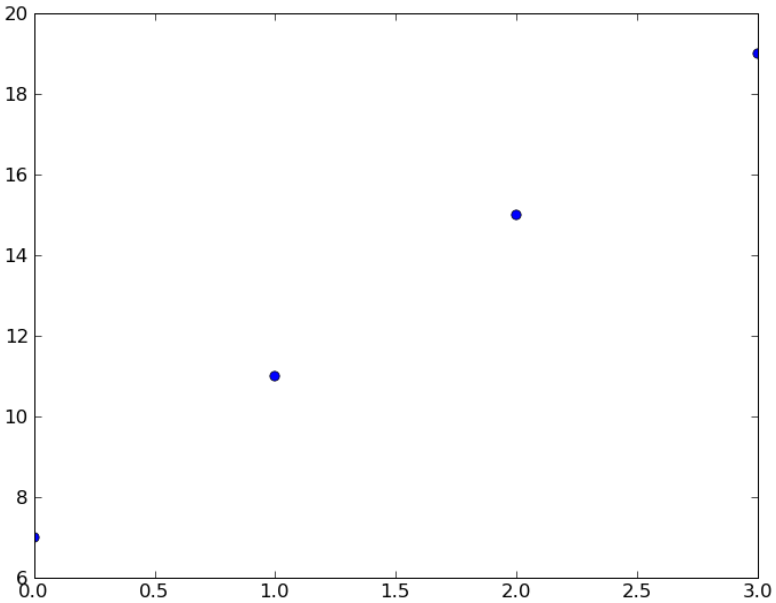
Plotting points

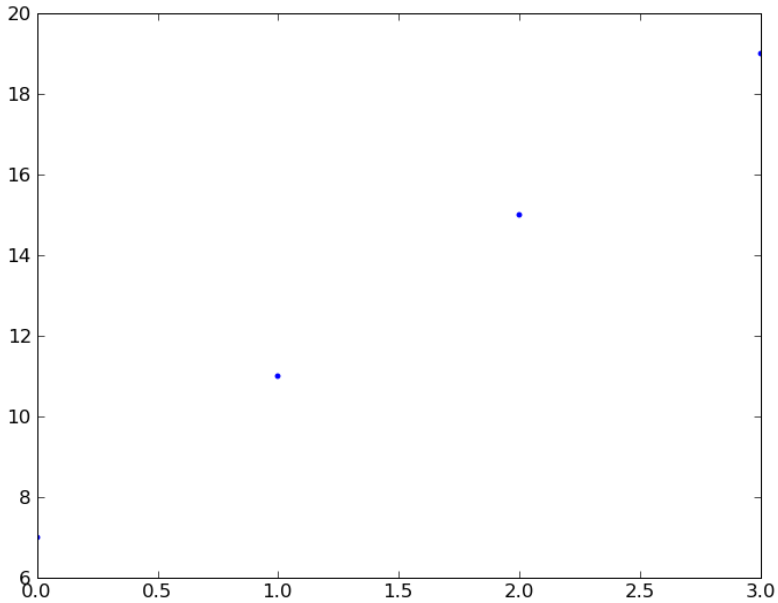
- What if we want to plot the points?

```
In [ ]: clf()

In [ ]: plot(time, distance, 'o')
Out[ ]: [<matplotlib.lines.Line2D object

In [ ]: clf()
In [ ]: plot(time, distance, '.')
Out[ ]: [<matplotlib.lines.Line2D object
```


20
18
16
14
12
10
8
6
0.0
0.5
1.0
1.5
2.0
2.5
3.0


20
18
16
14
12
10
8
6
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Additional Line Styles

- `'o'` - Filled circles
- `'.'` - Small Dots
- `'-'` - Lines
- `'--'` - Dashed lines

Outline

Lists: Introduction

```
In []: time = [0., 1., 2, 3]
```

```
In []: distance = [7., 11, 15, 19]
```

What are **time** and **distance**?

lists!!

Lists: Initializing & accessing elements

```
In []: mtlist = []
```

Empty List

```
In []: p = [ 2, 3, 5, 7]

In []: p[1]
Out[]: 3

In []: p[0]+p[1]+p[-1]
Out[]: 12
```

List: Slicing

Remember...

```
In []:  p = [ 2, 3, 5, 7]
```

```
In []: p[1:3]
Out[]: [3, 5]
```

A slice

```
In []: p[0:-1]
Out[]: [2, 3, 5]
In []: p[1:]
Out[]: [3, 5, 7]
```

List: Slicing . . .

Remember. . .

```
In []:  p = [ 2, 3, 5, 7]
```

```
In []: p[0:4:2]
Out[]: [2, 5]
In []: p[0::2]
Out[]: [2, 5]
In []: p[::2]
Out[]: [2, 5]
In []: p[::3]
Out[]: [2, 7]
In []: p[::-1]
Out[]: [7, 5, 3, 2]
```

list[initial:final:step]

List: Slicing

Remember...

```
In []:  p = [ 2, 3, 5, 7]
```

What is the output of the following?

```
In []: p[1::2]
```

```
In []: p[1:-1:2]
```

List operations

```
In []: b = [ 11, 13, 17]
In []: c = p + b

In []: c
Out[]: [2, 3, 5, 7, 11, 13, 17]

In []: p.append(11)
In []: p
Out[]: [ 2, 3, 5, 7, 11]
```

Question: Does `c` change now that `p` is changed? 10 m

Outline

Simple Pendulum - L and T

Let us look at the Simple Pendulum experiment.

L	T	T^2
0.2	0.90	
0.3	1.19	
0.4	1.30	
0.5	1.47	
0.6	1.58	
0.7	1.77	
0.8	1.83	

$L \alpha T^2$

Lets use lists

```
In []: L = [0.2, 0.3, 0.4, 0.5,
            0.6, 0.7, 0.8]

In []: t = [0.90, 1.19, 1.30,
            1.47, 1.58, 1.77,
            1.83]
```

Gotcha: Make sure `L` and `t` have the same number of elements

```
In []: print len(L), len(t)
```

Plotting L vs T^2

- We must square each of the values in `t`
- How do we do it?
- We use a `for` loop to iterate over `t`

Looping with `for`

```
In []: for time in t:
  ....:     print(time*time)
  ....:
  ....:
```

This will print the square of each item in the list, **t**

Plotting L vs T^2

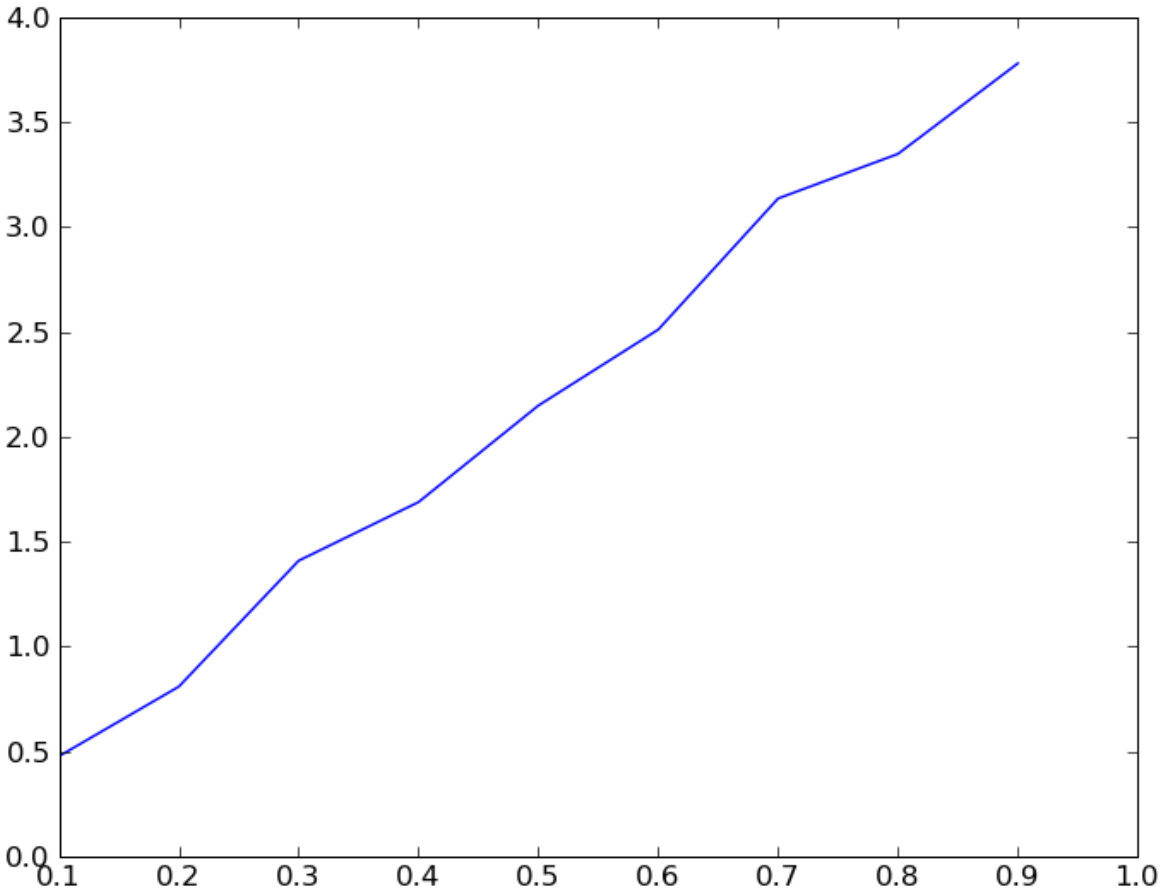
```
In [ ]: tsq = []

In [ ]: for time in t:
  ....:     tsq.append(time*time)
  ....:
  ....:
```

This gives **tsq** which is the list of squares of **t** values.

```
In [ ]: print(len(L), len(t), len(tsq))
Out[ ]: (7, 7, 7)

In [ ]: plot(L, tsq)
```



20 m

Don't repeat yourself: functions

Let us define a function to square the list

```
In []: def sqr(arr):
   ...:     result = []
   ...:     for x in arr:
   ...:         result.append(x*x)
   ...:     return result
   ...:

In []: tsq = sqr(t)
```

More on defining functions

- Consider the function `f(x) = x^2`
- Let's write a Python function, equivalent to this

```
In[]: def f(x):
....:     return x*x
....:

In[]: f(1)
In[]: f(2)
```

- `def` is a keyword
- `f` is the name of the function
- `x` the parameter of the function (local variable)
- `return` is a keyword

Aside: Exercise

- Write a function called **`mysum(a, b)`** that returns sum of two arguments.

```
In [ ]: def mysum(a, b):
   ...:     return a + b
   ...:
In [ ]: mysum(1, 2)

In [ ]: mysum([1, 2], [3, 4])
```

Aside: Exercise

- Write a function called **`mysum(a, b)`** that returns sum of two arguments.

```
In []: def mysum(a, b):
   ...:     return a + b
   ...:
In []: mysum(1, 2)

In []: mysum([1, 2], [3, 4])
```

This seems tedious

- Do we have to write a function just to get the square of a list?
- Lists
  - Nice
  - Not too convenient for math
  - Slow
- Enter NumPy arrays
  - Fixed size, data type
  - Very convenient
  - Fast

30 m

Outline

NumPy arrays

```
In []: t = array(t)

In []: tsq = t*t

In []: print(tsq)

In []: plot(L, tsq) # works!
```

Speed?

Lets use range to create a large list.

```
In [ ]: t = range(1000000)
```

```
In [ ]: tsq = sqr(t)
```

Now try it with

```
In [ ]: t = array(t)
```

```
In [ ]: tsq = t*t
```

...

IPython tip: Timing

Try the following:

```
In []: %timeit sqr(t)

In []: %timeit?
```

- **%timeit**: accurate, many measurements
- Can also use **%time**
- **%time**: less accurate, one measurement

40 m

Exercise

Find out the speed difference between the **`sqr`** function and **`t*t`** on the numpy array.

Solution

```
In []: t = linspace(0, 10, 100000)
In []: %timeit sqr(t)
In []: %timeit t*t
```

45 m

Summary

- Plot attributes
- plotting points
- Lists
- Defining simple functions
- Introduction to `numpy` arrays
- Timing with **`%timeit`**