Lecture 5

- Beginning NumPy
- Basic Plotting using MatPlotLib

Qingyang "Young" Xu

April 14 2015

Our feedback on your feedback

- Lectures too long. Need more time for lab.
  - Sure! We'll do our best to make lectures more concise (aiming towards 40/60).

- Need slides which summarizes the syntax.
  - No problem! We'll have such slide available during the lab.

- Python is fun! I'm excited to learn more!
  - Great! We'll make sure everyone stays excited!

What we've learned last week…

- Basic Unix Commands
  - navigate directories
  - create, delete, move files
- Error Handling and Debugging in Python
  - what's in the error message
  - how to debug based on the error message
- Basic Git Commands

Learning goals for today!

- After this lecture, you will…
- Learn what NumPy has to offer…
  - Numpy Arrays
  - functions and operations
- Simple plots with matplotlib
- Basic curve fitting with scipy

What's NumPy?

- Numerical extension of Python
- Basically the calculator with advanced features for specific purposes such as scientific research, financial accounting, etc.
- Main feature: NumPy Arrays
- Contains high level mathematical functions to operate these objects (mostly linear algebra and statistical operations)

Why NumPy?

- In most scientific experiments, the set of data is huge and therefore are most conveniently organized into arrays (often multidimensional).
- When we analyze data, it is also much more efficient to perform statistical analysis with these arrays as a whole.
- When we plot data, we are really "visualizing arrays" (most often 2D and 3D). Thus if we want to get good plots, it is very important to get a good array representation of the data.

Basic Examples of NumPy array

- Arrays in NumPy are assumed to be homogeneous and have a fixed size as declared, much like Java and C++.
- So all the freedom in using Python list is gone, but so does MOST of the responsibility to make sure that weird semantic errors does not occur!
- We will see an exception in a second…

How to use NumPy arrays?

- First things first:

```
import numpy (as np)
```

- Create a 1-dimensional array (or vector)

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

abc = np.array(["a","b","c"])

age = np.array(["age", 20])
```

- What will happen if I enter the line

```
print age[0] + age[1]?
```

Some very useful special arrays

- Array that is similar to range() function:

```
np.arange(10) # int less than 10

np.arange(3, 15) # int from 3 to 14

np.arange(1, 2, .1) # increment by .1
```

- Array that equally divides an interval:

```
tri_parte = np.linspace(0., 1., 4)

# this will create 3 sub-intervals
```

Basic 1-D Array Operations

- NumPy makes it really convenient to work with arrays. But sometimes it can be "too convenient" that we need to be careful.
- The "usual operations" are entry-wise.

```
import numpy as np
x = np.array([1,2,3])
y = np.array([2,5,7])
# addition/multiplication/squaring
# are entry-wise
print x + 1, y * 2, x**2
print x + y, x * y # NOT x dot y!
```

What's the next thing after vector?

- Matrices (or n-dimensional arrays, in general)
- Create two dimensional array of integers:
  - What is your guess and why?

```
mat = np.array([[1,2],[3,4]])

id = np.array([[1,0,0],[0,1,0],[0,0,1]])
```

  - This is really an "array of arrays".
  - What about higher dimensions?

Array iteration (accessing elements)

- Array iterations in Python carries over into NumPy. We will concentrate on 2D arrays.
- Accessing elements:

mat[1,1] or mat[1][1]

- Accessing rows:

mat[1] or mat[1,:]

iden[-1]

- Accessing columns:

iden[:,2]

- Accessing multiple rows:

iden[0:2]

Basic operations of NumPy arrays

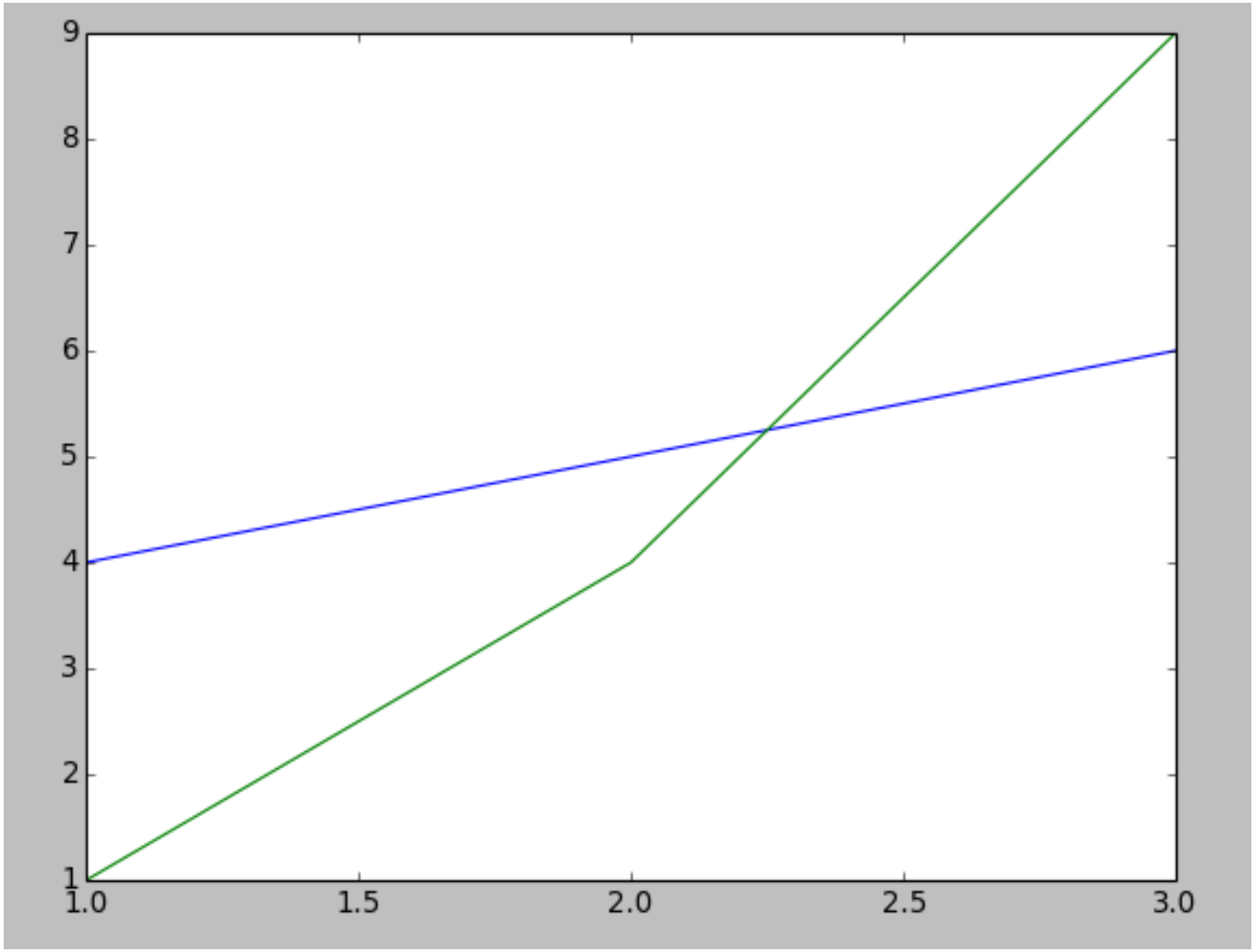
- Now that we have the basic objects of linear algebra, it's time for operations!
- NumPy basically supports all elementary mathematical functions. So when in doubt, just Google the documentation!

```
from numpy.linalg import solve, inv
mat.transpose()
inv(iden)          #returns the inverse matrix
b = np.array([2,3])
solve(mat,b)       #solves the system mat*x = b
```

Plotting with matplotlib

- matplotlib is a plotting library for Python.
- Plotting is super easy once you have the appropriate arrays to plot.

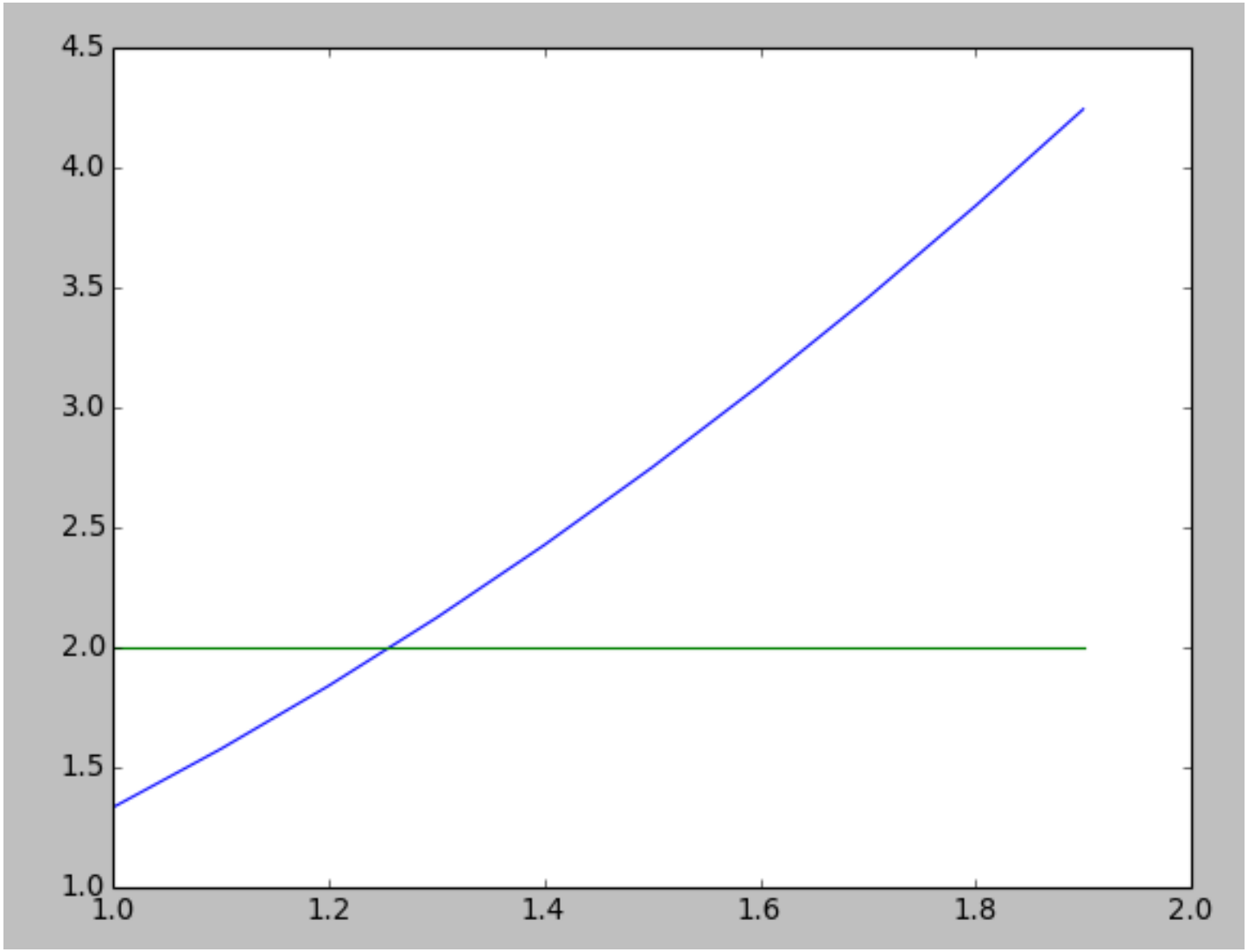
```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1,2,3])
y = np.array([4,5,6])
plt.plot(x, y)       # plots the curve
plt.plot(x, x**2)       # plots the curve
plt.show() # this allows you to see the plot
```


9
8
7
6
5
4
3
2
1
1.0
1.5
2.0
2.5
3.0

Another example

- It is easy to generalize this to create more complicated plots.

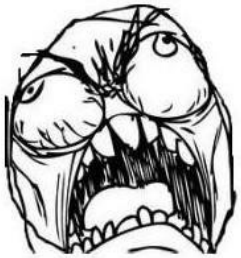
```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(1, 2, .1)
y = x ** 2 + x / 3
z  = x ** (1/2) + 1
plt.plot(x, y)
plt.plot(x, z)
plt.show()
```


4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.2
1.4
1.6
1.8
2.0

More questions to consider…

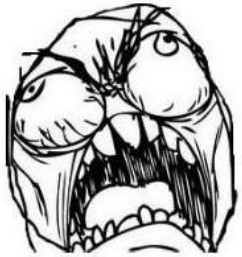
Q: How to label axes and curves?

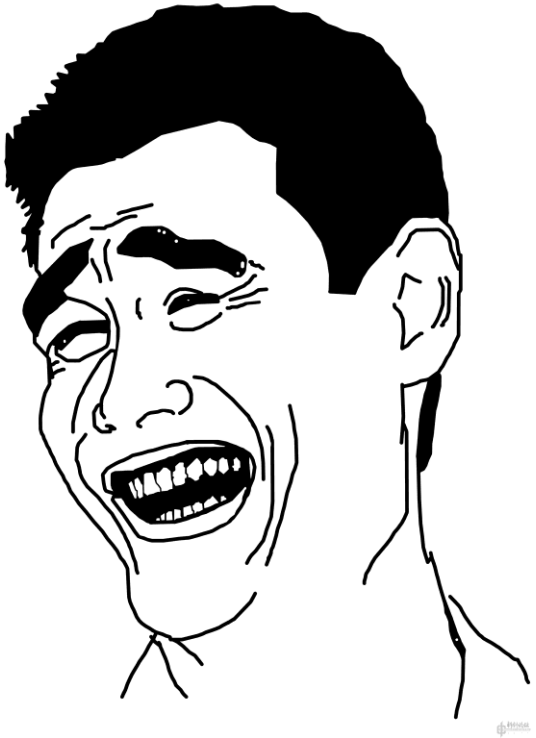
A: Google.



Q: How to plot histograms?

A: Google.





Q: How about #$%^&**&^%$?

A: You probably know the answer…

It doesn't take too much to be an expert!

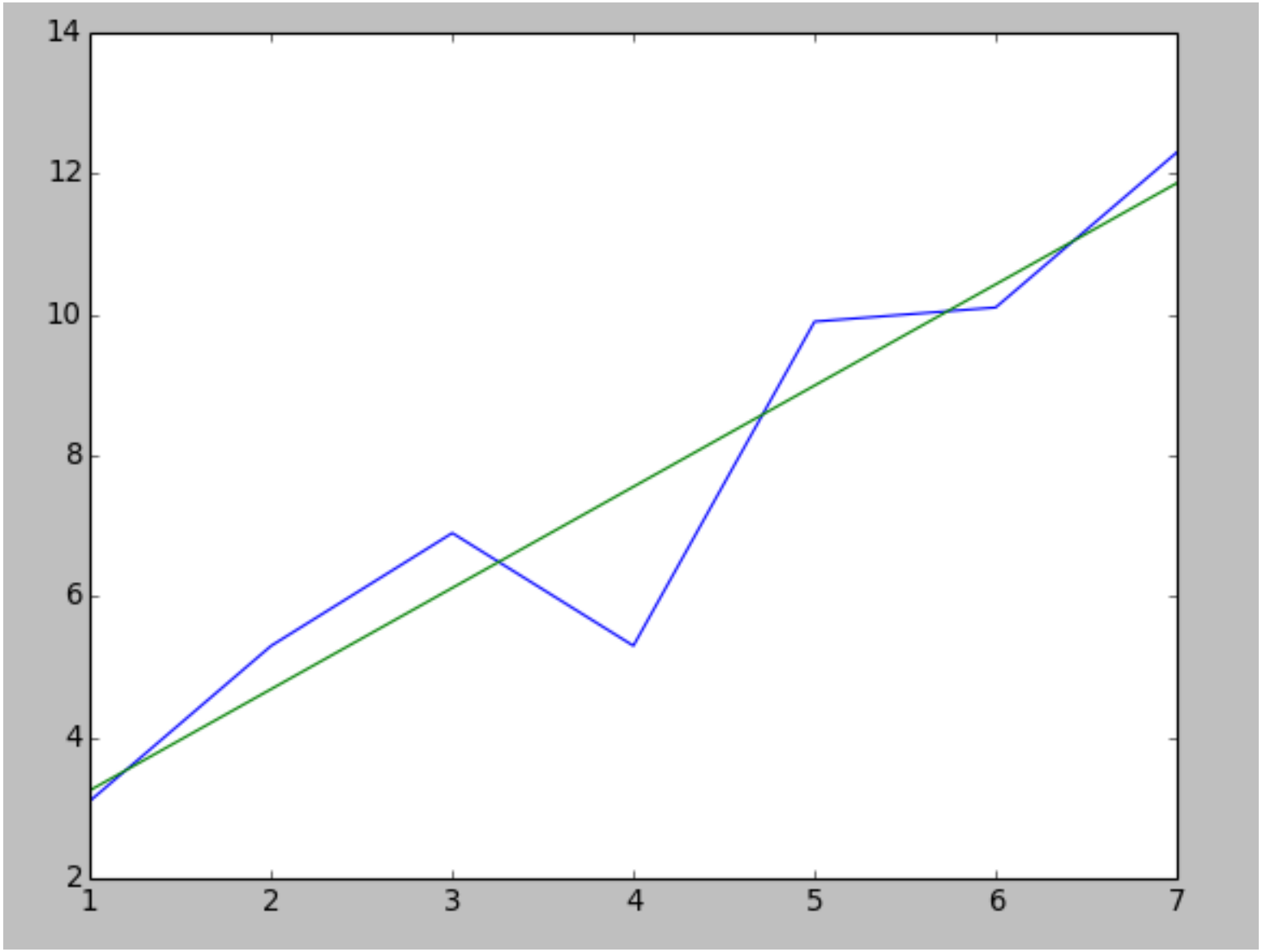
Curve-fitting with SciPy

- Scientific Python is the Python library for scientific computing, with modules for optimization, integration, signal and image processing, etc.
- We will learn to use curve fitting today.

```
from scipy.optimize import curve_fit
def linear_fit(x, a, b):
    return a * x + b
x_data = np.array([1, 2, 3, 4, 5, 6, 7])
y_data = np.array([3.1, 5.3, 6.9, 5.3, 9.9, 10.1, 12.3])

# curve_fit takes function and array-like data
# returns optimized parameters and covariance
opt, cov = curve_fit(linear_fit, x_data, y_data)

plt.plot(x_data, y_data)
plt.plot(x_data, a[0] * x_data + a[1])
plt.show()
```


14
12
10
8
6
4
2
1
2
3
4
5
6
7

Syntax Summary

NumPy Arrays

```
import numpy as np
x = np.array([x1, x2, ...])
y = np.arange(init, end, step)
```

matplotlib

```
import matplotlib.pyplot
as plt
plt.plot(array1, array2)
plt.plot(array3, array4)
plt.show()
```

SciPy Curve Fitting

```
from scipy.optimize import
curve_fit
x_data = np.array([...])
y_data = np.array([...])
def func(array, params):
    return ...
opt, cov =  curve_fit(func,
x_data, y_data)
# parameters in opt is in the
same order as in params
```

Looking forward…

- Next time, we'll learn…
  - more NumPy arrays (2D)
  - more SciPy functions
  - how to read documentation
  - how to design and manage data flow