Foundations of Computer Science in Python

Lecture 8: NumPy & SciPy

Today's Plan

- Introduction to ***NumPy*** & ***SciPy***
- Plotting
- Data Analysis

NumPy and SciPy

- **NumPy** and **SciPy** are packages for scientific computing that provide fast pre-compiled mathematical and numerical functions.
- The **NumPy** (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data.
- The **SciPy** (Scientific Python) package extends the functionality of NumPy with a large collection of useful algorithms, like minimization, regression, and other applied mathematical techniques

NumPy and SciPy

- **NumPy** and **SciPy** are open-source, and therefore provide a free Matlab alternative.
- The packages are popular among scientists, researchers and engineers who want to apply various mathematical methods on large datasets.

Read more here: *http://docs.scipy.org/doc/numpy/reference/*

Importing the required modules

- The packages we need are already included within the installation of Anaconda.
- Make sure you import any needed package at the beginning of your program in order to use its classes….
- For example:

```
import numpy as np, matplotlib, scipy
```

Alias

NumPy's main object is the **Array**

- NumPy's main object is the homogeneous multidimensional **array**.
- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In Numpy dimensions are called ***axes***.
- The number of axes is ***rank***.
  - *A **vector** is an array of rank 1*
  - *A 2D **matrix** is an array of rank 2.*

The Array object - Creation

```
>>> import numpy as np
>>> a = np.array([0, 1, 2])          # 1 x 3 array
array([0,1,2])
```

0	1	2

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array
array([[0,1,2],
       [3,4,5]])
```

0	1	2
3	4	5

Array – Attributes & Methods

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]]) # Creates a 2 x 3 array
array([[0,1,2],
       [3,4,5]])
>>> b.ndim  #number of dimensions
2
>>> b.shape  #dimension sizes
(2,3)
>>> len(b) # returns the size of the first dimension
2
>>> b.T          #Transpose
array([[0, 3],
       [1, 4],
       [2, 5]])
```

0	1	2
3	4	5

0	3
1	4
2	5

Creating an Array of a specific type

```
>>> a = np.array(range(10), float)
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> a.dtype
dtype('float64')
```

Creating an array of zeros or ones

```
>>> np.zeros(7, dtype=int)
array([0, 0, 0, 0, 0, 0, 0])

>>> np.ones((2,3), dtype=float)
 array([[ 1., 1., 1.],
        [ 1., 1., 1.]])
```

Creating arrays containing number sequences

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2,7)
array([2, 3, 4, 5, 6])
>>> np.arange(2,7,2) # start, end (exclusive), step
array([2, 4, 6])
```

Like range in Python, but returns a numpy array object

Creating arrays containing number sequences

```
>>> print (np.arange(1, 5.5, 0.5))  # start, end (exclusive), step
[ 1.   1.5  2.   2.5  3.   3.5  4.   4.5  5. ]
# slicing is similar to standard lists
>>> print (np.arange(1, 5.5, 0.5)[3:1:-1])
[ 2.5  2. ]
>>> print(np.linspace(1,5,9))  # start, end, number of points
[ 1.   1.5  2.   2.5  3.   3.5  4.   4.5  5. ]
# Arrays with random values between 0 and 1
>>> print (np.random.rand(5))
[ 0.53844313  0.3384871   0.5763536   0.29159273  0.43938366]
```

Additional ways to create arrays in NumPy:
http://docs.scipy.org/doc/numpy-1.10.1/reference/routines.array-creation.html#routines-array-creation

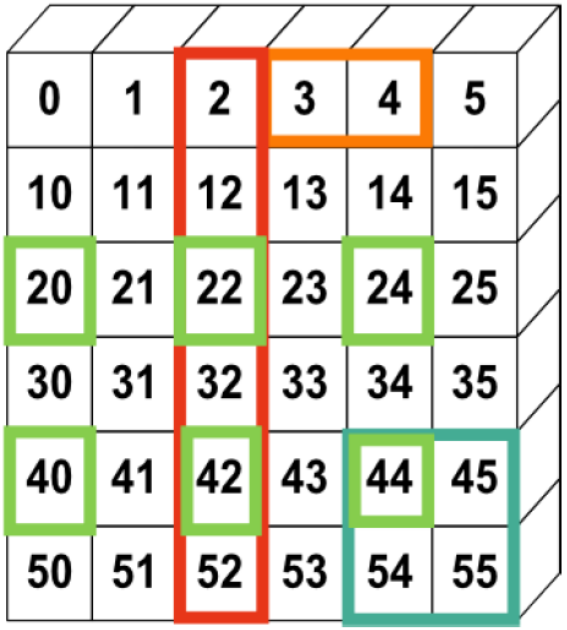
Indexing and slicing: similar to lists

```
>>> a = np.diag(np.arange(3))
array([[0, 0, 0],
[0, 1, 0],
[0, 0, 2]])
>>> a[1, 1]
1
>>> a[2, 1] = 0
>>> a[1, ]  # a[1] also works
array([ 0, 1, 0])
>>> a[:,1]
array([ 0,  1, 0])
>>> a[2][1:]
array([0,  2])
```

Slicing: specify which rows/columns to take by A[rows, columns]

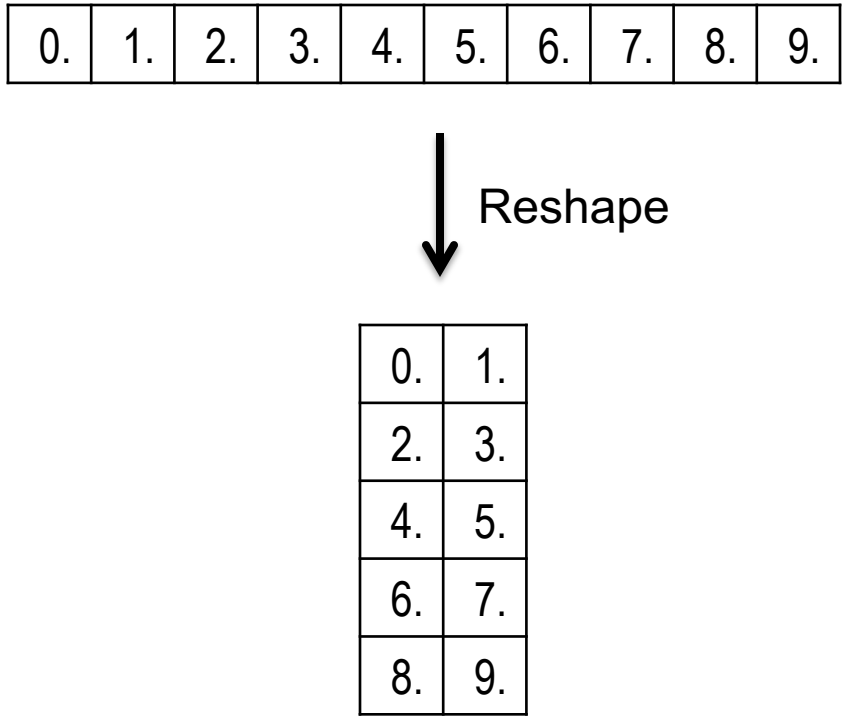
Indexing and slicing: matrices

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Reshaping an Array

```
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

>>> a = a.reshape((5, 2))
>>> a
array([[ 0., 1.],
       [ 2., 3.],
       [ 4., 5],
       [ 6., 7.],
       [ 8., 9.]])
>>> a.shape
(5, 2)
```



Example

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
>>> x.shape = (2,5) # Setting x's dimensions to (2,5)
>>> x[1,3]
8
>>> x[1,-1]
9
>>> x[0]
array([0, 1, 2, 3, 4])
```

Array Arithmetic

Arithmetic operators on arrays apply *elementwise*. A new array is created and filled with the result.

```
>>> x = np.array([1,5,2])
>>> y = np.array([7,4,1])
>>> x + y
array([8, 9, 3])
>>> x * y  # element by element multiplication ! Use np.dot(x,y) for matrix multiplication.
array([ 7, 20,  2])
>>> x - y
array([-6,  1,  1])
>>> x / y
array([0, 1, 2])
>>> x % y
array([1, 1, 0])
```

Note that here, x and y have the same size

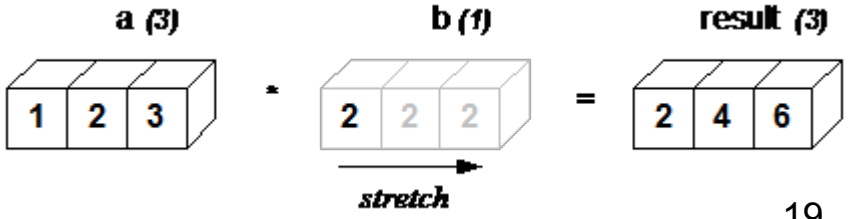
Broadcasting

- NumPy operations are usually done element-by-element which requires two arrays to have exactly the same shape:

```
>>> a = array([1.0,2.0,3.0])
>>> b = array([2.0,2.0,2.0])
>>> a * b
array([ 2.,  4.,  6.])
```

- But this will also work:

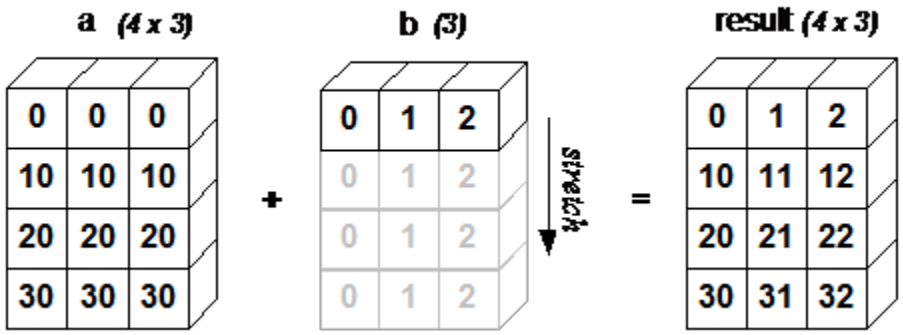
```
>>> a = array([1.0,2.0,3.0])
>>> b = 2.0
>>> a * b
array([ 2.,  4.,  6.])
```



Broadcasting

- Broadcasting is supported when the size of the *trailing* axes for both arrays in an operation is either the same or when one of them is one.

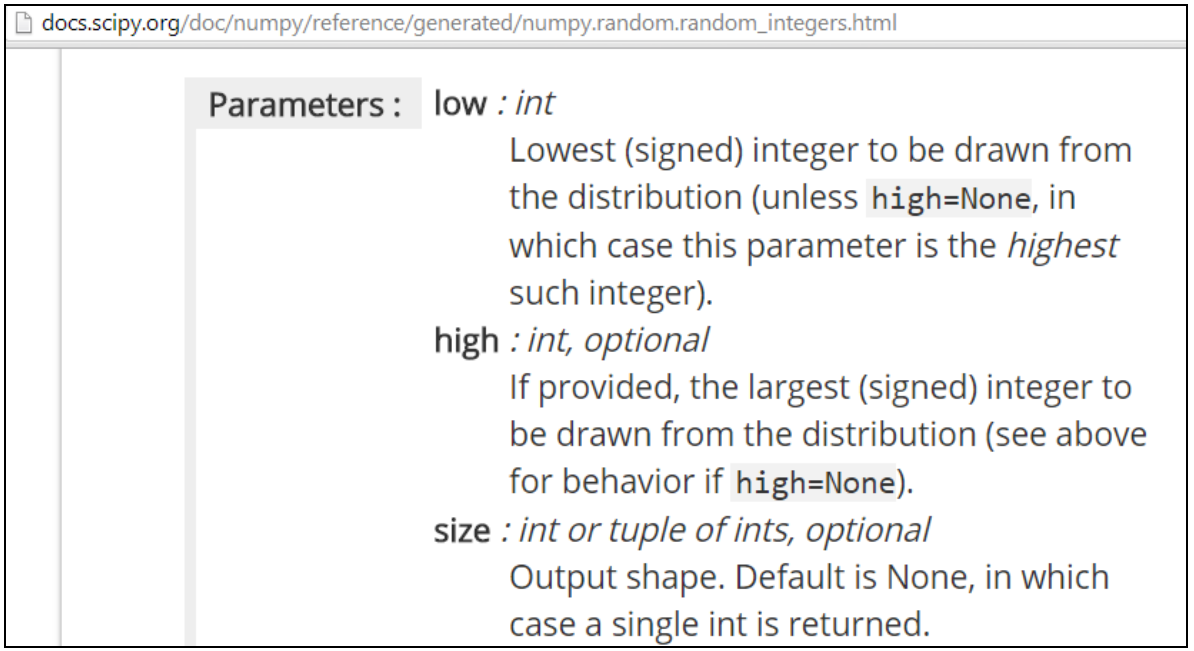
```
>>> a = array([[ 0.0, 0.0, 0.0],
...            [10.0,10.0,10.0],
...            [20.0,20.0,20.0],
...            [30.0,30.0,30.0]])
>>> b = array([1.0,2.0,3.0])
>>> a + b
array([[  1.,   2.,   3.],
       [ 11.,  12.,  13.],
       [ 21.,  22.,  23.],
       [ 31.,  32.,  33.]])
```



Comparisons and Boolean operations

```
>>> a = np.random.random_integers(0, 10, 15)
array([ 4,  9,  1,  7,  0,  9,  8, 10,  1,  0,  7,  7,  9,  5,  1])
```

Documentation - search np.random.integers in google



Comparisons and Boolean operations

```
>>> a = np.random.random_integers(0, 10, 15)
>>> b = np.random.random_integers(0, 10, 15)
>>> comp1 = a==b
>>> comp1
array([False, False, False, False, False,  True, False, False,
False,   False, False, False, False, False, False], dtype=bool)
>>> comp1.any()
True
>>> comp1.all()
False
>>> comp1.nonzero()
(array([5], dtype=int64),)
```

Any is true?

Are all true?

Get the True indices

Fancy indexing: Logical indexing

```
>>> a = np.random.random_integers(0, 20, 15)
array([10, 3, 8, 0, 19, 10, 11, 9, 10, 6, 0, 20, 12, 7, 14])
>>> (a % 3 == 0)
```

What will be the result?
What will be the type of the new object?

```
array([False, True, False, True, False, False, False, True,
False,True, True, False, True, False, False], dtype=bool)
```

Fancy indexing: Logical indexing

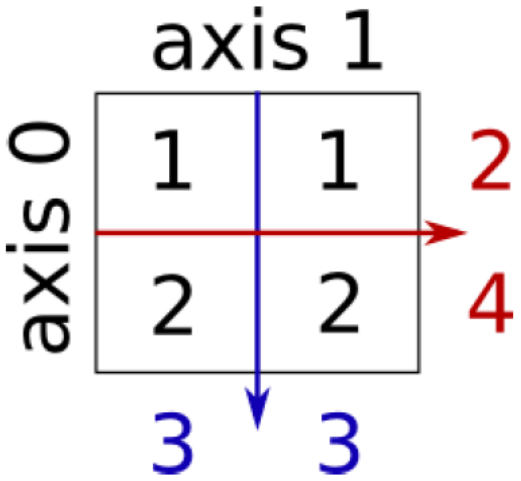
```
>>> a = np.random.random_integers(0, 20, 15)
array([10, 3, 8, 0, 19, 10, 11, 9, 10, 6, 0, 20, 12, 7, 14])
>>> (a % 3 == 0)
array([False, True, False, True, False, False, False, True,
    False,True, True, False, True, False, False], dtype=bool)
>>> mask = (a % 3 == 0)
# extract sub-array: this is a copy!
>>> extract_from_a = a[mask]
array([ 3, 0, 9, 6, 0, 12])
# change sub-array
>>> a[mask] = -1
array([10, -1, 8, -1, 19, 10, 11, -1, 10, -1, -1, 20, -1, 7, 14])
```

Fancy indexing II

```
# Indexing with array/list of integers
>>> a = np.arange(0, 100, 10)
# note: [2, 3, 2, 4, 2] is a Python list
>>> a[[2, 3, 2, 4, 2]]
array([20, 30, 20, 40, 20])
```

Reductions

```
>>> x = np.array([[1, 1], [2, 2]])
array([[1, 1],
       [2, 2]])
>>> x.sum() # works on the entire matrix
6

>>> x.sum(axis=0) # sum over the columns (first dimension)
array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
(3, 3)

>>> x.sum(axis=1) # sum over the rows (second dimension)
array([2, 4])
>>> x[0, :].sum(), x[1, :].sum()
(2, 4)
```



Also works with x.min, x.max, x.mean etc.

Sorting along an axis

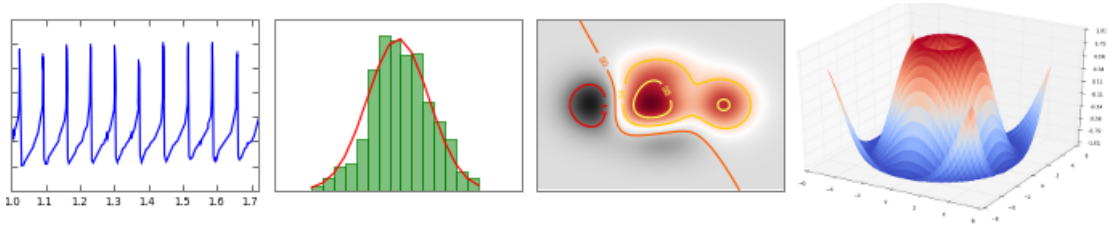
```
>>> a = np.array([[4, 3, 5], [1, 2, 1]])
>>> b = np.sort(a, axis=1)
>>> b
array([[3, 4, 5],
[1, 1, 2]])

>>> a.sort(axis=1)
>>> a
array([[3, 4, 5],
[1, 1, 2]])
```

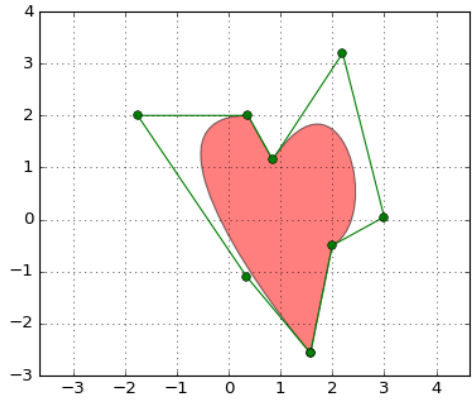
Array stacking

- Enables concatenating two arrays vertically (***vstack***) or horizontally (***hstack***)

```
>>> a = np.array((1,2,3))
>>> b = np.array((2,3,4))
>>> np.hstack((a,b))
array([1, 2, 3, 2, 3, 4])
>>> a = np.array([[1],[2],[3]])
>>> b = np.array([[2],[3],[4]])
>>> np.hstack((a,b))
array([[1, 2],
       [2, 3],
       [3, 4]])
```

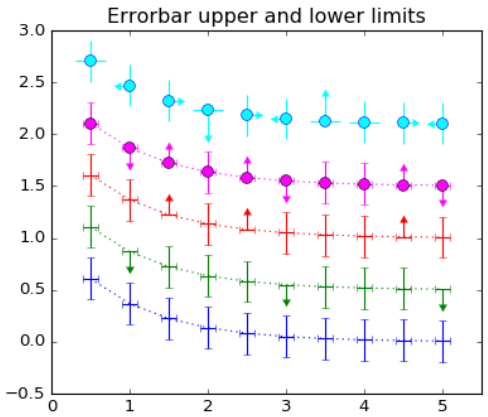
```
>>> a = np.array([1, 2, 3])
>>> b = np.array([2, 3, 4])
>>> np.vstack((a,b))
array([[1, 2, 3],
       [2, 3, 4]])
>>> a = np.array([[1], [2], [3]])
>>> b = np.array([[2], [3], [4]])
>>> np.vstack((a,b))
array([[1],
       [2],
       [3],
       [2],
       [3],
       [4]])
```



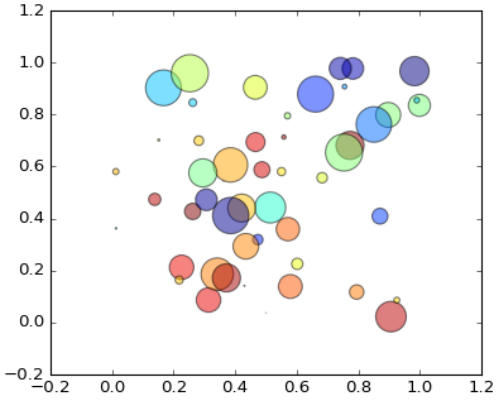


Plotting

matplotlib

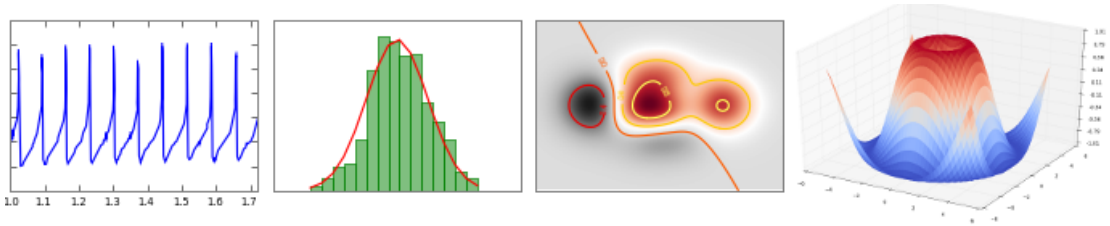


"A plot is worth a thousand words"



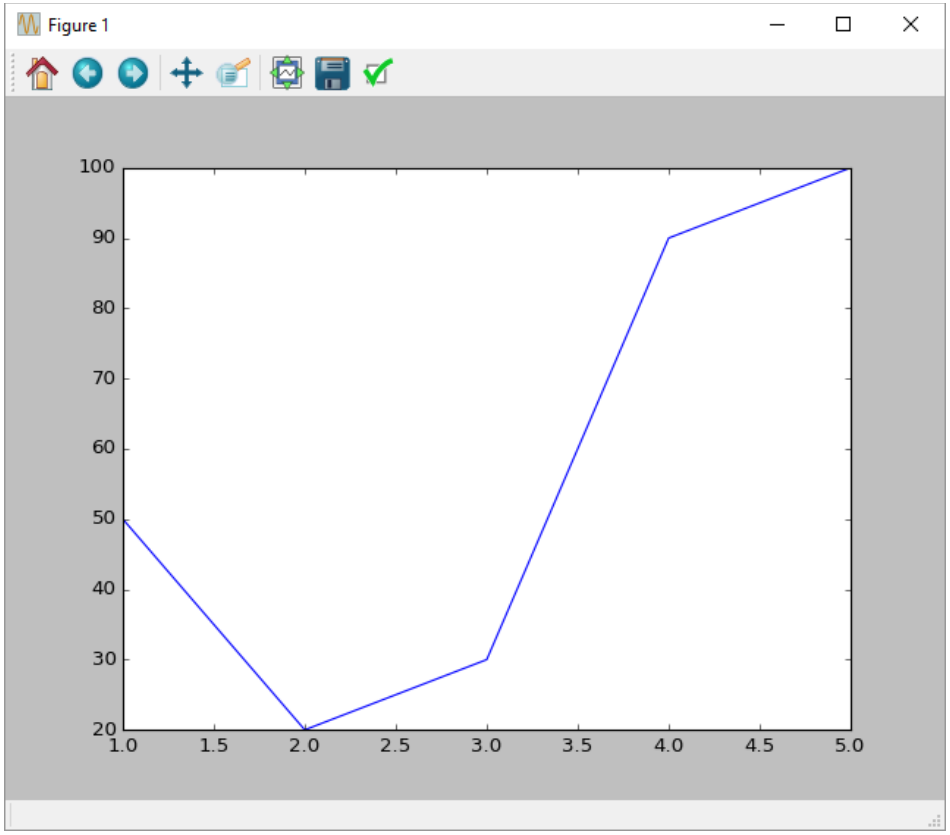
matplotlib

- **matplotlib** is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive.
- You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.
- Check this out: http://matplotlib.org/gallery.html



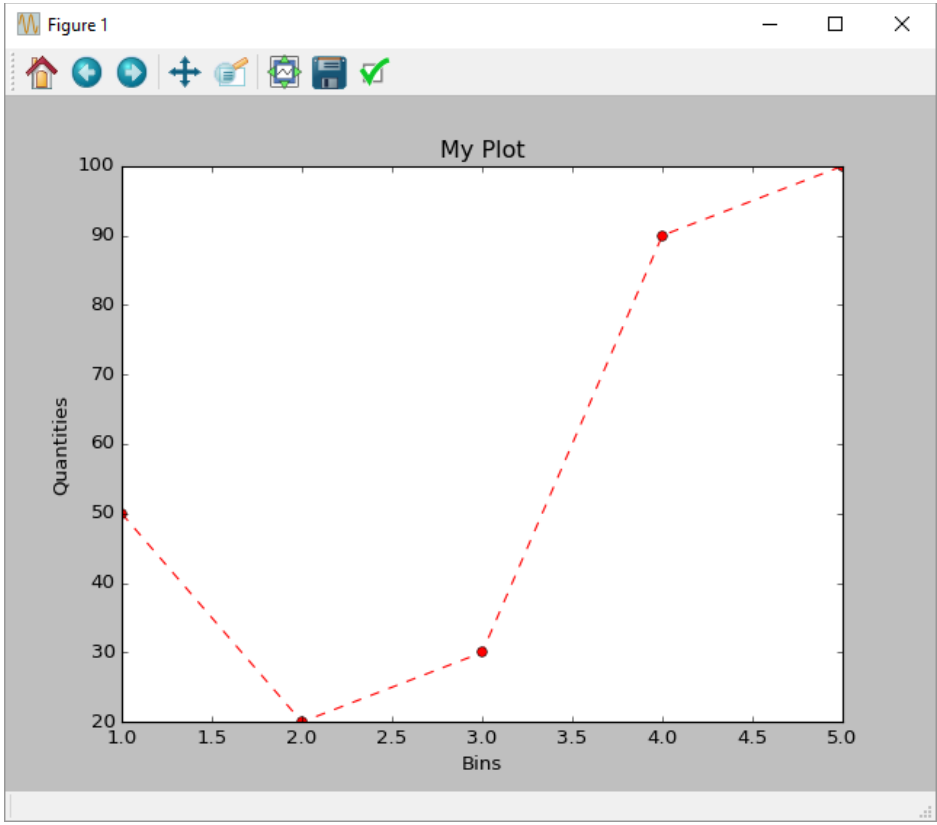
Example 1

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

x = np.array[5 ,4 ,3 ,2 ,1)
y = np.array([100 ,90 ,30 ,20 ,50
plt.plot( x,y)
plt.show()
```



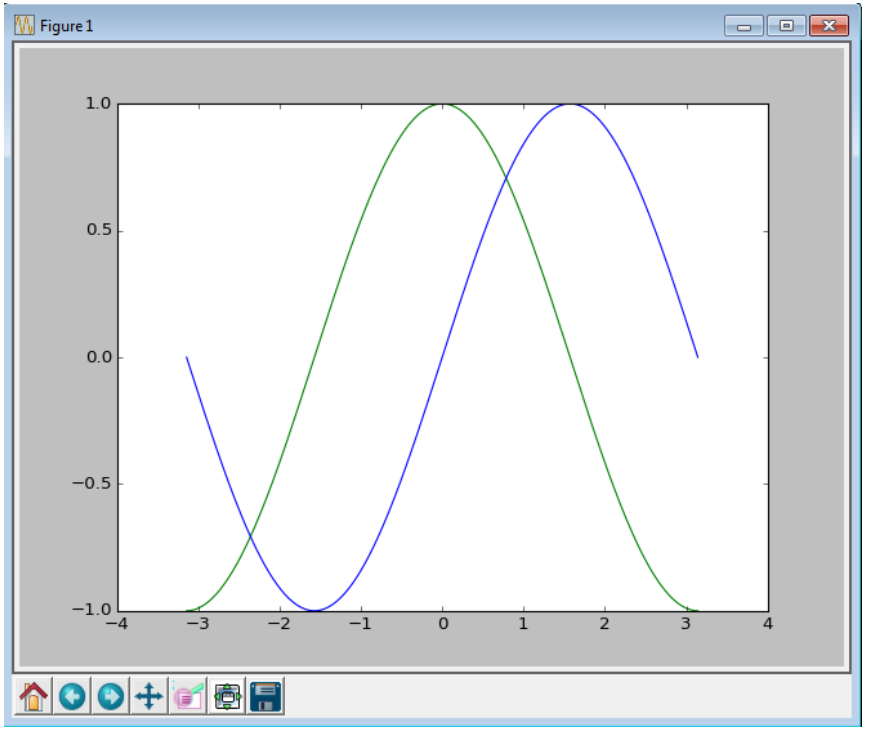
Example 1b

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

x = np.array ([1, 2, 3, 4, 5])
y = np.array ([50, 20, 30, 90, 100])
plt.plot (x,y,color='red',ls='--',marker='o')
plt.xlabel ('Bins')
plt.ylabel ('Quantities')
plt.title ('My Plot')
plt.show()
```



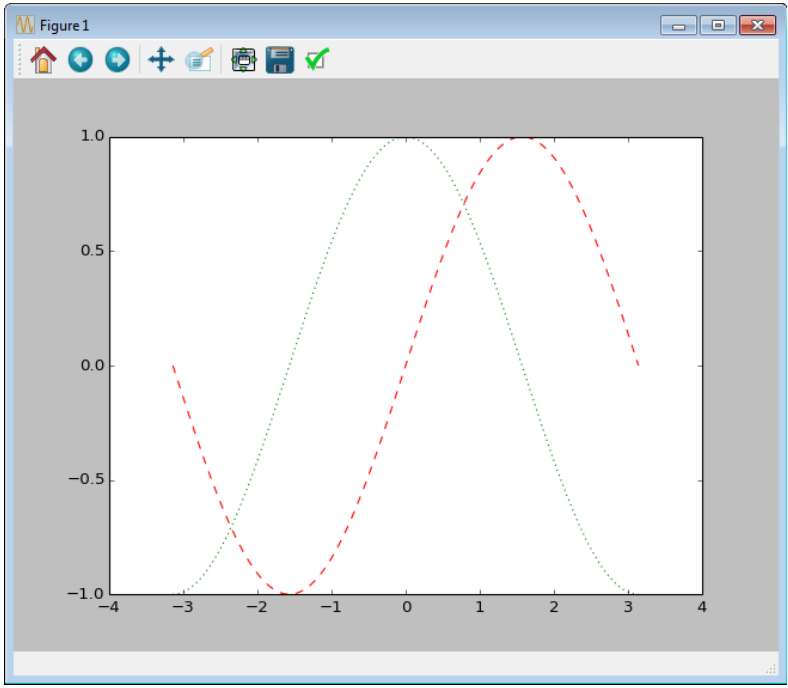
Example 2

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

x = np.linspace(-np.pi, np.pi)
y, z = np.cos(x), np.sin(x)

plt.plot(x, y, color = 'green')
plt.plot(x, z, color = 'blue')
plt.show()
```



Example 2b

```
x = np.linspace(-np.pi, np.pi)
y, z = np.cos(x), np.sin(x)
plt.plot(x, y, color = 'green')
plt.plot(x, z, color = 'r--')
plt.show()
```



character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker

Data analysis using Python

Real data analysis example

- Look at the file inflammation-01.csv (download the lecture's code)
- Tables as CSV files
  - Text files
  - Each row holds the same number of columns
  - Values are separated by commas

```
0,0,1,3,1,2,4,7,8,3,3,3,10,5,7,4,7,7,12,18,6,13,11,11,7,7,4,6,8,8,4,4,5,7,3,4,2,3,0,0
0,1,2,1,2,1,3,2,2,6,10,11,5,9,4,4,7,16,8,6,18,4,12,5,12,7,11,5,11,3,3,5,4,4,5,5,1,1,0,1
0,1,1,3,3,2,6,2,5,9,5,7,4,5,4,15,5,11,9,10,19,14,12,17,7,12,11,7,4,2,10,5,4,2,2,3,2,2,1,1
0,0,2,0,4,2,2,1,6,7,10,7,9,13,8,8,15,10,10,7,17,4,4,7,6,15,6,4,9,11,3,5,6,3,3,4,2,3,2,1
0,1,1,3,3,1,3,5,2,4,4,7,6,5,3,10,8,10,6,17,9,14,9,7,13,9,12,6,7,7,9,6,3,2,2,4,2,0,1,1
```

Real data analysis example

- Look at the file inflammation-01.csv
- The data: inflammation level in patients after a treatment (CSV) format
  - Each row holds information for a single patient
  - The columns represent successive days.
  - The first few lines in the file:

Days

Patients

0,0,1,3,1,2,4,7,8,3,3,3,10,5,7,4,7,7,12,18,6,13,11,11,7,7,4,6,8,8,4,4,5,7,3,4,2,3,0,0
0,1,2,1,2,1,3,2,2,6,10,11,5,9,4,4,7,16,8,6,18,4,12,5,12,7,11,5,11,3,3,5,4,4,5,5,1,1,0,1
0,1,1,3,3,2,6,2,5,9,5,7,4,5,4,15,5,11,9,10,19,14,12,17,7,12,11,7,4,2,10,5,4,2,2,3,2,2,1,1
0,0,2,0,4,2,2,1,6,7,10,7,9,13,8,8,15,10,10,7,17,4,4,7,6,15,6,4,9,11,3,5,6,3,3,4,2,3,2,1
0,1,1,3,3,1,3,5,2,4,4,7,6,5,3,10,8,10,6,17,9,14,9,7,13,9,12,6,7,7,9,6,3,2,2,4,2,0,1,1

Reading the data using NumPy

```
import numpy as np
fname = "inflammation-01.csv"
data = np.loadtxt(fname, delimiter=',')
print(data)
[[ 0. 0. 1. ..., 3. 0. 0.]
 [ 0. 1. 2. ..., 1. 0. 1.]
 [ 0. 1. 1. ..., 2. 1. 1.]
 ...,
 [ 0. 1. 1. ..., 1. 1. 1.]
 [ 0. 0. 0. ..., 0. 2. 0.]
 [ 0. 0. 1. ..., 1. 1. 0.]]

# properties of the data
print(type(data))
print(data.shape)
print("first value in data",data[0,0])
<type 'numpy.ndarray'>
(60L, 40L)
('first value in data', 0.0)
```

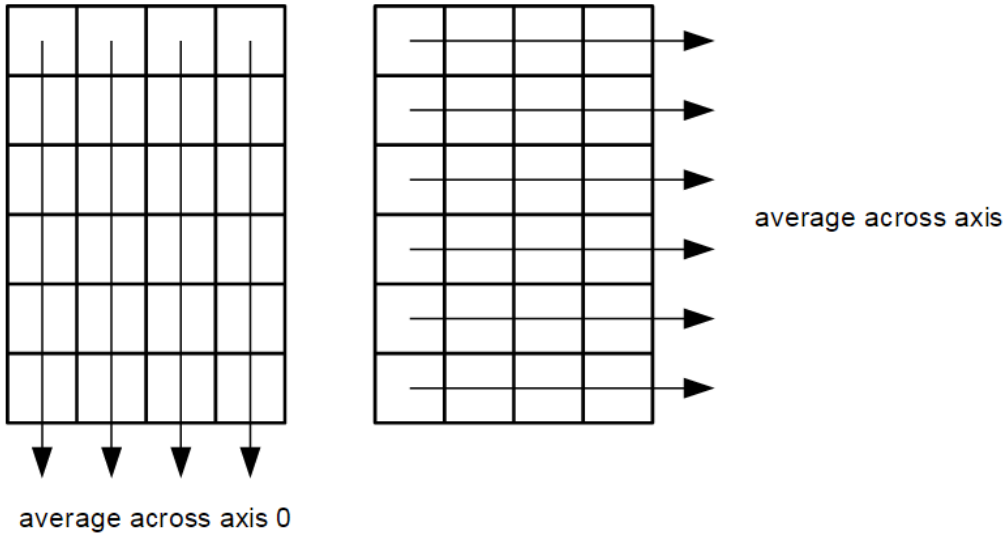
Tasks

- Remove the first and last 10 days
- Plot the average, min, and max inflammation score per day

Data trimming and plots

```
# remove the 10 first and last days
n,m = data.shape
data = data[:,10:(m-10)]
```

How can we get the average/min/max of each day?



Inflammation values per day

```
import matplotlib.pyplot as plt
plt.plot(data.mean(axis=0), label='avg')
plt.plot(data.max(axis=0), label='max')
plt.plot(data.min(axis=0), label='min')
plt.title('inflammation per day')
plt.legend()
plt.show()
```