UC Berkeley Extension

Python for Data Analysis and Scientific Computing

X433.3 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

Python for Data Analysis and Scientific Computing

Lecture 5 part 2 …

… show project presentations ...

Class exercise

- Solution 1/2

hw3_alex.py | lecture5.py | lecture6.py | Midterm.py

```
# ==========================
# Part 1 – create your data:
# ==========================

# 1. Include a section line with your name:
## HW 2: Alexander Iliev

# 2. Work only with these imports:
from numpy import matrix, array, random, min, max
import pylab as plb

# 3. Cerate a list A of 600 random numbers bound between (0:10):
A = list(random.random(600)*10)

# 4. Create an array B with 500 elements bound in the range [-3*pi:2*pi]:
B = plb.linspace(-plb.pi*3, plb.pi*2, 500)

# 5. Using if, for or while, create a function that overwrites every element
# in A that falls outside of the interval [2:9), and overwrite that element with
# the average between the smallest and largest element in A:
index = 0;
def my_function(x):
    for index, k in enumerate(x, start=0):
        if k < 2:
            x[index] = (min(x)+max(x))/2
        elif k >= 9:
            x[index] = (min(x)+max(x))/2
        # 6. Normalize each list element to be bound between [0:0.1]:
        x[index] = x[index] / 100
    return(x)

# 7. Return the result from the function to C:
C = my_function(A)

# 8. Cast C as an array:
C = array(C)

# 9. Add C to B (think of C as noise) and record the result in D:
D = C[0:len(B)] + B
```

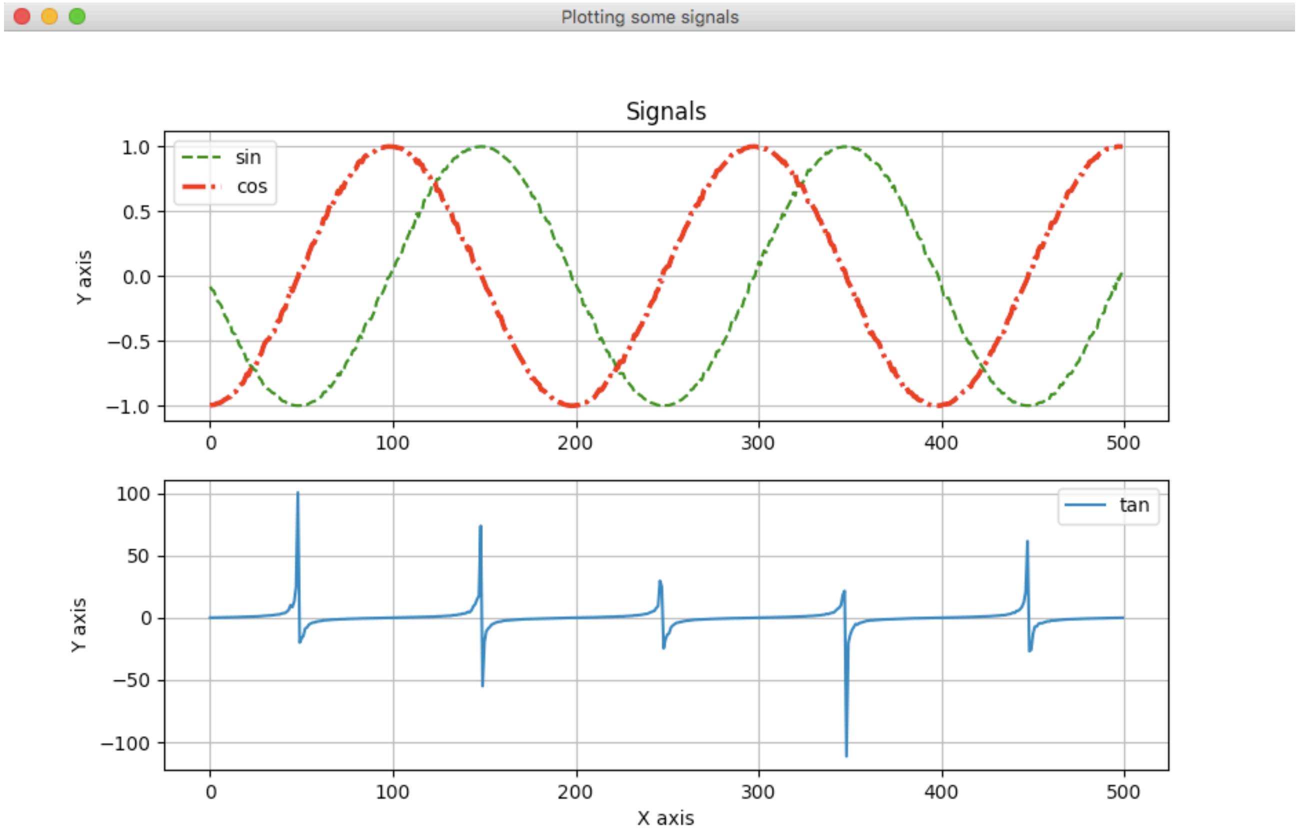
Class exercise

- Solution 2/2

```
# ==========================
# Part 2 - plotting:
# ==========================

# 10. Create a figure, give it a title and specify your own size and dpi:
plb.figure('Plotting some signals', figsize=(6,4), dpi=100)

# 11. Plot the sin of D, in the (2,1,1) location of the figure:
plb.subplot(2,1,1), plb.plot(plb.sin(D), color="green", linewidth=1.5, linestyle="--", label='sin')

# 12. Overlay a plot of cos of D, with different color, thickness and type of
# line:
plb.plot(plb.cos(D), color="red", linewidth=2.5, linestyle="-.", label='cos')

# 13. Create some space on top and bottom of the plot (on the y axis) and show
# the grid:
plb.ylim(-1.12, 1.12), plb.grid()

# 14. Specify the following: title, Y-axis label and legend to fit in the best way:
plb.ylabel('Y axis'), plb.title('Signals'), plb.legend(loc='best')

# 15. Plot the tan of D, in location (2,1,2) with grid showing, X-axis label,
# Y-axis label and legend on top right:
plb.subplot(2,1,2), plb.plot(plb.tan(D), label='tan'), plb.grid()
plb.xlabel('X axis'), plb.ylabel('Y axis'), plb.legend(loc='upper right')
```

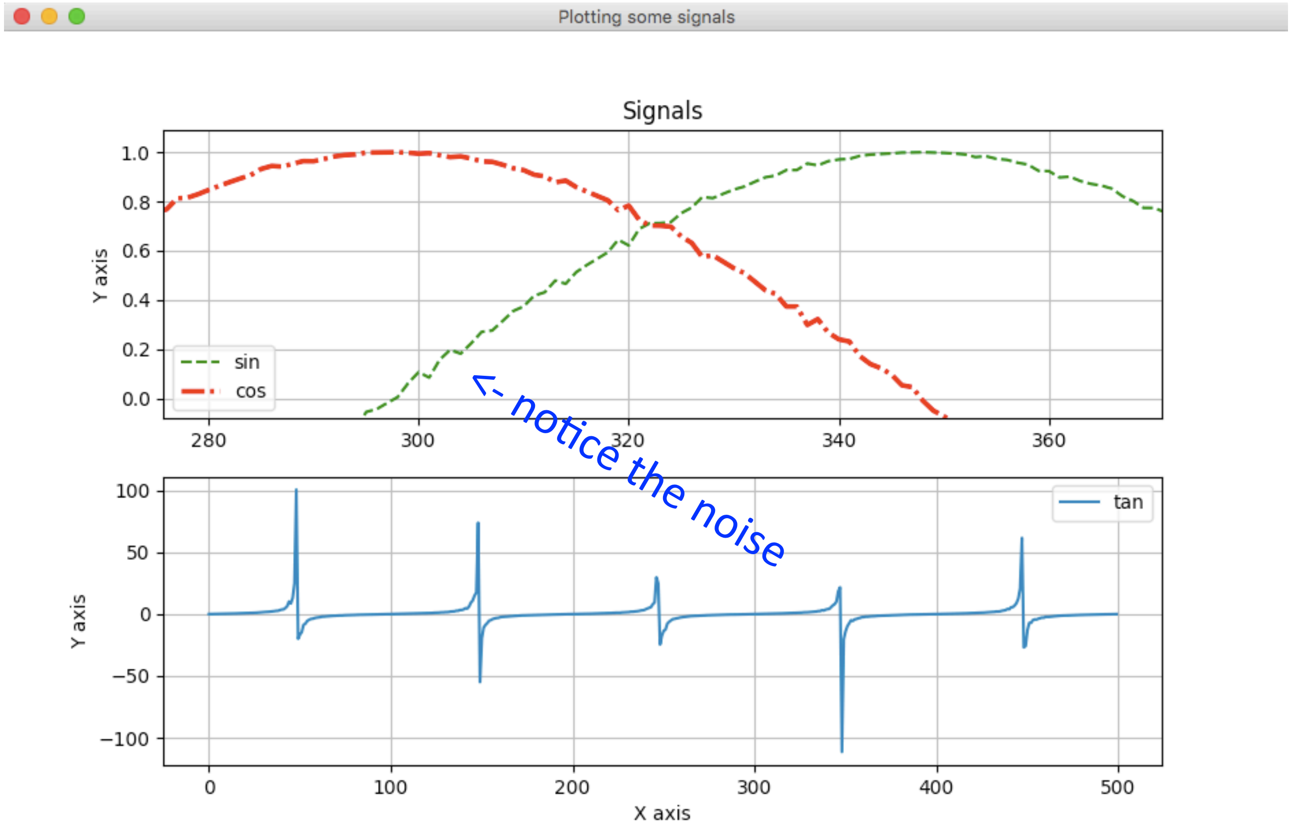
Class exercise

- Solution plots



Class exercise

- Solution plots



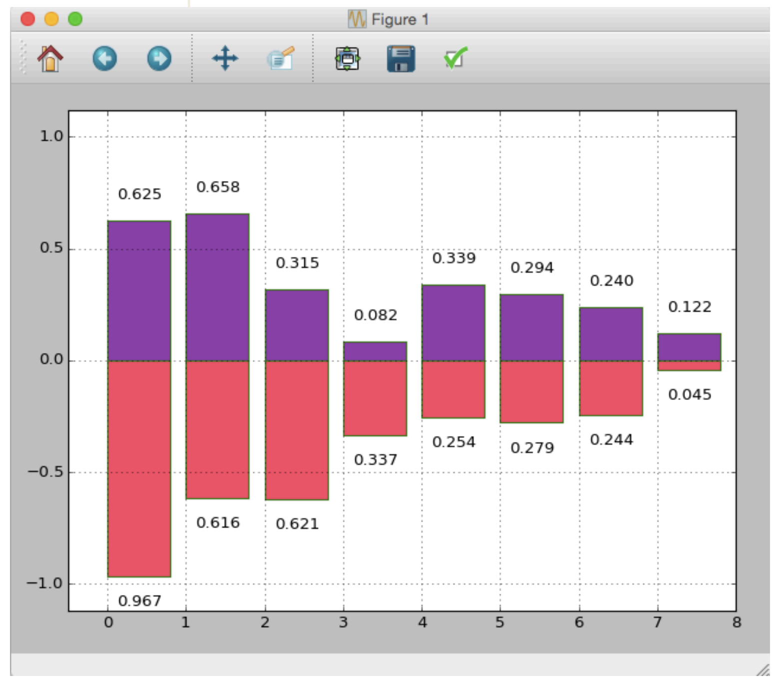
Advanced plotting

- Advanced plotting
  - There are many different types of 2-D and 3-D plotting choices in Matplotlib
  - Matplotlib provides a huge variety of plots aiming at specific visualization of results
  - Some of these types are:
    - Bar plots
    - Scatter plots
    - Imshow
    - Histogram
    - Pie charts
    - Contour plots
    - Polar axis
    - 3-D plots
    - *Text plots*
    - *Grids*
    - *Quiver plots*

Advanced plotting

- Advanced plotting: *bar plot*

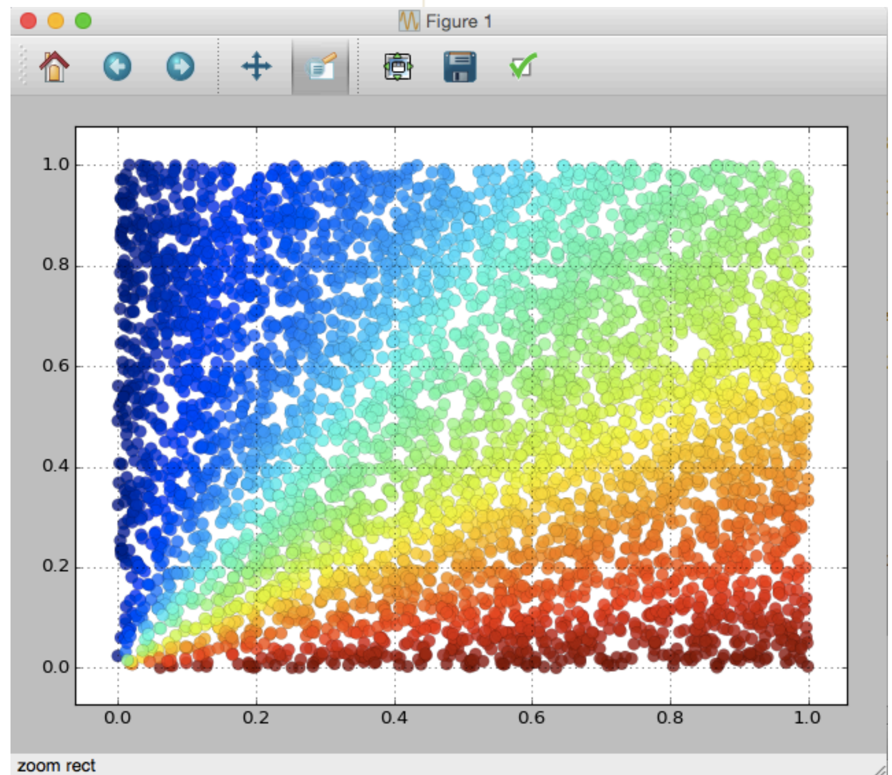
```python
## Advanced plotting:
# Bar plot:
import pylab as plb

k = 8
x = plb.arange(k)
y1 = plb.rand(k) * (1 - x / k)
y2 = plb.rand(k) * (1 - x / k)
plb.axes([0.075, 0.075 , .88, .88])

plb.bar(x, +y1, facecolor='#9922aa', edgecolor='green')
plb.bar(x, -y2, facecolor='#ff3366', edgecolor='green')

for a, b in zip(x, y1):
    plb.text(a+0.41, b+0.08, '%.3f' % b, ha='center', va='bottom')
for a, b in zip(x, y2):
    plb.text(a+0.41, -b-0.08, '%.3f' % b, ha='center', va= 'top')

plb.xlim(-.5, k), plb.ylim(-1.12, +1.12)
plb.grid(True)
plb.show()
```



Advanced plotting

- Advanced plotting: *scatter plot*

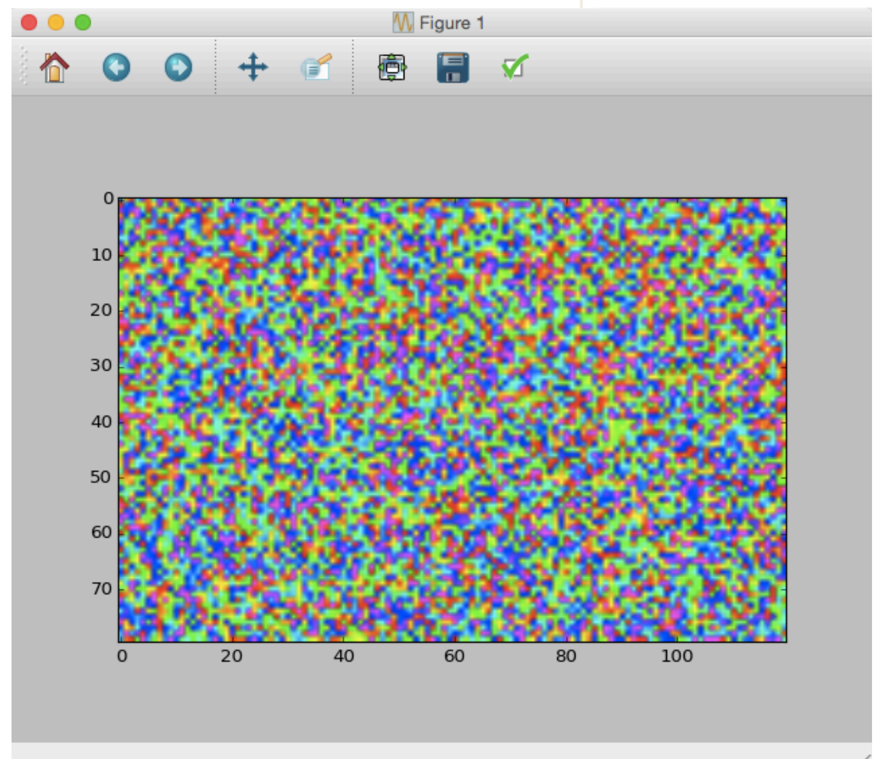
```python
## Scatter plot:
import pylab as plb

x = plb.rand(1,2,1500)
y = plb.rand(1,2,1500)
plb.axes([0.075, 0.075 , .88, .88])

plb.cla()   # clear the current axis
plb.scatter(x, y, s=65, alpha=.75, linewidth=.125,
    c=plb.arctan2(x, y))

plb.grid(True)
plb.xlim(-0.085,1.085), plb.ylim(-0.085,1.085)
plb.pause(1)
```



Advanced plotting

- Advanced plotting: *image plot*

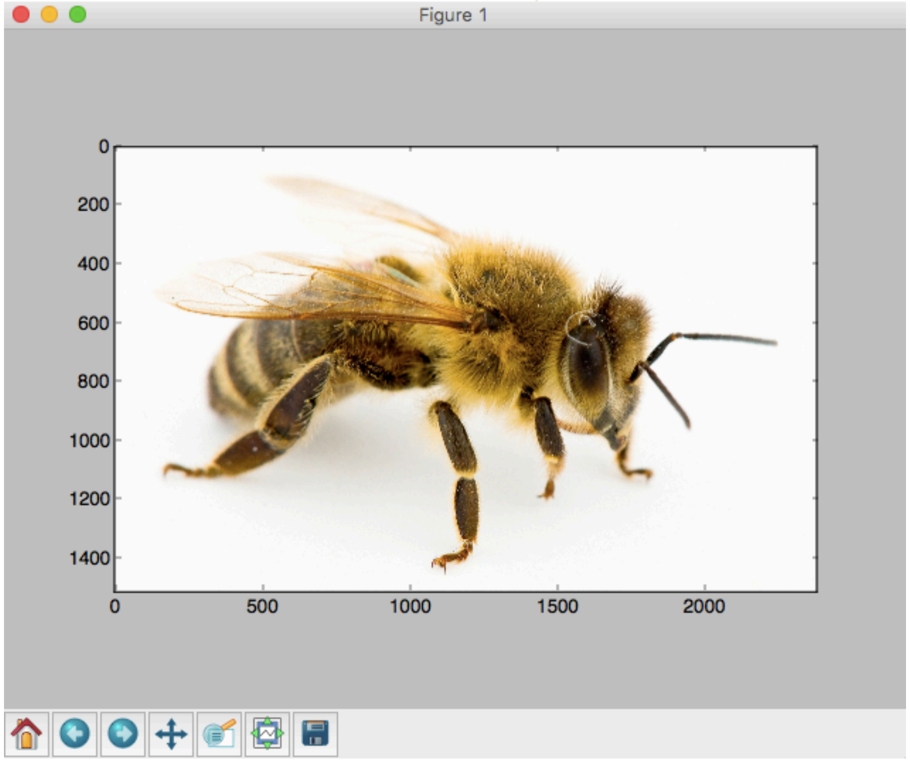
```
## Image 1/2:
plb.cla()
array = plb.random((80, 120))
plb.imshow(array, cmap=plb.cm.gist_rainbow) # with a specific colormap
plb.pause(1)
```



Advanced plotting

- Advanced plotting: *image plot*

```python
## Image 2/2:
import matplotlib.image as img
import matplotlib.pyplot as plt

image = img.imread('files/lecture5/bee.jpg')
plt.imshow(image)
plt.pause(1)
```



Advanced plotting

- Advanced plotting: *image plot 1/3*

```
# luminosity display using 1-channel only (no RGB color).
# A default colormap (lookup tabel = LUT) is applied called 'jet':
luminosity = image[:,:,0]
plt.imshow(luminosity)
plt.pause(5)

# Other colormaps can be:
plt.imshow(luminosity, cmap="hot")
plt.pause(5)
plt.imshow(luminosity, cmap="spectral")
plt.pause(5)
```

Advanced plotting

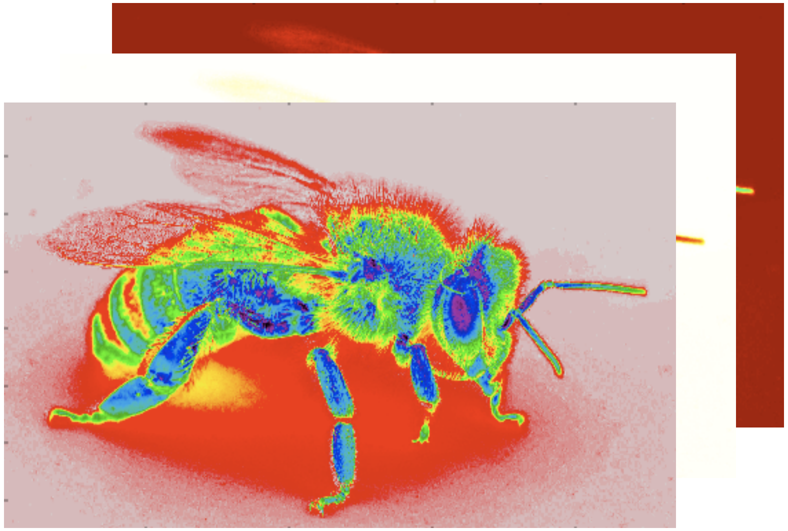
- Advanced plotting: *image plot 2/3*

```
# luminosity display using 1-channel only (no RGB color).
# A default colormap (lookup tabel = LUT) is applied called 'jet':
luminosity = image[:,:,0]
plt.imshow(luminosity)
plt.pause(5)

# Other colormaps can be:
plt.imshow(luminosity, cmap="hot")
plt.pause(5)
plt.imshow(luminosity, cmap="spectral")
plt.pause(5)
```

Advanced plotting

- Advanced plotting: *image plot 3/3*

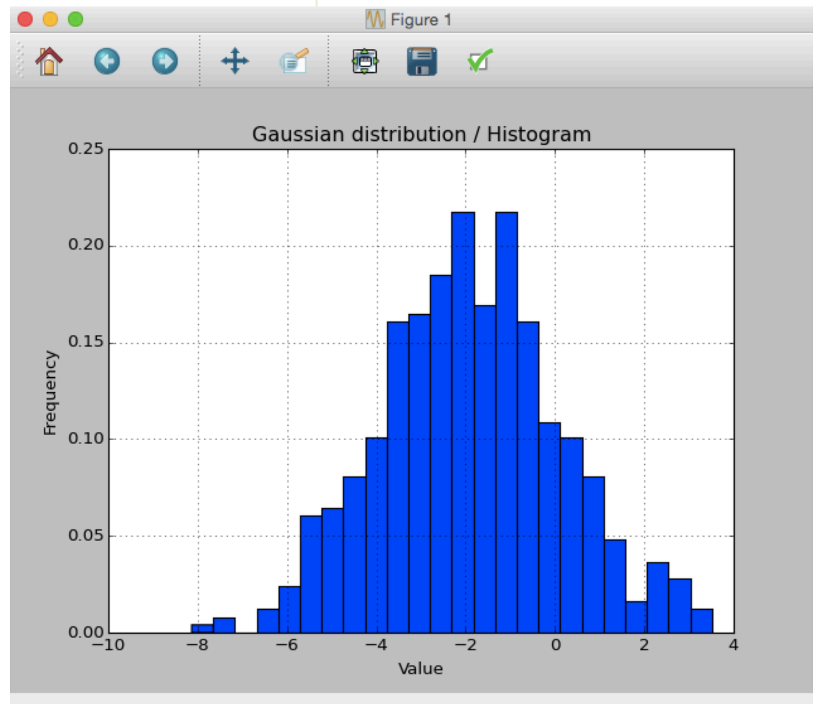
```
# luminosity display using 1-channel only (no RGB color).
# A default colormap (lookup tabel = LUT) is applied called 'jet':
luminosity = image[:,:,0]
plt.imshow(luminosity)
plt.pause(5)

# Other colormaps can be:
plt.imshow(luminosity, cmap="hot")
plt.pause(5)
plt.imshow(luminosity, cmap="spectral")
plt.pause(5)
```



Advanced plotting

- Advanced plotting: *histogram plot 1/2*

```
# Histogram 1/2:
import pylab as plb

plb.figure(1)
gaus_dist = plb.normal(-2,2,size=512) # create a random floating point vector

# plot the histogram with specific: bin number
plb.hist(gaus_dist, normed=True, bins=24) # default: bins=10, color='blue'

plb.title("Gaussian distribution / Histogram")
plb.xlabel("Value")
plb.ylabel("Frequency")
plb.grid(True)
plb.show()
```



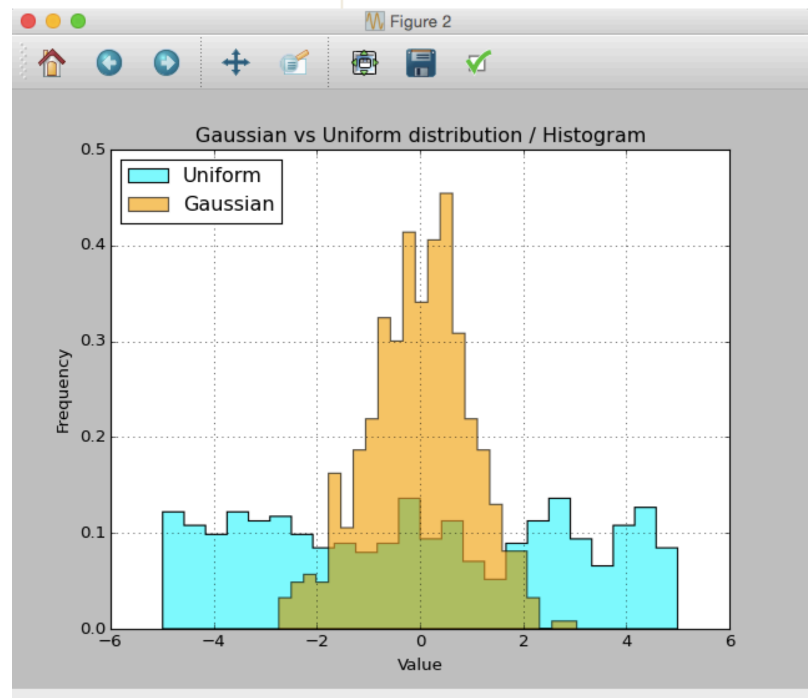
Histogram is great for visualizing statistical distribution of a set of variables in a given pool of samples, divided into classes called bins

Advanced plotting

- Advanced plotting: *histogram plot 2/2*

```
# Histogram 2/2:
plb.figure(2)
gaus_dist = plb.normal(size=512)
unif_dist = plb.uniform(-5,5,size=512) # create uniform distribution vector

# plot the histogram with specific: bin number, color, transparency, label
plb.hist(unif_dist, bins=24, histtype='stepfilled',
            normed=True, color='cyan', label='Uniform')
plb.hist(gaus_dist, bins=24, histtype='stepfilled',
            normed=True, color='orange', label='Gaussian', alpha=0.65)

plb.legend(loc='upper left')
plb.title("Gaussian vs Uniform distribution / Histogram")
plb.xlabel("Value")
plb.ylabel("Frequency")
plb.grid(True)
plb.show()
```

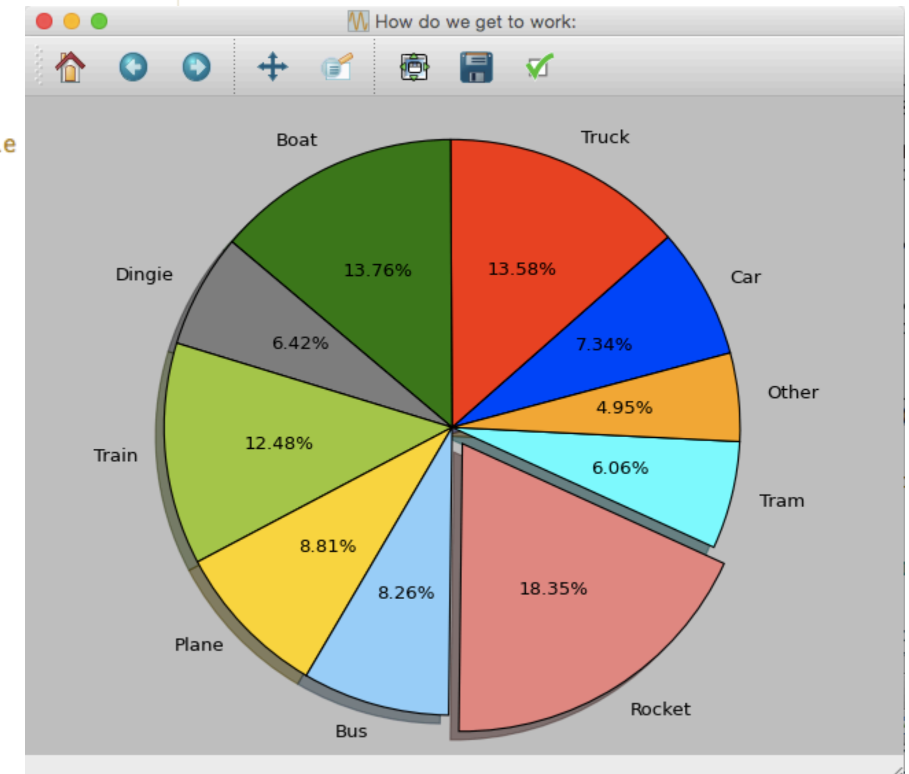
Histogram is great for visualizing statistical distribution of a set of variables in a given pool of samples, divided into classes called bins



Advanced plotting

- Advanced plotting: *pie chart*

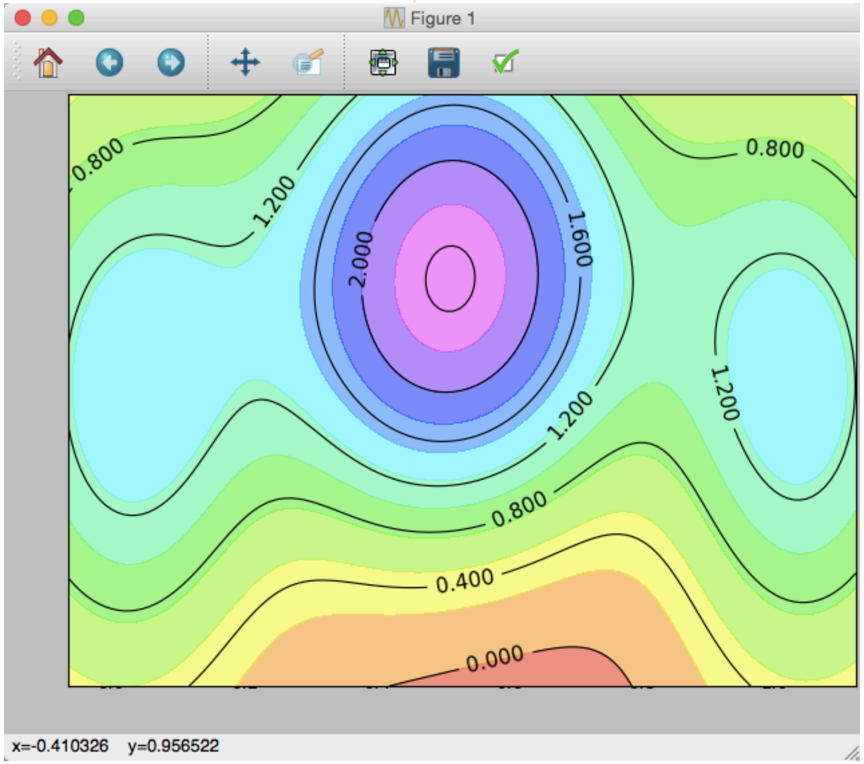
```python
# Pie chart:
plb.figure('How do we get to work:')
plb.axes([0.035, 0.035, 0.9, 0.9])

l = 'Car', 'Truck', 'Boat', 'Dingie', 'Train', 'Plane', 'Bus', 'Rocket', 'Tram', 'Other'
b = plb.round_(plb.random(10), decimals=2)
c = ['blue', 'red', 'green', 'gray', 'yellowgreen',
     'gold', 'lightskyblue', 'lightcoral', 'cyan', 'orange']
e = (0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0) # 'explode' the 8th slice only - 'Rocket'

plb.cla()
plb.pie(b, explode = e, labels=l, colors=c, radius=.75,
        autopct='%1.2f%%', shadow=True, startangle=15)

# we set the aspect ratio to 'equal' so the pie is drawn in a circle
plb.axis('equal')
plb.xticks(()); plb.yticks(())
plb.show()
```



Advanced plotting

- Advanced plotting: *contour plot*

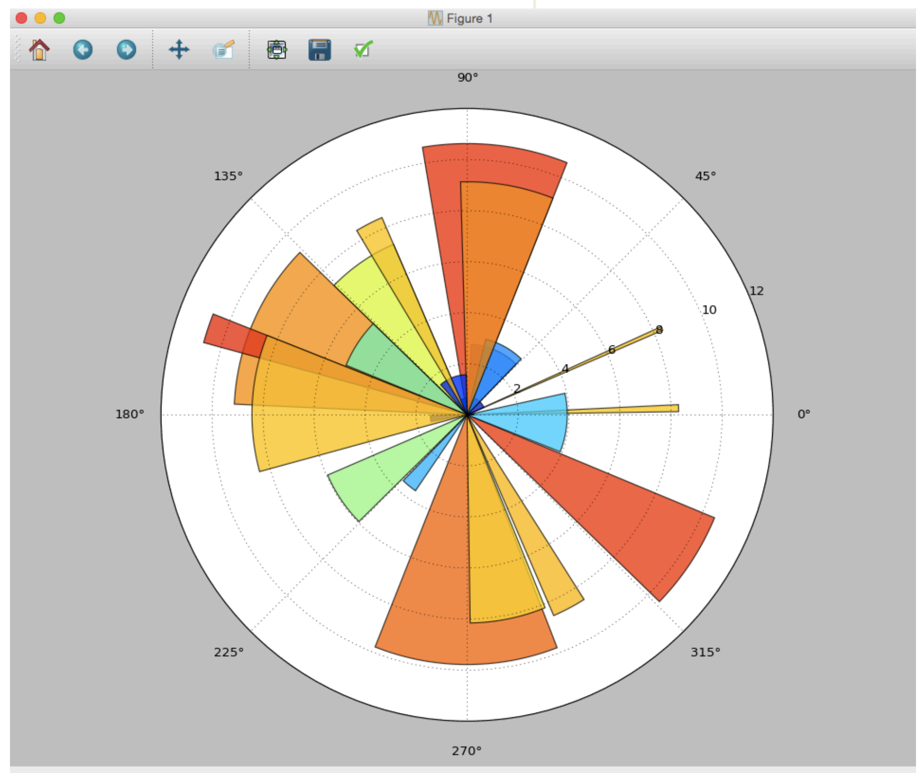
```python
# Contour plot:
import pylab as plb

def f(x,y):
    return (2 - x/3 + x**6 + 2.125*y) * plb.exp(-x**2 -y**2)

n = 128
x = plb.linspace(-2, 2, n)
y = plb.linspace(-1, 1, n)
A,B = plb.meshgrid(x, y)

plb.cla()
plb.axes([0.075, 0.075, 0.92, 0.92])

plb.contourf(A, B, f(A, B), 12, alpha=.50,
    cmap=plb.cm.gist_rainbow)
C = plb.contour(A, B, f(A, B), 8, colors='black',
    linewidth=.65)

plb.clabel(C, inline=1, fontsize=14)
plb.xticks(()); plb.yticks(())
plb.show()
```



Advanced plotting

- Advanced plotting: *polar plot*

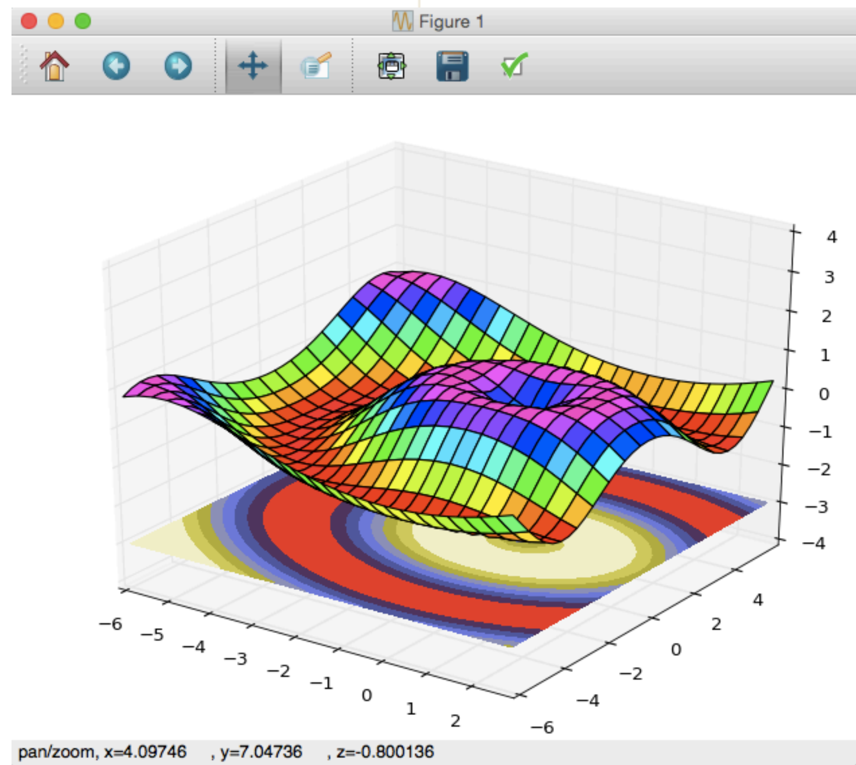
```
# Polar plot:
import pylab as plb

 plb.axes([0.065, 0.065, 0.88, 0.88], polar=True)

q = 24
t = plb.arange(0.015, 3*plb.pi, 3 * plb.pi / q)
rad = 12 * plb.rand(q)
w = plb.pi / 4 * plb.rand(q)
ba = plb.bar(t, rad, width = w)

for r,bar in zip(rad, ba):
    bar.set_facecolor(plb.cm.jet(r/12.))
    bar.set_alpha(0.75)

plb.show()
```



Advanced plotting

- Advanced plotting: *3-D plot*

```python
# 3-D plot:
import pylab as plb
from mpl_toolkits.mplot3d import Axes3D

ax = Axes3D(plb.figure())
x = plb.arange(-6, 3, 0.35)
y = plb.arange(-6, 6, 0.35)
x, y = plb.meshgrid(x, y)
k = plb.sqrt(x**2 + y**2)
z = plb.sin(k)

ax.plot_surface(x, y, z, rstride=2, cstride=1,
                cmap=plb.cm.gist_rainbow)
ax.contourf(x, y, z, zdir='z', offset=-3,
                cmap=plb.cm.gist_stern)
ax.set_zlim(-4, 4)

plb.show()
```



Discussion

Important:

- - dictionary keys are not sorted in Python: Standard Python dictionaries are NOT ordered. Even if you sorted the (key,value) pairs, preserving the order in dict is not possible. However, to get the order numerically or alphabetically (1st letter) based on the keys use the following:

```
In [1]: import collections
In [2]: a = {2:3, 1:89, 4:5, 3:0}
In [3]: ord = collections.OrderedDict(sorted(a.items()))
In [4]: ord
Out[4]: OrderedDict([(1, 89), (2, 3), (3, 0), (4, 5)])
```

- - to access dictionary keys if they are numbers and were assigned to a dict_key variable:

  the dict_values object does not support indexing, The method dict().values() is quite different between python 3 and python 2. <u>In python 2 it returns a list, and in python 3 it doesn't.</u> To fix this we cast to list:

```
In [41]: d = {1: 89, 2: 3, 3: 0, 4: 5}
In [42]: h = d.values()
In [43]: h
Out[43]: dict_values([89, 3, 0, 5])
In [44]: list(h)[0]
Out[44]: 89
```

Discussion

- What is a moving average?
  - In general it is a series of averages of different interval subsets of data points out of a full data set
  - Simple Moving Average (SMA) – is defined as unweighted mean of the previous data. Used in stock market to calculate the average closing price of a stock over specific time interval
  - Exponential Moving Average (EMA) – a.k.a. exponentially weighted moving average (EWMA), applies weighting factors which decrease exponentially and is a type of an IIR filter where the weighting for each older data decreases exponentially, but never reaching zero or: $(0 < \alpha <= 1]$
  - Cumulative Moving Average (CMA) – is an average of all presented data up to the current point
  - Weighted Moving Average (WMA) – it has multiplying factors serving as different weights to the data at different positions of a given sample frame
  - Modified Moving Average (MMA) – or running moving average (RMA), or smoothed moving average (SMMA) is like an exponential moving average, but with different degree of weighting decrease α

Class exercise

1. Import paylab
2. Create a dictionary with 5 keys and empty values in A
3. Using a function, assign random values to each key between [0:10], using a for loop and return the result in B
4. Using a function, change the value of any member in B that is less than 5 with the result from (4.1) (consider using an if statement in a loop):

   4.1 Using normal distribution with mean = 2 and std = 3 create an array of size 256 points

   4.2 Using a histogram with 12 bins, plot the result from 4.1 with a pause of 1 sec. Use proper labeling (figure, title, labels, legend, grid, etc.)
5. Assign the result from 4. in C
6. Update one of the keys in C with another using the pop feature
7. Update another key in C manually (add the new one and delete the old one)
8. Compare A, B and C using a short conditional expression