भारतीय प्रौद्योगिकी संस्थान तिरुपति

TIRUPATI

Python Scipy

Panchatcharam Mariappan

Assistant Professor

Department of Mathematics and Statistics,
IIT Tirupati

- NumPy/SciPy – numerical and scientific function libraries.
- Collection of mathematical algorithms
- NumPy extension
- Interactive python session by providing the user with high-level commands
- Manipulating and visualizing data
- Data processing
- System prototyping like MATLAB, SciLab

Simple Integral example

$$\int_a^b \sin x \, dx$$

scipy.integrate!

Methods for Integrating Functions given a function object:

Method	Explanation
`quad`	General purpose integration
`dblquad`	General purpose double integration
`tplquad`	General purpose triple integration
`fixed_quad`	Integrate function f(x) using Gaussian quadrature of order n
`quadrature`	Integrate with given tolerance using Gaussian quadrature
`romberg`	Integrate function using Romberg integration

Methods for Integrating Functions given a fixed samples:

Method	Explanation
`trapz`	Use trapezoidal rule to compute integral from samples
`cumtrapz`	Use trapezoidal rule to cumulatively compute integral
`simps`	Use Simpson's rule to compute integral from samples
`romb`	Use Romberg Integration to compute integral from (2**k + 1) evenly-spaced samples

Methods for Integrating Functions given a fixed samples:

```
>>> result = scipy.integrate.quad(np.sin, 0, np.pi)
>>> print(result)
(2.0, 2.220446049250313e-14) # 2 with a very small error margin!
>>> result = scipy.integrate.quad(np.sin, -np.inf, +np.inf)
>>> print(result)
(0.0, 0.0) # Integral does not converge
```

Methods for Integrating Functions given a fixed samples:

```
>>> sample_x = np.linspace(0, np.pi, 1000)
>>> sample_y = np.sin(sample_x) # Creating 1,000 samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
1.99999835177
>>> sample_x = np.linspace(0, np.pi, 1000000)
>>> sample_y = np.sin(sample_x) # Creating 1,000,000 samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
2.0
```

भारतीय प्रौद्योगिकी संस्थान तिरुपति

TIRUPATI

Python Matplotlib

Panchatcharam Mariappan

Assistant Professor

Department of Mathematics and Statistics,
IIT Tirupati

- Plotting library
- Huge number of examples for tacking unique problems

Matplotlib-plot function

- `import matplotlib as mpl`
- `import matplotlib.pyplot as plt`
- `plt.plot(x_values, y_values, format_string [, x, y, format, ])`

Matplotlib-plot function

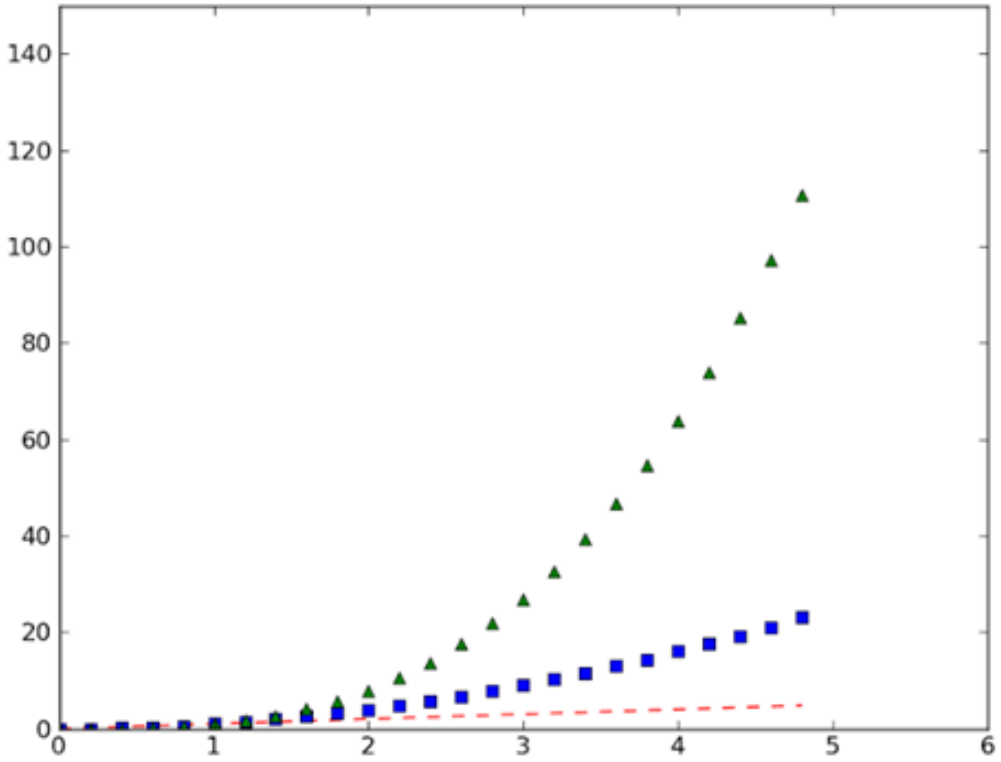
- `import matplotlib as mpl`
- `import matplotlib.pyplot as plt`
- `plt.plot(x_values, y_values, format_string [, x, y, format, ])`
- The format string argument associated with a pair of sequence objects indicates the color and line type of the plot (e.g. ‘bs’ indicates blue squares and ‘ro’ indicates red circles).
- Generally speaking, the x_values and y_values will be numpy arrays and if not, they will be converted to numpy arrays internally.
- Line properties can be set via keyword arguments to the plot function. Examples include label, linewidth, animated, color, etc…

Matplotlib-plot function

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

# evenly sampled time at .2 intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.axis([0, 6, 0, 150]) # x and y range of axis
plt.show()
```

Matplotlib-plot function

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

# evenly sampled time at .2 intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.axis([0, 6, 0, 150]) # x and y range of ax
plt.show()
```



Matplotlib-plot function

- It's important to note that a figure is a separate idea from how it is rendered. Pyplot convenience methods are used for creating figures and immediately displaying them in a pop up window. An alternative way to create this figure is shown below

```
import numpy as np
import matplotlib.figure as figure

t = np.arange(0, 5, .2)

f = figure.Figure()
axes = f.add_subplot(111)
axes.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
axes.axis([0, 6, 0, 150])
```

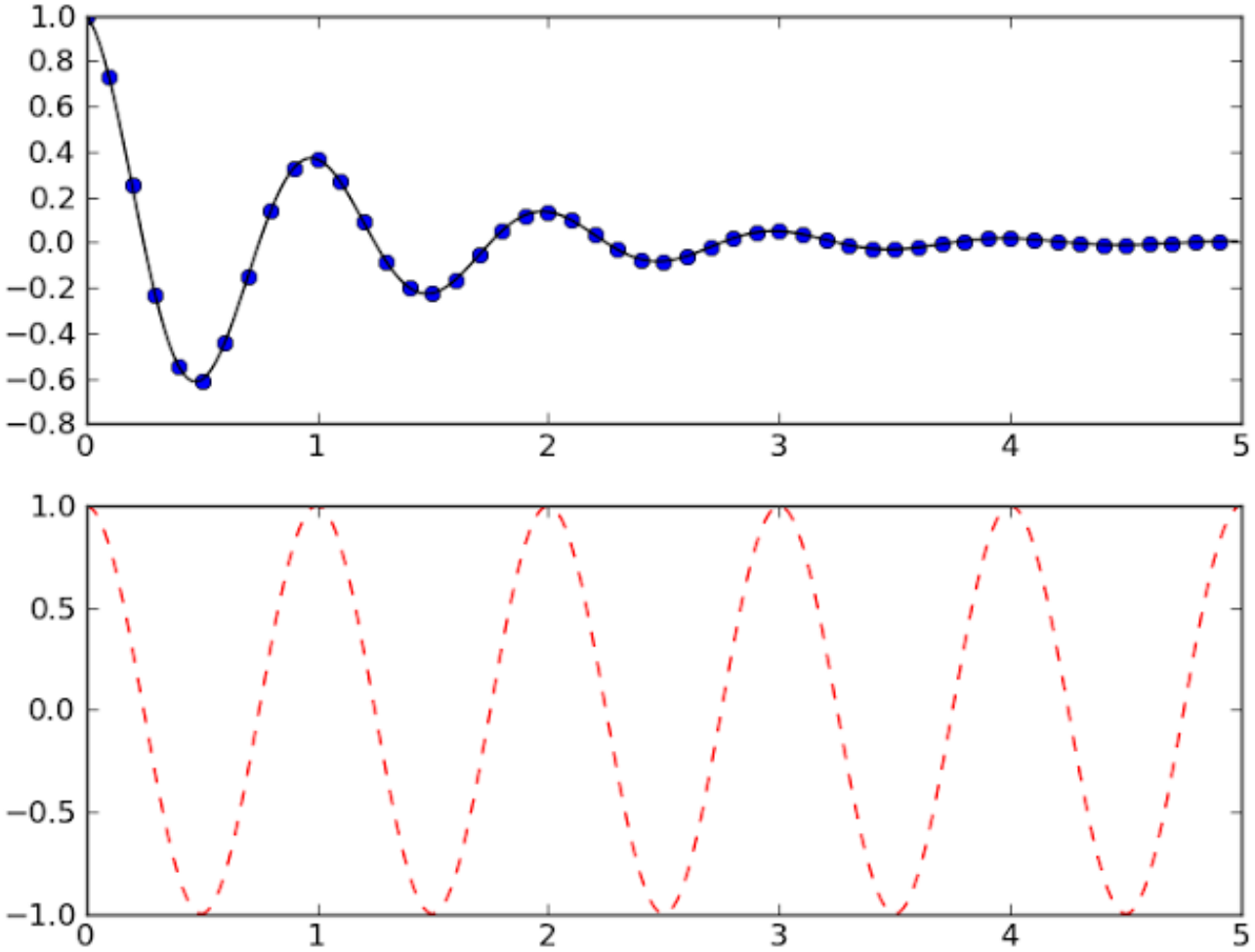
Matplotlib-subplot function

- A script can generate multiple figures, but typically you'll only have one.
- To create multiple plots within a figure, either use the subplot() function which manages the layout of the figure or use add_axes().

Matplotlib-plot function

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

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.figure(1) # Called implicitly but can use
              # for multiple figures
plt.subplot(211) # 2 rows, 1 column, 1st plot
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212) # 2 rows, 1 column, 2nd plot
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Matplotlib-plot function



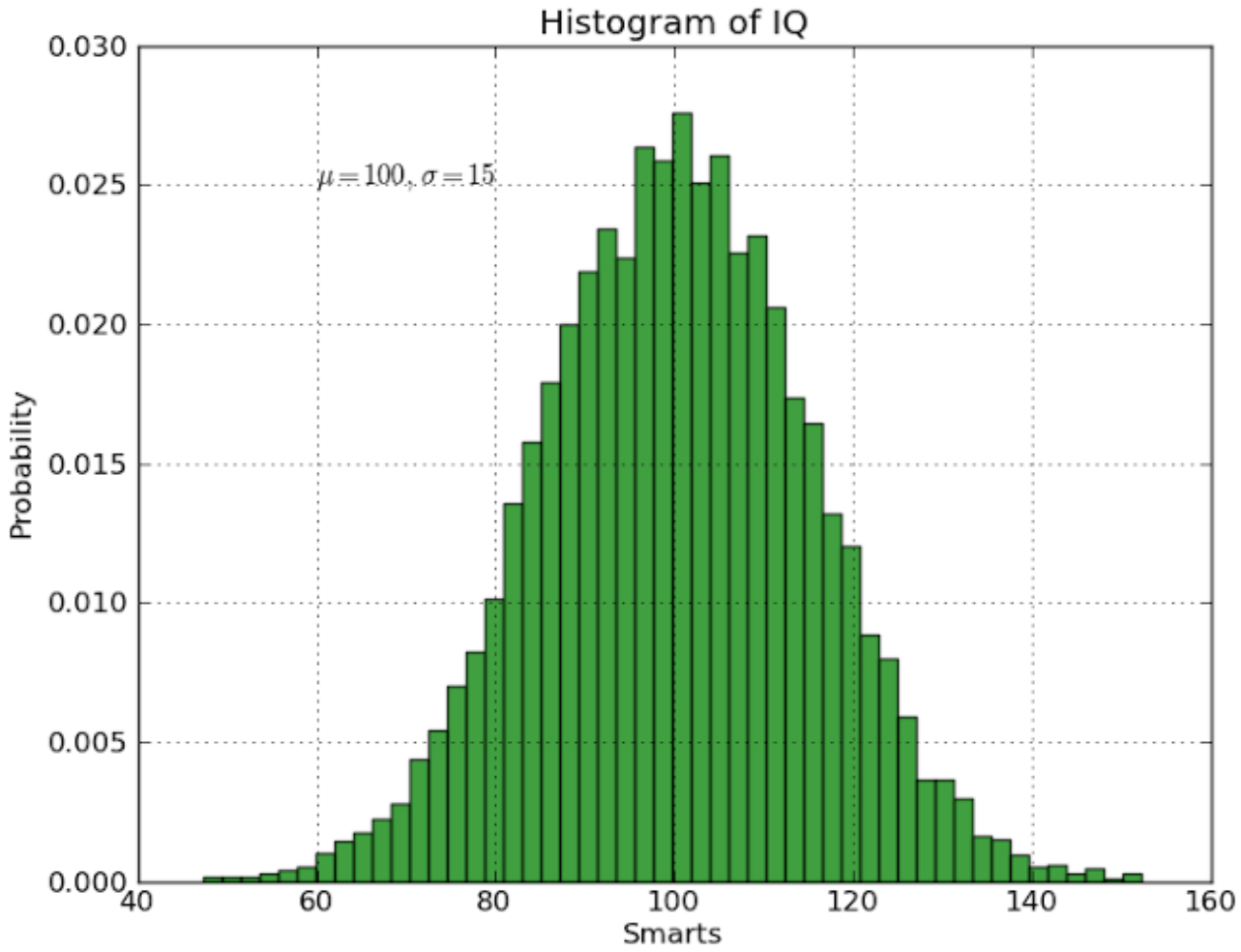
Matplotlib-plot function

- The `text()` command can be used to add text in an arbitrary location
- `xlabel()` adds text to x-axis.
- `ylabel()` adds text to y-axis.
- `title()` adds title to plot.
- `clear()` removes all plots from the axes.
- All methods are available on pyplot and on the axes instance generally.

Matplotlib-plot function

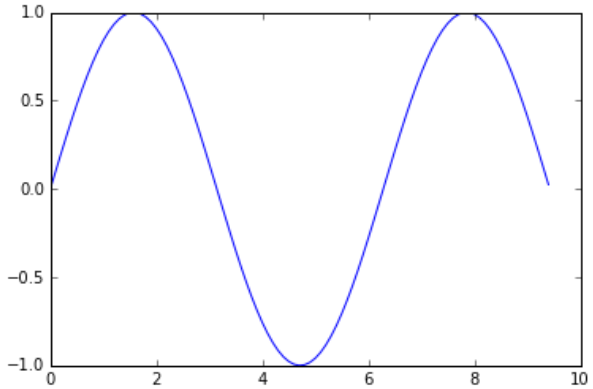
```
import numpy as np
import matplotlib.pyplot as plt
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g',
alpha=0.75)
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$') #TeX equations
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

Matplotlib-plot function



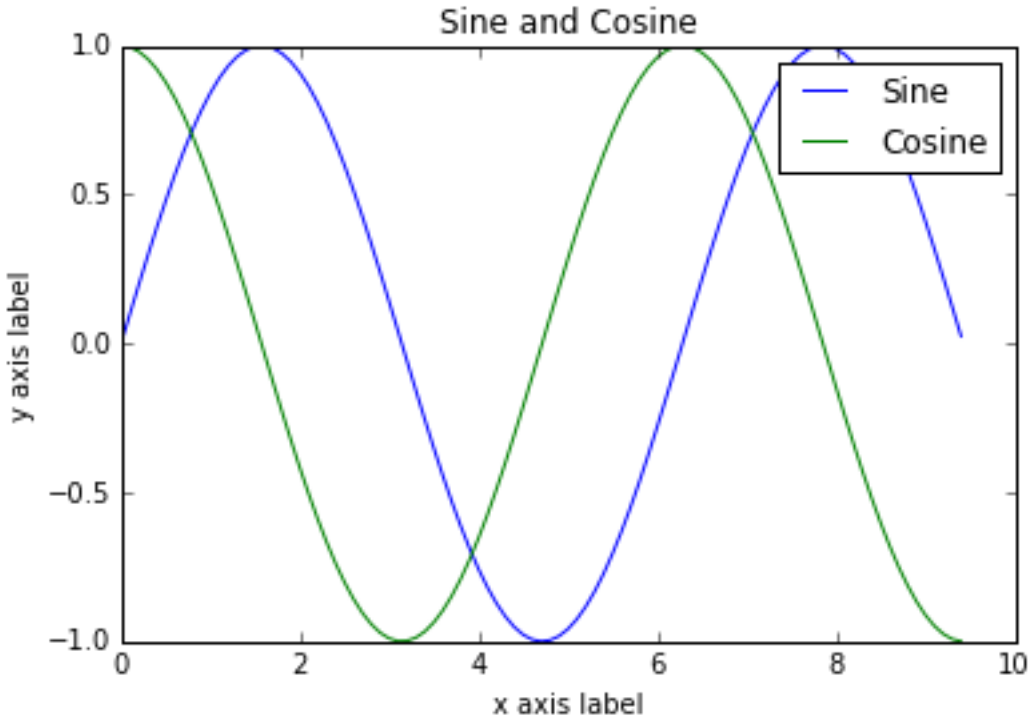
Matplotlib-plot function

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

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
# You must call plt.show() to make graphics appear.
```



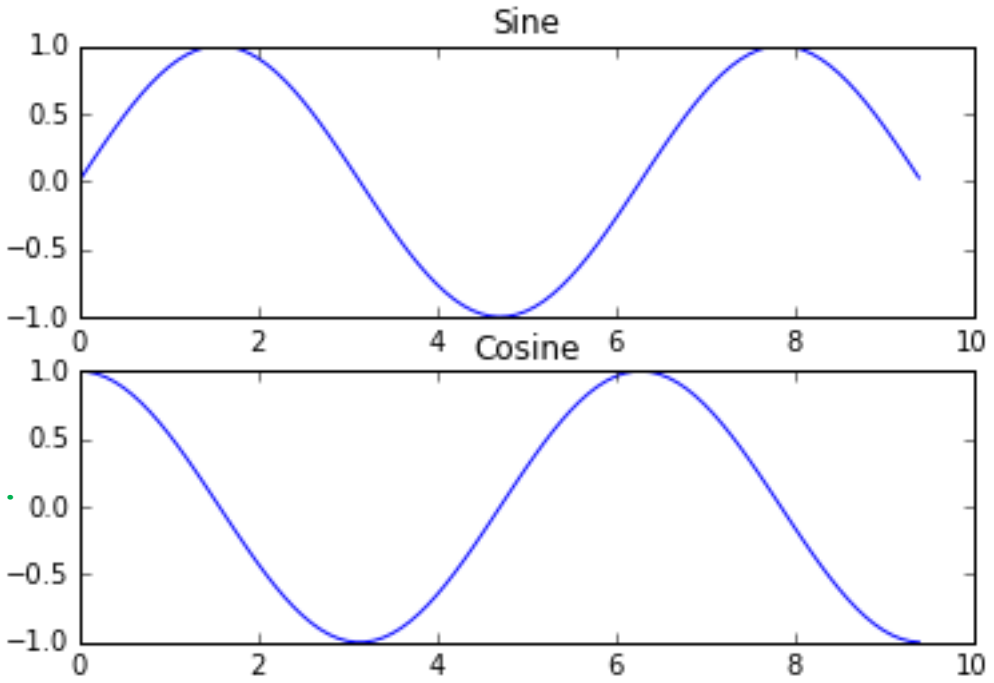
Matplotlib-plot function

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



Matplotlib-plot function

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



Matplotlib-plot function

```
import numpy as np
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt

img = imread('assets/cat.jpg')
img_tinted = img * [1, 0.95, 0.9]
# Show the original image
plt.subplot(1, 2, 1)
plt.imshow(img)

# Show the tinted image
plt.subplot(1, 2, 2)

# A slight gotcha with imshow is that it might give strange results
# if presented with data that is not uint8. To work around this, we
# explicitly cast the image to uint8 before displaying it.
plt.imshow(np.uint8(img_tinted))
plt.show()
```

End of Python Scipy and Matplotlib