Computational Astrophysics

E. Larrañaga

Observatorio Astronómico Nacional
Universidad Nacional de Colombia

January 2, 2019

Outline

Data Plotting

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

file_name = "plotdata.txt"
data = np.loadtxt(file_name, comments="#")
x = data[:,0]
y = data[:,1]

# plot the data
plt.plot(x,y)
plt.show()
```

Data Plotting Options

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

file_name = "plotdata.txt"
data = np.loadtxt(file_name, comments="#")
x = data[:,0]
y1 = data[:,1]
y2 = data[:,2]

# make the plots
fig, ax = plt.subplots()
p1 = ax.plot(x, y1, "r", linewidth = 2)
p2 = ax.plot(x, y2, "b", linewidth = 2)
```

Data Plotting Options (cont.)

```
# labels of the plot
ax.set(xlabel="X", ylabel="Y",
    title="Title of the plot")

# set x and y ranges
plt.xlim(min(x),max(x))
plt.ylim(min(y1*1.05),max(y1*1.05))

# plot's grid
ax.grid()
```

Data Plotting Options (cont.)

```
# show the plot
plt.show()

# save the plot as a pdf or png
fig.savefig("simpleplot.pdf")
fig.savefig("simpleplot.png")
```

Curve Plotting I

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

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

# range of the independent variable
# 50 points between 0 and 3
t = np.linspace(0, 3, 50)

# values of the function for the numbers in the range of t
y = np.zeros(len(t))
for i in range(len(t)):
    y[i] = f(t[i])

# plot and label
fig, ax = plt.subplots()
ax.plot(t, y, label='t^2*exp(-t^2)')
ax.legend()

# show the plot
plt.show()
```

Curve Plotting II

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

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

def g(t):
    return t*np.sin(2*t)

# range of the independent variable
# 50 points between 0 and 3
t = np.linspace(0, 3, 50)

# values of the function for the numbers in the range of t
y1 = f(t)
y2 = g(t)

# plot and label of the curve
fig, ax = plt.subplots()
ax.plot(t, y1, 'r-', label='f(t) = t^2*exp(-t^2)')
ax.plot(t, y2, 'bo', label='g(t) = t*sin(2t)')

# labels of the plot
ax.set(xlabel="t", ylabel="f(t), g(t)",
       title="Two curves")
ax.legend()

# show the plot
plt.show()
```

Curve Plotting III

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

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

def g(t):
    return t*np.sin(2*t)

# range of the independent variable
t = np.linspace(0, 3, 50)
# values of the function for the numbers in the range of t
y1 = f(t)
y2 = g(t)

# separate figures
plt.figure()

# First subplot
plt.subplot(2, 1, 1)
plt.plot(t, y1, 'r-', label='f(t) = t^2*exp(-t^2)')
plt.legend()
# Second subplot
plt.subplot(2, 1, 2)
plt.plot(t, y2, 'bo', label='g(t) = t*sin(2t)')
plt.legend()

# show the plot
plt.show()
```

Defining a Class

Defining a Class

1. Attributes
2.

Defining a Class

1. Attributes
2.

```
class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.planet_name = planet_name
        self.orbit_period = rev_period      # in Earth years
        self.mass = mass                    # in units of Earth mass
```

Defining a Class

1. Attributes
2.

```
class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.planet_name = planet_name
        self.orbit_period = rev_period      # in Earth years
        self.mass = mass                    # in units of Earth mass
```

Class Initialization

Defining a Class

1. Attributes
2.

```
class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.planet_name = planet_name
        self.orbit_period = rev_period      # in Earth years
        self.mass = mass                    # in units of Earth mass
```

Defining a Class

1. Attributes
2.

```
class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.planet_name = planet_name
        self.orbit_period = rev_period      # in Earth years
        self.mass = mass                    # in units of Earth mass


# defining Mars as a planet
mars = Planet("Mars", 1.88, 0.107)

# attributes of Mars
print(mars.rev_period)
print(mars.mass)
```

Defining a Class

1. Attributes
2. Methods

Defining a Class

1. Attributes
2. Methods

```
class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.planet_name = planet_name
        self.orbit_period = orbit_period  # in Earth years
        self.mass = mass # in units of Earth mass
```

Defining a Class

1. Attributes
2. Methods

```
class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.planet_name = planet_name
        self.orbit_period = orbit_period  # in Earth years
        self.mass = mass # in units of Earth mass

    def semimajoraxis(self):
        '''
        returns the semimajor axis of the planet in AU
        calculated using Kepler's third law
        '''
        return (self.orbit_period**(2./3.))
```

Defining a Class

1. Attributes
2. Methods

```
# defining Mars as a planet
mars = Planet("Mars", 1.88, 0.107)

# attributes of Mars
print(mars.orbit_period)
print(mars.mass)
print(mars.semimajoraxis())
```

Defining a Class

1. Attributes
2. Methods

```
# defining Mars as a planet
mars = Planet("Mars", 1.88, 0.107)

# attributes of Mars
print(mars.orbit_period)
print(mars.mass)
print(mars.semimajoraxis())


# defining Earth as a planet
earth = Planet("Earth", 1., 1.)

# attributes of Earth
print(earth.orbit_period)
print(earth.mass)
print(earth.semimajoraxis())
```

Defining a Subclass

```
import numpy as np

class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.name = planet_name
        self.orbit_period = orbit_period  # in Earth years
        self.mass = mass # in units of Earth mass

    def semimajoraxis(self):
        '''
        returns the semimajor axis of the planet in AU
        calculated using Kepler's third law
        '''
        return (self.orbit_period**(2./3.))
```

Defining a Subclass

```
import numpy as np

class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.name = planet_name
        self.orbit_period = orbit_period  # in Earth years
        self.mass = mass # in units of Earth mass

    def semimajoraxis(self):
        '''
        returns the semimajor axis of the planet in AU
        calculated using Kepler's third law
        '''
        return (self.orbit_period**(2./3.))

class Dwarf(Planet):
    def description(self):
        return self.name + " is a dwarf planet with a mass of " + str(self.mass) + " Earth masses."
```

Defining a Subclass

```
import numpy as np

class Planet(object):
    def __init__ (self, planet_name, orbit_period, mass):
        self.name = planet_name
        self.orbit_period = orbit_period  # in Earth years
        self.mass = mass # in units of Earth mass

    def semimajoraxis(self):
        '''
        returns the semimajor axis of the planet in AU
        calculated using Kepler's third law
        '''
        return (self.orbit_period**(2./3.))

class Dwarf(Planet):
    def description(self):
        return self.name + " is a dwarf planet with a mass of " + str(self.mass) + " Earth masses."

# defining Pluto as a dwarf planet
pluto = Dwarf("Pluto", 248.00, 0.00218)

# attributes of Pluto
print(pluto.orbit_period)
print(pluto.mass)
print(pluto.semimajoraxis())
print(pluto.description())
```

Writing and Using a Module

mymodule.py

```
def BalmerLines(n):
    '''
    returns the value of the wavelenght lambda (in meters)
    for a given value of n in the Balmer series
    '''
    if n < 3:
        return None
    else:
        R = 1.09677583E7 # in untis of m^-1
        lambda_inv = R * (1/4 - 1/n**2)
        return 1/lambda_inv
```

Writing and Using a Module

usingmymodule.py

```
import mymodule as mym

print("\nn        lambda (m)")
for n in range(2,16):
    print (n,"        ", mym.BalmerLines(n))

print()
```