MM220: Computational Lab

Amrita Bhattacharya

October 30, 2020

1 Introduction to python -II: Plotting and Differentiation

1.1 Locating, adding, and importing libraries

Python has strong in built libraries for numerical and scientific operations such as Numpy, Scipy, Sympy etc. Locate your libraries and add path of the libraries in your preamble (if required).

Listing 1: locate

```
locate numpy
```

Add the following in your preamble (if required), depending upon your search result to the path of the libraries;

Listing 2: path

```
#!/usr/bin/python
```

Import the libraries, depending on their need;

Listing 3: Import

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

x = np.arange(0,4*np.pi,0.1)   # start,stop,step
y = np.sin(x)

y = np.sin(x)
z = np.cos(x)

plt.plot(x,y,x,z)
plt.xlabel('x values from 0 to 4pi')
plt.ylabel('sin(x) and cos(x)')
plt.title('Plot of sin and cos from 0 to 4pi')
plt.legend(['sin(x)', 'cos(x)'])
plt.show()
```

Listing 4: harmonics

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

def func_duo(x):
     return  np.sin(x) , np.cos(x)

def comp_func(x, a, b):
    return a*np.sin(x) + b*np.cos(x)

xdata = np.linspace(-np.pi, np.pi, 300)
sin, cos = func_duo(xdata)
y_comp = comp_func(xdata, 2, 1)

plt.plot(xdata, sin, ".", label="sin");
plt.plot(xdata, cos, ".", label="cos");
plt.plot(xdata, y_comp, ".", label="Data");
plt.xlabel("x")
plt.ylabel("y")
plt.legend(( 'sin(x)', 'cos(x)','harmonics(x)'))
plt.savefig('harmonics.png')
plt.show();
```

The output of the given program should be as follows;

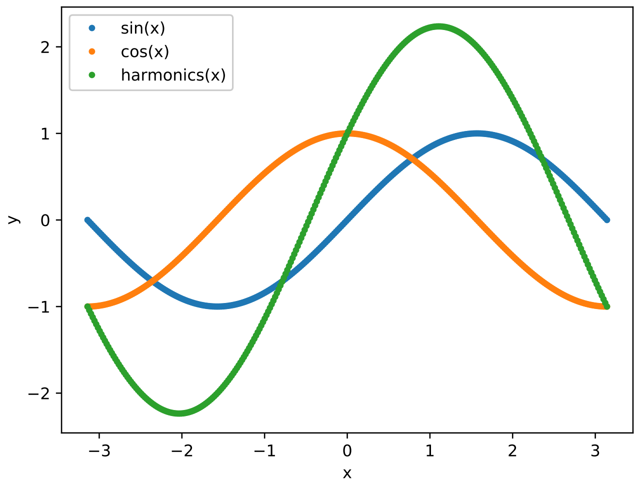

Figure 1: Plot of the harmonics.

Listing 5: polyfit

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

# This is the function we are trying to fit to the data.
def func(x, a, b, c):
     return a * np.exp(-b * x) + c

# Generate some data
xdata = np.linspace(0, 4, 50)
y = func(xdata, 2.5, 1.3, 0.5)
y_noise = 0.2 * np.random.normal(size=xdata.size)
ydata = y + y_noise

# Plot the actual data
plt.scatter(xdata, ydata)

# The actual curve fitting happens here
p1 = np.polyfit(xdata,ydata,1)
print(p1, np.poly1d(p1))
p2 = np.polyfit(xdata,ydata,2)
p3 = np.polyfit(xdata,ydata,3)
plt.plot(xdata, np.polyval(p1,xdata), 'r-')
plt.plot(xdata, np.polyval(p2,xdata), 'b-')
plt.plot(xdata, np.polyval(p3,xdata), 'g-')
# Show the graph
plt.xlabel("x")
plt.ylabel("y")
plt.legend(('data', 'p_1', 'p_2', 'p_3'))
plt.savefig("polyfit.png")
plt.show();
```

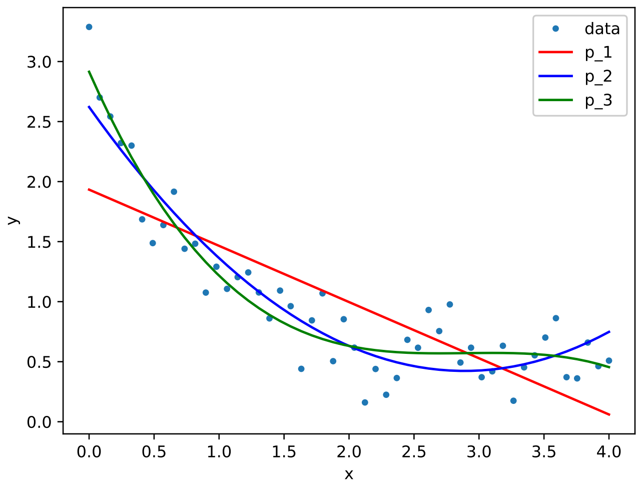

Figure 2: Plot showing the polynomial fit for the function.

1.2 Numerical Differentiation

Listing 6: Numerical

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

########## Definition of the functions #############
def func(x):
    fn1 = 4*x**2 +2
    fn2 = np.sin(x)
    return fn1, fn2

####### Numerical derivative of the functions ######
def nd_func(x):
    h = 1e-5
    ndfn1 = (func(x+h)[0]-func(x-h)[0])/(2*h)
    ndfn2 = (func(x+h)[1]-func(x-h)[1])/(2*h)
    return ndfn1, ndfn2

######### plotting ###################
x = np.linspace(-6, 6)
plt.plot(x,func(x)[0])
plt.xlabel("x")
plt.ylabel("y")
plt.plot(x,nd_func(x)[0])
plt.legend(('fn(x)', 'd_fn(x)'))
plt.savefig('num_diff_fx.eps')
plt.show()

x = np.linspace(-2*np.pi,2*np.pi)
plt.plot(x,func(x)[1])
plt.xlabel("x")
plt.ylabel("y")
plt.plot(x,nd_func(x)[1])
plt.legend(('sin(x)', 'cos(x)'))
plt.savefig('num_diff_harmonics.eps')
plt.show()
```

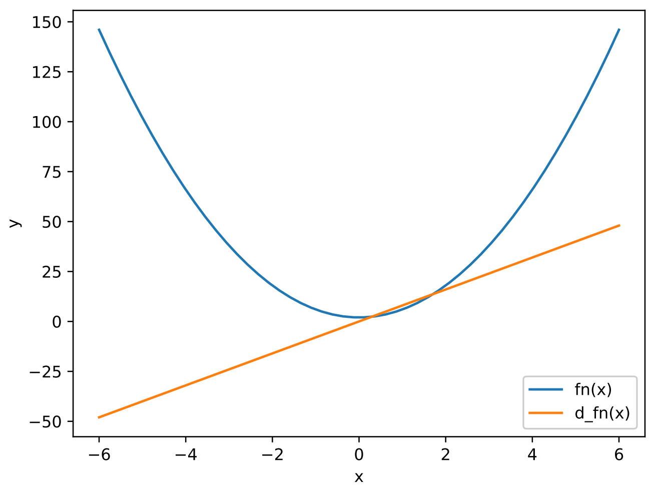

Figure 3: Numerical differentiation of function

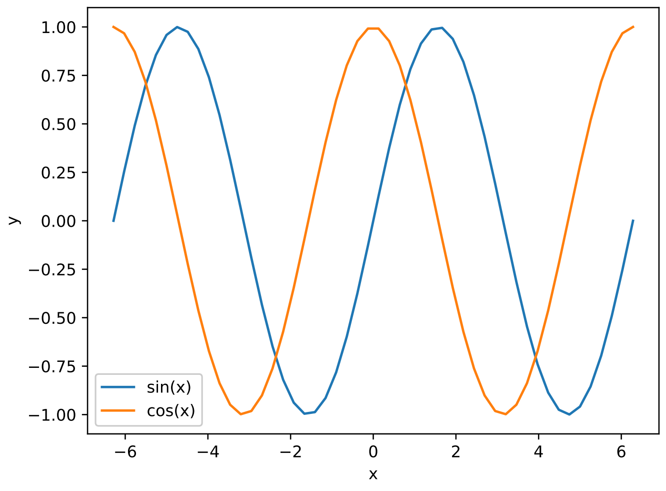

Figure 4: Numerical differentiation of harmonics.

1.3 Symbolic Differentiation

Listing 7: Symbolic

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

#################### Symbolic derivative of the functions ############
def diff_func(x):
  x = sympy.symbols('x')
  fn1 = 4*x**2 +2
  fn2 = 50*sympy.sin(x)
  dfn1 = sympy.diff(fn1, x)
  dfn2 = sympy.diff(fn2, x)
  return fn1, dfn1, fn2, dfn2

x = sympy.symbols('x')
fn1, dfn1, fn2, dfn2 = diff_func(x)

print fn1, dfn1, fn2, dfn2
print diff_func(x)[0], diff_func(x)[1], diff_func(x)[2], diff_func(x)[3]

##################### plotting the functions ##########################
fx1=[]
diffx1=[]
fx2=[]
diffx2=[]
for i in np.linspace(-6,6):
  fx1.append(fn1.evalf(subs={x: i}))
  diffx1.append(dfn1.evalf(subs={x: i}))
  fx2.append(fn2.evalf(subs={x: i}))
  diffx2.append(dfn2.evalf(subs={x: i}))

x = np.linspace(-6,6)
plt.plot(x,fx1)
plt.plot(x,diffx1)
plt.xlabel("x")
plt.ylabel("y")
plt.plot(x,fx2)
plt.plot(x,diffx2)
plt.legend(('fx1', 'dfx1', 'fx2', 'dfx2'))
plt.savefig('sym_diff.png')
plt.show()
```

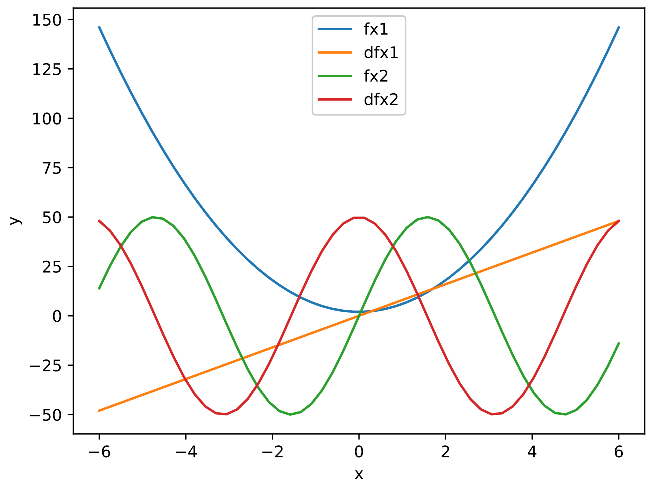

Figure 5: Symbolic differentiation.

2 Assignments

(i) Write a program that does the following:
(a) creates a function, $f(x) = ae^x + be^{-x}$ that takes the parameters a = 2, 4 and b = 3, 6
(b) plots the function for $-6 < x < 6$.
(c) fits the function with polynomials of the order two, three and four.
(ii) The Lennard-Jones (LJ) model is used to depict the inter atomic force/ potential between two atoms. It consists of a steep repulsive term, and smooth attractive term. The 12-6 LJ potential is given by the following equation:

$$\mathrm{V}(r) = \left[\left(\frac{\mathrm{B}}{r^{12}}\right) - \left(\frac{\mathrm{A}}{r^6}\right)\right] \tag{1}$$

Say,
$\mathrm{A} = 1.024 \times 10^{-23}$ (J nm^6)
$\mathrm{B} = 1.582 \times 10^{-26}$ (J nm^{12})
The force between the atom is given by $\mathrm{F}(r) = -\frac{d\mathrm{V}}{dr}$. At equilibrium separation, the potential between the two atoms is minimum, which implies the force is equal to zero, i.e. $\mathrm{F}(r) = 0$.
Write a program that does the following:
(a) plots the LJ potential $\mathrm{V}(r)$ as a function of r (in nm) varying in range $0.3 \leqslant r \leqslant 0.8$.
(b) plots the force $\mathrm{F}(r)$ between the atoms as a function of r.
(c) calculates the equilibrium separation r_0 between the atoms for which the potential is minimum.

Hints: It is easier if you divide A and B by the Boltzmann's constant, $1.381 \times 10^{-23}\mathrm{JK}^{-1}$ so as to measure $\mathrm{V}(r)$ in units of K.