EE21b043_week6

D.Somesh Reddy <ee21b043@smail.iitm.ac.in>

March 24, 2023

[1]:
```
%matplotlib ipympl
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from mpl_toolkits.mplot3d import Axes3D
```

[2]:
```
import numpy as np
from numpy import cos, sin, pi, exp
```

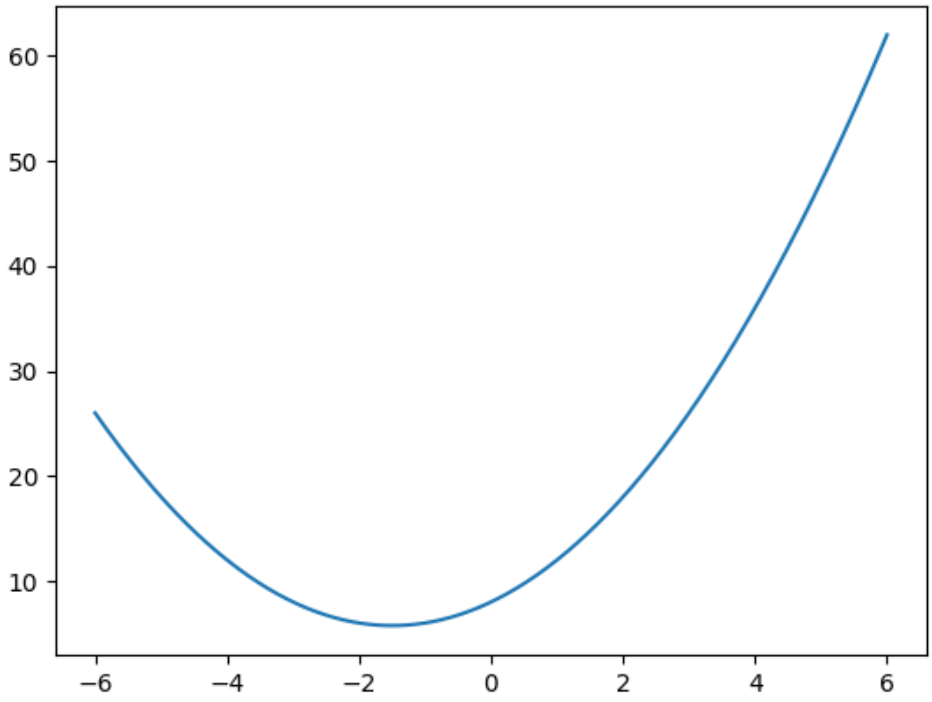
0.1 Problem 1

[3]:
```
def df1(x):
    return 2*x+3
```

[4]:
```
def f1(x):
    return x ** 2 + 3 * x + 8
```

[13]:
```
xbase = np.linspace(-6, 6, 1000)
ybase=f1(xbase)
bestx = 4
fig, ax = plt.subplots()
ax.plot(xbase, ybase)
xall, yall = [], []
lnall,  = ax.plot([], [], 'ro')
lngood, = ax.plot([], [], 'go', markersize=10)

# Learning rate
lr = 0.1


def onestepderiv(frame):
    global bestcost, bestx, lr
    x = bestx - df1(bestx) * lr
    bestx = x
    y = f1(x)
    lngood.set_data(x, y)
    xall.append(x)
    yall.append(y)
```

```
    lnall.set_data(xall, yall)
    # return lngood,

ani= FuncAnimation(fig, onestepderiv, frames=range(100), interval=100,␣
 ↪repeat=False)


plt.show()
```



[14]:
```
print(f"local minimum is found at x = {bestx}")
```

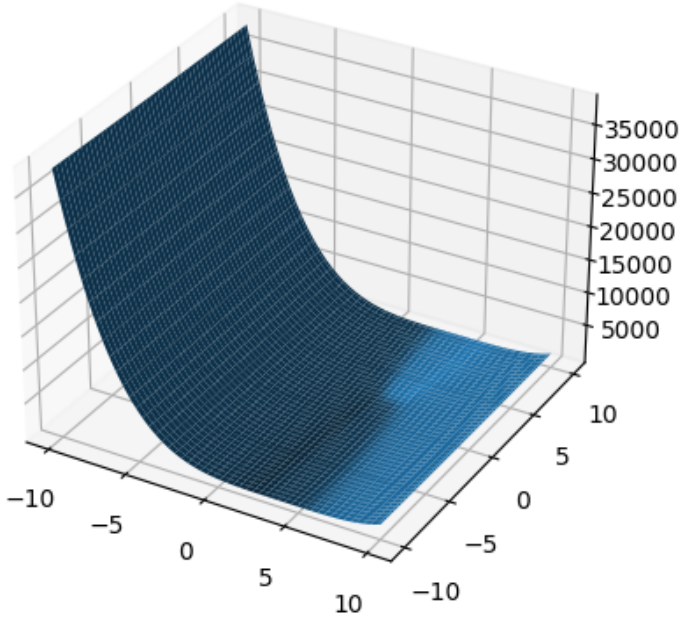
```
local minimum is found at x = -1.499999999103704
```

0.2 Problem 2

[15]:
```
def f3(x, y):
    return x**4 - 16*x**3 + 96*x**2 - 256*x + y**2 - 4*y + 262

def df3_dx(x, y):
    return 4*x**3 - 48*x**2 + 192*x - 256
```

```
def df3_dy(x, y):
    return 2*y - 4
```

[24]:
```
xbase = np.linspace(-10, 10, 100)
ybase = np.linspace(-10, 10, 100)
xmesh, ymesh = np.meshgrid(xbase, ybase)
zmesh = f3(xmesh, ymesh)
bestx, besty = -4,0

# Learning rate
lr = 0.001

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(xmesh, ymesh, zmesh)

xall, yall, zall = [], [], []
lnall,  = ax.plot([], [], [], 'ro')
lngood, = ax.plot([], [], [], 'go',markersize=10)

def onestepderiv(frame):
    global bestcost, bestx, besty, lr
    x = bestx - df3_dx(bestx, besty) * lr
    y = besty - df3_dy(bestx, besty) * lr
    bestx, besty = x, y
    z = f3(x, y)
    lngood.set_data([x], [y])
    lngood.set_3d_properties([z])
    xall.append(x)
    yall.append(y)
    zall.append(z)
    lnall.set_data(xall, yall)
    lnall.set_3d_properties(zall)
    # return lngood,

ani = FuncAnimation(fig, onestepderiv, frames=range(10000), interval=0.1,␣
 ↪repeat=False)

plt.show()
```



[25]:
```
print(f"local minimum is found at x = {bestx} and y = {besty}")
```

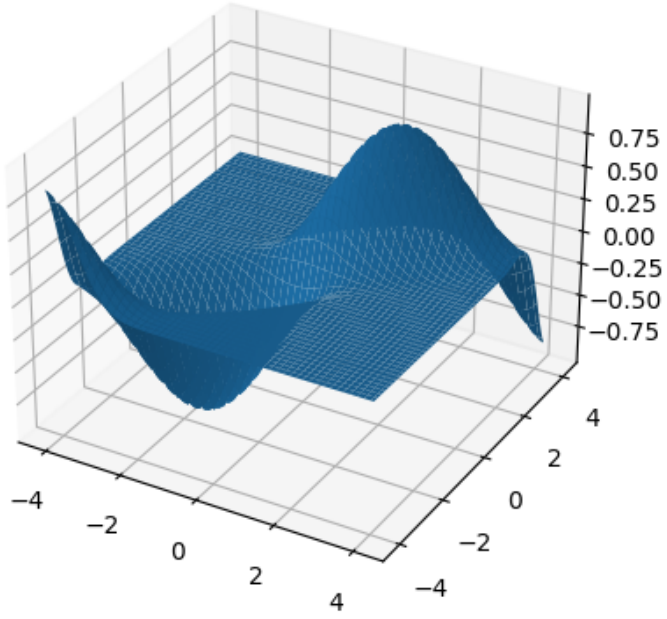
```
local minimum is found at x = 3.8400367975496237 and y = 1.9998850047625931
```

0.3 Problem 3

[26]:
```
def f4(x,y):
    return exp(-(x - y)**2)*sin(y)

def df4_dx(x, y):
    return -2*exp(-(x - y)**2)*sin(y)*(x - y)

def df4_dy(x, y):
    return exp(-(x - y)**2)*cos(y) + 2*exp(-(x - y)**2)*sin(y)*(x - y)
```

[33]:
```
xbase = np.linspace(-pi-1, pi+1, 100)
ybase = np.linspace(-pi-1, pi+1, 100)
xmesh, ymesh = np.meshgrid(xbase, ybase)
zmesh = f4(xmesh, ymesh)
bestx, besty = -0.1,-0.1
# Learning rate
```

```
lr = 0.05

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(xmesh, ymesh, zmesh)

xall, yall, zall = [], [], []
lnall,  = ax.plot([], [], [], 'ro')
lngood, = ax.plot([], [], [], 'go', markersize=10)

def onestepderiv(frame):
    global bestcost, bestx, besty, lr
    x = bestx - df4_dx(bestx, besty) * lr
    y = besty - df4_dy(bestx, besty) * lr
    bestx, besty = x, y
    z = f4(x, y)
    lngood.set_data([x], [y])
    lngood.set_3d_properties([z])
    xall.append(x)
    yall.append(y)
    zall.append(z)
    lnall.set_data(xall, yall)
    lnall.set_3d_properties(zall)
    # return lngood,


ani = FuncAnimation(fig, onestepderiv, frames=range(1000), interval=0.1,␣
 ↪repeat=False)


# from matplotlib.animation import PillowWriter

# writer = PillowWriter(fps=25)
# ani.save("problem3.gif", writer=writer)


plt.show()
```



[34]:
```
print(f"local minimum is found at x = {bestx} and y = {besty}")
```

```
local minimum is found at x = -1.7638959233829723 and y = -1.5708291519457998
```

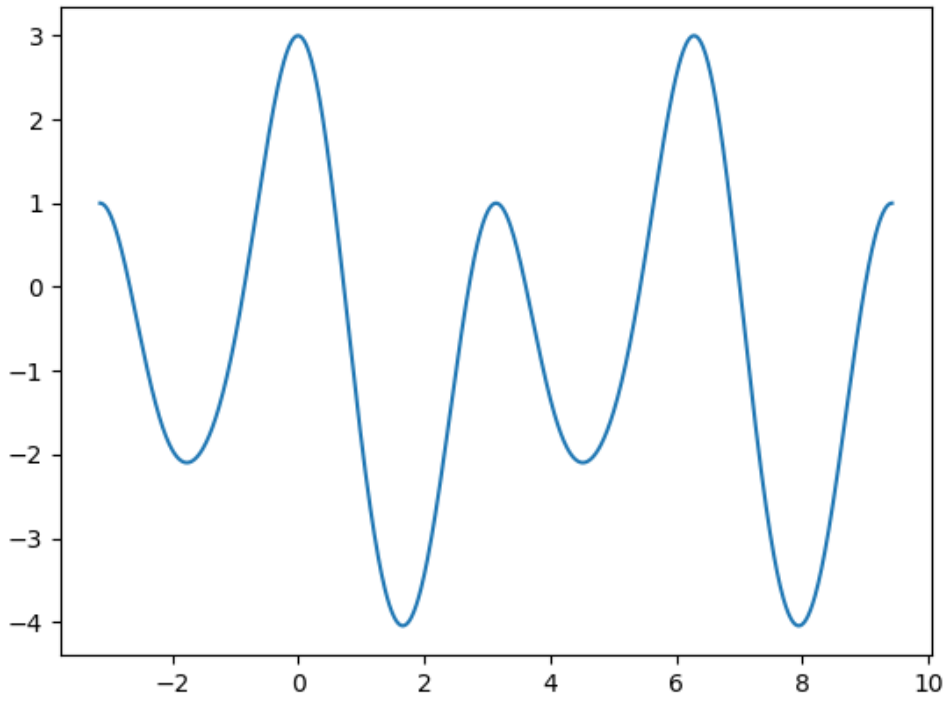
0.4 problem 4

[28]:
```
def f5(x):
    return cos(x)**4 - sin(x)**3 - 4*sin(x)**2 + cos(x) + 1
```

[29]:
```
def df5(x):
    return -1*sin(x)*(4*cos(x)**3+3*sin(x)*cos(x)+8*cos(x)+1)
```

[30]:
```
xbase = np.linspace(-pi, 3*pi, 1000)
ybase=f5(xbase)
bestx = 2.9
fig, ax = plt.subplots()
ax.plot(xbase, ybase)
xall, yall = [], []
lnall,  = ax.plot([], [], 'ro')
lngood, = ax.plot([], [], 'go', markersize=10)
```

```
# Learning rate
lr = 0.05


def onestepderiv(frame):
    global bestcost, bestx, lr
    x = bestx - df5(bestx) * lr
    bestx = x
    y = f5(x)
    lngood.set_data(x, y)
    xall.append(x)
    yall.append(y)
    lnall.set_data(xall, yall)
    # return lngood,

ani= FuncAnimation(fig, onestepderiv, frames=range(100), interval=1000,␣
 ↪repeat=False)

# from matplotlib.animation import PillowWriter

# writer = PillowWriter(fps=25)
# ani.save("problem4.gif", writer=writer)



plt.show()
```



[31]:
```
print(f"local minimum is found at x = {bestx}")
```

```
local minimum is found at x = 1.6616610958583544
```

0.5 Explanation

The animation is created using the FuncAnimation class of Matplotlib. The animation runs for 1000 frames with an interval of 100 milliseconds between each frame. On each frame, the onestepderiv function is called, which updates the current solution bestx using the gradient descent algorithm, computes the corresponding function value y, and updates the plots showing the current point in red and the trajectory of the optimization process in green.

The xbase and ybase arrays are used to create the initial plot of the function being optimized.

0.6 Gradient descent function for several variables

[ ]:
```
def gradient_descent(initial_values, main_func, gradient_func, learning_rate=0.
↪1, num_iterations=100):

    values = np.array(initial_values)
    for i in range(num_iterations):
        gradient = gradient_func(*values)
```

```
        values = values - learning_rate * gradient
    return values.tolist(), main_func(*values)
```

The function updates the values of the input variables using the formula: new_values = old_values - learning_rate * gradient

At each iteration, it computes the gradient of the function at the current values and updates the values accordingly. The function returns the final values of the variables and the minimum value of the function obtained at the end of the iterations.