Kritik Assignment 8

March 7, 2024

[9]:
```python
#4a
def fun_1(x,y): #From the assignment, just defining the function
    return x**2+y**2

def grad_f_1(x,y):
    x0=0.1 #defining x0 and y0 values given in assignment
    y0=0.1
    grad_x = 2*x #setting grad_x and grad_y to be partial derivatives of fun_1
    grad_y = 2*y
    learning_rate = 0.1 #setting learning rate as given in assignment
    for i in range (10): #10 represents the max iterations
        x = x0 - learning_rate * grad_x #using equation from assignment to find␣
 ↪x and y
        y = y0 - learning_rate * grad_y
    return x, y

answer_x, answer_y = grad_f_1(0.1,0.1)
print ("Results of gradient descent are:", answer_x, answer_y)

#This same code was used for 4b-4e. The only thing that changes is the function␣
 ↪for 4d
#4e and thus the partial derivatives too, and the x0, y0, learning rate and max␣
 ↪iterations.
```

```
Results of gradient descent are: 0.08 0.08
```

[10]:
```python
#4b
def fun_1(x,y):
    return x**2+y**2

def grad_f_1(x,y):
    x0=-1
    y0=1
    grad_x = 2*x
    grad_y = 2*y
    learning_rate = 0.01
    for i in range (100):
```

```
        x = x0 - learning_rate * grad_x
        y = y0 - learning_rate * grad_y
    return x, y

answer_x, answer_y = grad_f_1(-1,1)
print ("Results of gradient descent are:", answer_x, answer_y)
```

```
Results of gradient descent are: -0.98 0.98
```

[11]:
```
#4c
import numpy as np

def fun_2 (x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)




def grad_f_2 (x,y):
    x0= 0
    y0 = 1
    grad_x = 2 * x * np.exp(-x**2 - (y - 2)**2) + 4 * x * np.exp(-x**2 - (y +␣
 ↪2)**2)
    grad_y = 2 * (y - 2) * np.exp(-x**2 - (y - 2)**2) + 4 * (y + 2) * np.
 ↪exp(-x**2 - (y + 2)**2)
    learning_rate = 0.01
    for i in range (10000):
        x = x0 - learning_rate*grad_x
        y = y0 - learning_rate*grad_y
    return x,y

answer_x, answer_y = grad_f_2(0,1)
print ("Results of gradient descent are:", answer_x, answer_y)
```

```
Results of gradient descent are: 0.0 1.0073427796469385
```

[12]:
```
#4d
import numpy as np

def fun_2 (x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)

def grad_f_2 (x,y):
    x0= 0
    y0 = -1
    grad_x = 2 * x * np.exp(-x**2 - (y - 2)**2) + 4 * x * np.exp(-x**2 - (y +␣
 ↪2)**2)
```

```
    grad_y = 2 * (y - 2) * np.exp(-x**2 - (y - 2)**2) + 4 * (y + 2) * np.
 ↪exp(-x**2 - (y + 2)**2)
    learning_rate = 0.01
    for i in range (1000):
        x = x0 - learning_rate*grad_x
        y = y0 - learning_rate*grad_y
    return x,y

answer_x, answer_y = grad_f_2(0,-1)
print ("Results of gradient descent are:", answer_x, answer_y)
```

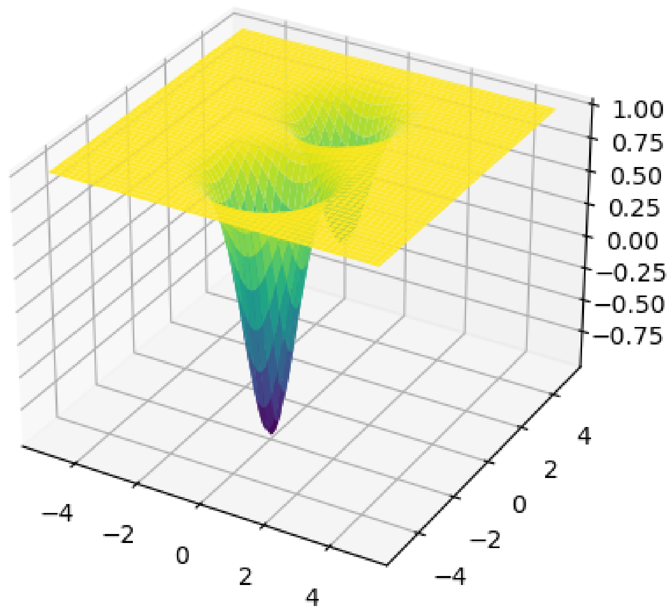
```
Results of gradient descent are: 0.0 -1.0147077730586125
```

[13]:

```
#4e
import numpy as np
from mpl_toolkits import mplot3d #for 3D plots
import matplotlib.pyplot as plt #usual matplotlib
%matplotlib widget
X=np.linspace(-5,5,100)
Y=np.linspace(-5,5,100)
x,y=np.meshgrid(X,Y)
z= 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2) #adding function equation
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
```

[13]:

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7fe48c0c5190>
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−4
−2
0
2
4
4
2
0
−2
−4

[ ]: