NUMERICAL OPTIMIZATION

[Practical Submission]

Name: Upasna Kukreti

Roll No. : 16126

Semester : 3 (B)

Practicals must be done in Python

Practical list

1. WAP for finding optimal solution using Line Search method.
2. WAP to solve a LPP graphically.
3. WAP to compute the gradient and Hessian of the function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

4. WAP to find Global Optimal Solution of a function
   $f(x) = -10Cos(\pi x - 2.2) + (x + 1.5)x$ algebraically
5. WAP to find Global Optimal Solution of a function
   $f(x) = -10Cos(\pi x - 2.2) + (x + 1.5)x$ graphically
6. WAP to solve constraint optimization problem.

Practical -1 : WAP for finding optimal solution using Line Search Method.

Code:-

```
def gradient(x):
  res=2*x + 2
  return res

def line_search(x,lr,it):
  for i in range(it):
    x= x-(lr*gradient(x))
  return x



print("x: ",line_search(0,0.000001,100),"\nF(x): x^2 +2x +1")
```

Output:

```
x:  -0.00019998020129353733
F(x): x^2 +2x +1
```

Practical -2 : WAP to solve a LPP graphically.

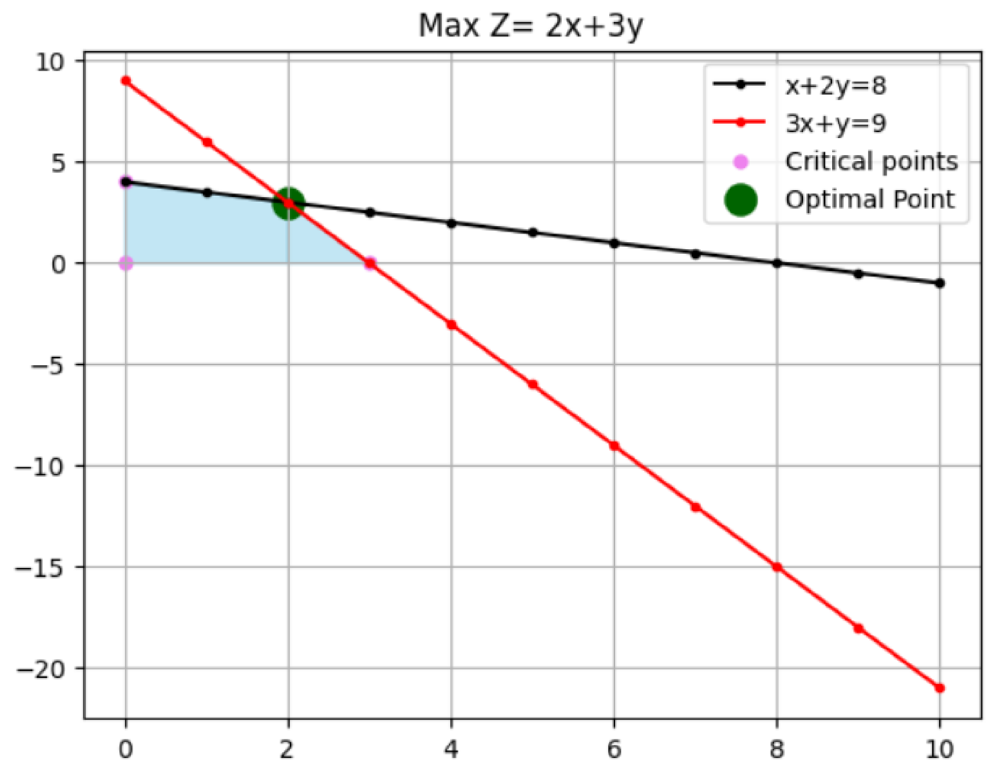
Code:

```
import matplotlib.pyplot as plt
import numpy as np
c1=2
c2=3
a1=1
a2=2
a3=3
a4=1
b1=8
b2=9
x1=[i for i in range(0,11)]
y1=[(b1-a1*i)/a2 for i in x1]
plt.plot(x1,y1,marker='.',c='black',label='x+2y=8')
x2=[i for i in range(0,11)]
y2=[(b2-a3*i)/a4 for i in x2]
plt.plot(x2,y2,marker='.',c='red',label='3x+y=9')
plt.grid()
plt.fill_between(x1,0,y1,where=[y1[i]<=y2[i] for i in
range(len(x1))],color='skyblue',alpha=0.5)
plt.fill_between([2,3],0,y2[2:4],color='skyblue',alpha=0.5)
print("Critical points: \nA(0,4)\nB(2,3) \nC(3,0)\nD(0,0)")
points=[[0,4],[2,3],[3,0],[0,0]]
x=[i[0] for i in points]
y=[i[1] for i in points]

plt.scatter(x,y,marker='o',color='violet',label='Critical points',s=25)

Z_vals=[c1*i[0]+c2*i[1] for i in points]

z_max=max(Z_vals)
ind=Z_vals.index(z_max)
opt_pt=points[ind]
plt.scatter(opt_pt[0],opt_pt[1],marker='o',s=150,color='darkgreen',label='
Optimal Point')
print("\nMax Z=", z_max , " , at point: ",opt_pt)
plt.title("Max Z= 2x+3y")
```

```
plt.legend()
plt.show()
```

Output:

```
Critical points:
A(0,4)
B(2,3)
C(3,0)
D(0,0)

Max Z= 13  , at point:  [2, 3]
```



Code (using Pulp library):

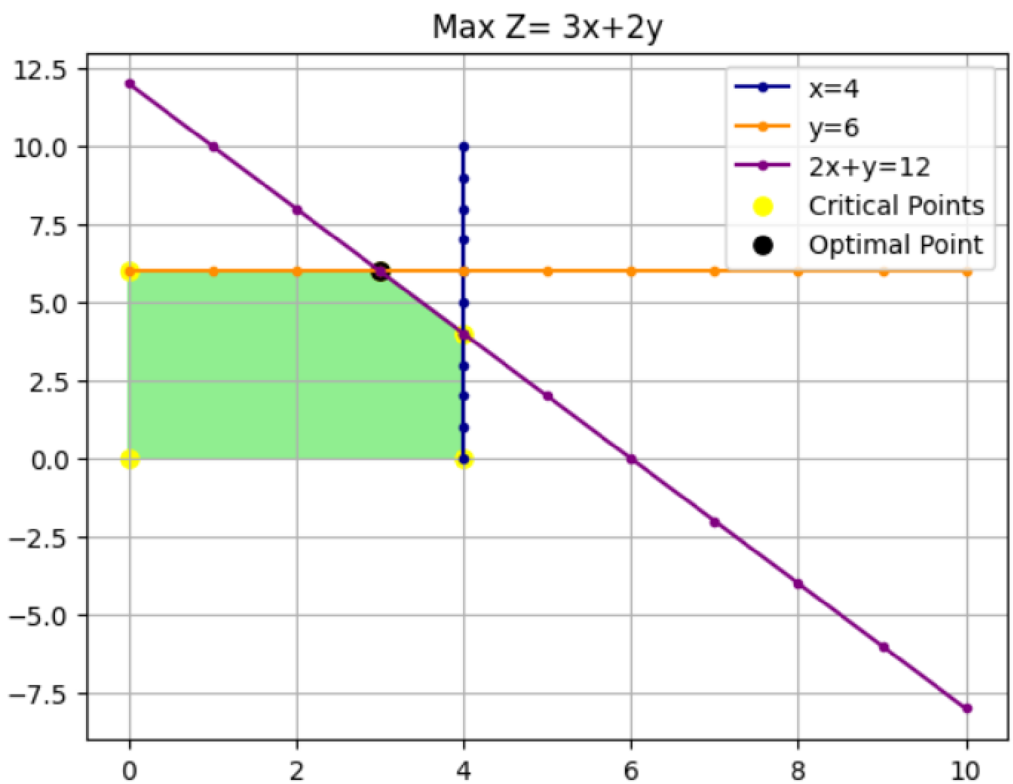
```
! pip install pulp #installing pulp library
import pulp
import matplotlib.pyplot as plt

lp_problem= pulp.LpProblem("LPP",pulp.LpMaximize)  #Creating an instance for our LPP problem
x=pulp.LpVariable("x", lowBound=0)  #defining variables
y=pulp.LpVariable("y", lowBound=0)    #lowbound serves as a constraint to limit the value of the variables to positive values only
```

```
z=3*x+2*y
lp_problem += z   #feeding objective function to the instance created above
lp_problem += x<=4    #feeding constraints to our LPP instance
lp_problem += y<=6
lp_problem += 2*x+y<=12
lp_problem.solve()    #using in-built .solve() to solve our LPP
#print(lp_problem.status)  #access the status of LPP
print("Status of solution of LPP: ",pulp.LpStatus[lp_problem.status])
#function to decipher the status code of our LPP
print("x of Optimal Point: ",x.varValue)   #accessing the variable value of our variables
print("Y of Optimal Point: ",y.varValue)
print("Max Z= ",pulp.value(lp_problem.objective))  #function that accesses our objective function and gives its maximum value
x_opt=x.varValue
y_opt=y.varValue
c1=3
c2=2
a1=1
a2=1
a3=2
a4=1
b1=4
b2=6
b3=12
x1=[4 for i in range(0,11)]
y1=[i for i in range(0,11)]
x2=[i for i in range(0,11)]
y2=[6 for i in range(0,11)]
x3=[i for i in range(0,11)]
y3=[(b3-a3*x)/a4 for x in x3 ]
plt.plot(x1,y1,marker='.',c='darkblue',label='x=4')
plt.plot(x2,y2,marker='.',c='darkorange',label='y=6')
plt.plot(x3,y3,marker='.',c='purple',label='2x+y=12')
x=[0,4,4,3,0]
y=[0,0,4,6,6]
plt.fill(x,y,c='lightgreen')  #an alternative for filbetween, it directly takes list of points and fills the region enclosed by the points
plt.scatter(x,y,marker='o',c='yellow',s=45,label="Critical Points")
```

```
plt.scatter(x_opt,y_opt,marker='o',c='black',s=45,label='Optimal Point')
plt.legend()
plt.title("Max Z= 3x+2y")
plt.grid()
plt.show()
```

Output:

```
Collecting pulp
  Downloading PuLP-2.7.0-py3-none-any.whl (14.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.3/14.3 MB 62.6 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-2.7.0
Status of solution of LPP:  Optimal
x of Optimal Point:  3.0
Y of Optimal Point:  6.0
Max Z=  21.0
```



Practical -3: WAP to compute the gradient and Hessian of the function:

$100(x_2-x_1^2)^2+(1-x_1)^2$

Code:

```
import sympy as sp
from sympy import *
init_printing(use_unicode=True)

x1,x2= symbols('x1 x2')
f=100*(x2-(x1)**2)**2+(1-x1)**2
gradf=[diff(f,x1),diff(f,x2)]

hessian_mtr=[[diff(gradf[0],x1),diff(gradf[0],x2)],[diff(gradf[1],x1),diff
(gradf[1],x2)]]
print("f: 100(x2-x1**2)**2+(1-x1)**2\n")
print("Gradient of f: ",gradf)
print("\nHessian Matrix:" )
for i in hessian_mtr:
  print(i)
```

Output:

```
f: 100(x2-x1**2)**2+(1-x1)**2

Gradient of f:  [-400*x1*(-x1**2 + x2) + 2*x1 - 2, -200*x1**2 + 200*x2]

Hessian Matrix:
[1200*x1**2 - 400*x2 + 2, -400*x1]
[-400*x1, 200]
```

Practical -4: WAP to find Global Optimal Solution of the following function algebraically:

$-10\cos(\pi x-2.2)+(x+1.5)x$

Code (using minimize):

```
import numpy as np
from scipy.optimize import minimize
def func(x):
  return -10*np.cos(np.pi*x-2.2)+(x+1.5)*x
x0=0

result=minimize(func,x0)
print("Global Optimal Solution: ",result.x)
print("Function Value:  ",func(result.x))
```

Output:

```
Global Optimal Solution:  [0.67143793]
Function Value:   [-8.50098642]
```

Code (using differential evolution):

```
import numpy as np
from scipy.optimize import differential_evolution
def func(x):
  return -10*np.cos(np.pi*x-2.2)+(x+1.5)*x
bounds=[(-10,10)]
result=differential_evolution(func,bounds)
global_min_x=result.x
global_min_f=result.fun
print("Global minimum solution: ",global_min_x)
print("Global minimum function value: ",global_min_f)
```

Output:

```
Global minimum solution:  [-1.28879779]
Global minimum function value:  -10.266312448524529
```

Practical -5: WAP to find Global Optimal Solution of the following function graphically:

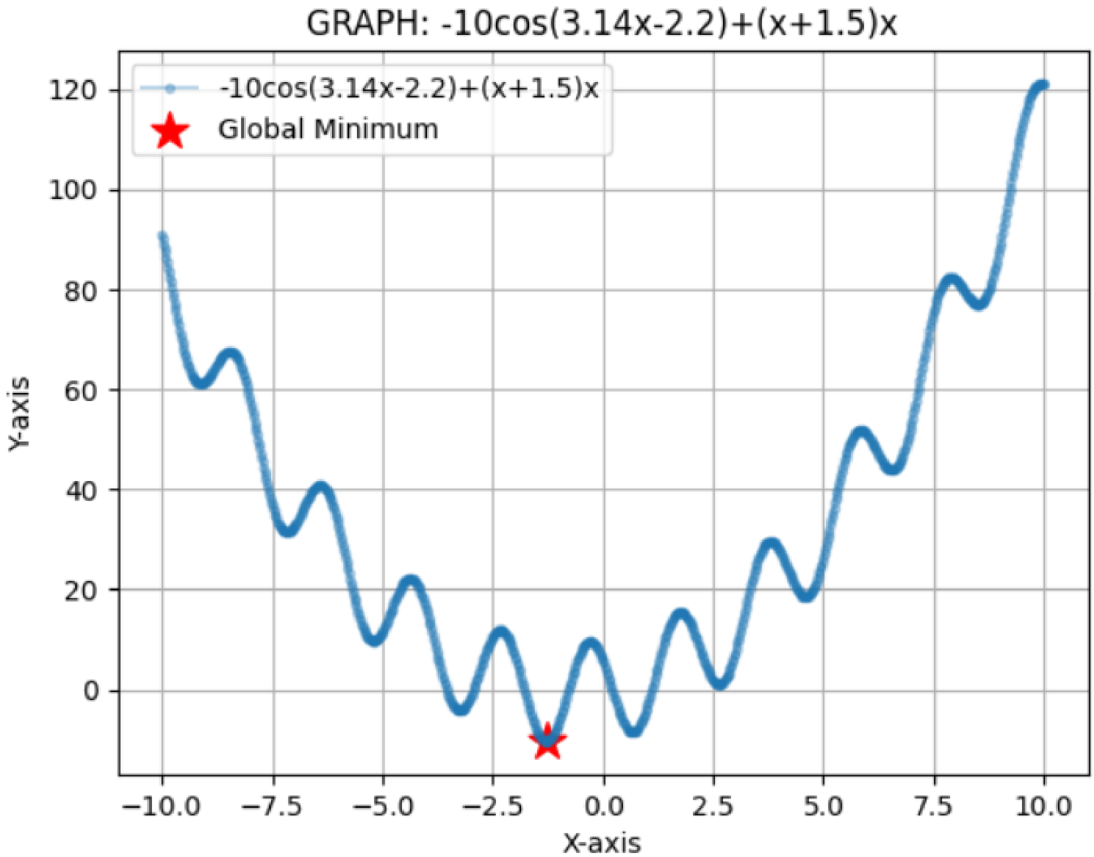
$-10\cos(\pi x-2.2)+(x+1.5)x$

Code:

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

def obj_func(x):
  return -10*np.cos(np.pi*x-2.2)+(x+1.5)*x
x=np.linspace(-10,10,1000)
y=obj_func(x)
plt.plot(x,y,marker='.',label="-10cos(3.14x-2.2)+(x+1.5)x",alpha=0.3)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("GRAPH: -10cos(3.14x-2.2)+(x+1.5)x")

min_y=min(y)
print("Minimum_y : ",min_y)
min_y_ind=np.argmin(y)
min_x=x[min_y_ind]
print("Minimum_x : ",min_x)
plt.scatter(min_x,min_y,marker='*',c='red',s=200,alpha=1,label="Global
Minimum")
plt.grid(True)
plt.legend()

plt.show()
```

Output:

```
Minimum_y :  -10.265999559988813
Minimum_x :  -1.2912912912912908
```



Practical -6: WAP to solve a constraint optimization problem

Code:

```
from sympy import symbols,diff,solve,Matrix

x,y,l=symbols('x y lambda')
f=x**2+y**2   #Objective function
g=x+y-1    #Constraint

L=f-l*g     #langragian function

partials=[diff(L,var) for var in (x,y,l)] #partial derivatives with
respect to x,y,l
solution=solve(partials,(x,y,l),dict=True)[0] #solve partial derivatives
wrt x,y and l

optimal_x=solution[x] #solutions were stored in a dictionary so using key
x will give value associated with x
optimal_y=solution[y]
#print("Solution:-\nx:",optimal_x,"\ny:", optimal_y)

hessian_list=[]
for var2 in (x,y,l):   #can be done with list comprehension but for loop
is used for easy understanding
    row=[]
    for var1 in (x,y,l):
        row.append(diff(L.diff(var1),var2))  #First, L will be
differentiated wrt var1, then it will be differented wrt var2
    hessian_list.append(row)  #above value is append into row[], then
after completion of for loop, row is apend to hessian matrix


hessian_matrix=Matrix(hessian_list)
print("Hessian matrix: ",hessian_matrix)
hessian_det=hessian_matrix.det()
print("Hessian determinant: ",hessian_matrix.det())

if(hessian_det>0):
```

```python
    print("-->Local Minima")
elif(hessian_det<0):
    print("-->Local Maxima")
else:
   print("-->Test Fails")

print("Optimal Solution:-")
print(f"x :{optimal_x}")
print("y: "+str(optimal_y))
```

Output:

```
Hessian matrix:  Matrix([[2, 0, -1], [0, 2, -1], [-1, -1, 0]])
Hessian determinant:  -4
-->Local Maxima
Optimal Solution:-
x :1/2
y: 1/2
```