ME598/494 Homework 2

1. (20 points) Show that the stationary point (zero gradient) of the function

$$f(x_1, x_2) = 2x_1^2 - 4x_1x_2 + 1.5x_2^2 + x_2$$

is a saddle (with indefinite Hessian).

Find the directions of downslopes away from the saddle. To do this, use Taylor's expansion at the saddle point to show that

$$f(x_1, x_2) = f(1, 1) + (a\partial x_1 - b\partial x_2)(c\partial x_1 - d\partial x_2),$$

with some constants a, b, c, d and $\partial x_i = x_i - 1$ for $i = 1, 2$. Then the directions of downslopes are such $(\partial x_1, \partial x_2)$ that

$$f(x_1, x_2) - f(1, 1) = (a\partial x_1 - b\partial x_2)(c\partial x_1 - d\partial x_2) < 0.$$

Ans Given: $f(x_1, x_2) = 2x_1^2 - 4x_1x_2 + 1.5x_2^2 + x_2$

$\therefore \nabla f = \begin{bmatrix} 4x_1 - 4x_2 \\ -4x_1 + 3x_2 + 1 \end{bmatrix}$

$\therefore$ The point at which gradient ∇f is zero:

$\therefore \nabla f = \begin{bmatrix} 4x_1 - 4x_2 \\ -4x_1 + 3x_2 + 1 \end{bmatrix} = 0$

- $4x_1 - 4x_2 = 0$

$\therefore x_1 = x_2$

- $-4x_1 + 3x_2 + 1 = 0$

$\therefore -4(x_2) + 3x_2 + 1 = 0$

$-4x_2 + 3x_2 + 1 = 0$

$x_2 = 1$

$\Rightarrow \quad x_2 = 1 \qquad \text{and} \quad x_1 = 1$

$\therefore$ Stationary Point $x^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\rightarrow$ Determining the Hessian matrix :

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_2^2} \end{bmatrix}$$

$$H = \begin{bmatrix} 4 & -4 \\ -4 & 3 \end{bmatrix}$$

$$\therefore \quad \begin{vmatrix} 4-\lambda & -4 \\ -4 & 3-\lambda \end{vmatrix} = 0$$

$$\therefore \quad (4-\lambda)(3-\lambda) - 16 = 0$$

$$\therefore \quad \lambda^2 - 7\lambda - 4 = 0$$

$$\therefore \quad \lambda = \left(\frac{7 \pm \sqrt{65}}{2} \right)$$

$$\Rightarrow \lambda_1 = \frac{7 - \sqrt{65}}{2} < 0$$

and

$$\lambda_2 = \frac{7 + \sqrt{65}}{2} > 0$$

$\rightarrow$ Since λ_1 is negative, while λ_2 is positive. Hence, the Hessian matrix is indefinite, so the stationary point is a saddle point.

→ Direction of Downslope :

For the saddle point $x^* = [1 \quad 1]^T$, Jaylor Expansion is :

$$f(x,y) = f(1,1) + \nabla f\Big|_{(1,1)}^{T} (x - x^*) + \frac{1}{2}(x-x^*)^T H\Big|_{(1,1)} (x - x^*)$$

$$\therefore f(x,y) = f(1,1) + 0 + \frac{1}{2}\begin{bmatrix} x_1 - 1 \\ x_2 - 1 \end{bmatrix}^T \begin{bmatrix} 4 & -4 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} x_1 - 1 \\ x_2 - 1 \end{bmatrix}$$

→ Given, $x_1 - 1 = \partial x_1$ & $x_2 - 1 = \partial x_2$

$$\therefore f(x_1, x_2) = f(1,1) + \frac{1}{2}\begin{bmatrix} \partial x_1 & \partial x_2 \end{bmatrix} \begin{bmatrix} 4 & -4 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} \partial x_1 \\ \partial x_2 \end{bmatrix}$$

$$\therefore f(x,y) = f(1,1) + \frac{1}{2}\begin{bmatrix} (4\partial x_1 - 4\partial x_2) & (-4\partial x_1 + 3\partial x_2) \end{bmatrix} \begin{bmatrix} \partial x_1 \\ \partial x_2 \end{bmatrix}$$

$$\therefore f(x,y) = f(1,1) + \frac{1}{2}\left[\partial x_1(4\partial x_1 - 4\partial x_2) + \partial x_2(-4\partial x_1 + 3\partial x_2)\right]$$

$$\therefore f(x,y) = f(1,1) + \frac{1}{2}\left[4(\partial x_1)^2 - 4(\partial x_2 \partial x_1) + 3(\partial x_2)^2 - 4(\partial x_1 \partial x_2)\right]$$

$$\therefore f(x,y) = f(1,1) + \frac{1}{2}\left[4\partial x_1^2 + 3\partial x_2^2 - 8\partial x_1 \partial x_2\right]$$

$$\therefore f(x,y) = f(1,1) + \frac{1}{2}(2\partial x_1 - \partial x_2)(\partial x_1 - 3\partial x_2)$$

$$\therefore f(x,y) - f(1,1) = \frac{1}{2}(2\partial x_1 - \partial x_2)(\partial x_1 - 3\partial x_2) \quad < 0$$

→ To get the downslope

Either • $(2\partial x_1 - \partial x_2) < 0$ and $(2\partial x_1 - 3\partial x_2) > 0$

OR

• $(2\partial x_1 - \partial x_2) > 0$ and $(2\partial x_1 - 3\partial x_2) < 0$

2. (a) (10 points) Find the point in the plane $x_1 + 2x_2 + 3x_3 = 1$ in $\mathbb{R}^3$ that is nearest to the point $(-1, 0, 1)^T$. Is this a convex problem? Hint: Convert the problem into an unconstrained problem using $x_1 + 2x_2 + 3x_3 = 1$.

(b) (40 points) Implement the gradient descent and Newton's algorithm for solving the problem. Attach your codes in the report, along with a short summary of your findings. The summary should include: (1) The initial points tested; (2) corresponding solutions; (3) A log-linear convergence plot.

(a)

Ans Defining the problem :

→ The distance between the two points $(-1\ 0\ 1)^T$ and $(x_1\ x_2\ x_3)^T$ should be minimum such that (x_1, x_2, x_3) lies on the plane $x_1 + 2x_2 + 3x_3 = 1$ in $\mathbb{R}^3$

∴ Square of Distance between two points $= (x_1+1)^2 + (x_2+0)^2 + (x_3-1)^2$

$$\Rightarrow \min_{\{x_1\ x_2\ x_3\}} \left[(x_1+1)^2 + (x_2)^2 + (x_3-1)^2\right]$$

$$\text{s.t} \quad x_1 + 2x_2 + 3x_3 = 1$$

→ To make the problem unconstrained, let us substitute :

$x_1 = 1 - (2x_2 + 3x_3)$ in the distance equation

∴ $f(x_2, x_3) = (1 - 2x_2 - 3x_3 + 1)^2 + x_2^2 + (x_3 - 1)^2$

∴ $\frac{\partial f}{\partial x_2} = (-4)(2 - 2x_2 - 3x_3) + 2x_2$

• $\frac{\partial f}{\partial x_2} = 10x_2 + 12x_3 - 8 = 0$ ——— (1)

∴ $\frac{\partial f}{\partial x_3} = -6(2 - 2x_2 - 3x_3) + 2(x_3 - 1)$

• $\frac{\partial f}{\partial x_3} = 12x_2 + 20x_3 - 14 = 0$ ——— (2)

→ Solving ① and ② for x_2 and x_3:

$$x_2 = \frac{-1}{7} \quad \text{and} \quad x_3 = \frac{11}{14}$$

Using these values in equation of plane we get

$$x_1 = \frac{-15}{14}$$

⇒ Point nearest to $[-1 \ 0 \ 1]^T$ on the plane $x_1 + 2x_2 + 3x_3 = 1$ $= \left[\frac{-15}{14} \quad \frac{-1}{7} \quad \frac{11}{14}\right]^T$

→ Determining the Hessian of the unconstrained function:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} \end{bmatrix}$$

$$= \begin{bmatrix} 10 & 12 \\ 12 & 20 \end{bmatrix}$$

→ Eigen value of H matrix:

$$\begin{vmatrix} 10-\lambda & 12 \\ 12 & 20-\lambda \end{vmatrix} = 0$$

$\therefore (20-\lambda)(10-\lambda) - 144 = 0$

$\therefore 200 - 20\lambda - 10\lambda - 144 = 0$

$\therefore \lambda^2 - 30\lambda + 56 = 0$

$\therefore (\lambda - 28)(\lambda - 2) = 0$

⇒ $\lambda_1 = 28$ & $\lambda_2 = 2$ ⇒ λ_1 & $\lambda_2 > 0$

→ Thus, Hessian matrix is positive definite, making the problem a convex problem

(b) Code :

Git Hub Link:
https://github.com/monarkparekh/MAE-598__Design-Optimization/tree/Assignment-2

I have also attached a PDF copy of the code below. However, the fringes in the below copy of the code may not be visible, to see the complete code please use the link above

MAE 598 Design Optimization: Assignment 2, Question 2b

Name: Monark Parekh
ASU ID: 1222179426

Importing the required libraries

In [299…

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

Defining the Function, Gradient and Hessian matrix

In [300…

```
function = lambda x: (2-2*x[0]-3*x[1])**2 + (x[0])**2 + (x[1]-1)**2

def gradient(x):
  return np.array([(10*x[0]+12*x[1]-8), (12*x[0]+20*x[1]-14)])

Hessian = np.array([[10,12],[12,20]])
```

Line Search Algorithm

In [301…

```
def linesearchalgorithm(x):
    alpha = 1
    t = 0.3
    beta = 0.5
    temp = -1 * gradient(x)
    # Defining the function to calculate phi(alpha)
    def pi(alpha, x):
      return function(x) - alpha * t * np.matmul(np.transpose(gradient(x)), temp)
    # Comparing phi(alpha) and f(x - alpha*gradient)
    while pi(alpha, x) < function(x + alpha*temp):
      alpha = beta * alpha
    # Returning the Final value of alpha
    return alpha
```

Gradient Descent Algorithm

In [302…

```
def gradientdescentalgorithm(function,gradient,Hessian,x0,maximum_iteration):
  epsilon = (10)**-6
  X = []
  gradient_normal = []
  X.append(x0)
  for k in range(0,maximum_iteration):
      gradient_normal.append(np.linalg.norm(gradient(X[k])))

      if gradient_normal[k] <= epsilon:
        x1 = 1-(2*X[k][0]+3*X[k][1])
        error_values = [abs(function(X[i])-function([-1/7,11/14])) for i in range(len(X))]
        X = np.insert(X,0,x1,axis=1)
        print(f'\nThe Gradient Descent Algorithm has converged to a point in {k} iterations\n')
        print(f'Current Gradient Normal is {gradient_normal[k]}\n[x1,x2,x3] = {X[k]}')
        return X,gradient_normal,(k),error_values

      alpha = linesearchalgorithm(X[k])
      new_X = X[k] - alpha*gradient(X[k])
      X.append(new_X)
```

```
    if k == (maximum_iteration - 1):
      print(f"Number of iterations has exceeded the maxinimum iterations, still not converged
      return X,gradient_normal,(k)
```

In [303…

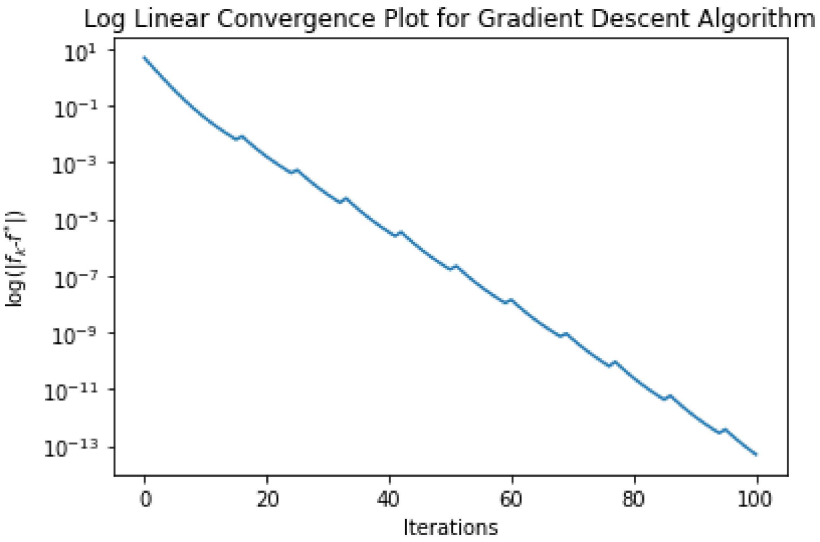
```
X_val,gradient_normal_val,(k),error_values_all  =  gradientdescentalgorithm(function,gradient,H
```

```
The Gradient Descent Algorithm has converged to a point in 100 iterations

Current Gradient Normal is 9.516108713790478e-07
[x1,x2,x3] = [-1.07142846 -0.142857    0.78571416]
```

Log Linear Convergence Plot for Gradient Descent Algorithm

In [304…

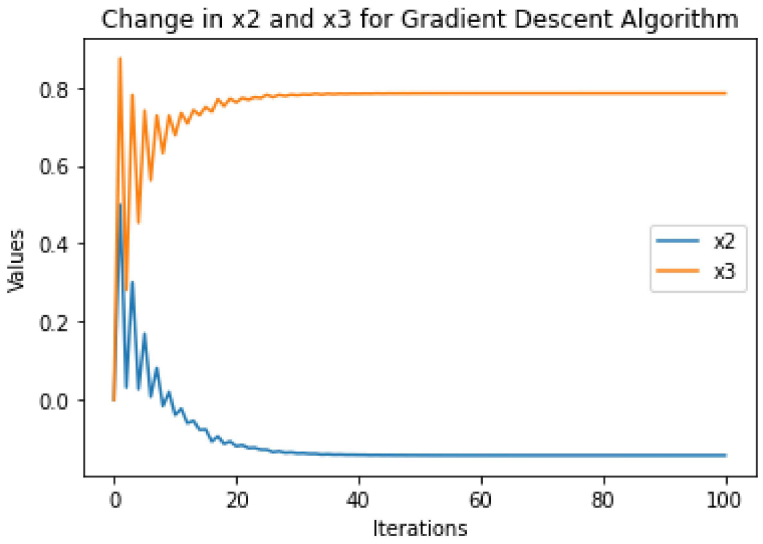
```
plt.yscale("log")
plt.plot(error_values_all)
plt.title('Log Linear Convergence Plot for Gradient Descent Algorithm')
plt.xlabel('Iterations')
plt.ylabel('log(|$f_k$-$f^{*}$|)')
plt.show()
```



Change in x2 and x3 for Gradient Descent Algorithm

In [305…

```
X2_values = []
q = [X2_values.append(i[1]) for i in X_val]
X3_values = []
q = [X3_values.append(i[2]) for i in X_val]
plt.plot(range(0,k+1),X2_values)
plt.plot(range(0,k+1),X3_values)
plt.title('Change in x2 and x3 for Gradient Descent Algorithm')
plt.xlabel('Iterations')
plt.ylabel('Values')
plt.legend(["x2","x3"])
plt.show()
```



Newton's Algorithm

In [306…

```
def newtonsalgorithm(function,gradient,Hessian,x0,maximum_iteration):
  epsilon = (10)**-6
  X = []
  gradient_normal = []
  X.append(x0)
  for k in range(0,maximum_iteration):
      gradient_normal.append(np.linalg.norm(gradient(X[k])))

      if gradient_normal[k] <= epsilon:
        x1 = 1-(2*X[k][0]+3*X[k][1])
        error_values = [abs(function(X[i])-function([-1/7,11/14])) for i in range(len(X))]
        X = np.insert(X,0,x1,axis=1)
        print(f'\nThe Newtons Algorithm has converged to a point in {k} iterations\n')
        print(f'Current Gradient Normal is {gradient_normal[k]}\n[x1,x2,x3] = {X[k]}')
        return X,gradient_normal,(k),error_values

      new_X = X[k] - np.matmul(np.linalg.inv(Hessian), gradient(X[k]))
      X.append(new_X)

      if k == (maximum_iteration - 1):
        print(f"Number of iterations has exceeded the maxinimum iterations, still not converged
        return X,gradient_normal,(k)
```

In [307…

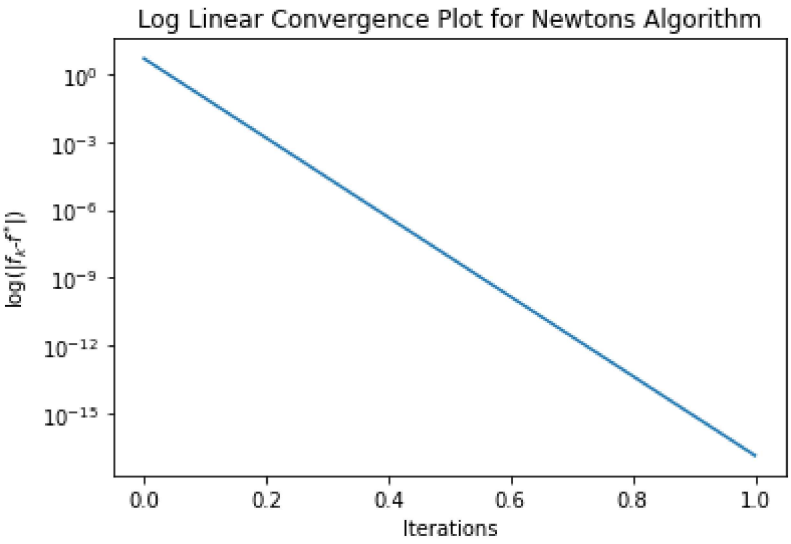
```
X_val,gradient_normal_val,(k),error_values_all  =  newtonsalgorithm(function,gradient,Hessian,r
```

```
The Newtons Algorithm has converged to a point in 1 iterations

Current Gradient Normal is 3.972054645195637e-15
[x1,x2,x3] = [-1.07142857 -0.14285714  0.78571429]
```

Log Linear Convergence Plot for Newtons Algorithm

In [308…

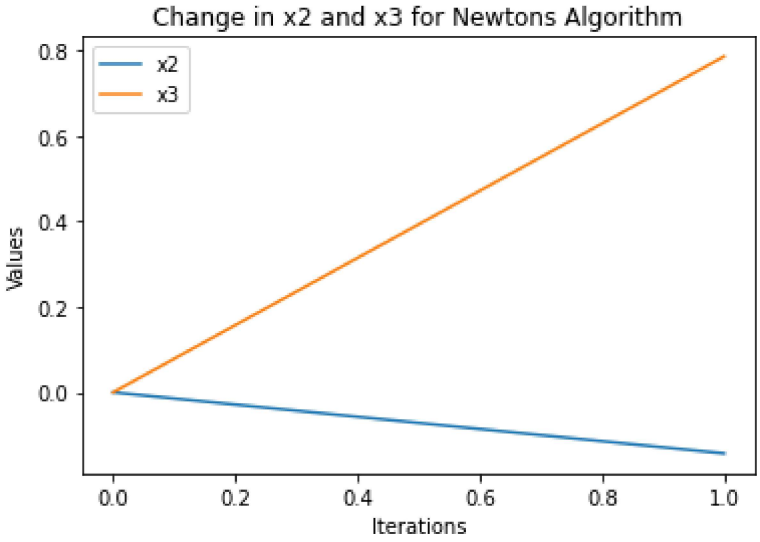
```
plt.yscale("log")
plt.plot(error_values_all)
plt.title('Log Linear Convergence Plot for Newtons Algorithm')
plt.xlabel('Iterations')
plt.ylabel('log(|$f_k$-$f^{*}$|)')
plt.show()
```



Change in x2 and x3 for Newtons Algorithm

In [309…

```
X2_values = []
q = [X2_values.append(i[1]) for i in X_val]
X3_values = []
q = [X3_values.append(i[2]) for i in X_val]
plt.plot(range(0,k+1),X2_values)
plt.plot(range(0,k+1),X3_values)
plt.title('Change in x2 and x3 for Newtons Algorithm')
plt.xlabel('Iterations')
plt.ylabel('Values')
plt.legend(["x2","x3"])
plt.show()
```



SUMMARY FOR GRADIENT DESCENT ALGORITHM

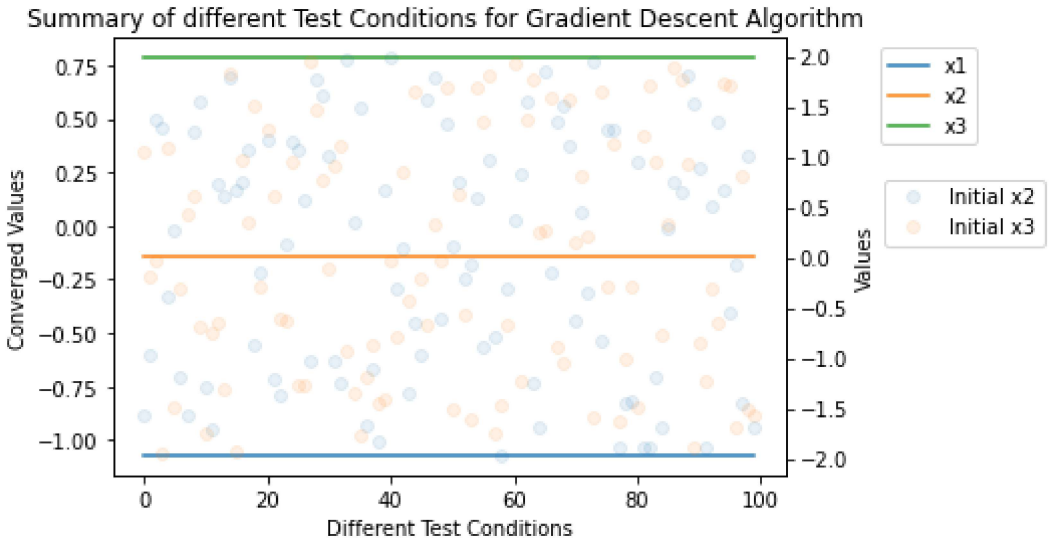
Different initial points are tested and their corresponding results are shown:

In [ ]:

```
all_initial_values = []
all_final_values = []
all_numberofiterations = []
for i in range(0,100):
  x0 = 4*np.random.rand(2)-2
  all_initial_values.append(x0)
  X_val,gradient_normal_val,(k),error_values_all  =  gradientdescentalgorithm(function,gradient
  all_final_values.append(X_val[k])
  all_numberofiterations.append(k)
```

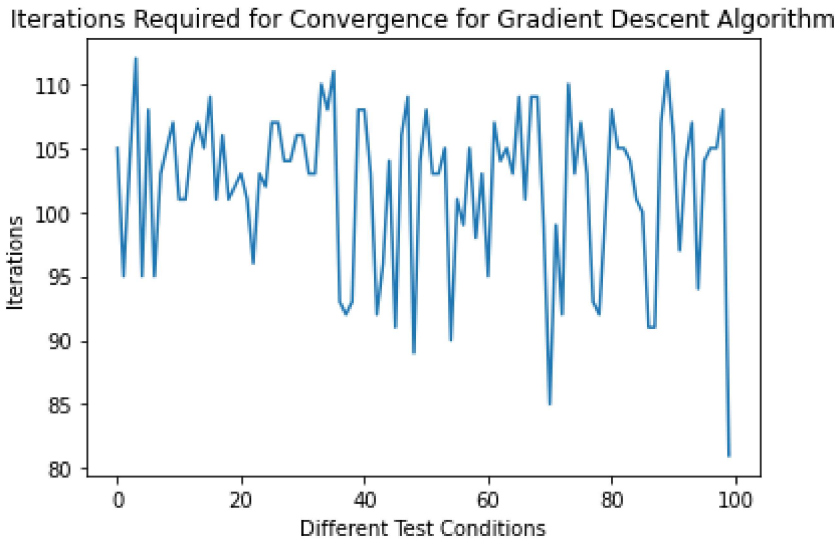
In [311…

```
all_X1_final_values = []
q = [all_X1_final_values.append(i[0]) for i in all_final_values]
all_X2_final_values = []
q = [all_X2_final_values.append(i[1]) for i in all_final_values]
all_X3_final_values = []
q = [all_X3_final_values.append(i[2]) for i in all_final_values]

all_X2_initial_values = []
q = [all_X2_initial_values.append(i[0]) for i in all_initial_values]
all_X3_initial_values = []
q = [all_X3_initial_values.append(i[1]) for i in all_initial_values]

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(range(0,100),all_X1_final_values)
ax1.plot(range(0,100),all_X2_final_values)
ax1.plot(range(0,100),all_X3_final_values)
ax2.plot(range(0,100),all_X2_initial_values,'o',alpha=0.1)
ax2.plot(range(0,100),all_X3_initial_values,'o',alpha=0.1)
plt.title('Summary of different Test Conditions for Gradient Descent Algorithm ')
ax1.set_xlabel('Different Test Conditions')
ax1.set_ylabel('Converged Values')
ax2.set_ylabel('Values')
ax1.legend(["x1","x2","x3"], bbox_to_anchor = (1.3, 1))
ax2.legend(['Initial x2', 'Initial x3'], bbox_to_anchor = (1.4, 0.7))
plt.show()
```



In [312…

```
plt.plot(range(0,100), all_numberofiterations)
plt.title('Iterations Required for Convergence for Gradient Descent Algorithm')
plt.xlabel('Different Test Conditions')
plt.ylabel('Iterations')
plt.show()
```



Determining the the overall impact of initial guess for Gradient Descent Method

In [321…

```
standard_deviation_x1 = 0
standard_deviation_x2 = 0
standard_deviation_x3 = 0
mean_x1 = sum(all_X1_final_values)/len(all_X1_final_values)
mean_x2 = sum(all_X2_final_values)/len(all_X2_final_values)
mean_x3 = sum(all_X3_final_values)/len(all_X3_final_values)
for i in all_X1_final_values:
  standard_deviation_x1 = standard_deviation_x1 + ( i - mean_x1)**2
  standard_deviation_x1 = (standard_deviation_x1/len(all_X1_final_values))**(1/2)

for i in all_X2_final_values:
  standard_deviation_x2 = standard_deviation_x2 + ( i - mean_x2)**2
  standard_deviation_x2 = (standard_deviation_x2/len(all_X2_final_values))**(1/2)

for i in all_X3_final_values:
  standard_deviation_x3 = standard_deviation_x3 + ( i - mean_x3)**2
  standard_deviation_x3 = (standard_deviation_x3/len(all_X3_final_values))**(1/2)

print(f"The Mean of x1 is {mean_x1}, whereas the Standard Deviation is {standard_deviation_x1}"
print(f"The Mean of x2 is {mean_x2}, whereas the Standard Deviation is {standard_deviation_x2}"
print(f"The Mean of x3 is {mean_x3}, whereas the Standard Deviation is {standard_deviation_x3}"
```

```
The Mean of x1 is -1.0714285852862813, whereas the Standard Deviation is 0.010000000000007813
The Mean of x2 is -0.14285709634050625, whereas the Standard Deviation is 0.01000000000001888
The Mean of x3 is 0.7857142593224314, whereas the Standard Deviation is 0.010000000000008913
```

SUMMARY FOR NEWTON'S ALGORITHM

Different initial points are tested and their corresponding results are shown:

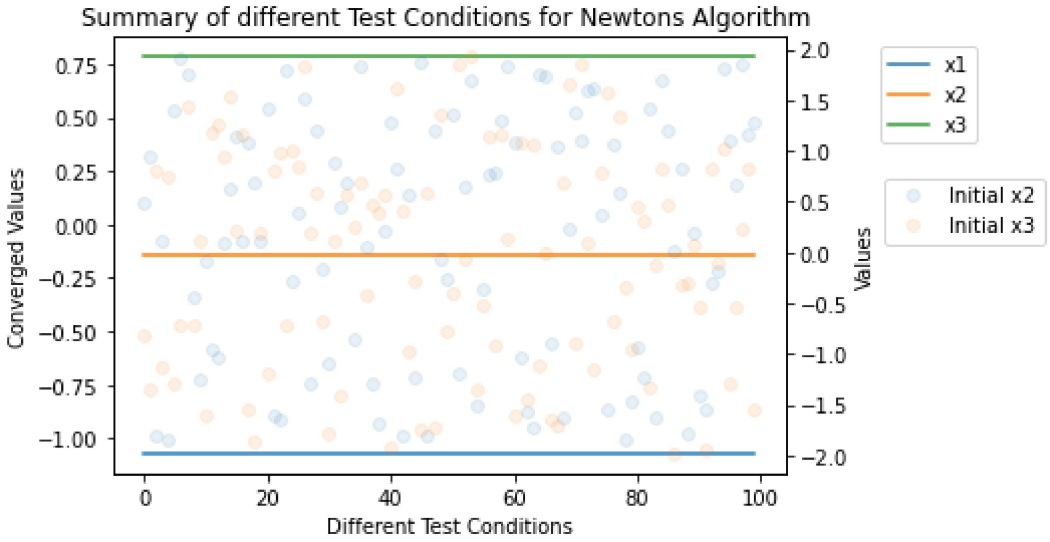
In [ ]:

```
all_initial_values = []
all_final_values = []
all_numberofiterations = []
for i in range(0,100):
  x0 = 4*np.random.rand(2)-2
  all_initial_values.append(x0)
  X_val,gradient_normal_val,(k),error_values_all  =  newtonsalgorithm(function,gradient,Hessiar
  all_final_values.append(X_val[k])
  all_numberofiterations.append(k)
```

In [323…

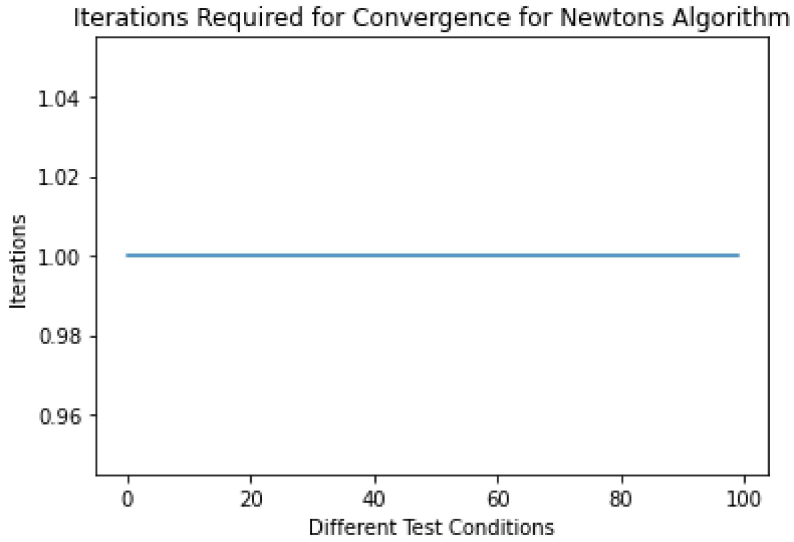
```
all_X1_final_values = []
q = [all_X1_final_values.append(i[0]) for i in all_final_values]
all_X2_final_values = []
q = [all_X2_final_values.append(i[1]) for i in all_final_values]
```

```
all_X3_final_values = []
q = [all_X3_final_values.append(i[2]) for i in all_final_values]

all_X2_initial_values = []
q = [all_X2_initial_values.append(i[0]) for i in all_initial_values]
all_X3_initial_values = []
q = [all_X3_initial_values.append(i[1]) for i in all_initial_values]

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(range(0,100),all_X1_final_values)
ax1.plot(range(0,100),all_X2_final_values)
ax1.plot(range(0,100),all_X3_final_values)
ax2.plot(range(0,100),all_X2_initial_values,'o',alpha=0.1)
ax2.plot(range(0,100),all_X3_initial_values,'o',alpha=0.1)
plt.title('Summary of different Test Conditions for Newtons Algorithm ')
ax1.set_xlabel('Different Test Conditions')
ax1.set_ylabel('Converged Values')
ax2.set_ylabel('Values')
ax1.legend(["x1","x2","x3"], bbox_to_anchor = (1.3, 1))
ax2.legend(['Initial x2', 'Initial x3'], bbox_to_anchor = (1.4, 0.7))
plt.show()
```



In [324…

```
plt.plot(range(0,100), all_numberofiterations)
plt.title('Iterations Required for Convergence for Newtons Algorithm')
plt.xlabel('Different Test Conditions')
plt.ylabel('Iterations')
plt.show()
```



Determining the the overall impact of initial guess for Newton's Method

In [325…

```
standard_deviation_x1 = 0
standard_deviation_x2 = 0
standard_deviation_x3 = 0
mean_x1 = sum(all_X1_final_values)/len(all_X1_final_values)
mean_x2 = sum(all_X2_final_values)/len(all_X2_final_values)
mean_x3 = sum(all_X3_final_values)/len(all_X3_final_values)
for i in all_X1_final_values:
  standard_deviation_x1 = standard_deviation_x1 + ( i - mean_x1)**2
  standard_deviation_x1 = (standard_deviation_x1/len(all_X1_final_values))**(1/2)

for i in all_X2_final_values:
  standard_deviation_x2 = standard_deviation_x2 + ( i - mean_x2)**2
  standard_deviation_x2 = (standard_deviation_x2/len(all_X2_final_values))**(1/2)

for i in all_X3_final_values:
  standard_deviation_x3 = standard_deviation_x3 + ( i - mean_x3)**2
  standard_deviation_x3 = (standard_deviation_x3/len(all_X3_final_values))**(1/2)

print(f"The Mean of x1 is {mean_x1}, whereas the Standard Deviation is {standard_deviation_x1}"
print(f"The Mean of x2 is {mean_x2}, whereas the Standard Deviation is {standard_deviation_x2}"
print(f"The Mean of x3 is {mean_x3}, whereas the Standard Deviation is {standard_deviation_x3}"
```

```
The Mean of x1 is -1.0714285714285705, whereas the Standard Deviation is 0.01
The Mean of x2 is -0.14285714285714285, whereas the Standard Deviation is 0.01
The Mean of x3 is 0.7857142857142863, whereas the Standard Deviation is 0.01
```

I used 100 random initial points to determine the results for both the Gradient Descent Algorithm and Newton's Algorithm. Both the algorithms give the convergence point of:

[x1, x2, x3] = [-1.07142857, -0.14285714, 0.78571429]

Also, the standard deviation for 100 different results obtained for 100 different initial conditions is very small (~0.01).

However, the number iterations required for convergence for Gradient Descent Algorithm varies with the initial point, on the other hand the Newton's Algorithm always converges in 1 iteration, regardless of the inital point.

3. (5 points) Prove that a hyperplane is a convex set. Hint: A hyperplane in $\mathbb{R}^n$ can be expressed as: $\mathbf{a}^T\mathbf{x} = c$ for $\mathbf{x} \in \mathbb{R}^n$, where $\mathbf{a}$ is the normal direction of the hyperplane and c is some constant.

Ans Hyperplane H is defined as:

$$H = \left\{ \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n \;\middle|\; \overbrace{a_1x_1 + a_2x_2 + \cdots + a_nx_n}^{a^Tx} = c \right\}$$

where $a_1, \cdots, a_n \neq 0$ and $c \in \mathbb{R}$

→ Let us consider $x_1, x_2 \in H$

when, $x_1 \in H \Rightarrow a^Tx_1 = c$

& $x_2 \in H \Rightarrow a^Tx_2 = c$

→ A line segment that joins x_1 and x_2 can be defined as:

$v = \lambda x_1 + (1-\lambda)x_2$ where $\lambda \in [0, 1]$

thus,

$$\begin{aligned} a^Tv &= a^T(\lambda x_1 + (1-\lambda)x_2) \\ &= a^T\lambda x_1 + a^T(1-\lambda)x_2 \\ &= a^T(\lambda x_1) + (1-\lambda)(a^Tx_2) \\ &= \lambda c + (1-\lambda)c \\ &= \lambda c + c - \lambda c \end{aligned}$$

$\Rightarrow a^Tv = c$

→ Thus $v \in H$, i.e $\lambda x_1 + (1-\lambda)x_2 \in H$, where $0 \leq \lambda \leq 1$

⇒ Using the defination of a convex set we can conclude that H is a convex set

4. (15 points) Consider the following illumination problem:

$$\min_{\mathbf{p}} \quad \max_{k}\{h(\mathbf{a}_k^T\mathbf{p}, I_t)\}$$
$$\text{subject to:} \quad 0 \le p_i \le p_{\max},$$

where $\mathbf{p} := [p_1, ..., p_n]^T$ are the power output of the n lamps, $\mathbf{a}_k$ for $k = 1, ..., , m$ are fixed parameters for the m mirrors, I_t the target intensity level. $h(I, I_t)$ is defined as follows:

$$h(I, I_t) = \begin{cases} I_t/I & \text{if } I \le I_t \\ I/I_t & \text{if } I_t \le I \end{cases}$$

(a) (5 points) Show that the problem is convex.

(b) (5 points) If we require the overall power output of any of the 10 lamps to be less than p^*, will the problem have a unique solution?

(c) (5 points) If we require no more than 10 lamps to be switched on ($p > 0$), will the problem have a unique solution?

(a)

Ans Let us take $h(a^T p, I_t)$

Gradient : $g = \frac{\partial h}{\partial p} = \left(\frac{\partial h}{\partial I}\right)\left(\frac{\partial a^T p}{\partial p}\right) = h' a$

Hessian : $H = \frac{\partial^2 h}{\partial p^2} = \left(\frac{\partial h'}{\partial I}\right)\left(\frac{\partial a^T p}{\partial p}\right) = h'' a a^T$

→ To prove the problem is a convex problem we need to show that Hessian matrix is positive definite.

→ Let λ be an eigen value of aa^T and q be the eigen vector

then, $(aa^T)q = \lambda q$

$\therefore \quad q^T a a^T q = \lambda q^T q$

$\therefore \quad \lambda = \frac{q^T a a^T q}{q^T q}$

Let, $w = a^T q \quad ; \quad w^T = q^T a$

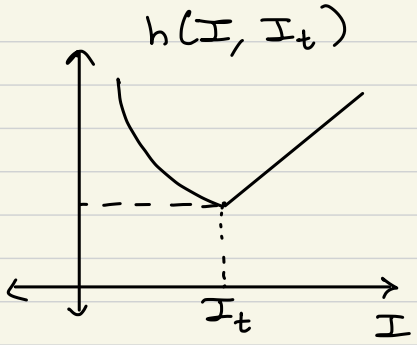
$$\therefore \lambda = \frac{w^T w}{q^T q}$$

→ Consider the i^{th} element of the above multiplication:

$$w_i^T w_i = w_i^2$$

and, $$q_i^T q_i = q_i^2$$

→ Here since all the values are squared we can conclude that $\lambda \geq 0$. So the Hessian matrix is positive definite which means that $h(a^T p, I_t)$ is convex, but not strictly convex and $\max_K \{h(a_K^T p, I_t)\}$ is convex function



→ For the conditions:

$$h(I, I_t) = \frac{I}{I_t} \quad \text{when} \quad I < I_t$$

By using $I = a_K^T p = a_{K_1} p_1 + \cdots\cdots + a_{Kn} p_n$

$$h(I, I_t) = \frac{a_{K_1} p_1 + \cdots\cdots + a_{Kn} p_n}{I_t}$$

and $$h(I, I_t) = \frac{I_t}{I} \quad \text{when} \quad I_t < I$$

$$\therefore h(I, I_t) = \frac{I_t}{a_{K_1} p_1 + \cdots + a_{Kn} p_n}$$

Also,

$$h''(I, I_t) = \begin{cases} \frac{2 I_t}{I^3}, & I \leq I_t \\ 0, & I \geq I_t \end{cases}$$

$\Rightarrow h'' \geq 0$

So for the function $h(I, I_t)$ to be convex $I > 0$ has to true, means $a_K^T p > 0$

This is valid for all k^{th} term in $h(a_K^T, I_t)$ as by property we know the max of convex set is convex

thus, $\max_K \{h(a_K^T p, I_t)\}$ is a convex function

when $a_K^T p > 0$

→ The constraint $0 \leq p_i < p_{max}$ is also the convex constraint because p_i is the two half plain $p_i \geq 0$ and $p_i < p_{max}$

So we can conclude that the Problem is convex problem

(b)

Ans If any of 10 lamps from the n lamps has power less than p^*

→
$$\left.\begin{aligned} p_1 + p_2 + \cdots + p_{10} &\leq p^* \\ &\vdots \\ p_{n-10} + p_{n-9} + \cdots + p_n &\leq p^* \end{aligned}\right\} \text{(1)}$$

We have C_{10}^{n} combination of lamps that power are less than p^*

Converting equations (1) in vector form

$$\begin{bmatrix} 1 & 1 & \cdots & 0 & 0 \end{bmatrix}_{1\times n} \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}_{n\times 1} \leq p^*$$

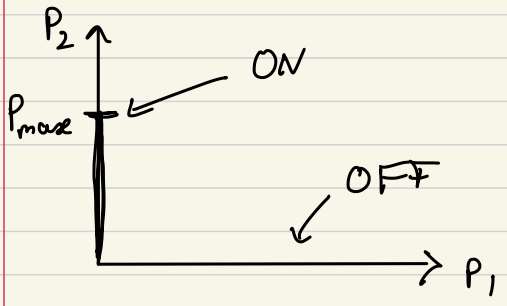
$$\begin{bmatrix} 0 & 1 & 1 & \cdots & 0 & 0 \end{bmatrix}_{1\times n} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}_{n\times 1} \leq p^* \quad \Big| \quad \begin{bmatrix} 0 & \cdots & 1 & 1 \end{bmatrix}_{1\times n} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} \leq p^*$$

So by looking we can say that the formulated constraints are in a linear combination, which shows that the problem is convex and the nature of feasible solution does not change the convexity of problem, so it has a unique solution

(C)

Ans If we require no more than 10 lamps to be switched on then we cannot say how many solution we can have, because their can be n number of ways to start lamps.

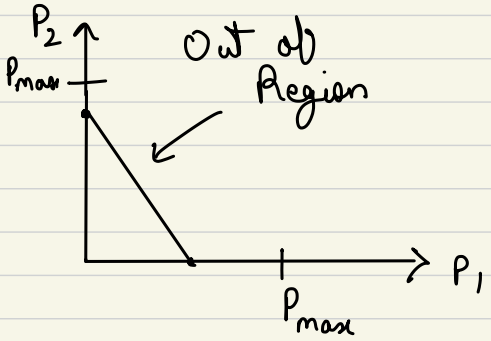
For example:

- 

Here, $P_2 \rightarrow$ ON
$P_1 \rightarrow$ OFF
$\Rightarrow$ Convex Set

- 

Here, $P_2 \rightarrow$ OFF
$P_1 \rightarrow$ ON
$\Rightarrow$ Convex Set

- 

When we choose both P_1 & P_2 to be ON the point on any location between P_1 & P_2 is out of the set
$\Rightarrow$ Non - Convex

$\rightarrow$ Thus, by using the constant of not more than 10 lamps to be switched on, the original problem becomes a non-convex problem & we cannot tell how many local solutions we have

$\Rightarrow$ Thus, it cannot have a unique solution

5. (10 points) Let $c(x)$ be the cost of producing x amount of product A and assume that $c(x)$ is differentiable everywhere. Let y be the price set for the product. Assuming that the product is sold out. The total profit is defined as

$$c^*(y) = \max_x \{xy - c(x)\}.$$

Show that $c^*(y)$ is a convex function with respect to y.

Ans Given problem:

$$c^*(y) = \max_x \{xy - c(x)\}$$

→ Let us consider the i^{th} element of the function

$$f_i = (x_i y_i - c(x_i))$$

→ Determining gradient

$$g = \frac{\partial f_i}{\partial y_i} = x_i$$

→ Determining Hessian:

$$H = \frac{\partial^2 f_i}{\partial y_i^2} = 0$$

Since the Hessian is zero we can say that the function is linear function and a convex function.

→ Thus, we have a set of functions as follows:

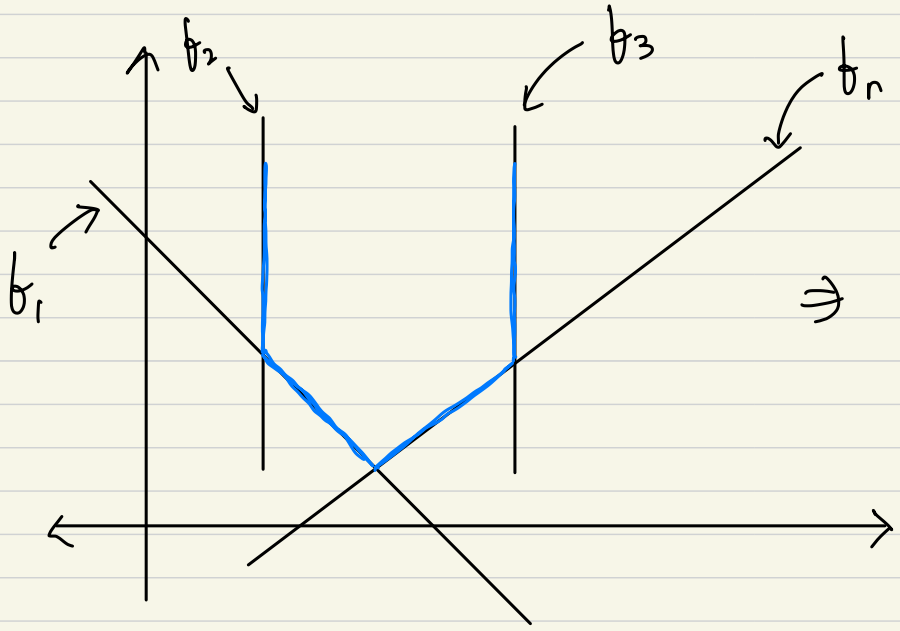
$$f_1 = (x_1 y_1 - c(x_1))$$
$$f_2 = (x_2 y_2 - c(x_2))$$
$$f_3 = (x_3 y_3 - c(x_3))$$
$$\vdots$$
$$f_n = (x_n y_n - c(x_n))$$

All of these functions are linear and convex in y_i

→ We have a set of functions that are linear and convex, therefore by the property we can say that, the max of convex set is convex itself

$$\Rightarrow \max_x \{xy - c(x)\} \text{ is convex w.r.t } y$$

→ Graphically we show that :



$\Rightarrow c^*(y)$ is convex