Homework-2

Name: Gaurav Lavadiya
ASU ID: 1222768924

Problem 1

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

$$\nabla_x f(x_1, x_2) = \begin{bmatrix} 4x_1 - 4x_2 \\ -4x_1 + 3x_2 + 1 \end{bmatrix}$$

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

- Stationary point is when gradient is zero

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

Solving for x_1, x_2

$$\begin{bmatrix} 4 & -4 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\left[\begin{array}{cc|c} 4 & -4 & 0 \\ -4 & 3 & -1 \end{array}\right] \longrightarrow \left[\begin{array}{cc|c} 4 & -4 & 0 \\ 0 & -1 & -1 \end{array}\right]$$

$$\begin{aligned} -x_2 &= -1 \\ 4x_1 - 4x_2 &= 0 \end{aligned} \Rightarrow \begin{aligned} x_2 &= 1 \\ x_1 &= 1 \end{aligned}$$

$\Rightarrow$ Stationary Points $= \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

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

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

$$(4-\lambda)(3-\lambda) - 16 = 0$$
$$12 - 7\lambda + \lambda^2 - 16 = 0$$

$$\boxed{\lambda^2 - 7\lambda - 4 = 0}$$

→ So, eigen values are following

$$\lambda_1 = \frac{7+\sqrt{65}}{2} \quad \& \quad \lambda_2 = -\frac{(-7+\sqrt{65})}{2}$$

$$\approx -0.531128 \qquad \approx 7.531128$$

– So, one eigen value is positive another is negative thats why Hassian is indefinite and so, that the sationary point is a saddle point.

→ direction of downslop away from saddle point

- $f(x_1,x_2) = f(1,1) + \nabla f^T_{(x_1,x_2)} (x-x^*) + \frac{1}{2}(x-x^*)^T H (x-x^*)$

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

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

- lets say $x_1 - 1 = \partial x_1$
  $x_2 - 1 = \partial x_2$ then

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

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

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

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

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

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

$$\Rightarrow f(x_1, x_2) - f(1,1) = \frac{1}{2}(2\partial x_1 - \partial x_2)(2\partial x_1 - 3\partial x_2)$$

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

so, $(2\partial x_1 - \partial x_2)(2\partial x_1 - 3\partial x_2) < 0$

this to be true

$$(2\partial x_1 - \partial x_2) < 0 \text{ and } (2\partial x_1 - 3\partial x_2) > 0$$

or

$$(2\partial x_1 - \partial x_2) > 0 \text{ and } (2\partial x_1 - 3\partial x_2) < 0$$

0 Problem 2

(a) find the point in plane

$x_1 + 2x_2 + 3x_3 = 1$ in R^3 that is nearest to the point $(-1, 0, 1)^T$

→ from eq. of distance two points

$$\min\{(x_1+1)^2 + (x_2)^2 + (x_3-1)^2\} \quad -(1)$$

→ making problem unconstrained using plane equation

$$x_1 + 2x_2 + 3x_3 = 1$$
$$x_1 = 1 - (2x_2 + 3x_3) \quad -(2)$$

→

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

$$\nabla f = \begin{bmatrix} -4(2-2x_2-3x_3) + 2x_2 \\ -6(2-2x_2-3x_2) + 2(x_3-1) \end{bmatrix}$$

$$-4(2-2x_2-3x_3) + 2x_2 \Rightarrow 10x_2 + 12x_3 - 8 = 0 \quad -(3)$$
$$-6(2-2x_2-3x_3) + 2(x_3-1) \Rightarrow 12x_2 + 20x_3 - 14 = 0 \quad -(4)$$

→ Solving (3) & (4) for stationary point

$$\left[\begin{array}{cc|c} 10 & 12 & 8 \\ 12 & 20 & 14 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} 1 & 6/5 & 4/5 \\ 12 & 20 & 14 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} 1 & 6/5 & 4/5 \\ 0 & 28/5 & 22/5 \end{array}\right]$$

$$x_3 = \frac{11}{14} \qquad x_2 = \frac{-1}{7} \qquad x_1 = 1 - 2\left(\frac{-1}{7}\right) - 3\left(\frac{11}{14}\right)$$

$$x_1 = -15/14$$

$x_1 = \frac{-15}{14}$ $x_2 = \frac{-1}{7}$ $x_3 = \frac{11}{14}$

→ now

$$\nabla f = \begin{bmatrix} 10x_2 + 12x_3 - 8 \\ 12x_2 + 20x_3 - 14 \end{bmatrix}$$

Hessian

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

→ $H - \lambda I = \begin{bmatrix} 10-\lambda & 12 \\ 12 & 20-\lambda \end{bmatrix}$

$\det(H - \lambda I) = 0$

$$(10-\lambda)(20-\lambda) - 144 = 0$$
$$200 - 30\lambda + \lambda^2 - 144 = 0$$

$$\lambda^2 - 30\lambda + 56 = 0$$
$$(\lambda - 28)(\lambda - 2) = 0$$

$\lambda_1 = 28$ & $\lambda_2 = 2$

⇒ Code for Problem 2(b) is on next Page

→ both the eiger values are positive hence Hassian is Possive definite and given Problem is Convex Problem

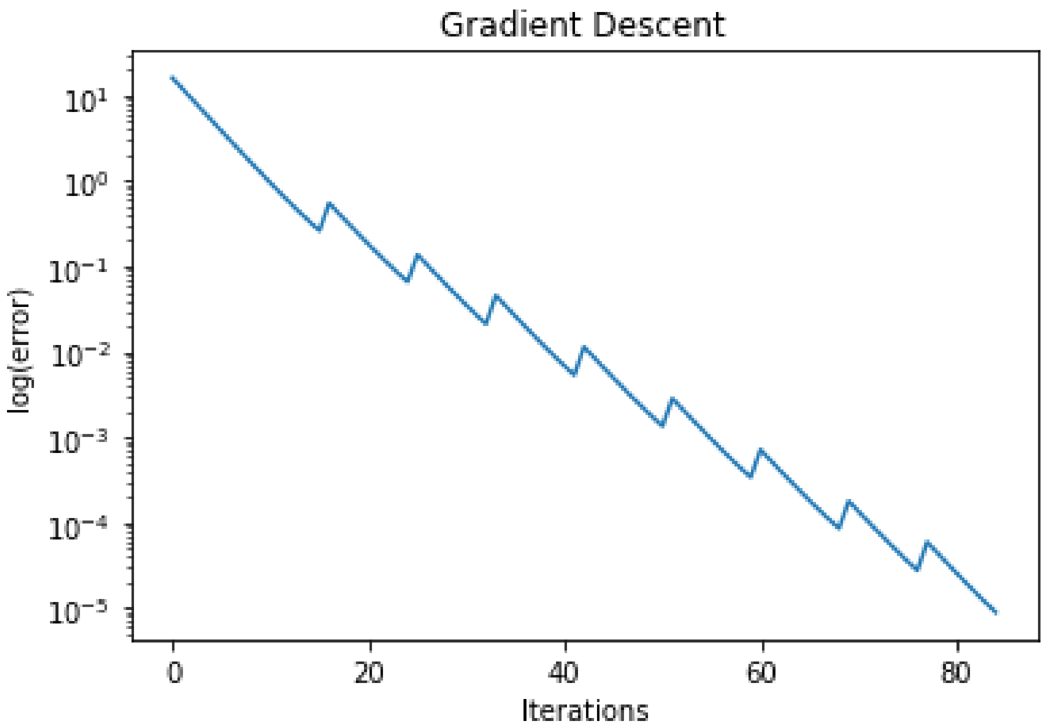
Name: Gaurav Lavadiya

ASU ID: 1222768924

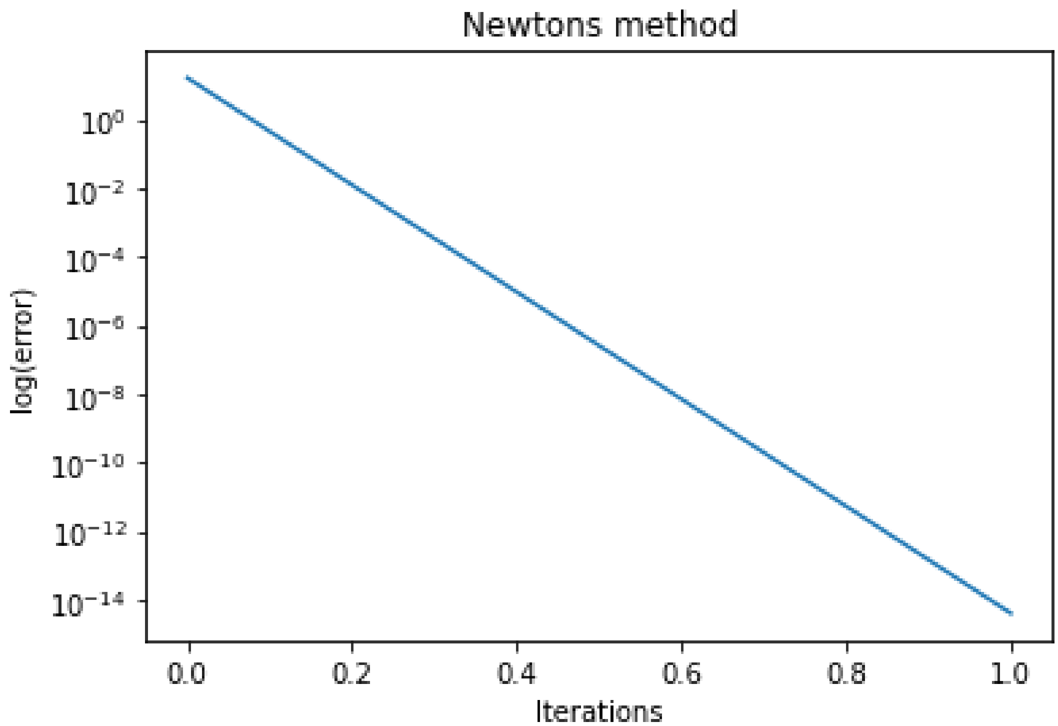
```
import numpy as np
import scipy
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# function f(x)
def f(x):
  return (2-2*x[0]-3*x[1])**2 + x[0]**2 + (x[1]-1)**2
# gradiant g(x)
def grad(x):
  return np.array([(10*x[0]+12*x[1]-8) , (12*x[0]+20*x[1]-14)])
#hessian
H = np.array([[10,12],[12,20]])

#intial value
x0 = np.array([0,0])
print('intial value=',x0)

#gradient descent

m=0
iter = [m]  #to store number of iterations
res = [x0]  #to store results
x = x0

error = np.linalg.norm(grad(x))
error_str = []
error_str.append(error)

def linesearch(x):
  alpha = 1
  t = 0.3
  b = -1*grad(x)
  def phi(alpha, x):
      return f(x) - t *alpha*np.matmul(np.transpose(grad(x)),b)
  while phi(alpha,x) < f(x+alpha*b):
    alpha = 0.5*alpha
  return alpha

while error >= 0.00001:
  alpha = linesearch(x)
  x = x - alpha*grad(x)
  res.append(x)
  error = np.linalg.norm(grad(x))
  error_str.append(error)
```

```
  m = m+1
  iter.append(m)

print('\nnumber of iterations=',m)
print('x1=',1-(2*res[-1][0]+3*res[-1][1]))
print('x2=',res[-1][0])
print('x3=',res[-1][1])

plt.title('Gradient Descent')
plt.xlabel('Iterations')
plt.ylabel('log(error)')
plt.plot(error_str)
plt.yscale('log')
plt.show()


#newton method

Hinv = np.linalg.inv(H)
n=0
itern = [n]
resn = [x0]
x = x0
error = np.linalg.norm(grad(x))
error_str = []
error_str.append(error)

while error >= 0.000001:
  alpha = linesearch(x)
  x = x - np.dot(Hinv,grad(x))
  res.append(x)
  error = np.linalg.norm(grad(x))
  error_str.append(error)
  n = n+1
  iter.append(n)

print('number of iterations=',n)
print('x1=',1-(2*res[-1][0]+3*res[-1][1]))
print('x2=',res[-1][0])
print('x3=',res[-1][1])


plt.title('Newtons method')
plt.xlabel('Iterations')
plt.ylabel('log(error)')
plt.plot(error_str)
plt.yscale('log')
plt.show()
```

```
intial value= [0 0]

number of iterations= 84
x1= -1.0714275611385826
x2= -0.14285548982167645
x3= 0.7857128469273119
```




```
number of iterations= 1
x1= -1.071428571428572
x2= -0.1428571428571428
x3= 0.7857142857142858
```

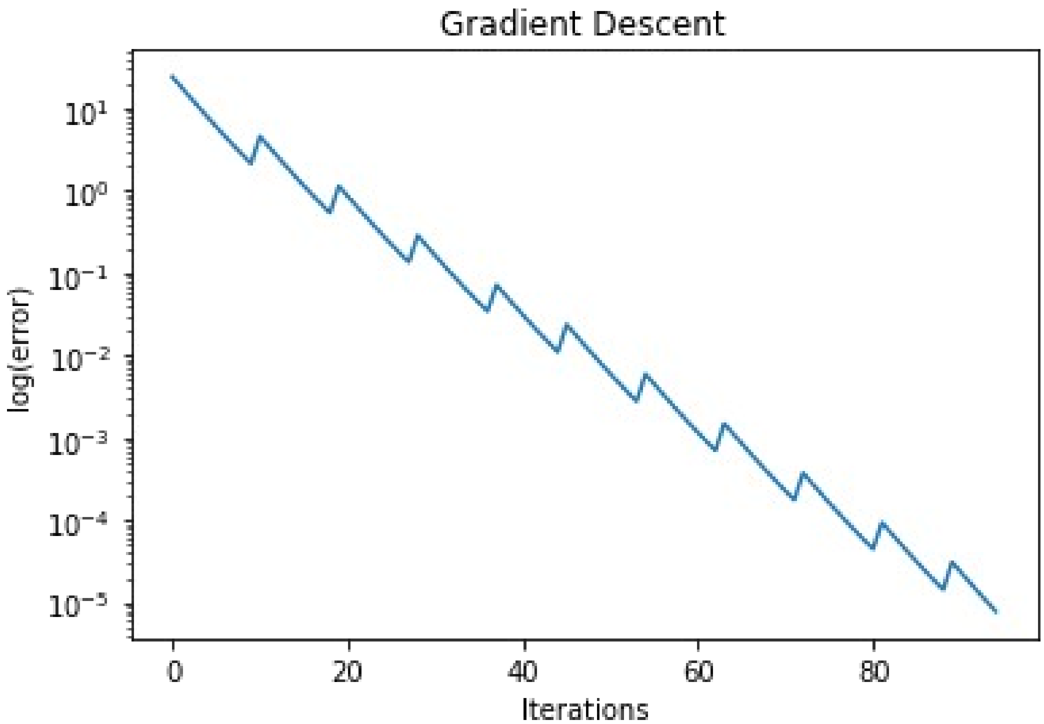



```
x0 = np.array([1,-1])
print('intial value=',x0)

#gradient descent

m=0
iter = [m]  #to store number of iterations
res = [x0]  #to store results
x = x0

error = np.linalg.norm(grad(x))
error_str = []
error_str.append(error)

def linesearch(x):
  alpha = 1
  t = 0.3
  b = -1*grad(x)
  def phi(alpha, x):
      return f(x) - t *alpha*np.matmul(np.transpose(grad(x)),b)
  while phi(alpha,x) < f(x+alpha*b):
    alpha = 0.5*alpha
  return alpha

while error >= 0.00001:
  alpha = linesearch(x)
  x = x - alpha*grad(x)
```

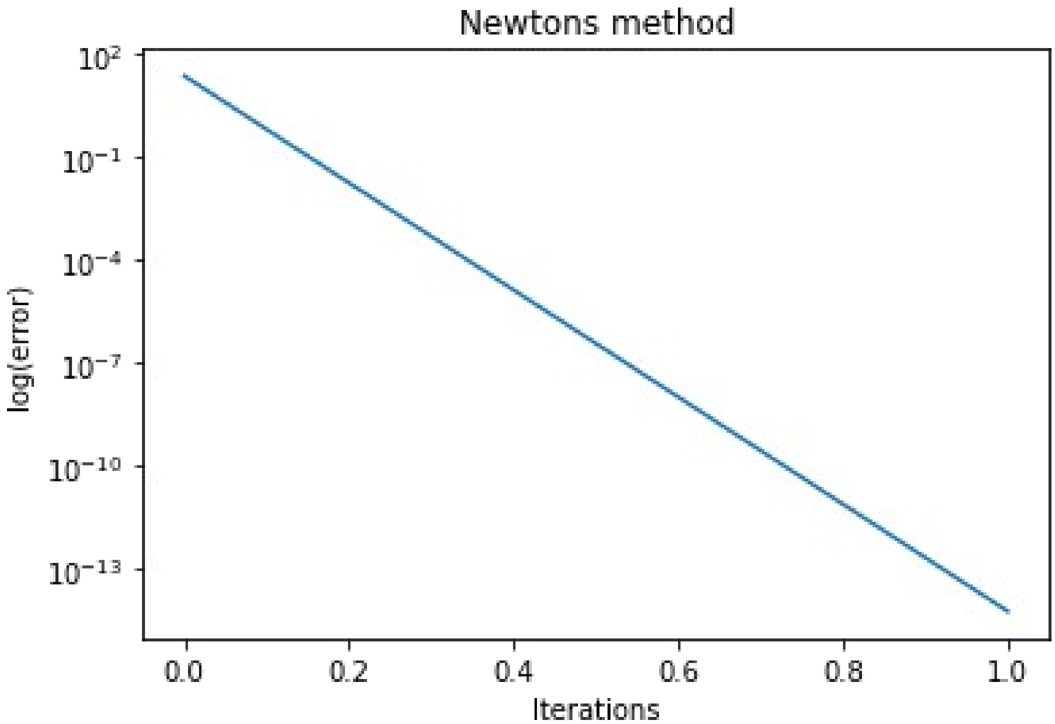
```
    res.append(x)
    error = np.linalg.norm(grad(x))
    error_str.append(error)
    m = m+1
    iter.append(m)

print('\nnumber of iterations=',m)
print('x1=',1-(2*res[-1][0]+3*res[-1][1]))
print('x2=',res[-1][0])
print('x3=',res[-1][1])

plt.title('Gradient Descent')
plt.xlabel('Iterations')
plt.ylabel('log(error)')
plt.plot(error_str)
plt.yscale('log')
plt.show()


#newton method

Hinv = np.linalg.inv(H)
n=0
itern = [n]
resn = [x0]
x = x0
error = np.linalg.norm(grad(x))
error_str = []
error_str.append(error)

while error >= 0.000001:
  alpha = linesearch(x)
  x = x - np.dot(Hinv,grad(x))
  res.append(x)
  error = np.linalg.norm(grad(x))
  error_str.append(error)
  n = n+1
  iter.append(n)

print('number of iterations=',n)
print('x1=',1-(2*res[-1][0]+3*res[-1][1]))
print('x2=',res[-1][0])
print('x3=',res[-1][1])


plt.title('Newtons method')
plt.xlabel('Iterations')
plt.ylabel('log(error)')
plt.plot(error_str)
```

```
plt.yscale('log')
plt.show()
```

```
intial value= [ 1 -1]

number of iterations= 94
x1= -1.0714276341112745
x2= -0.14285606366951104
x3= 0.7857132538167656
```




```
number of iterations= 1
x1= -1.0714285714285707
x2= -0.14285714285714368
x3= 0.785714285714286
```

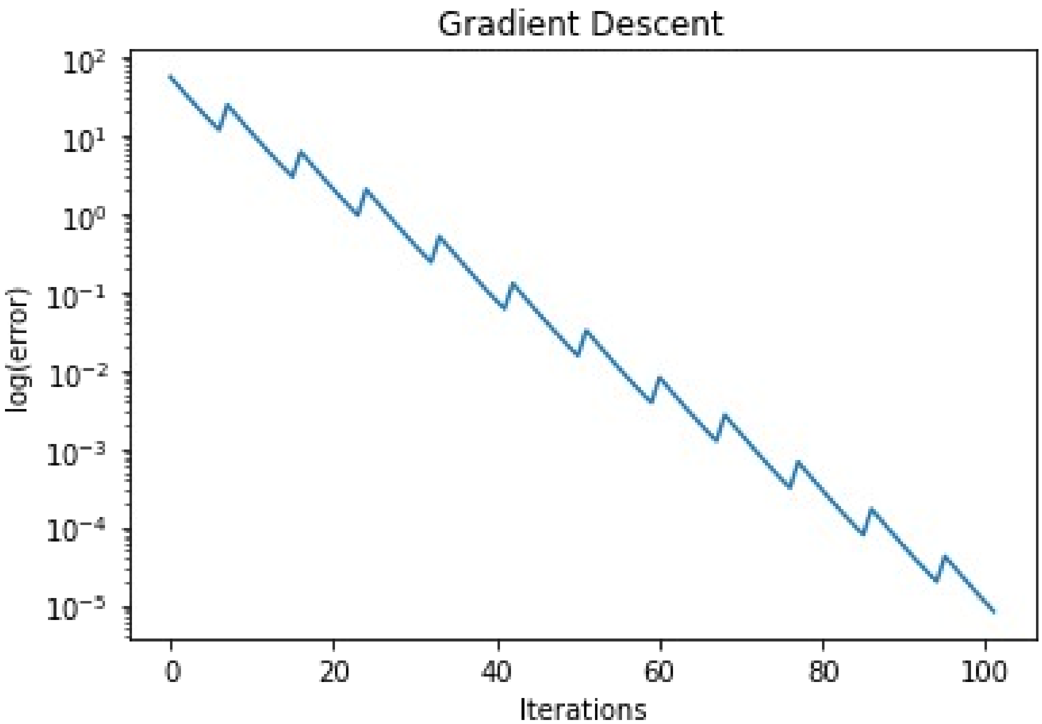



```
x0 = np.array([5,-5])
print('intial value=',x0)

#gradient descent

m=0
iter = [m]  #to store number of iterations
res = [x0]  #to store results
x = x0

error = np.linalg.norm(grad(x))
error_str = []
error_str.append(error)

def linesearch(x):
  alpha = 1
  t = 0.3
  b = -1*grad(x)
  def phi(alpha, x):
      return f(x) - t *alpha*np.matmul(np.transpose(grad(x)),b)
  while phi(alpha,x) < f(x+alpha*b):
    alpha = 0.5*alpha
  return alpha

while error >= 0.00001:
  alpha = linesearch(x)
  x = x - alpha*grad(x)
```

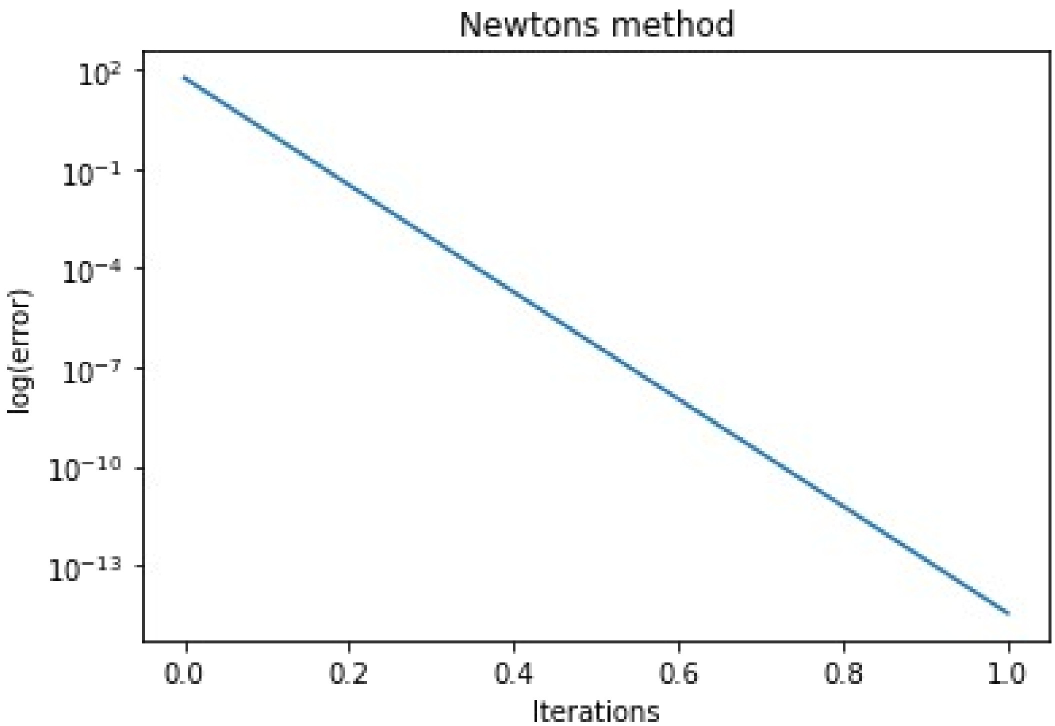
```
  res.append(x)
  error = np.linalg.norm(grad(x))
  error_str.append(error)
  m = m+1
  iter.append(m)

print('\nnumber of iterations=',m)
print('x1=',1-(2*res[-1][0]+3*res[-1][1]))
print('x2=',res[-1][0])
print('x3=',res[-1][1])

plt.title('Gradient Descent')
plt.xlabel('Iterations')
plt.ylabel('log(error)')
plt.plot(error_str)
plt.yscale('log')
plt.show()


#newton method

Hinv = np.linalg.inv(H)
n=0
itern = [n]
resn = [x0]
x = x0
error = np.linalg.norm(grad(x))
error_str = []
error_str.append(error)

while error >= 0.000001:
  alpha = linesearch(x)
  x = x - np.dot(Hinv,grad(x))
  res.append(x)
  error = np.linalg.norm(grad(x))
  error_str.append(error)
  n = n+1
  iter.append(n)

print('number of iterations=',n)
print('x1=',1-(2*res[-1][0]+3*res[-1][1]))
print('x2=',res[-1][0])
print('x3=',res[-1][1])


plt.title('Newtons method')
plt.xlabel('Iterations')
plt.ylabel('log(error)')
plt.plot(error_str)
```

```
plt.yscale('log')
plt.show()
```

```
intial value= [ 5 -5]

number of iterations= 101
x1= -1.0714295316269307
x2= -0.14285540680579217
x3= 0.7857134484128383
```




```
number of iterations= 1
x1= -1.0714285714285712
x2= -0.14285714285714413
x3= 0.7857142857142865
```



#findings:

1. from the resutls we got for 3 different intial values number of iterations changes in gradient descent while for newtons methon there is no change in number of iterations

2. also as gradient descent converges liners it need more internation compare to newtons method with quadratic covergence.

3. regardless of number of iternations both method converge to the same points

⇒ code for Problem 2(b) is on next Page

→ both the eiger values are Positive hence Hassian is Possive definite and given Problem is Convex Problem

○ Problem 3

→ Pove that hyperplane ($a^T x = c$) is a convex set.

- hyperplane (H) $a^T x = c$ for $x \in \mathbb{R}^n$ where a is the normal direction of the hyper plane and c is some constant

- let's assume $x_1, x_2 \in H$

So, that $a^T x_1 = c$ & $a^T x_2 = c$

- from convex set defination

$$y = \lambda x_1 + (1-\lambda) x_2 \quad : \lambda \in [0,1]$$

- lets prove that y is also on the hyper plane

$$y = \lambda x_1 + (1-\lambda) x_2$$

$$a^T y = a^T \lambda x_1 + a^T (1-\lambda) x_2$$

$$a^T y = \lambda a^T x_1 + (1-\lambda) a^T x_2$$

$$a^T y = \lambda c + (1-\lambda) c$$

$$a^T y = \lambda c + c - \lambda c$$

$$\boxed{a^T y = c}$$

→ this suggest that y is also on the hyperplane

so from defination of convex set we can say that hyperplan is a convex set.

○ Problem 4

- illumination problem $\min\limits_{p} \max\limits_{k} \{h(a_k^T p, I_t)\}$

- p is power output $p = [p_1, \ldots p_n]^T$ of n lamps

- a_k for $k = [1, \ldots, m]$ are fixed parameters for the m mirrors

- I_t the target intensity level

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

(a) Show that the Problem is convex

$$\min_{p} \max_{k} \{h(a_k^T p, It)\}$$

$f = h(a^T p, It)$

Gradient $= \frac{\partial f}{\partial p} = \frac{\partial h}{\partial p} \cdot \frac{\partial a^T p}{\partial p} = h' \cdot a$

Hassion $= \frac{\partial f^2}{\partial^2 p} = h'' \cdot a \cdot a^T$

here $h'' \geq 0$

- Lemma: $A \in \mathbb{R}^{m \times m}$ if $\forall w \in \mathbb{R}^m$ $w \neq 0$ $w^T A w \geq 0$
  $A = A^T$ then A is P.S.d

$w^T a . a^T w = y^2 \geq 0$ $\quad\quad \because a^T w = y = w^T a$

- Since y^2 then $\lambda \geq 0$, so H is P.S.d. means

$h(a^T p, It)$ is convex but not stricly convex

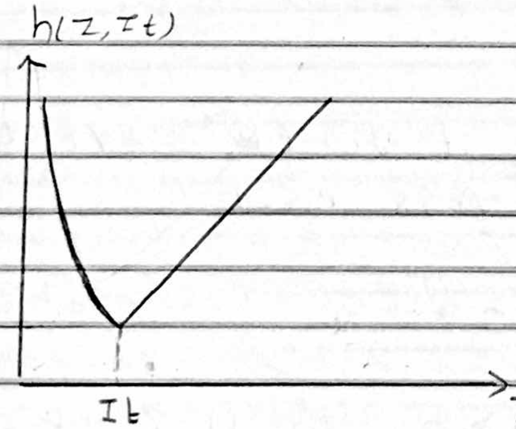
$\max\{h(a^T p, It)$ is also convex function

$I = a_k^T P = a_{k_1} P_1 + a_{k_2} P_2 + \dots + a_{k_n} P_n$

when $I \geq I_t$ $\quad h = \frac{I}{I_t} = \frac{a_{k1}p_1 + \dots + a_{kn}p_n}{I_t}$

when $I \leq I_t$ $\quad h = \frac{I_t}{I} = \frac{I_t}{a_{k1}p_1 + \dots + a_{kn}p_n}$

- we can say h to be convex $I > 0$, so when the substitution is done, $a^T p > 0$
- Ploting the function also show the convex behavior.



- SO, Problem is convex but not strickly convex
- also in this case Problem will have unique solution

(b) 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?

- Problem has n lump and in given condition any 10 lamp have less power than P^*
- which is a type of contraint for this problem
- this contrain is linear contraint and we know that linear contraints are convex

⇒ So, does not change the nature of the problem so, problem is still a convex problem with unique solution

(c) if we require no more than 10 lamps to be switched on $(P>0)$, will the problem have unique solution?

- we can not tell, how many solution we can get, there are number of ways to turn on lights which could result in non-unique solution
- So, for this case problem don't or may not have a unique solution.

Problem 5

→ $c(x)$ cost of prodcing x amont of Product A

→ y price set for the product

→ total Profit defined as $C^*(y) = \max_x \{xy - c(x)\}$

– considering i^{th} element of the function

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

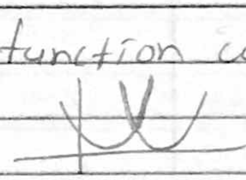
– gradient $g = \frac{\partial f_i}{\partial y_i} = x_i$

– Hasian $H = 0$

→ Hasian of the given function is zero which indicates that function is a linear function and we know that the linear function are convex functions

– so f_i is an convex fuction

– we know that taking max of all function will give a convex function as well



→ So we can say that given function $C^*(y)$ is a convex function