Vinit Salian
1222246623

H.W2

Design Optimization

1)

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

To check whether the point is saddle or not, we need to show, that the eigen values of the hessian matrix are $\lambda_i > 0$ & $\lambda_i < 0$. When this happen, it is a Hessian matrix.

The eigen values of hessian matrix basically give information about the curvature of the function, so when our eigen values are $\lambda_i \geq 0$ & $\lambda < 0$, this mean on side the function has a positive curvature (bending up), while on the other side side (bending down), it has a negative curvature. Thus it must be a saddle point.

To start with we will find the gradient of the function

$$\therefore \nabla f = \begin{bmatrix} \frac{df}{dx_1} \\ \frac{df}{dx_2} \end{bmatrix}$$

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

To ~~show~~ for zero gradient, we will equate this to zero

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

$\therefore 4x_1 - 4x_2 = 0 \rightarrow x_1 = x_2 \quad -(1)$

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

$-x_2 + 1 = 0$ using (1)

$x_2 = 1$

$\therefore x_1 = 1$

Thus $(1, 1)$ is the stationary point of the function $x^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Calculating Hessian

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

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

To compute eigen values of this

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

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

determinant $= 0$

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

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

~~$\lambda^2 - 7\lambda + 24 = 0$~~

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

$\therefore \lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{7 \pm \sqrt{49 - 4(-4)}}{2} = \frac{7 \pm \sqrt{65}}{2}$

$= \frac{7 \pm 8.062}{2}$

$\lambda_1 = \frac{7 + 8.062}{2}$ $\quad$ $\lambda_2 = \frac{7 - 8.062}{2}$

$\lambda_1 = 7.531$ $\quad$ $\lambda_2 = -0.531$

Thus looking at λ_1 & λ_2, we can say that the stationary point of function is saddle, since one eigen value > 0 & another eigen value is less than zero.

→ Doing taylor series expansion of the function

$\therefore f(x) = f(x_0) + \nabla f\big|_{x_0}^T (x - x_0) + \frac{1}{2}(x - x_0)^T H\big|_{x_0} (x - x_0)$

$x_0 = (1,1)$

$$f(x) = f(1,1) + \begin{bmatrix} 4x_1 - 4x_2 \\ -4x_1 + 3x_2 + 1 \end{bmatrix}^T_{x_0} \begin{bmatrix} x_1 - 0 \\ x_2 - 0 \end{bmatrix}$$

$$+ \frac{1}{2} \begin{bmatrix} x_1 - 1 & x_2 - 1 \end{bmatrix} \begin{bmatrix} 4 & -4 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} x_1 - 1 \\ x_2 - 1 \end{bmatrix}$$

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

let $x_1 - 1 = dx_1$

$x_2 - 1 = dx_2$

$$\therefore f(x) = f(1,1) + \frac{1}{2} \begin{bmatrix} dx_1 & dx_2 \end{bmatrix} \begin{bmatrix} 4 & -4 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix}$$

$$\therefore f(x) = f(1,1) + \frac{1}{2} \begin{bmatrix} 4dx_1 - 4dx_2 & -4dx_1 \\ & +3dx_2 \end{bmatrix} \begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix}$$

$$f(x) = f(1,1) + \frac{1}{2} \begin{bmatrix} 4dx_1^2 - 4dx_1 dx_2 & -4dx_1 dx_2 + 3dx_2^2 \end{bmatrix}$$

$$f(x) = f(1,1) + \frac{1}{2} \left(4dx_1^2 - 8dx_1 dx_2 + 3dx_2^2 \right)$$

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

$$f(x) = f(1,1) + \left(2\delta x_1^2 - 3\delta x_1\delta x_2 - \delta x_1\delta x_2 + \frac{3}{2}\delta x_2^2\right)$$

$$f(x) = f(1,1) + \left(\delta x_1(2\delta x_1 - 3\delta x_2) - \frac{\delta x_2}{2}(2\delta x_1 - 3\delta x_2)\right)$$

$$\therefore f(x) = f(1,1) + \left(2\delta x_1 - 3\delta x_2\right)\left(\delta x_1 - \frac{\delta x_2}{2}\right)$$

This is similar to the form

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

Thus $a = 2$, $b = 3$, $c = 1$, $d = \frac{1}{2}$

→ To find the direct of the downslope, we are going in a direction such that the function value decreases

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

$$\therefore \underbrace{(2\delta x_1 - 3\delta x_2)}_{m}\underbrace{\left(\delta x_1 - \frac{\delta x_2}{2}\right)}_{n} < 0$$

For this term should be less than zero, there are two possible case

either m is +ve & n is −ve Case 1

or m is −ve n is +ve Case 2

Case 1

m is +ve & n is -ve

$\therefore 2\partial x_1 - 3\partial x_2 > 0$

$\therefore 2\partial x_1 > 3\partial x_2$

$\partial x_1 - \frac{\partial x_2}{2} < 0$

$\therefore \partial x_1 < \frac{\partial x_2}{2}$

Case 2

m is -ve & n is +ve

$\therefore 2\partial x_1 - 3\partial x_2 < 0$

$\therefore 2\partial x_1 < 3\partial x_2$

$\partial x_1 - \frac{\partial x_2}{2} > 0$

$\therefore \partial x_1 > \frac{\partial x_2}{2}$

2)

a) The plane is given by $x_1 + 2x_2 + 3x_3 = 1$

We have to find a point closest to $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

Thus we have to minimize the distance between the point.

$\therefore$ distance $= \sqrt{(x_1+1)^2 + (x_2-0)^2 + (x_3-1)^2}$

So the objective function would be

$$\min_{x_1, x_2, x_3} (x_1+1)^2 + x_2^2 + (x_3-1)^2$$ → taking the square off

using the equation of plane

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

substituting above

$$(x_1+1)^2 + x_2^2 + (x_3-1)^2$$

$$= (2 - 2x_2 - 3x_3)^2 + (x_2)^2 + (x_3-1)^2$$
$$= 4 - 4x_2 - 6x_3$$

$$\min_{x_2, x_3} \quad (2 - 2x_2 - 3x_3)^2 + (x_2)^2 + (x_3 - 1)^2$$

∴ To minimize it

At the minimum $\nabla f = 0$

$$\therefore \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = 0$$

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

$$\nabla f = \begin{bmatrix} 10x_2 + 12x_3 - 8 \\ 12x_2 + 20x_3 - 14 \end{bmatrix} = 0$$

$$\therefore \quad 10x_2 + 12x_3 = 8 \quad -(1)$$
$$12x_2 + 20x_3 = 14 \quad -(2)$$

Solve (1) & (2),

we get

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

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

$$\therefore x = \begin{bmatrix} \frac{-15}{14} \\ \frac{-1}{7} \\ \frac{11}{14} \end{bmatrix}$$ this is closest to the point $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

$\therefore$ Calculate Hessian

$$H = \begin{bmatrix} \frac{\partial f}{\partial x_2^2} & \frac{\partial f}{\partial x_2 \partial x_3} \\ \frac{\partial f}{\partial x_2 \partial x_3} & \frac{\partial f}{\partial x_3^2} \end{bmatrix} = \begin{bmatrix} 10 & 12 \\ 12 & 20 \end{bmatrix}$$

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

$$\therefore H - \lambda I = \begin{bmatrix} 10-\lambda & 12 \\ 12 & 20-\lambda \end{bmatrix} = 0$$

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

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

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

$$\lambda^2 - 28\lambda - 2\lambda + 56 = 0$$

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

$$\therefore \lambda = 2, \quad \lambda = 28$$

$\lambda_1 > 0 \qquad \lambda_2 > 0$

Thus the eigen values of the Hessian matrix is positive, which means the Hessian matrix is positive definite, Hence it is a convex problem

Can also be visualized as positive curvature, so function bending upwards and hence also a convex problem.

Q2)

b) I have taken 2 cases for each Newtons method & Gradient descent

for Gradient descent:

Case 1:

The parameters are as follows.

$x = [0, 0]$

$t = 0.8$

& alpha = 1

Case 2

$x = [0.2, 4]$, $t = 0.9$

& alpha = 1

Also the observance is that, we get a linear straight line with decreasing slope, in each case, which is because the error decreases down to zero. and ~~the~~.

Newton's Method

Case 1:

$x = [2, 1]$ initial guess

$t = 0.5$, alpha = 1

Case 2:

$x = [1, 7]$ as initial guess

$t = 0.4$, alpha = 1

As observed, since we are using a second order approximation to approximate a quadratic function, the Newton's method converges in just 1 step.

Also the values in the code, are the initial guess point & just after x changes, to see the difference, however it is just executed in a single step.

3) A hyperplane is given by

$$a^T x = c \qquad x \in R^n$$

a is the direction of the normal and c some constant.

Let x_1 be a point in the hyperplane, if it is then it will satisfy the equation of the plane.

$$\therefore a^T x_1 = c \quad - (1)$$

Let x_2 be another point in the hyperplane, if it is then it will also satisfy the equation

$$\therefore a^T x_2 = c \quad - (1)$$

By definition, if we want to prove a set is convex we must show that

for $x_1, x_2 \in S \rightarrow$ where S is a convex set.

$$\lambda x_1 + (1-\lambda) x_2 \in S \quad \forall \lambda \in [0,1]$$

Thus $\lambda x_1 + (1-\lambda) x_2$, would be any point on the line drawn connecting points x_1 & x_2.

Thus let's put this point in the equation of plane & see, whether it satisfies it or not

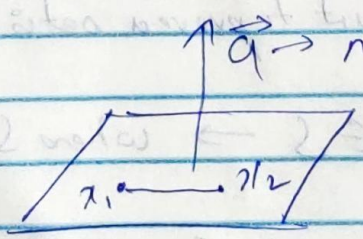
$$\therefore a^T(\lambda x_1 + (1-\lambda)x_2)$$
$$\therefore \lambda a^T x_1 + a^T x_2 - \lambda a^T x_2$$

from (1) & (2) we know that

$$a^T x_1 \text{ & } a^T x_2 = c$$
$$\therefore \lambda c - \lambda c + c$$
$$\therefore c$$

This satisfies the equation of the plane, Thus it is a convex set.

This can also be visualized mathematically



If this is a hyperplane with a as normal, & x_1, x_2 belonging to the plane, the line drawn between x_1 & x_2, will always lies on that plane, be a part of it & hence it is always a convex set.

4)

$$\min_{P} \max_{K} \{ h(a_K^T P, I_t) \}$$

subject to $0 \leq p_i \leq p_{max}$

$P = [p_1 \ \dots \ p_n]^T$ power output of n lamps,

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

To start with

$$\min_{P} \max_{K} \{ h(a_1^T P, I_t), h(a_2^T P, I_t), h(a_3^T P, I_t) \dots h(a_K^T P, I_t) \}$$

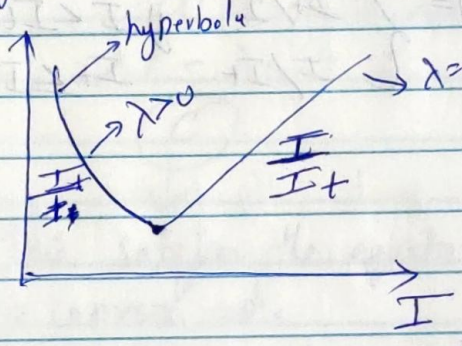
We know that if a function is convex and $\max(f_1, f_2)$ is ~~also~~ convex, then $\max(f_1, f_2, f_3, f_4)$ will also be convex.

Thus, we will start by showing with $h(a_i^T P, I_t)$ is convex

To show this, let look the function

$$h(I, I_+) = \begin{cases} I_+/I & , I \leq I_+ \\ I/I_+ & , I > I_+ \end{cases}$$

This function will look like



Thus left side of the graph represents $\frac{I_+}{I}$ & right side presents $\frac{I}{I_+}$

Looking the left side of the graph, we can say that, this graph is bending upwards, which mean $\lambda > 0$, which jumps to the conclusion, Hessian would be positive definite.

For right side of the function, which is a linear function

we know that $\lambda = 0$, since there is no curvature & hence this would be a positive semi definite Hessian matrix

So since in the overall function, we have $\lambda > 0$ as well $\lambda = 0$, we will call ~~it~~ the Hessian to positive semi definite. & this then, helps us jump to the conclusion that ~~hessian~~ function is convex

Now, coming to it mathematical

we already know $h(I_1, I_t)$ w.r.t is convex. however, we must also prove its convex w.r.t p.

$$I = a^T p$$

$$\therefore \frac{\partial h}{\partial p} = \frac{dh}{dI} \cdot \frac{dI}{dp} = h' a$$

$$\frac{d^2 h}{\partial p^2} = \frac{dh'}{dI} \cdot \frac{d\, a^T p}{dp} \cdot a^T = h'' a a^T$$

now we know that the function which is w.r.t I is definitely convex

Thus we would only have to prove aa^T is also convex

$$H(P) = h'' \sum_{k=1}^{m} a_k a_k^T$$

Let $w \in R^m$ & $w \neq 0$ s.t. $||w|| = 1$

We can write Hessian as

$$H(P) = w^T \left(\sum a_k a_k^T \right) w$$

$$= \sum w^T a_k a_k^T w$$

Let $a_k^T w = y_k$

$$H(P) = \sum y_k^T y_k$$

$$H(P) = \sum y_k^2$$

↓

Now talking about this term, if we prove this +ve, then the problem is convex.

$\therefore a_k^T w = y_k$

This would always lead to Hessian either be positive definite or positive semi definite.

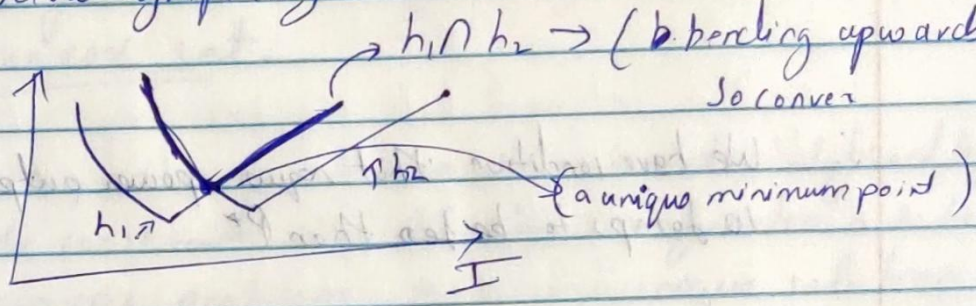
Hence the overall function $\frac{\partial^2 h}{\partial^2 p} = h'' a_k a_k^T$

This is always convex

Since $\max(h_1, h_2, h_3, \ldots, h_n)$ where $h_1, h_2, h_3 \ldots h_n$ are convex, so overall function is convex

* Also coming to the point, whether the problem has a unique solution or not.

Since we know that $h_1, h_2, h_3 \ldots h_\infty$ are convex & $\max\{h_1, h_2, h_3, h_4 \ldots\}$ are also convex we can expect the function to look something like below graphically



Now, if we look at this section, the function has a unique minimum point, so similarly if we take various such $\max(h_1, h_2, h_3, h_4)$ the function will ultimately look somewhat similar to above one and just that intersection point would be unique which will corresponding to the one at the intersection of decreasing and increasing function value. and this point of intersection, the minimum function value is unique. Thus the problem will have a unique solution.

b) Thus our originally formulated problem is a convex problem, with a unique solution.

Also, if we some constraints, & if we show the feasible domain defined by that constraint, is a convex set. the problem will still have a unique solution

∴ We have condition that requires power output of 10 lamps to be less than P^*

Say $P_1 + P_2 + P_3 + P_4 \cdots P_{10} \leq P^*$

This can be written thus $\underbrace{[1\ 1\ 1\ 1\ 1 \cdots]}_{A} \underbrace{\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}}_{x} \leq \underbrace{P^*}_{b}$

$$\therefore Ax - b \leq 0$$

If we look at this, it is linear function & linear functions will always have zero curvature, which means $\lambda = 0$

and if $d=0$, we know that Hessian would be Positive Semi definite and hence the function is convex.

Thus considering the constraint, the function is linear and hence the function defined by the constraint is convex.

Thus feasible domain of set by this constraint is a convex set.

Hence, as long we have a convex set, defined by the constraint and the objective function is having a convex problem, having a unique solution. It will always maintain the uniqueness of the problem.

Thus this problem will have a unique solution.

■

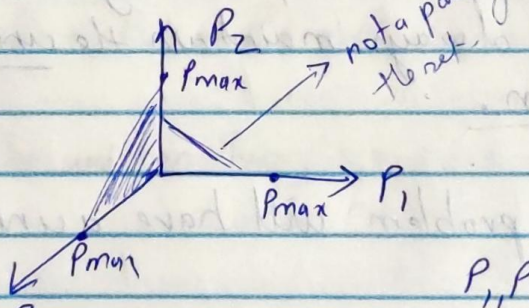
c)

For this constraint we have the condition that no more than 10 lamps to be switched on.

In this case, we can visulation that in the set of P_i, where

$$0 < P_i < P_{max},$$

In this set each bulb can have max P_{max} to represent it graphically, I will consider the case of 3 lights.



P_1, P_2, P_3 corresponds to the power of the lamp

Now, if we say at time of all the 3 lights, only 2 ~~two~~ lights can be one, so constraint put the condition that there will some linear combination of P_2 & P_3, but there won't be any combinatio of P_1 with P_2 & P_3

the dark region in the graph represents that so among P_2 & P_3, there are linear combinations, but P_1 lies only on 1 axis along P_1 and thus if try to draw a line connecting P_1 and P_2 some points on it, the points on the line doesn't belong to the ~~convex~~ set.

Thus this ~~feed~~ set won't be convex, and if we extend the same idea to a n dimensional space

where $P = [P_1, P_2, P_3 \cdots P_n]^T$
out of which only 10 dampers are allowed to be switched on, it will also follow a similar patter and thus it won't be a convex set.

and if the feasible domain of X is not a convex set, then the objective problem will no longer have a unique solution.

Mathematically, it can also be said that $\lambda x_1 + (1-\lambda) x_2$ does not belong to that set, which why it is not a convex set

5)

$C(x)$ cost of producing x amount of A

y price set for the product

$$C^*(y) = \max_x \left(xy - C(x) \right)$$

Now this function is $x(y) - C(x)$

for eg for x_1

$$x_1(y) - C(x_1)$$

This as we can is a linear function w.r.t xy

So, if a function is linear, we know that $\lambda = 0$ (curvature is zero) & Hessian is a positive semi definite matrix. Hence the problem defined would be convex.

Similarly for $x_2, x_3, x_4 \cdots x_n$

this can be written as

$$\left.\begin{aligned} x_2(y) - C(x_2) \\ x_3(y) - C(x_3) \\ \vdots \\ x_n(y) - C(x_n) \end{aligned}\right\} \text{linear \& thus convex}$$

Also we know that if the functions are convex, the max of convex functions will also be convex

$$\therefore C^*(y) = \max_x \left(x_1 y - C(x_1), x_2(y) - C(x_2), \ldots, x_n(y) - C(x_n) \right)$$

They would be convex

and thus the overall function is a convex function w.r.t y

In [2]:

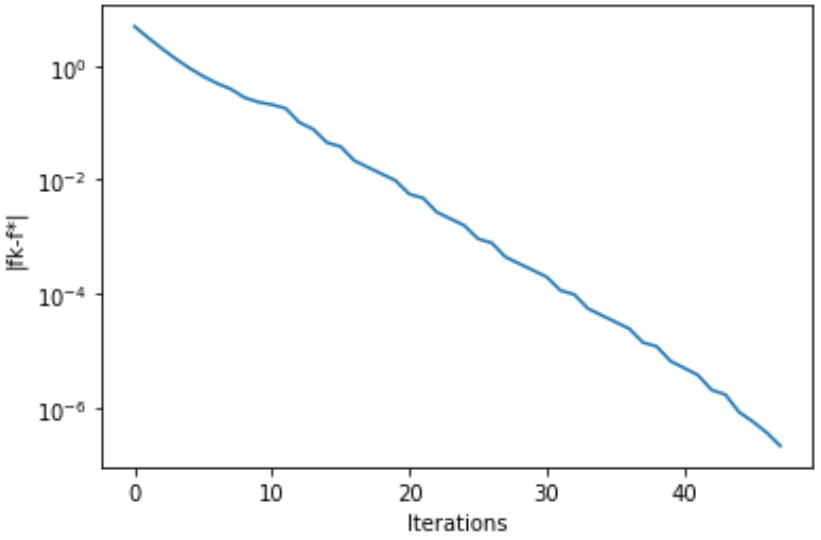
```
# gradient descent using inexact line search (Case1 using initial guess as [0,0] and t=0.8)


import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt

x = [0,0]                 # Taking a initial guess

x_values=[]               # Taking a array for storing the updated values of x

def fun(x):               # Defining function for calculating the equation defined
    f = (2- 2*x[0] - 3*x[1])**2 + x[0]**2  + (x[1] -1)**2
    return f

f_values=[]               # Taking a array for storing the function values for updated version of x

def gd(x):
    dx1 = 10*x[0] -8 + 12*x[1]
    dx2= 20*x[1] - 14 + 12*x[0]
    return np.array([dx1,dx2])

gd_values=[]            # Taking a array for storing the gradient values for updated version of x


def phi(x,alpha):       #defining the function phi
    p= fun(x) - alpha*(norm(gd(x))**2)*0.8
    return p


def l_s(x):             # Defining a function for line search
    alpha=1
    while fun(x-alpha*gd(x)) > phi(x,alpha):

        alpha=0.5*alpha
    return alpha

c=0   # count for iteration
it=[] # Array fro storing iteration values

while norm(gd(x)) > 1e-3:    # Checking the condition for gradient
```

```
    a=l_s(x)
    it.append(c)
    gd_values.append(norm(gd(x)) )
    x_values.append(x)
    f_values.append(fun(x))

    x=x-a*gd(x)
    c=c+1

print(f"The values for the gradient are  = \n{gd_values}\n")
print(f"The values for the function values are  = \n{f_values}\n")
print(f"The values for the updated x are  = \n{x_values}\n")




x=len(f_values)
f_s=f_values[x-1]

error=[]

for i in range(x-1):
    f_abs= abs(f_values[i]-f_s)
    error.append(f_abs)



it=it[0:len(error)]
plt.yscale('log')
plt.plot(it,error)   # Plotting the graph

slope,intercept=np.polyfit(it,error,1)
print(f"The slope of the line is = {slope}\n")
plt.xlabel('Iterations'),plt.ylabel('|fk-f*|')
```

```
The values for the gradient are  =
[16.1245154965971, 12.614785471818378, 9.87697731212742, 7.743197667103332, 6.082476752716368, 4.792707096384696,
3.7943233818049036, 3.025320082778326, 1.873516249860259, 1.2988293325275042, 1.0366564138131307, 0.8492566328170
452, 0.6547373430829303, 0.568399719027217, 0.5123857031858294, 0.3925407870646592, 0.3075158835568518, 0.2657830
735087359, 0.23042008834275993, 0.20023063868916788, 0.1770028525979969, 0.13859551192125175, 0.1079913778420324,
0.09347447284508632, 0.0811295146307457, 0.07346828986671582, 0.05599813166236172, 0.04392528543983886, 0.0379508
5785089465, 0.032892536504332937, 0.028577125729741363, 0.02536764050977013, 0.019771066082178327, 0.015422772205
```

```
249092, 0.013345368469533736, 0.011580111167310277, 0.010069728015533531, 0.008777319139599143, 0.006981042216623
581, 0.005419127282041094, 0.004695530621676484, 0.0040786302218086224, 0.003636000192839218, 0.00282041260355417
76, 0.002202679755839597, 0.0019053732456841656, 0.0016529345454449167, 0.0014370769534121255, 0.0012575074718758
723]

The values for the function values are  =
[5, 3.18994140625, 2.0817477703094482, 1.4020445959758945, 0.9839740429966497, 0.7256909804992941, 0.565030985190
8131, 0.46404891141297655, 0.34918996180385603, 0.3034019413822135, 0.28004954078795996, 0.2507996041383271, 0.17
273522862176324, 0.14889986224970841, 0.11644916534576877, 0.10962854219199918, 0.09305697225158432, 0.0879563610
612387, 0.08406496707064065, 0.08109335936334468, 0.07702176618118542, 0.07619576122686951, 0.0741254234576243,
0.07348993185092437, 0.07300487848871565, 0.07234551585811352, 0.07220576203826456, 0.0718686973680288, 0.0717648
829800968, 0.0716856887828428, 0.07162521811566744, 0.07154247358953367, 0.0715255611743389, 0.07148344929332645,
0.07147051561235464, 0.0714606446234818, 0.07145310478797545, 0.07144272866031172, 0.07144067543022464, 0.0714354
1481398137, 0.0714338031088117, 0.07143257255372853, 0.07143089104689426, 0.07143054471185331, 0.0714296881372751
3, 0.07142942490490983, 0.0714292240255264, 0.07142907059731514, 0.07142885970483925]

The values for the updated x are  =
[[0, 0], array([0.0625  , 0.109375]), array([0.10986328, 0.19580078]), array([0.14542389, 0.26428223]), array([0.
17178619, 0.31872964]), array([0.19098449, 0.36219818]), array([0.20460775, 0.39707492]), array([0.21389699, 0.42
522498]), array([0.2257459 , 0.47098649]), array([0.22716314, 0.50022586]), array([0.22287655, 0.52006219]), arra
y([0.20820431, 0.54894461]), array([0.124532  , 0.61427662]), array([0.08599204, 0.62803184]), array([0.03645422,
0.67896418]), array([0.02045071, 0.67844123]), array([-0.02277453,  0.70166208]), array([-0.03478701,  0.7166653
7]), array([-0.05054416,  0.72192391]), array([-0.06039699,  0.73242714]), array([-0.08354146,  0.74195478]), arr
ay([-0.0856678 ,  0.74706109]), array([-0.09917469,  0.75791006]), array([-0.10562305,  0.7599035 ]), array([-0.1
0953627,  0.76424141]), array([-0.11897805,  0.76794228]), array([-0.11977577,  0.77009513]), array([-0.12519875,
0.77452096]), array([-0.12784025,  0.77526882]), array([-0.12939171,  0.77706298]), array([-0.13131913,  0.777778
04]), array([-0.13383727,  0.78031164]), array([-0.13462999,  0.78030584]), array([-0.13680127,  0.78148622]), ar
ray([-0.13741514,  0.78222939]), array([-0.13820272,  0.78250401]), array([-0.13870403,  0.78302604]), array([-0.
13986305,  0.78351698]), array([-0.13997472,  0.78376751]), array([-0.14065759,  0.78431081]), array([-0.1409797
,  0.78441549]), array([-0.14117901,  0.7846309 ]), array([-0.1416516 ,  0.78482215]), array([-0.14169378,  0.784
92766]), array([-0.14196804,  0.78514919]), array([-0.14209991,  0.78518873]), array([-0.14217902,  0.78527775]),
array([-0.14227544,  0.78531483]), array([-0.14240338,  0.78544092])]

The slope of the line is = -0.03445134535007397
```

Out[2]: (Text(0.5, 0, 'Iterations'), Text(0, 0.5, '|fk-f*|'))



In [4]:

```
# gradient descent using inexact line search (Case 2: Taking the initial guess as [2,4] and the t parameter as 0.


import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt

x = [2,4]                  # Taking a initial guess

x_values=[]                # Taking a array for storing the updated values of x

def fun(x):                # Defining function for calculating the equation defined
    f = (2- 2*x[0] - 3*x[1])**2 + x[0]**2  + (x[1] -1)**2
    return f

f_values=[]                # Taking a array for storing the function values for updated version of x

def gd(x):
    dx1 = 10*x[0] -8 + 12*x[1]
    dx2= 20*x[1] - 14 + 12*x[0]
    return np.array([dx1,dx2])

gd_values=[]            # Taking a array for storing the gradient values for updated version of x
```

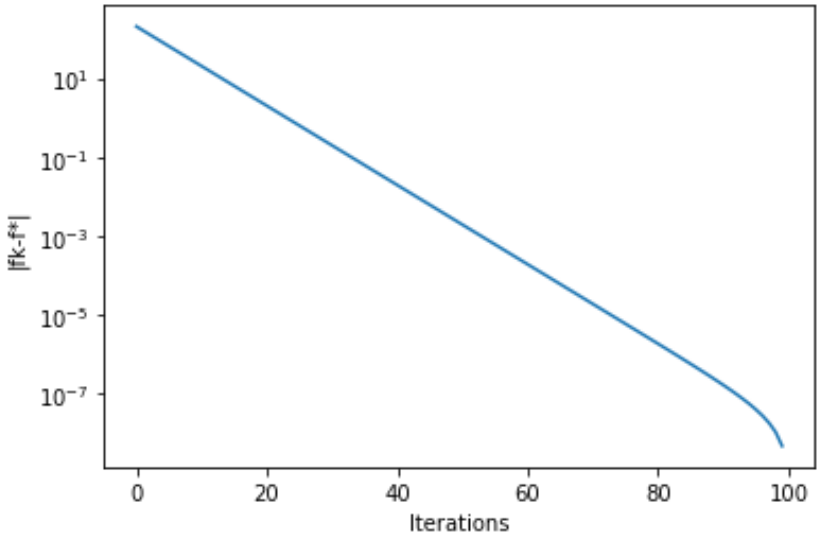
```python
def phi(x,alpha):       #defining the function phi
    p= fun(x) - alpha*(norm(gd(x))**2)*0.9
    return p


def l_s(x):            # Defining a function for line search
    alpha=1
    while fun(x-alpha*gd(x)) > phi(x,alpha):

        alpha=0.5*alpha
    return alpha

c=0   # count for iteration
it=[] # Array fro storing iteration values

while norm(gd(x)) > 1e-3:    # Checking the condition for gradient

    a=l_s(x)
    it.append(c)
    gd_values.append(norm(gd(x)) )
    x_values.append(x)
    f_values.append(fun(x))

    x=x-a*gd(x)
    c=c+1

print(f"The values for the gradient are  = \n{gd_values}\n")
print(f"The values for the function values are  =\n{f_values}\n")
print(f"The values for the updated x are  = \n{x_values}\n")




x=len(f_values)
f_s=f_values[x-1]

error=[]

for i in range(x-1):
    f_abs= abs(f_values[i]-f_s)
    error.append(f_abs)
```

```
it=it[0:(len(error))]
plt.yscale('log')
plt.plot(it,error)    # Plotting the graph

slope,intercept=np.polyfit(it,error,1)
print(f"The slope of the line is = {slope}\n")
plt.xlabel('Iterations'),plt.ylabel('|fk-f*|')
```

```
The values for the gradient are  =
[108.16653826391968, 96.33582314130346, 85.7990924852234, 76.41481674465209, 68.05694616320577, 60.6132176766051
3, 53.98364699322645, 48.07918560334231, 42.82052467797674, 38.13702979132303, 33.96579215789708, 30.250783640627
084, 26.942104179933498, 23.995311535253276, 21.370824336084944, 19.033390424325656, 16.951613346665038, 15.09753
0636873554, 13.446238223465508, 11.975555917773969, 10.66572948926744, 9.499165326378812, 8.460194118806129, 7.53
4860387061707, 6.710735032226832, 5.976748388077024, 5.3230415331311, 4.740833865444884, 4.22230516141185, 3.7604
905343824293, 3.349186882184352, 2.9828695669454395, 2.6566182080607823, 2.366050591554133, 2.1072638081029007,
1.8767818290916463, 1.6715088165347494, 1.4886875397262587, 1.3258623400686997, 1.1808461466236866, 1.05169109933
67197, 0.9366623853467666, 0.8342149369494647, 0.7429726782206176, 0.661710041540238, 0.5893355057467733, 0.52487
69348057191, 0.46746852006134404, 0.416339150679634, 0.3708020560740484, 0.33024558119094977, 0.2941249707481903
3, 0.2619550520726077, 0.23330371825216506, 0.20778612406833602, 0.18505951674836352, 0.16481863210400957, 0.1467
9159421763296, 0.1307362636000782, 0.11643698476881921, 0.1037016895597286, 0.09235931726413286, 0.08225751693836
816, 0.07326060102323385, 0.06524772278631687, 0.05811125310656403, 0.05175533479803367, 0.04609459505449942, 0.0
4105299872041393, 0.03656282698536821, 0.0325637677838453, 0.029002105682486335, 0.025830000373465327, 0.02300484
4082617083, 0.020488689261081775, 0.01824773887315077, 0.01625189243389887, 0.014474341698939536, 0.0128912105756
18725, 0.011481234418909568, 0.010225474404340653, 0.009107063141365083, 0.00811097811027821, 0.00722383987946678
8, 0.006433732392650392, 0.0057300429122042204, 0.005103319468683212, 0.004545143901795875, 0.004048018787536324,
0.003605266732649825, 0.003210940683766598 7, 0.0028597440464805455, 0.0025469595413972284, 0.002268385841557651,
0.0020202811401382656, 0.0017993128904352964, 0.0016025130430441215, 0.0014272381789622178, 0.001271134003137796,
0.0011321037215452116, 0.0010082798770021202]

The values for the function values are  =
[209, 165.7962646484375, 131.5265048444271, 104.34328960926359, 82.78121287609801, 65.67789566270567, 52.11132397
659442, 41.35014443358282, 32.81423810173598, 26.043447166147512, 20.67276851631183, 16.412676979857697, 13.03351
7457899821, 10.353124565604618, 8.22700237149644, 6.540534840085924, 5.202807054550576, 4.141704130916706, 3.3000
23613610444, 2.632391777495198, 2.102817598896947, 1.68275253389067, 1.3495515094264616, 1.085252161651995, 0.875
6065120135087, 0.7093128802568089, 0.5774066279185479, 0.4727768882098052, 0.3897832299300919, 0.323951590342497
5, 0.27173308521063827, 0.2303126938108796, 0.1974575054178587, 0.17139634646060112, 0.15072429923107755, 0.13432
69648930104, 0.1213203879241678, 0.1110034034095753, 0.10281983830022251, 0.09632852896421429, 0.091179538721858
6, 0.0870952932879196, 0.08385561716124287, 0.08128586429611281, 0.07924750319777107, 0.07763064889881788, 0.0763
4813922662587, 0.07533083602229182, 0.07452389800693995, 0.07388382437122751, 0.07337610971243111, 0.072973383900
```

```
3147, 0.07265393659475647, 0.07240054687411226, 0.07219955488134538, 0.0720401254417703, 0.07191366395515421, 0.0
718133530738027, 0.07173378518964477, 0.071670670918251, 0.07162060786459898, 0.07158089720509817, 0.071549398198
0869, 0.07152441277968366, 0.07150459402372857, 0.07148887353102884, 0.07147640383357243, 0.07146651270880784, 0.
07145866694114177, 0.07145244357709216, 0.07144750712450504, 0.07144359146667893, 0.071440485516416, 0.0714380218
3662973, 0.07143606761406496, 0.07143451749953542, 0.07143328792870868, 0.07143231261727892, 0.07143153898768044,
0.0714309253347104, 0.0714304385772642, 0.0714300524749833, 0.07142974621367688, 0.0714295032832608, 0.0714293105
8772323, 0.07142915773913881, 0.07142903649767143, 0.07142894032737657, 0.07142886404385901, 0.07142880353478957,
0.07142875553821562, 0.07142871746671449, 0.07142868726790896, 0.07142866331382729, 0.07142864431314078, 0.071428
62924155137, 0.07142861728657238, 0.07142860780372895, 0.07142860028181525, 0.07142859431533641, 0.07142858958264
839]

The values for the updated x are  =
[[2, 4], array([1.765625 , 3.6484375]), array([1.55688477, 3.33532715]), array([1.37097549, 3.05646324]), array
([1.20540005, 2.80810007]), array([1.05793442, 2.58690163]), array([0.92659784, 2.38989676]), array([0.8096262,
2.2144393]), array([0.70544834, 2.05817251]), array([0.61266493, 1.91899739]), array([0.5300297 , 1.79504455]), a
rray([0.4564327 , 1.68464905]), array([0.39088537, 1.58632806]), array([0.33250729, 1.49876093]), array([0.280514
3 , 1.42077145]), array([0.23420805, 1.35131208]), array([0.19296654, 1.28944982]), array([0.15623583, 1.2343537
4]), array([0.12352254, 1.1852838 ]), array([0.09438726, 1.14158089]), array([0.06843865, 1.10265798]), array([0.
04532817, 1.06799226]), array([0.0247454 , 1.03711811]), array([0.00641388, 1.00962081]), array([-0.00991264,  0.
98513104]), array([-0.02445345,  0.96331983]), array([-0.03740385,  0.94389422]), array([-0.0489378 ,  0.9265932
9]), array([-0.05921023,  0.91118465]), array([-0.06835911,  0.89746133]), array([-0.07650733,  0.885239  ]), arr
ay([-0.08376435,  0.87435348]), array([-0.09022762,  0.86465857]), array([-0.09598397,  0.85602404]), array([-0.1
0111073,  0.84833391]), array([-0.10567674,  0.84148489]), array([-0.10974335,  0.83538498]), array([-0.11336517,
0.82995225]), array([-0.11659085,  0.82511372]), array([-0.11946373,  0.82080441]), array([-0.12202238,  0.816966
42]), array([-0.12430119,  0.81354822]), array([-0.12633074,  0.81050389]), array([-0.12813832,  0.80779252]), ar
ray([-0.12974819,  0.80537772]), array([-0.13118198,  0.80322703]), array([-0.13245895,  0.80131157]), array([-0.
13359625,  0.79960562]), array([-0.13460916,  0.79808625]), array([-0.13551129,  0.79673307]), array([-0.1363147
4,  0.79552789]), array([-0.13703032,  0.79445453]), array([-0.13766762,  0.79349856]), array([-0.13823523,  0.79
264716]), array([-0.13874075,  0.79188888]), array([-0.13919098,  0.79121353]), array([-0.13959197,  0.7906120
5]), array([-0.1399491 ,  0.79007636]), array([-0.14026716,  0.78959926]), array([-0.14055044,  0.78917434]), arr
ay([-0.14080274,  0.78879589]), array([-0.14102744,  0.78845884]), array([-0.14122756,  0.78815866]), array([-0.1
414058,  0.7878913]), array([-0.14156454,  0.78765319]), array([-0.14170592,  0.78744112]), array([-0.14183183,
0.78725225]), array([-0.14194398,  0.78708404]), array([-0.14204385,  0.78693422]), array([-0.14213281,  0.786800
79]), array([-0.14221203,  0.78668195]), array([-0.14228259,  0.78657611]), array([-0.14234543,  0.78648185]), ar
ray([-0.1424014,  0.7863979]), array([-0.14245125,  0.78632313]), array([-0.14249564,  0.78625654]), array([-0.14
253518,  0.78619723]), array([-0.1425704 ,  0.78614441]), array([-0.14260176,  0.78609736]), array([-0.14262969,
0.78605546]), array([-0.14265457,  0.78601815]), array([-0.14267673,  0.78598491]), array([-0.14269646,  0.785955
31]), array([-0.14271403,  0.78592895]), array([-0.14272969,  0.78590547]), array([-0.14274363,  0.78588456]), ar
ray([-0.14275604,  0.78586594]), array([-0.1427671 ,  0.78584935]), array([-0.14277695,  0.78583458]), array([-0.
14278572,  0.78582142]), array([-0.14279353,  0.7858097 ]), array([-0.14280049,  0.78579927]), array([-0.1428066
9,  0.78578997]), array([-0.1428122 ,  0.78578169]), array([-0.14281712,  0.78577432]), array([-0.1428215 ,  0.78
576775]), array([-0.1428254 ,  0.78576191]), array([-0.14282887,  0.7857567 ]), array([-0.14283196,  0.7857520
6]), array([-0.14283472,  0.78574793]), array([-0.14283717,  0.78574425])]

The slope of the line is = -0.5536993766115718
```

Out[4]: (Text(0.5, 0, 'Iterations'), Text(0, 0.5, '|fk-f*|'))



In [5]:

```
# Newton's method using inexact line search ( Case1 Taking initial guess as [2,1] and t=0.5)


import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm
from numpy.linalg import inv

x = np.array([2,1])
h=np.array([[10,12],[12,20]])
hinv=inv(h)

x_values=[]

def fun(x):
    f = (2- 2*x[0] - 3*x[1])**2 + x[0]**2  + (x[1] -1)**2
    return f

f_values=[]

def gd(x):
    dx1 = 10*x[0] -8 + 12*x[1]
    dx2= 20*x[1] - 14 + 12*x[0]
```

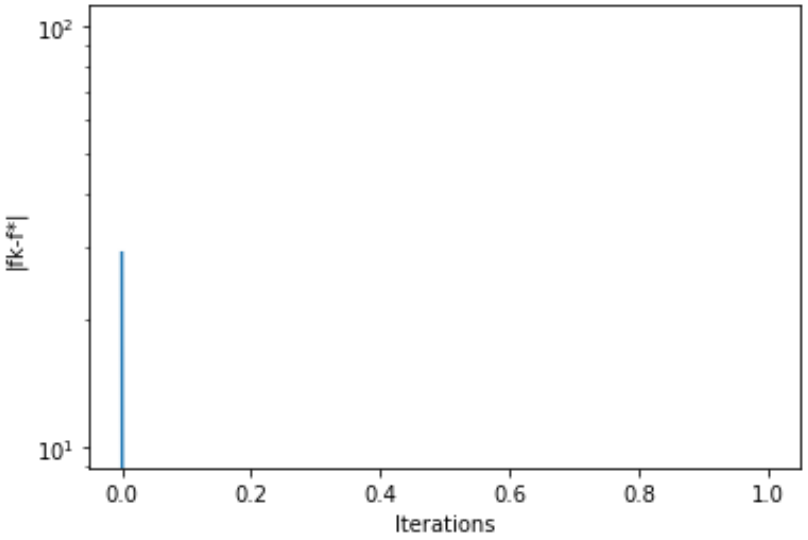
```
    return np.array([dx1,dx2])

gd_values=[]


def phi(x,alpha,hinv):


    grad = gd(x)
    gradt=np.transpose(grad)

    p= fun(x) - alpha*(grad@hinv@gradt)*0.5

    return p

def l_s(x,hinv):
    alpha=1
    hg=np.matmul(hinv,gd(x))

    while fun(x-alpha*(hg) )  > phi(x,alpha,hinv):

        alpha=0.5*alpha
    return alpha


def newton_method(x,hinv):
    d= gd(x)
    step_size=np.matmul(hinv,d)
    return step_size




c=0
it=[]

while norm(gd(x)) > 1e-3:


    gd_values.append(norm(gd(x)))
    x_values.append(x)
    f_values.append(fun(x))
```

```
    it.append(c)


    alpha=  l_s(x,hinv)
    step_size= newton_method(x,hinv)
    x= x- alpha*step_size


    gd_values.append(norm(gd(x)))
    x_values.append(x)
    f_values.append(fun(x))

    c=c+1
    it.append(c)

print(f"The values for the gradient are  = \n{gd_values}\n")
print(f"The values for the function values are  =\n{f_values}\n")
print(f"The values for the updated x are  = \n{x_values}\n")




x=len(f_values)
f_s = f_values[x-1]

error=[]

for i in range(x):
    f_abs= abs(f_values[i]-f_s)
    error.append(f_abs)



plt.yscale('log')
plt.plot(it,error)

slope,intercept=np.polyfit(it,error,1)
plt.xlabel('Iterations'),plt.ylabel('|fk-f*|')

print(" The Observation is that the algorithm is completed just in one step, reason being the function we are app
```

```
The values for the gradient are  =
```

```
[38.41874542459709, 3.972054645195637e-15]

The values for the function values are  =
[29, 0.07142857142857142]

The values for the updated x are  =
[array([2, 1]), array([-0.14285714,  0.78571429])]
```

The Observation is that the algorithm is completed just in one step, reason being the function we are approximating is quadratic function and we are using a second order taylor series expansion for that



In [6]:

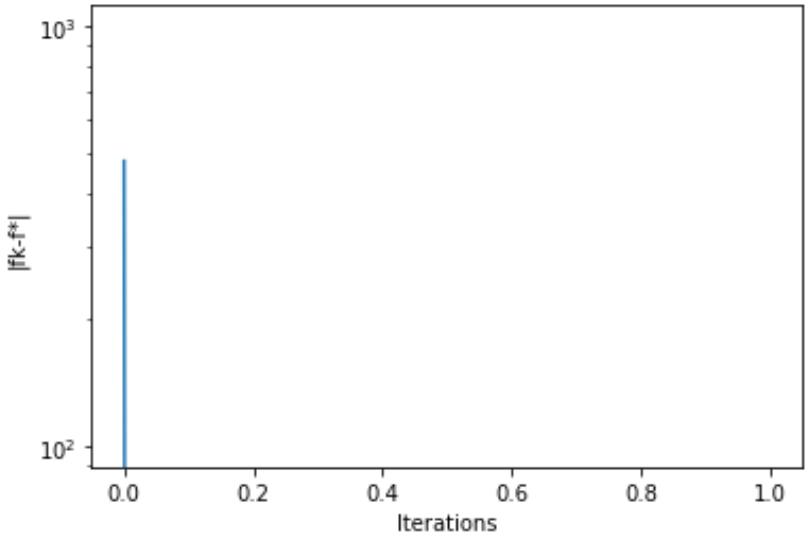
```python
# Newton's method using inexact Case 2:line search, taking initial guess as [ 1,7] and t=0.4


import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm
from numpy.linalg import inv

x = np.array([1,7])
h=np.array([[10,12],[12,20]])
hinv=inv(h)

x_values=[]

def fun(x):
```

```
    f = (2- 2*x[0] - 3*x[1])**2 + x[0]**2  + (x[1] -1)**2
    return f

f_values=[]

def gd(x):
    dx1 = 10*x[0] -8 + 12*x[1]
    dx2= 20*x[1] - 14 + 12*x[0]
    return np.array([dx1,dx2])

gd_values=[]


def phi(x,alpha,hinv):


    grad = gd(x)
    gradt=np.transpose(grad)

    p= fun(x) - alpha*(grad@hinv@gradt)*0.4

    return p

def l_s(x,hinv):
    alpha=1
    hg=np.matmul(hinv,gd(x))

    while fun(x-alpha*(hg) )  > phi(x,alpha,hinv):

        alpha=0.5*alpha
    return alpha


def newton_method(x,hinv):
    d= gd(x)
    step_size=np.matmul(hinv,d)
    return step_size




c=0
```

```
it=[]

while norm(gd(x)) > 1e-3:


    gd_values.append(norm(gd(x)))
    x_values.append(x)
    f_values.append(fun(x))
    it.append(c)


    alpha=  l_s(x,hinv)
    step_size= newton_method(x,hinv)
    x= x- alpha*step_size


    gd_values.append(norm(gd(x)))
    x_values.append(x)
    f_values.append(fun(x))

    c=c+1
    it.append(c)

print(f"The values for the gradient are  = \n{gd_values}\n")
print(f"The values for the function values are  =\n{f_values}\n")
print(f"The values for the updated x are  = \n{x_values}\n")




x=len(f_values)
f_s = f_values[x-1]

error=[]

for i in range(x):
    f_abs= abs(f_values[i]-f_s)
    error.append(f_abs)



plt.yscale('log')
```

```
plt.plot(it,error)

slope,intercept=np.polyfit(it,error,1)
plt.xlabel('Iterations'),plt.ylabel('|fk-f*|')
print(" The Observation is that the algorithm is completed just in one step, reason being the function we are app
```

```
The values for the gradient are  =
[162.60381299342276, 9.559374836295625e-14]

The values for the function values are  =
[478, 0.07142857142857142]

The values for the updated x are  =
[array([1, 7]), array([-0.14285714,  0.78571429])]

 The Observation is that the algorithm is completed just in one step, reason being the function we are approximat
ing is quadratic function and we are using a second order taylor series expansion for that
```



In [ ]: