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

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

✓

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

so we get $x_1 = x_2$

$$\Downarrow$$

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

$$\underline{x_2 = 1 = x_1}$$

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

* One λ is (+) other is (−) meaning the Hessian is indefinite aka saddle point

For a downslope, $\partial x' = x - x'$

and using taylor's expansion,

$$\underline{\partial x'} = \nabla f' \partial x' + \frac{1}{2} \partial x'^T H \partial x'$$

$$= (a \partial x_1 - b \partial x_2) \cdot (c \partial x_1 - d \partial x_2) < 0$$

So either:

1) $a \partial x_1 - b \partial x_2 > 0$ <u>AND</u> $c \partial x_1 - d \partial x_2 < 0$

<u>OR</u>

2) $a \partial x_1 - b \partial x_2 < 0$ <u>AND</u> $c \partial x_1 - d \partial x_2 > 0$

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) optimize near point $X_1 = -1,\ X_2 = 0,\ X_3 = 1$

so... $(X_1+1)^2 + X_2^2 + (X_3-1)^2 = 0$ ← we can minimize!

s.t. $X_1 + 2X_2 + 3X_3 - 1 = 0$

* similar to HW1, this is a simple unconstrained optimization prob so coding this in python we get...

$X_1 = -1.07121$
$X_2 = -0.14285$
$X_3 = 0.7857$

2b)

↓

```
## This is code for problem 2a

import numpy as np
from scipy.optimize import minimize

# this line describes the bounds for each x value
b = (-10,10)
bounds = [b,b,b]

# these lines define an equality constraints which equals 0
def cont1(x):
    return x[0]+2*x[1]+3*x[2]-1

# define the function to minimize
def func(x):
    x1 = x[0]
    x2 = x[1]
    x3 = x[2]
    ff = pow((x1+1),2) + pow((x2),2) + pow((x3-1),2)
    return ff
    #return (np.x1-np.x2)^2 + (np.x2+np.x3-2)^2 + (np.x4-1)^2 + (np.x5-1)^2

# this code is for making a directory of constraints
direct = ({'type':'eq', 'fun':cont1})

# initial guesses
x0 = [1,1,1]

# load guesses into an array
guess = np.array([x0[0],x0[1],x0[2]])

# find solution
sol = minimize(func,guess,method='SLSQP',bounds = bounds,constraints=direct)
print(sol)
```

```
     fun: 0.07142857142857142
     jac: array([-0.14285714, -0.28571427, -0.42857141])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([-1.07142858, -0.14285714,  0.78571429])
```

Start of descent code

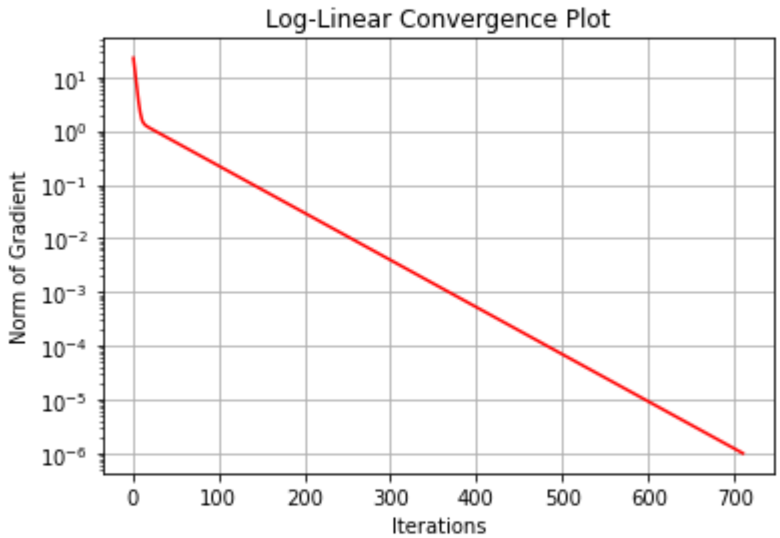
```
import numpy as np
from scipy.optimize import minimize
import pdb
```

```
import matplotlib.pyplot as plt

# inital guess and counter
x0 = [1,1]
x2 = x0[0]
x3 = x0[1]
i = 0;

# define the function
def f(x2,x3):
  return (pow((2-2*x2-3*x3),2) + pow((x2),2) + pow((x3-1),2))

# define the gradient
def g(x2,x3):
  return [2*(2-2*x2-3*x3)*-2+2*x2,2*(2-2*x2-3*x3)*-3+2*(x3-1)]

# define the hessian
h = ([10,12],[12,20])

# find gradient at x2 and x3 and the norm
g0 = g(x2,x3)
g_norm = pow(((pow(g0[0],2))+(pow(g0[1],2))),0.5)

# create structures to store iteration, and g_norms
iteration = []
g_norms = []

# this is the while condition below
while g_norm > 1e-6:

# counter and alpha
   i = i + 1;
   alpha = 0.01;

# implement gradient descent method (g take x2 and x3 as inputs)
   g_n = g(x2,x3)
   alpha_g_n = [alpha*g_n[0],alpha*g_n[1]]
   x0 = np.subtract(x0, alpha_g_n)
   x2 = x0[0]
   x3 = x0[1]

# store current iteration, and g_norms
   iteration.append(i)
   g_norms.append(g_norm);

# Update g_norm to test criteria again
   g0 = g(x2,x3)
   g_norm = pow(((pow(g0[0],2))+(pow(g0[1],2))),0.5)


# print solutions
x1 = 1-2*x2-3*x3
```

```
print('i = ',i)
print('x1 = ',x1)
print('x2 = ',x2)
print('x3 = ',x3)


#perform log transformation on both x and y
print()
plt.yscale("log")
plt.plot(iteration,g_norms,'-r')
plt.xlabel('Iterations')
plt.ylabel('Norm of Gradient')
plt.title("Log-Linear Convergence Plot")
plt.grid()
```

```
i =  710
x1 =  -1.0714285714285716
x2 =  -0.14285673471362936
x3 =  0.7857140136186102
```

Plot for descent code



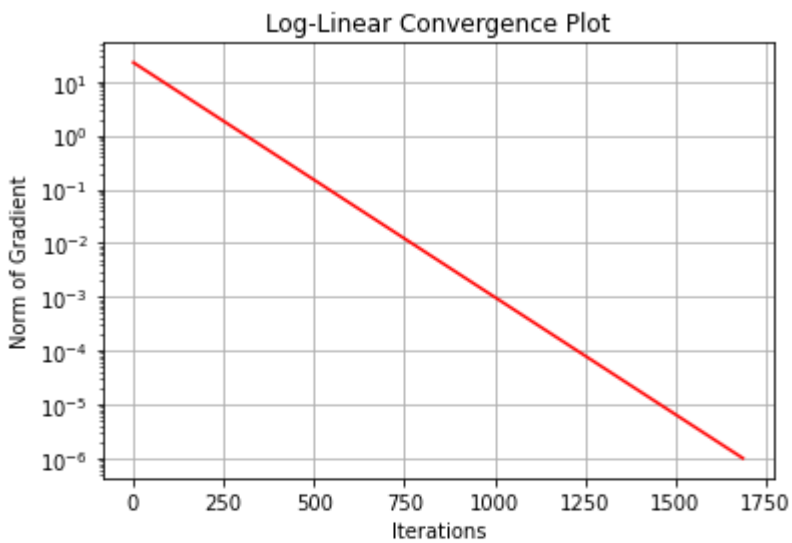
Start of newton's method code

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

# inital guess and counter
x0 = [1,1]
x2 = x0[0]
x3 = x0[1]
i = 0;
```

```
# define the function
def f(x2,x3):
  return (pow((2-2*x2-3*x3),2) + pow((x2),2) + pow((x3-1),2))

# define the gradient
def g(x2,x3):
  return [2*(2-2*x2-3*x3)*-2+2*x2,2*(2-2*x2-3*x3)*-3+2*(x3-1)]

# define the hessian
h = ([10,12],[12,20])

# find gradient at x2 and x3 and the norm
g0 = g(x2,x3)
g_norm = pow(((pow(g0[0],2))+(pow(g0[1],2))),0.5)

# create structures to store iteration, and g_norm
iteration = []
g_norms = []

# this is the while condition below
while g_norm > 1e-6:

# counter
   i = i + 1;

# alpha is the step that is really small for Newton's method
   alpha = 0.01;

# implement newtons method (g take x2 and x3 as inputs)
   g_n = g(x2,x3)
   new = np.matmul(np.linalg.inv(h),g_n)
   alpha_g_n = [alpha*new[0],alpha*new[1]]
   x0 = np.subtract(x0, alpha_g_n)
   x2 = x0[0]
   x3 = x0[1]

# store current iteration, and g_norms
   iteration.append(i)
   g_norms.append(g_norm);

# Update g_norm to test criteria again
   g0 = g(x2,x3)
   g_norm = pow(((pow(g0[0],2))+(pow(g0[1],2))),0.5)


# print solutions
x1 = 1-2*x2-3*x3
print('i = ',i)
print('x1 = ',x1)
print('x2 = ',x2)
```

```
print('x3 = ',x3)


#perform log transformation on both x and y
print()
plt.yscale("log")
plt.plot(iteration,g_norms,'-r')
plt.xlabel('Iterations')
plt.ylabel('Norm of Gradient')
plt.title("Log-Linear Convergence Plot")
plt.grid()
```

```
i =  1686
x1 =  -1.0714286995417814
x2 =  -0.14285709286174392
x3 =  0.785714295088423
```



Plot of newton's method

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.

For any 2 points, x_1 and x_2

$a^T x_1 = c$ and $a^T x_2 = c$

now lets say,

* proof of convexity

$$a^T\left[\lambda x_1 + (1-\lambda)x_2\right] = c$$

* $a^T x_1 = c$, $a^T x_2 = c$ so

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

$$\lambda c + (1-\lambda)c = c \equiv \underline{c = c} \checkmark$$

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

This means $\lambda x_1 + (1-\lambda)x_2$ exists in H and therefore H (hyperplane) is convex

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) solve min max$\{ h(a_0^T P, It) \}$

max$\{h_1, h_2, ... h_3\}$ Finds largest value h

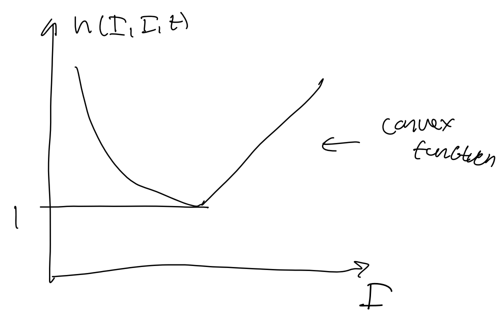
$\downarrow$

we have

max$\{h(a_1^T P, It), h(a_2^T P, It) ... \}$

$h(I, It) = \quad It/I \; , \; I \le It$

$\qquad\qquad\quad I/It \quad I > It$



Show $h(a^T P, It)$ is convex w/ respect to P

by $\dfrac{\partial h}{\partial P} = \dfrac{\partial h}{\partial I} \cdot \dfrac{\partial a^T P}{\partial P} = h' \cdot a$

1D function

$\dfrac{\partial^2 h}{\partial P^2} = \dfrac{\partial h'}{\partial I} \cdot \dfrac{\partial a^T P}{\partial P} a^T = h'' \cdot a \cdot a^T$

h is convex

$\Downarrow$

$h'' \ge 0$

b) For any 10 lights,

$$C_1^{10}\begin{cases} P_1 + \dots + P_{10} \le P^* \\ P_2 + \dots + P_{11} \le P^* \end{cases}$$

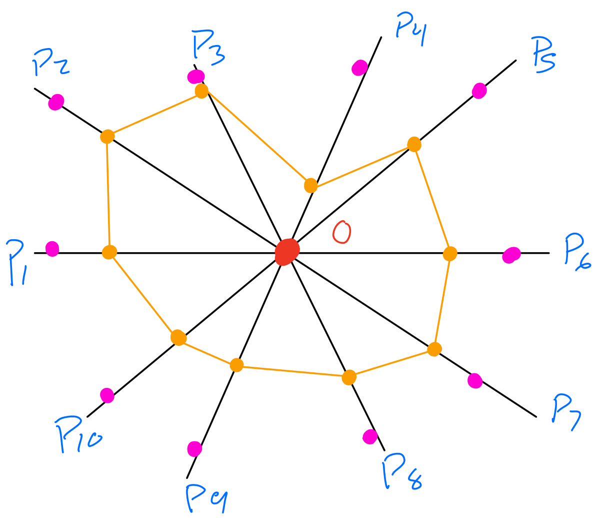
$\downarrow$ matrix form

$$\begin{bmatrix} \underbrace{1 \dots 1}_{\text{10 1's (they're on)}}, & \underbrace{0 \dots 0}_{\text{(they're off)}} \end{bmatrix} \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix}$$

these are linear constraints! Meaning that the problem still is convex and has a unique solution!

c) Not sure

By using this constraint, which is non convex, then we don't know how many solutions there are since the problem is non-convex now



If you draw a line between the intensities (orange) then there will be areas not in the set so it is non convex!

[set is between "0" and P_{max} (pink dot)]

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.

5) $\underbrace{x}_{\text{slope}}\underbrace{y}_{\text{variable}} - \underbrace{c(x)}_{\text{constant}} \in$ linear w/ respect to y

$\underbrace{\max\{x_1 y - c(x_1),\ x_2 y - c(x_2), \ldots\}}_{\text{still convex function}}$

for any y we take corresponding x, we apply def. of convex

so,

$c^*(y) = \underbrace{\max\{\text{convex function}\}}$

↳ this is a convex fuction!

Because all "max()" does is find the largest value and since every entry is a line function with respect to y, $\max\{xy - c(x)\}$

$\equiv$ convex function