Problem 1

(i)

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

The stationary point of the function is obtained by taking the gradient and setting it to zero:

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

Solving for x_1 and x_2 gives a stationary point (1,1).

Indefiniteness of the Hessian can be shown by computing its eigenvalues: \

$H = \begin{bmatrix} 4 & -4 \\ -4 & -1 \end{bmatrix}$

$\begin{vmatrix} 4-\lambda & -4 \\ -4 & -1-\lambda \end{vmatrix} = 0$

$\lambda_1 = 10.54, \lambda_2 = -4.54$

Since the eigenvalues are both positive and negative, the Hessian is indefinite.

To find the direction of the slopes awy from the saddle point (1,1) we need to find the points such that the difference f - f(1,1) < 0:

$f - f(1,1) = \frac{f_{x_1x_1}(1,1)(x_1-1)^2}{2} + \frac{f_{x_1x_2}(1,1)(x_1-1)(x_2-1)}{2} + \frac{f_{x_2x_2}(1,1)(x_2-1)^2}{2}$

$f - 0.5 = 2(x_1-1)^2 - 2(x_1-1)(x_2-1) - 0.5(x_2-1)^2 < 0$

Problem 2

Part a: (Analytical)

The problem asks to find a point closest to (-1, 0, 1) such that it is constrained to lie on the plane $x_1 + 2x_2 + 3x_3 = 1$.

The problem can be formulated as:

$\underset{x}{\operatorname{argmin}} L(x_0, x)$

st $x_1 + 2x_2 + 3x_3 = 1$

Using Lagrange multipliers on the equality constraint:

$\underset{x}{\operatorname{argmin}} L(x, x_0) = (x_{01} - x_1)^2 + (x_{02} - x_2)^2 + (x_{03} - x_3)^2 = \lambda g(x)$

$\nabla L(x) = \lambda \nabla g(x)$

$g(x) = c$

The constraint equation works out to be:

$\nabla L(x) = \begin{bmatrix} 2x_1 + 2 \\ 2x_2 \\ 2x_3 - 2 \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda \\ 2\lambda \end{bmatrix}$

Together with the equation $x_1 + 2x_2 + 3x_3 - 1 = 0$, solving for the x's we get:

$x = \begin{bmatrix} -1 \\ \frac{-1}{8} \\ \frac{3}{4} \end{bmatrix}$

Problem 3

(i) Let f(x) and g(x) be two convex functions defined on the convex set X.

The linear combination h(x) is defined as:

$h(x) = af(x) + bg(x)$

If f(x) and g(x) are convex then:

$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$

$g(\lambda x_1 + (1-\lambda)x_2) \leq \lambda g(x_1) + (1-\lambda)g(x_2)$

$af(\lambda x_1 + (1-\lambda)x_2) + bg(\lambda x_1 + (1-\lambda)x_2) \leq a(\lambda f(x_1) + (1-\lambda)f(x_2)) + b(\lambda g(x_1) + (1-\lambda)g(x_2))$

The right side can be rewritten as:

$\lambda(af(x_1) + bg(x_1)) + (1-\lambda)(af(x_2) + bg(x_2))$

From the definition of h(x), the right hand side can be transformed to:

$\lambda h(x_1) + (1-\lambda)h(x_2)$

Problem 5

(i) Formulation of optimization problem:

$\min_{p_j} \sum_{k=1}^{n} (I_k - I_t)^2$

st $0 \leq p_j \leq p_{max}$

(ii) A formulation is convex if the objective is a convex function and the constraints have a feasible set that is convex.

Consider the function: $f(p) = (a_k^T p - I)^2$

$\nabla f = 2a_k a_k^T p - 2Ia_k$

$\nabla^2 f = 2a_k a_k^T$

Lemma: If $d^T H d \geq 0$ for all d $\neq 0$ then H is positive semi definite.

In this case, applying the lemma gives us $d^T H d = 2d^T a_k a_k^T d$, which is always non negative.

Moreover, the summation of k convex functions is also convex. Therefore the objective is convex. The feasible set from the constraints is convex because they form a segmented space that is a hypercube.

(iii) If we require the power output of any of the n lamps to be less than p*, the modified optimization problem is:

$\min_{p_j} \sum_{k=1}^{n} (I_k - I_t)^2$

st $0 \leq p_j \leq p_{max}$,

$\sum_{j=1}^{k} p_j < p^*, k = 1, ..., n$

The second constraint can be further split into n separate constraints. Unique solutions are possible if all the constraints are convex sets. Consider the constraint $\sum_{j=1}^{l} p_j < p^*, l \epsilon k$.

The summation produces an l-dimensional half space of a hyperplane, which is a convex set.

Therefore the problem has a unique solution.

(iv) If only half the lamps can be on at the same time, then the problem has a combinatorial constraint, where out of n lamps, any n/2 of them can be switched on. The constraint to be added to the formulation is:

$\min_{p_j} \sum_{k=1}^{n} (I_k - I_t)^2$

st $0 \leq p_j \leq p_{max}$,

$||x||_0 \leq \frac{n}{2}$

The l-0 norm is not convex, therefore the problem does not have a unique solution.

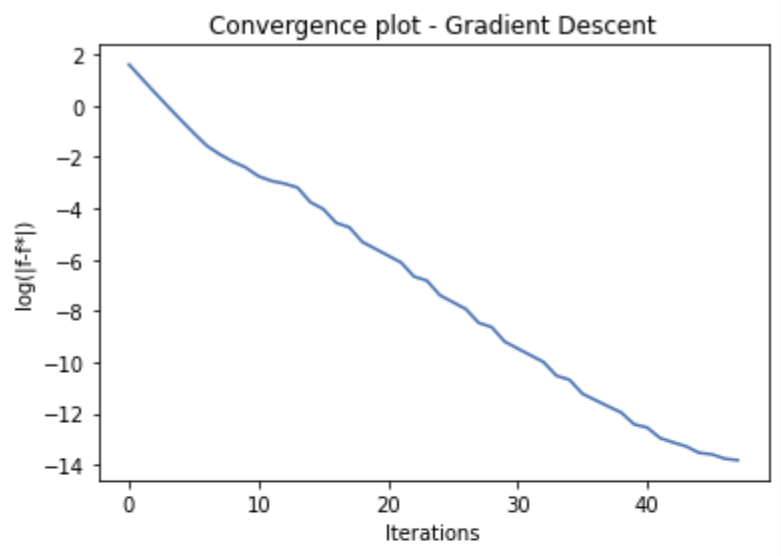
```
# PROBLEM 2 : CODE FOR GRADIENT DESCENT
print("Gradient descent solution")
import numpy as np

def objective(x):
    return (2 - (2*x[0]) - (3*x[1]))**2 + (x[0])**2 + (x[1] - 1)**2
```

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

def phi(a,x):
    obj = objective(x)
    corr = a*0.8*np.dot(grad(x), grad(x))
    return obj - corr

def line_search(x):
    a = 1  # initialize step size
    while np.linalg.norm(phi(a,x))<objective(x-a*grad(x)):  # if f(x+a*d)>phi(a) then
backtrack. d is the search direction
        a = 0.5*a
    return a

eps = 1e-3
x0 = [0,0]
k = 0
x = x0
soln = [objective(x)]

error = np.linalg.norm(grad(x))

while error >= eps and k < 10000:  # keep searching while gradient norm is larger than eps
    a = line_search(x)
    x = x - a*grad(x)
    error = np.linalg.norm(grad(x))
    soln.append(objective(x))
    k+=1
    if k % 1000 == 0: print("Iteration:", k, "alpha:", a, "Error:", error, "x:", x)

import matplotlib.pyplot as plt
y = [np.log(np.linalg.norm(s - objective(x) + 1e-6)) for s in soln]
fig = plt.figure()
plt.plot(y)
plt.xlabel('Iterations')
plt.ylabel('log(|f-f*|)')
plt.title('Convergence plot - Gradient Descent')
x1 = 1 - 2*x[0] - 3*x[1]
x = [x1, x[0], x[1]]
print("Final solution:", x)

##########################################
# Problem 2: Newton's method
print("Newton's method solution: ")
import numpy as np

def objective(x):
    return (2 - (2*x[0]) - (3*x[1]))**2 + (x[0])**2 + (x[1] - 1)**2
```

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

def hessian():
    return np.asarray([[10,12],[12,20]])

def phi(a,x):
    obj = objective(x)
    corr = a*0.8*np.matmul(np.matmul(grad(x).reshape(1,2),np.linalg.inv(hessian())),
grad(x).reshape(2,1))
    return obj - corr

def line_search(x):
    a = 1  # initialize step size
    while phi(a,x)<objective(x-a*grad(x)):  # if f(x+a*d)>phi(a) then backtrack. d is the
search direction
        a = 0.5*a
    return a

eps = 1e-3
x0 = [0,0]
k = 0
x = np.asarray(x0).reshape(2,1)
soln = [objective(x)]

lam = np.matmul(np.matmul(grad(x).reshape(1,2),np.linalg.inv(hessian())),
grad(x).reshape(2,1))
while lam**2/2 >= eps and k < 10000:  # keep searching while gradient norm is larger than eps
    a = line_search(x)
    a = 1
    x = x - a*np.matmul(np.linalg.inv(hessian()), grad(x).reshape(2,1))
    soln.append(objective(x))
    lam = np.matmul(np.matmul(grad(x).reshape(1,2),np.linalg.inv(hessian())),
grad(x).reshape(2,1))
    k+=1
    if k % 1 == 0: print("Iteration:", k, "alpha:", a, "Error:", error, "x:", x)

import matplotlib.pyplot as plt
fig = plt.figure()

y = [np.log(np.linalg.norm(s - objective(x) + 1e-6)) for s in soln]
plt.plot(y)
plt.xlabel('Iterations')
plt.ylabel('log(|f-f*|)')
plt.title('Convergence plot - Newton')

x1 = 1 - 2*x[0] - 3*x[1]
x = [x1, x[0], x[1]]
print("Final solution:", x)
```

```
Gradient descent solution
Final solution: [-1.0713988626792919, -0.14249520055213166, 0.7854630879278517]
Newton's method solution:
Iteration: 1 alpha: 1 Error: 0.000910695792946387 x: [[-0.14285714]
 [ 0.78571429]]
Final solution: [array([-1.07142857]), array([-0.14285714]), array([0.78571429])]
```

</>



</>