Assignment – 4

By: Maor Ashkenazi, Pan Eyal

Question 1

Section A

We will define the Lagrangian function for our constrained optimization problem as follows:

$$\underset{\boldsymbol{x}\in R^3,\lambda\in R}{\mathcal{L}(\boldsymbol{x},\lambda)} = x_1x_2 + x_2x_3 + x_1x_3 + \lambda(x_1 + x_2 + x_3 - 3)$$

The LICQ conditions holds because we only have one constraint.

We will use KKT to search for suspicious critical point.

$$\nabla_{\mathbf{x}}\mathcal{L} = \begin{bmatrix} x_2 + x_3 + \lambda \\ x_1 + x_3 + \lambda \\ x_1 + x_2 + \lambda \end{bmatrix} = 0$$

$$C^{eq}(\boldsymbol{x}) = 0 \rightarrow x_1 + x_2 + x_3 - 3 = 0$$

From equations 1 & 2:

$$x_2 + x_3 + \lambda = x_1 + x_3 + \lambda = 0 \Rightarrow x_1 = x_2$$

Also, from equations 2 & 3:

$$x_1 + x_3 + \lambda = x_1 + x_2 + \lambda = 0 \Rightarrow x_3 = x_2$$

Hence,

$$x_1 = x_2 = x_3$$

And:

$$x_1 + x_2 + x_3 - 3 = 0 \Rightarrow 3x_1 = 3 \Rightarrow \boldsymbol{x_1} = \boldsymbol{x_2} = \boldsymbol{x_3} = \mathbf{1}$$

$$\lambda + 2 = 0 \ \Rightarrow \boldsymbol{\lambda} = -\mathbf{2}$$

Therefore, the critical point is $(1,1,1)$.

Section B

We will find the hessian matrix of the Lagrangian:

$$\nabla_{\mathbf{x}}^2 \mathcal{L} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

We will show that for every direction $\boldsymbol{y}$ that satisfies the condition (i.e., $\langle \boldsymbol{y}, \nabla C \rangle = 0$), $\boldsymbol{y}^T \nabla_{\mathbf{x}}^2 \mathcal{L} \boldsymbol{y} < 0$, thus proving that the critical point is a maximum.

$$\langle \boldsymbol{y}, \nabla C \rangle = y_1 + y_2 + y_3 = 0$$

$$\boldsymbol{y}^T \nabla_{\mathrm{x}}^2 \mathcal{L} \boldsymbol{y} = [y_1, y_2, y_3] \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_2 \end{bmatrix} = [y_2 + y_3 \quad y_1 + y_3 \quad y_1 + y_2] \begin{bmatrix} y_1 \\ y_2 \\ y_2 \end{bmatrix} =$$

$$y_1(y_2 + y_3) + y_2(y_1 + y_3) + y_3(y_1 + y_2) =$$

$$y_1 y_2 + y_1 y_3 + y_2 y_1 + y_2 y_3 + y_3 y_1 + y_3 y_2 =$$

$$2(y_1 y_2 + y_2 y_3 + y_1 y_3) = (*)$$

Since $y_1 + y_2 + y_3 = 0 \rightarrow \Rightarrow \; y_1 = -y_2 - y_3$, Therefore:

$$(*) = 2\big((-y_2 - y_3)y_2 + y_2 y_3 + (-y_2 - y_3)y_3\big) =$$

$$2(-y_2^2 - y_2 y_3 + y_2 y_3 - y_2 y_3 - y_3^2) =$$

$$-2(y_2^2 + 2y_2 y_3 + y_3^2) + 2y_2 y_3 =$$

$$-2(y_2 + y_3)^2 + 2y_2 y_3 =$$

$$-2\left[\underbrace{(y_2 + y_3)^2 - y_2 y_3}_{(**)}\right]$$

If y_2, y_3 have different signs, it's clear that the expression $(**)$ is positive, thus $\boldsymbol{y}^T \nabla_{\mathbf{x}}^2 \mathcal{L} \boldsymbol{y} < \mathbf{0}$.

If y_2, y_3 have the same sign, the expression $(**)$ (which can also be written as $y_2^2 + y_2 y_3 + y_3^2$) is positive, thus $\boldsymbol{y}^T \nabla_{\mathbf{x}}^2 \mathcal{L} \boldsymbol{y} < \mathbf{0}$.

Question 2

Section A

We need to find the minimum number of activate inequality constraints for which KKT holds.

To begin, we will check the case where there aren't any active inequality constraints, and define the Lagrangian function as follows:

$$\underset{\boldsymbol{x}\in R^2,\lambda^{eq}\in R}{\mathcal{L}(\boldsymbol{x},\lambda)} = (x_1+x_2)^2 - 10(x_1+x_2) + \lambda^{eq}(3x_1+x_2-6)$$

We will not specifically mention the inequality constraints Lagrangian multipliers in the Lagrangian, because we assumed they aren't active.

We will use KKT to search for suspicious critical point.

$$\nabla_{\mathbf{x}}\mathcal{L} = \begin{bmatrix} 2x_1+2x_2-10+3\lambda^{eq} \\ 2x_1+2x_2-10+\lambda^{eq} \end{bmatrix} = 0$$

$$C^{eq}(\boldsymbol{x}) = 0 \rightarrow 3x_1+x_2-6=0$$

From equations 1 & 2:

$$2x_1+2x_2-10+3\lambda^{eq} = 2x_1+2x_2-10+\lambda^{eq} = 0 \Rightarrow \lambda^{eq} = 0$$

$$2x_1+2x_2-10+\lambda^{eq} = 0 \Rightarrow x_1+x_2 = 5$$

From the equality constraint:

$$3x_1+(5-x_1)-6 = 0 \Rightarrow x_1 = 0.5 \Rightarrow x_2 = 4.5$$

We can see that the first inequality constraint does not hold for this solution ($x_1^2+x_2^2 > 5$). Thus, KKT does not hold.

We'll check the case where only the first inequality constraint holds, and define the Lagrangian function as follows:

$$\underset{\boldsymbol{x}\in R^2,\lambda^{eq}\in R}{\mathcal{L}(\boldsymbol{x},\lambda)} = (x_1+x_2)^2 - 10(x_1+x_2) + \lambda^{eq}(3x_1+x_2-6) + \lambda^{ieq}(x_1^2+x_2^2-5)$$

$$\nabla_{\mathbf{x}}\mathcal{L} = \begin{bmatrix} 2x_1+2x_2-10+3\lambda^{eq}+2\lambda^{ieq}x_1 \\ 2x_1+2x_2-10+\lambda^{eq}+2\lambda^{ieq}x_2 \end{bmatrix} = 0$$

$$C^{eq}(\boldsymbol{x}) = 0 \rightarrow 3x_1+x_2-6=0$$

$$C_1^{ieq}(\boldsymbol{x}) = 0 \rightarrow x_1^2+x_2^2-5=0$$

From equations 1 & 2:

$$2x_1+2x_2-10+3\lambda^{eq}+2\lambda^{ieq}x_1 = 2x_1+2x_2-10+\lambda^{eq}+2\lambda^{ieq}x_2 = 0 \Rightarrow$$

$$\lambda^{eq} = \lambda^{ieq}(x_2-x_1)$$

From the constraints we get:

$$x_2 = 6 - 3x_1 \Rightarrow$$

$$x_1^2 + (6 - 3x_1)^2 - 5 = 0 \Rightarrow$$

$$x_1^2 + 36 - 36x_1 + 9x_1^2 - 5 = 0 \Rightarrow$$

$$10x_1^2 - 36x_1 + 31 = 0$$

$$x_1 = \begin{cases} 1.4258 \\ 2.1742 \end{cases} \Rightarrow x_2 = \begin{cases} 1.7226 \\ -0.5226 \end{cases}$$

We'll compute the Lagrange multipliers for the point $(2.1742, -0.5226)$:

$$\lambda^{eq} = -2.6968\lambda^{ieq}$$

$$2x_1 + 2x_2 - 10 + \lambda^{eq} + 2\lambda^{ieq}x_2 = 0 \Rightarrow -6.6968 - 2.6968\lambda^{ieq} - 1.0452\lambda^{ieq} = 0$$

$$\Rightarrow 3.742\lambda^{ieq} = -6.6968$$

$$\lambda^{ieq} < 0$$

In contrary to the KKT conditions, thus this point is not a critical point.

We'll compute the Lagrange multipliers for the point $(1.4258, 1.7226)$:

$$\lambda^{eq} = 0.2968\lambda^{ieq}$$

$$2x_1 + 2x_2 - 10 + \lambda^{eq} + 2\lambda^{ieq}x_2 = 0 \Rightarrow -3.7032 + 0.2968\lambda^{ieq} + 3.4452\lambda^{ieq} = 0$$

$$\Rightarrow 3.742\lambda^{ieq} = 3.7032$$

$$\lambda^{ieq} = 0.9896, \qquad \lambda^{eq} = 0.2937$$

Also, the second inequality holds ($-x_1 < 0$). This means that all the KKT conditions hold for $(1.4258, 1.7226)$, and it is a critical point.

Section B

We will find the Hessian for the Lagrangian:

$$\nabla_{\mathbf{x}}^2 \mathcal{L} = \begin{bmatrix} 2 + 2\lambda^{ieq} & 2 \\ 2 & 2 + 2\lambda^{ieq} \end{bmatrix} = \begin{bmatrix} 3.9792 & 2 \\ 2 & 3.9792 \end{bmatrix}$$

From linear algebra, we know that a matrix is positive definite if the determinants of all upper left matrices are positive. Since this is a $2x2$ matrix and the top left value is positive, it's easy to check:

$$\det(\nabla_{\mathbf{x}}^2 \mathcal{L}) = 3.9792^2 - 2^2 > 0$$

We can also check the eigenvalues of the hessian, to see that they are all positive:

$$\lambda_1 = \frac{1237}{625}, \lambda_2 = \frac{3737}{625}$$

Thus, the Hessian is SPD, and we can conclude that for every $y \in R^2$, it holds that $y^T \nabla_{\mathbf{x}}^2 \mathcal{L} y > 0$. This means that the critical point is a **minimum**.

Section C

The unconstrained minimization problem that corresponds to the problem above, using the penalty method is:

$$\min_{x\in R^2}(x_1+x_2)^2-10(x_1+x_2)+\mu\left(\rho(3x_1+x_2-6)+\rho(\max\{0,x_1^2+x_2^2-5\})+\rho(\max\{0,-x_1\})\right)=$$

$$\min_{x\in R^2}\underbrace{(x_1+x_2)^2-10(x_1+x_2)+\mu((3x_1+x_2-6)^2+\max\{0,x_1^2+x_2^2-5\}^2+\max\{0,-x_1\}^2)}_{f_\mu(x)}$$

Section D

We find the gradient for steepest descent:

$$\nabla_x f_\mu=\begin{bmatrix}2(x_1+x_2)-10+\mu\left(6(3x_1+x_2-6)+I_{ieq1>0}*\left(4x_1(x_1^2+x_2^2-5)\right)+I_{ieq2>0}*2x_1\right)\\ 2(x_1+x_2)-10+\mu\left(2(3x_1+x_2-6)+I_{ieq1>0}*\left(4x_2(x_1^2+x_2^2-5)\right)\right)\end{bmatrix}$$

Now we implement the function and its gradient in python:

```
import numpy as np

def f(x, mu):
    return (x[0] + x[1])**2 - 10*(x[0] + x[1]) + mu*((3*x[0]+x[1]-6)**2 + max(0, x[0]**2 + x[1]**2 - 5)**2 + max(0,-x[0])**2)

def g(x, mu):
    ieq_1 = 4*x[0]*(x[0]**2 + x[1]**2 - 5) if (x[0]**2 + x[1]**2 - 5) > 0 else 0
    ieq_2 = 2*x[0] if -x[1] > 0 else 0
    g1 = 2*(x[0] + x[1]) - 10 + mu*(6*(3*x[0]+x[1]-6) + ieq_1 + ieq_2)
    g2 = 2*(x[0] + x[1]) - 10 + mu*(2*(3*x[0]+x[1]-6) + ieq_1)
    return np.array([g1, g2])
```

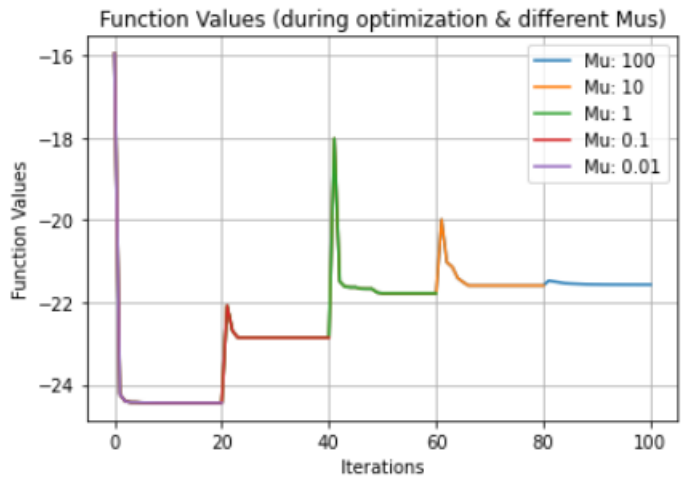
Next, we implemented the line search & optimization iterations

```
def linesearch(iterations, x, mu, d, grad, alpha=1, beta=0.5, c=1e-4):
    for j in range(iterations):
        if f(x+alpha*d, mu) <= f(x, mu) + c*alpha*(d@grad):
            break
        else:
            alpha = alpha*beta
    return alpha

def opt(iterations, x, mus):
    fs = [f(x, mus[0])]
    mus_intervals = [0]

    for mu in mus:
        for i in range(iterations):
            grad = g(x, mu)
            d = -grad
            alpha = linesearch(100, x, mu, d, grad)

            x = x + alpha*d

            fs.append(f(x, mu))

            if np.linalg.norm(grad) < 1e-3:
                break
        mus_intervals.append(mus_intervals[-1] + i + 1)

    return x, fs, mus_intervals
```

We weren't sure what was required in the stopping condition, since we didn't want to assume that we have the optimal solution in-hand. Instead, we used the stopping condition showed in class.

We ran each μ for 20 iterations and received the following result: $(1.4246, 1.7256)$, which is very similar to the one we found analytically.

Here is the required plot of function values, during the optimization iterations. The updated μ's can be seen in the different graph part colors:



Question 3

Section A

We will find the closed form solution for:

$$\min_{x \in R} \left\{ \frac{1}{2} h x^2 - g x \right\} \; s.t. \, a \leq x \leq b \; for \; a < b, h > 0$$

To begin, we find the derivative of $f(x) = \frac{1}{2} h x^2 - g x$ without considering the constraints.

$$f'(x) = hx - g$$

$$\Rightarrow hx - g = 0 \; \Rightarrow x = \frac{g}{h}$$

In addition,

$$f''(x) = h$$

Since $h > 0$, $\frac{g}{h}$ is a minimum critical point. And because $f(x)$ is convex, $\frac{g}{h}$ is a global minimum.

If $a \leq \frac{g}{h} \leq b$, we found a solution for the constrained minimization problem. Otherwise, since the function is convex, we know that the solution for the constrained problem will be on the closest "border". We formalize this as follows:

$$x_{opt} = \begin{cases} \frac{g}{h} \mid a \leq \frac{g}{h} \leq b \\ a \mid \quad \frac{g}{h} \leq a \\ b \mid \quad \frac{g}{h} \geq b \end{cases}$$

Section B

$$f(x) = \frac{1}{2} x^T H x - x^T g$$

$$\underbrace{=}_{\substack{opening\ the \\ matmul}} \frac{1}{2} \sum_{i=1}^{n} x_i \sum_{j=1}^{n} h_{ij} x_j - \sum_{i=1}^{n} x_i g_i$$

$$= \sum_{i=1}^{n} \left\{ \frac{1}{2} \left(\sum_{j=1}^{n} h_{ij} x_j \right) - g_i \right\} x_i$$

$$= \sum_{i=1}^{n} \left\{ \frac{1}{2} \left(h_{ii} x_i + \left(\sum_{j \neq i}^{n} h_{ij} x_j \right) \right) - g_i \right\} x_i$$

$$= \sum_{i=1}^{n} \left\{ \frac{1}{2} h_{ii} x_i^2 + \left[\frac{1}{2} \left(\sum_{j \neq i}^{n} h_{ij} x_j \right) - g_i \right] x_i \right\}$$

$$= \underbrace{\sum_{i=1}^{n} \frac{1}{2} h_{ii} x_i^2}_{(*)} + \underbrace{\sum_{i=1}^{n} \left[\frac{1}{2} \left(\sum_{j \neq i}^{n} h_{ij} x_j \right) - g_i \right] x_i}_{(**)}$$

To minimize this function for a scalar x_k (we changed the index for clarity), the important observation is that in each iteration, we assume that all other variables $x_i \; s.t \; i \neq k$ are **known.** This means that every expression that **does not** include x_k is a scalar and can't be modified thus can be ignored in the minimization problem.

In $(*)$ the only expression that is relevant for minimization is when $i = k$.

In $(**)$ it's a bit more complex:

1. When $i = k$, the entire expression is relevant.
2. When $i \neq k$, the only expression that is relevant to us is when $j = k$. In addition, we can ignore the $(-g_i)$ inside the parenthesis, since after the multiplication it results in a constant expression $(-g_i x_i)$.

Thus, we get the following minimization problem:

$$\min_{x_k \in R} \left\{ \underbrace{\frac{1}{2} h_{kk} x_k^2}_{from\ (*)} + \underbrace{\left[\frac{1}{2} \left(\sum_{j \neq k}^{n} h_{kj} x_j \right) - g_k \right] x_k}_{from\ (**)_1} + \underbrace{\sum_{i \neq k}^{n} \frac{1}{2} (h_{ik} x_k) x_i}_{from\ (**)_2} \right\} \underbrace{=}_{h\ is\ symmetric}$$

$$\min_{x_k \in R} \left\{ \frac{1}{2} h_{kk} x_k^2 + \left[\frac{1}{2} \left(\sum_{j \neq k}^{n} h_{kj} x_j \right) - g_k \right] x_k + \frac{1}{2} \left(\sum_{j \neq k}^{n} h_{kj} x_j \right) x_k \right\} =$$

$$\min_{x_k \in R} \left\{ \frac{1}{2} h_{kk} x_k^2 + \left[\left(\sum_{j \neq k}^{n} h_{kj} x_j \right) - g_k \right] x_k \right\}$$

Also, because we only consider a single index, only its constraint is relevant: - $a_k \leq x_k \leq b_k$.

To define the update step for the coordinate descent algorithm, we notice that our minimization problem is in the same format as the scalar minimization problem in the previous section:

$$\min_{x_k \in R} \left\{ \frac{1}{2} \underbrace{h_{kk}}_{previous\ h} x_k^2 - \underbrace{\left[-\left(\sum_{j \neq k}^{n} h_{kj} x_j \right) + g_k \right]}_{previous\ g} x_k \right\}$$

Thus, the update step is as follows:

$$x_k = \begin{cases} \dfrac{-\left[\left(\sum_{j\neq k}^{n} h_{kj}x_j\right) - g_k\right]}{h_{kk}} & \mid a_k \leq \dfrac{-\left[\left(\sum_{j\neq k}^{n} h_{kj}x_j\right) - g_k\right]}{h_{kk}} \leq b_k \\ a_k & \mid \dfrac{-\left[\left(\sum_{j\neq k}^{n} h_{kj}x_j\right) - g_k\right]}{h_{kk}} \leq a_k \\ b_k & \mid \dfrac{-\left[\left(\sum_{j\neq k}^{n} h_{kj}x_j\right) - g_k\right]}{h_{kk}} \geq b_k \end{cases}$$

Section C

The following code implements coordinate descent for our constrained minimization problem:

```
def f(x):
    return 0.5*x.transpose()@H@x - x.transpose()@g

def update_index(x, k):
    value = -(np.sum([H[k,j]*x[j] for j in range(len(a)) if j!=k]) - g[k])/H[k, k]
    return max(a[k], min(b[k], value))
```

```
def opt(iterations, x):
    fs = [f(x)]

    for i in range(iterations):
        prev_x = x.copy()
        for coord_ind in range(len(a)):
            x[coord_ind] = update_index(x, coord_ind)

            fs.append(f(x))

        if np.linalg.norm(x-prev_x) < 1e-3:
            print(f"Converged after {i} iterations")
            break

    return x, fs
```

Section D

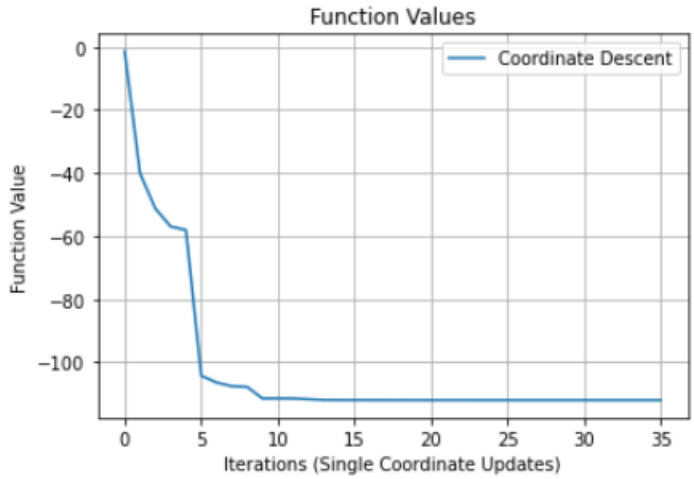
The following code runs the coordinate descent algorithm on the given parameters

```
import numpy as np
H = np.array([[5, -1, -1, -1, -1],
              [-1, 5, -1, -1, -1],
              [-1, -1, 5, -1, -1],
              [-1, -1, -1, 5, -1],
              [-1, -1, -1, -1, 5]])
g = np.array([18, 6, -12, -6, 18])
a = np.array([0, 0, 0, 0, 0])
b = np.array([5, 5, 5, 5, 5])
```

```
x, fs = opt(1000, np.random.random(len(a)))
```

```
Converged after 6 iterations
```

The following plot shows the function values during the optimization (each iteration is a single coordinate update):



The solution we obtained is:

$$\boldsymbol{x_{opt}} = [\mathbf{5}, \mathbf{3.6666}, \mathbf{0.6666}, \mathbf{1.6666}, \mathbf{5}]$$