**University of Illinois** **Fall 2023**

# ECE 490: Programming Assignment 3
# Method of Multipliers

**Due: Monday, 17 April 2023 5:00pm**

The assignment will be done in groups of 3. The code must be uploaded to Gradescope by the deadline. One group member will be required to run the code in front of a TA during the assigned office hour slots (**in-person**) and the grade will be handed out for all group members right then. Please look at the course website for the group assignment and the allotted time slots.

1. [**Method of Multipliers**]
   Consider a linear equality constrained optimization problem of the form

$$\min x^\top Q x$$
$$s.t.\ Ax = b$$

where $Q \succeq 0$ and $A$ is an $m \times n$ full rank matrix with $m < n$. The goal of this programming assignment is to implement the method of multipliers (see Lec 20 and Sec 5.2.2 of the textbook: D. Bertsekas. Nonlinear Programming) to solve this problem. Write a python function that will take as input the matrices $Q$ and $A$, the vector $b$, and a constant $\epsilon$ (used to define a stopping criterion for the algorithm). Use the stopping condition $\|Ax^{(k)} - b\|_2 < \epsilon$. Note that the stopping conditions states that the algorithm is terminated when the Lagrange multipliers do not change significantly. Use gradient descent with Armijo's rule with appropriate parameters to solve the optimization problem at iteration $k$ given by:

$$x^{(k)} \in \arg\min_x L_{c_k}(x, \lambda^{(k)}),$$

where $L_c(\cdot,\cdot)$ is the *augmented Lagrangian function* as defined in the notes or textbook. The output should be:

(a) The optimal value of $x$ found using the method of multipliers.

(b) The number of iterations taken.

(c) Plot the relative error $\frac{\|x^* - x^{(k)}\|_2}{\|x^*\|_2}$ versus the iteration number $k$, for different choices of the sequence $\{c_k\}$ (in the same plot). Here, $x^*$ is the optimal solution.

**Hint:** Experiment with various choices of $\{c_k\}$, as there is no universally acceptable way to choose it. Some examples of $\{c_k\}$:

(a) $c_k =$ constant.

(b) $c_{k+1} = c_k + \beta,\ \beta > 0$.

(c) $c_{k+1} = \beta c_k$, $\beta$ close to 1.

(d) $c_{k+1} = \begin{cases} \beta c_k, & \|h(x^{(k)})\|_2 > \gamma \|h(x^{(k-1)})\|_2 \\ c_k, & \text{otherwise} \end{cases}$,
where $h(x) = Ax - b$.

Use the following code snippet to generate $Q, A, b$:

```python
import numpy as np

def get_Q_A_b(m,n):
    Q = np.random.rand(n,n)-0.5
    Q = 10*Q @ Q.T + 0.1*np.eye(n)
    A = np.random.normal(size=(m,n))
    b = 2*(np.random.rand(m)-0.5)
    return Q,A,b

#Example:
np.random.seed()
Q,A,b = get_Q_A_b(10,25)
```