University of Illinois **Fall 2021**

ECE 490: Programming Assignment 2
Constrained Optimization and Newton's method

Due: Wednesday, 22 March 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. **[Projected gradient descent]**
Write your own code to solve (1) using projected gradient descent with fixed step-size, chosen in accordance with convergence results. The set $\mathcal{S} = \{x : x \in \mathbb{R}^n, \ell[i] \le x[i] \le u[i], \ \forall i \in [n]\}$ is a box-constrained set with the ith coordinate of x bounded between $\ell[i]$ and $u[i]$. Choose the stopping criterion $||x_k - x_{k+1}||_2 \le \epsilon$. You can set $\epsilon = 0.01$ in the simulations. Justify why this is a valid stopping criterion and why we cannot use a stopping criterion based on the gradient for this problem.

$$\min_{x \in \mathcal{S}} \frac{1}{2} x^T Q x + b^T x + c := f(x) \tag{1}$$

Use the file *asgn_source.py* (found in assignment files) to generate Q, b, c and also the box constraints. An example is given in the following code snippet (also found in *projected_gd.py* in the assignment files) which has to be used to generate the solution. The arrays *constraint_min* and *constraint_max* in the snippet contain the box constraints $\ell[i], u[i]$ as the ith coordinate respectively. (Note : You can also use *scipy.optimize* to check your answer).

```python
import asgn_source as asou
import numpy as np
import scipy

def project_coordinate(x,constraint_min,constraint_max,n):
    ### write your code here
    return x_projected

def run_projected_GD(constraint_min,constraint_max,Q,b,c,n):
    ### write your code here
    return result

if __name__ == '__main__':
    n =25
    np.random.seed()
    constraint_min,constraint_max, Q_val,b_val, c_val = asou.get_parameters(n)
    armijo_sol = run_projected_GD(constraint_min,constraint_max,Q_val,b_val, c_val,n)
```

2. **[Newton's Method]**
Consider Newton's method with stepsize $\alpha = 1$, i.e.,

$$z_{k+1} = z_k - \left(\nabla^2 f(z_k)\right)^{-1} \nabla f(z_k).$$

Suppose we apply this method to the function $f : \mathbb{R}^2 \to \mathbb{R}$ given by

$$f(z) = x^3 y - x y^3 - y, \quad z = (x, y).$$

   (a) Find the stationary points of f.

   (b) Write a function in Python, which takes as input arguments the initialization point z_0 and the number of iterations N, and outputs the last iterate z_N of Newton's method (for the given fucntion and stepsize). For various initialization points in $[-1, 1] \times [-1, 1]$, color z_0 according to the stationary point of f nearest to z_N. The code for plotting the resulting image is provided below. If plotting takes a lot of memory or time, reduce the value of n (note that default is 500 and this will take a few minutes).

```python
import numpy as np
import matplotlib.pyplot as plt

def newton_method(z0: np.array, N=50)->np.array:
    # Write your code here.
    # If needed, you can define other functions as well to be used here.
    # Input z0 and output zN should be numpy.ndarray objects with 2 elements:
    # e.g. np.array([x,y]).
    return zN

def plot_image(s_points: np.array, n=500, domain=(-1, 1, -1, 1)):
    m = np.zeros((n, n))
    xmin, xmax, ymin, ymax = domain
    for ix, x in enumerate(np.linspace(xmin, xmax, n)):
        for iy, y in enumerate(np.linspace(ymin, ymax, n)):
            z0 = np.array([x,y])
            zN = newton_method(z0)
            code = np.argmin(np.linalg.norm(s_points-zN,ord=2,axis=1))
            m[iy, ix] = code

    plt.imshow(m, cmap='brg')
    plt.axis('off')
    plt.savefig("q2_hw3.png")

if __name__ == '__main__':
    # Example of usage.
    # In this example, the stationary points are (0,0), (1,1), (2,2) and (3,3).
    # Replace these with the ones obatined in part a).
    stationary_points= np.array([[0,0],[1,1],[2,2],[3,3]])
    plot_image(stationary_points)
```

What will happen during the demonstration session:

You will be given asked to add a line to your code: **numpy.random.seed()** to fix the matrices Q, b, c produced, and you will be asked to use your code with a value of n to generate the solutions. You are expected to be able to comfortably explain your code to the TA, and answer some follow-up questions.