Assignment 1
E0-230 Computational Methods of Optimization

Instructions

- This is an individual assignment, all the work must be your own.
- No copying or sharing of code is allowed. Any form of academic dishonesty will result in ZERO marks for this assignment.
- The deadline for submission is 13^{th} September, 2024.
- Submissions will be accepted till 17^{th} September, 2024, with a penalty of 5 points per day.
- All the code must be written in Python, along with appropriate comments. The code needs to be submitted for any problem that requires any implementation or programming. Follow the naming convention suggested in the question and feel free to pass additional arguments if necessary with mention. These codes can be reused in subsequent assignments.
- The outputs of the coding problems and other analytical solutions should be reported in a single LaTeX-generated PDF file.
- Submit a single zip file in the `Last_Five_digits_of_sr_no_CMO_A1` which should contains one .py file and LaTeX-generated PDF report.

Oracle Access Instructions

- Based on your OS, download the corresponding .zip file and extract it to get the folder `CMO_A1`.
- You need to run your .py file in the extracted `CMO_A1` folder.
- Make sure to include the following lines of code:

```
from CMO_A1 import f1, f2, f3, f4
import numpy as np
```

- All the oracle outputs are dependent on the last five digits of your sr.no, which should be passed in int format.

1 Convex and Coercive functions (5 points)

1. **(2 points)** Write code to check whether the functions `f1` and `f2` are convex or not in an interval of [-2,2]. Report which functions are strictly convex. Both functions are one dimensional. Write a function `isConvex(function name, interval)` that takes the mentioned arguments and returns either true or false. To call `f1, f2` you need to pass two arguments, i. SR.No(int format), ii. x value. eg: `fx = f1(Sr.No, x)`

   (a) Report how you tested the convexity.

   (b) Report which function is convex or strongly convex or not a convex along with the values of x^* and $f(x^*)$ and explain how you evaluated. Is x^* unique?

2. **(3 points)** Given `f3`, a quartic polynomial, write two functions `isCoercive(function name)` that returns whether the function is coercive or not, `FindStationaryPoints(function name)` that returns a dictionary with keys as Roots, Minima, LocalMaxima. Call `f3` like how you called `f1 or f2`.

   (a) Report how you tested the coercivity.

   (b) Report all the stationary points and roots and explain how you evaluated them.

2 Gradient Descent (15 points)

1. Given a function `f4` of the form $\frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x} + \mathbf{b}^\top\mathbf{x}$, where $\mathbf{A} \in \mathbb{R}^{5\times5}$ and $\mathbf{b}, \mathbf{x} \in \mathbb{R}^5$.Consider the initialization as $\mathbf{x_0} = [0.0, 0.0, 0.0, 0.0, 0.0]$ The function call for `f4` is `fx, gradfx = f4(Sr.No, x)`, where `x` is a NumPy array like $\mathbf{x_0}$. Write code for the following methods.

   (a) **(3 points)** Gradient Descent with fixed step size: Implement an optimization algorithm, start from $\mathbf{x_0}$ and consider $\alpha = 10^{-5}$, direction at every iteration $\mathbf{p_k} = -\nabla f(\mathbf{x}_k)$, update rule is $\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha \cdot \mathbf{p_k}$.Name your function as `ConstantGradientDescent(alpha, initialx)`.

   (b) **(4 points)** Gradient Descent with Diminishing step-size: Consider the same setup as above, but the step size $\alpha_k = {}^{\alpha_0}/_{(k+1)}$ and $\alpha_0 = 10^{-3}$. Run this algorithm for minimum of 10000 iterations, and report the values of x_T and $f(x_T)$ where T=10000. Are these matching with the previous answer, if not what might be the reason. Name your function as `DiminishingGradientDescent(InitialAlpha, initialx)`

   (c) **(4 points)** Inexact Line Search Using Wolfe Conditions: In the previous two questions we are kind of making α as deterministic. Now set the alpha in every iteration using the following two conditions, with $c_1 + c_2 = 1$ and $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$:

$$f(\mathbf{x}_k + \alpha_k\mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1\alpha_k\mathbf{p}_k^\top\nabla f(\mathbf{x}_k), \tag{1}$$

   and,

$$-\mathbf{p}_k^\top\nabla f(\mathbf{x}_k + \alpha_k\mathbf{p}_k) \leq -c_2\mathbf{p}_k^\top\nabla f(\mathbf{x}_k) \tag{2}$$

   We choose the α in every iteration as:

$$\alpha_k = 1$$

   while any of (i) or (ii) not satisfied:

$$\alpha = \gamma \cdot \alpha$$

End while

Report the values of x^* and $f(x^*)$. Name your function as `InExactLineSearch(c1, c2, gamma)`.

(d) **(4 points)** Exact Line Search: Unlike doing a search for α in every iteration, we solve a optimization problem over α like follows, $\alpha_k = \arg\min_{\alpha} f(\mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k))$.The solution of the optimization problem for the quadratic form is

$$\alpha_k = -\frac{\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k}{\mathbf{p}_k^\top A \mathbf{p}_k}$$

where $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$. Name your function as `ExactLineSearch()`.

For all the above methods report $\mathbf{x}^*$, $f(\mathbf{x}^*)$, number of iterations (T) and plot the following at each iteration:

- $||\nabla f(\mathbf{x}_k)||_2$,
- $f(\mathbf{x}_k) - f(\mathbf{x}_T)$,
- The ratio ${}^{f(\mathbf{x}_k)-f(\mathbf{x}_T)}/_{f(\mathbf{x}_{k-1})-f(\mathbf{x}_T)}$,
- $||\mathbf{x}_k - \mathbf{x}_T||_2^2$, and,
- The ratio ${}^{||\mathbf{x}_k-\mathbf{x}_T||_2^2}/_{||\mathbf{x}_{k-1}-\mathbf{x}_T||_2^2}$.

3 Perturbed Gradient Descent (10+5 points)

In this exercise, we will try to check if the intentional addition of noise to gradient descent can help us avoid saddle points. But before doing so, let us revise the necessary and sufficient conditions for identifying (local) minima of continuously differentiable functions. We will start with the first order necessary conditions which state that if a point $\boldsymbol{x}^*$ is a local minimum in an open neighbourhood of $\boldsymbol{x}^*$, then it is a stationary point, i.e., $\nabla f(\boldsymbol{x}^*) = 0$. Note, however, that the converse may not always be true.

To find if a point is indeed a minimum, we would need additional information on the Hessian of the function at that point. Supposing that $\nabla^2 f$ exists and is continuous in an open neighbourhood of $\boldsymbol{x}^*$:

- If $\boldsymbol{x}^*$ is a local minimum, then $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite.
- Conversely, if $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite at a point $\boldsymbol{x}^*$, then $\boldsymbol{x}^*$ is a strict local minimiser of f.
- Alternatively, a stationary point is a local maximiser if the Hessian is negative semidefinite, or a *saddle point* if the Hessian has a mix of non-positive and non-negative eigenvalues at that point.

Please refer to Chapter 2 of Nocedal and Wright for a deeper and more thorough exposition.

To escape saddle points, we will make use of perturbed gradient descent, whose update-equation is given by:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha_t \left(\nabla f(\boldsymbol{\theta}^{(t)}) + \boldsymbol{\zeta}^{(t)} \right),$$

where, $\alpha_t \geq 0$, and $\boldsymbol{\zeta}^{(t)}$ is an artificially-added noise term. Here, we may choose to add uncorrelated Gaussian noise to each coordinate such that: $\boldsymbol{\zeta}^{(t)} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2 \boldsymbol{I})$[1]. The step-size α_t and the noise-variance may either be constant, or may diminish at each iteration as:

$$\beta_t = \frac{\beta_0}{(t+1)}.$$

Now, consider the following function:

$$f(x, y) = e^{xy},$$

with $x, y \in \mathbb{R}$.

1. **(1 point)** Find all the stationary points of $f(x, y)$ and categorise them as minima, maxima, or saddle points.
2. **(2 points)** Analytically show that on starting gradient descent at an initial point $x = y$, one stays on the line $x = y$.
3. **(1 point)** Create a contour plot of $f(x, y)$, with $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$.
4. **(1 point)** Run gradient descent with a fixed step-size on $f(x, y)$ starting with a point such that $x = y$. Plot the function value at each iteration, and the trajectory of the points on the contour plot.
5. **(1 point)** Run gradient descent with a decreasing step-sizes on $f(x, y)$ starting with a point such that $x = y$. Plot the function value at each iteration, and the trajectory of the points on the contour plot.
6. **(1 point)** Run perturbed gradient descent with a fixed step-size and noise-variance on $f(x, y)$ starting with a point such that $x = y$. Plot the "expected" function value at each iteration, and the trajectory of the points on the contour plot.
7. **(1 point)** Run perturbed gradient descent with a fixed step-size and decreasing noise-variance on $f(x, y)$ starting with a point such that $x = y$. Plot the "expected" function value at each iteration, and the trajectory of the points on the contour plot.
8. **(1 point)** Run perturbed gradient descent with decreasing step-sizes and a fixed noise-variance on $f(x, y)$ starting with a point such that $x = y$. Plot the "expected" function value at each iteration, and the trajectory of the points on the contour plot.
9. **(1 point)** Run perturbed gradient descent with decreasing step-sizes and noise-variances on $f(x, y)$ starting with a point such that $x = y$. Plot the "expected" function value at each iteration, and the trajectory of the points on the contour plot.
10. **(Bonus question, 5 points)** Find the relation between $\mathbb{E}\left[f^{(t+1)}\right]$ and $\mathbb{E}\left[f^{(t)}\right]$.

[1] One may use the `random.multivariate_normal()` function from `numpy` to generate samples from a normal distribution.

4 Zeroth-order Optimisation (10 points)

Until now, we have been relying on gradients of functions to find their minimisers. Such methods are referred to as first-order methods because they use the first derivatives. Similarly, methods that also utilise the Hessian are called second-order methods. But, what if we do not have access to the gradients? Such situations may arise due to a variety of factors:

- if the gradients simply don't exist, i.e., the function is not differentiable (in these cases, however, one may resort to using the subgradients. But this is out of the scope of this exercise),
- or, the gradients may exist, but computing them is expensive (temporally or monetarily),
- or, as it happens very frequently, we are attempting to optimise a *black-box*.

All is, however, not lost; because there exist methods that aid in optimisation without the use of gradients. These are referred to as zeroth-order methods. In this exercise, we will learn about two such methods that work on functions $f : \mathbb{R} \to \mathbb{R}$. These methods find the minima (or maxima) of the function f in the interval $[a, b]$, with a major caveat being that the function is *unimodal* (i.e., only one minimum (or maximum) exists) in the given range. If multiple extrema are present, then these methods converge to any of them, and not necessarily the best one.

4.1 Golden Section Search

The method is so named because the search-interval reduces by a factor of $\rho = (1 - \phi)$ at each iteration, where $\phi = {}^{(1+\sqrt{5})}/_{2}$ is the golden ratio. This method uses only a single evaluation of the function at each iteration, and proceeds as follows:

1. Start with the initial search-interval $[a, b]$.
2. Define $x_1 := \rho a + (1 - \rho)b$ and $x_2 := (1 - \rho)a + \rho b$.
3. Consider the tuple $[a, x_1, x_2, b]$.
4. If $f(x_1) \leq f(x_2)$, change the tuple to $[a, x_3, x_1, x_2]$, where $x_3 := \rho a + (1 - \rho)x_2$. Else, if $f(x_2) \leq f(x_1)$, change the tuple to $[x_1, x_2, x_3, b]$, where $x_3 := \rho x_1 + (1 - \rho)b$.
5. If convergence is reached, return the final interval; or go back to step 3 with the new tuple.

4.2 Fibonacci Search

This method is similar to golden section search, with the slight modification that instead of using a constant ρ, its value is updated at each iteration t to get:

$$\rho_t = 1 - \frac{F_{T-t+1}}{F_{T-t+2}},$$

where T is the total number of iterations, and $\{F\}$ is the Fibonacci sequence with $F_{-1} = 0$ and $F_0 = 1$. Also, the final search interval after T iterations is given by ${}^{(b-a)}/_{F_{T+1}}$.

Consider the function:

$$f(x) = x(x - 1)(x - 3)(x + 2);\ x \in \mathbb{R}.$$

1. **(2 points)** Find all the extrema of $f(x)$.

2. Now, restrict yourself to the interval $[1, 3]$, with the objective of finding the minimum within this interval with a range of 10^{-4} using:

   (a) **(4 points)** Golden section search. First, report the number of iterations required to achieve the aforesaid precision. Then, implement the method and tabulate the intervals at each iteration. Also, plot $f(a_t)$, $f(b_t)$, $(b_t - a_t)$ and $(b_t - a_t)/(b_{t-1} - a_{t-1})$ at each iteration.

   (b) **(4 points)** Repeat all of the above for Fibonacci search.