E7: Introduction to Computer Programming for Scientists and Engineers

University of California, Berkeley, Fall 2020
Instructor: Shaofan Li

Final Project
Optimization Using Gradient Descent

Labs 012 and 018

Due: Dec 14, 2020

The purpose of this final project is to develop a MATLAB code to do optimization on a multivariate function by using gradient descent/ascent algorithms.

Before starting, make sure to review the final project introduction slides, which introduces essential concepts needed to complete the assignment.

Use only functions contained in the basic MATLAB installation. Only exception is the **diff** function of the Symbolic Math Toolbox. You may use this function to compute the exact derivatives of a symbolic expression.

You are not allowed to ask for debugging help, but feel free reach out to Caglar (caglar.tamur@berkeley.edu) or your lab GSIs, if you have any conceptual questions.

For the final project, Please submit all your MATLAB files with your final report and the group presentation slides.

1 Gradient Descent Algorithm

Gradient Descent is an iterative optimization algorithm used to find the local minimum of a differentiable function. It works by starting with an initial guess and moving in the opposite direction of the gradient of the function. Direction of the gradient is the direction where we have 'steepest increase' in the function, so it makes sense to follow it's opposite direction to reach a local minimum. The movement in direction of the gradient is scaled by a factor called the *learning rate* (or step size).

The **Gradient Descent** algorithm can be expressed mathematically as follows:

$$x_{n+1} = x_n - \gamma \nabla f(x_n)$$

where γ is the learning rate and ∇ is the gradient operator.
Here, x_n is the current approximation for the local minimum and x_{n+1} is the next approximation.
If we are interested in the local maximum instead, we can simply move in the direction of the gradient. This gives us **Gradient Ascent** algorithm:

$$x_{n+1} = x_n + \gamma \nabla f(x_n)$$

Notice that only difference is the plus sign in front of the learning rate.

Two-variable case
If the objective function we are optimizing is of two variables, $f(x, y)$, the algorithms can be expressed as:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} \pm \gamma \begin{bmatrix} \dfrac{\partial f}{\partial x}(x_n, y_n) \\ \dfrac{\partial f}{\partial y}(x_n, y_n) \end{bmatrix}$$

A point (x_n, y_n) in $\mathbb{R}^2$ is called a **critical point** of the function $f(x, y)$ if the gradient of the function at that point is zero, e.g. $\nabla f(x, y) = 0$.
In order to classify a critical point, we can look at the **Hessian** of the function:

$$H = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

At a critical point, we can have three cases:

1. If $det(H) > 0$ and $\dfrac{\partial^2 f}{\partial x^2} > 0 \quad \longrightarrow \quad$ Local Minimum
2. If $det(H) > 0$ and $\dfrac{\partial^2 f}{\partial x^2} < 0 \quad \longrightarrow \quad$ Local Maximum
3. If $det(H) < 0 \quad \longrightarrow \quad$ Saddle Point

This classification method is known as the *second partial derivative test*.
Note that if the $det(H) = 0$, test is inconclusive.

2 Your Task

Given the two variable, non-convex objective function:

$$f(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+y)^2-y^2}$$

Find all of the local minimum/maximum points of the function by using Gradient Descent/Ascent algorithms over the given domain:

$$D = \{(x,y)| \in \mathbb{R}^2 : -3 \leq x, y \leq 3\}$$

For each local maximum and minimum, report your initial guess (x_0, y_0), the learning rate γ you have used and the total number of steps at convergence.

3 Hints

- Visualizing the objective function will be very helpful. You can use the **surfc** and **meshgrid** functions to plot the objective function in the given domain. This may help you to pick suitable initial guesses.
- If the learning rate is too small, your algorithm may need a lot of steps before converging to a local extrema. On the other hand, if the learning rate is too big, it may simply diverge. Therefore, you have to come up with a suitable learning rate. This can be easily achieved by trial and error. Start with some initial learning rate and adjust it according to the behavior you see.
- You will need to evaluate several partial derivatives of the objective function. Do not try to do this by hand, as the expression will be rather complicated and prune to errors. You are allowed to use the **diff** function of the MATLAB's Symbolic Math Toolbox. If you don't have this toolbox installed, feel free to use any other method or tool (e.g. Wolfram). Here is an example on how to use **diff** function:

```
>> syms x y
>> f = @(x,y) x^3 - 2*y^2 + 4*x*y;
>> diff(f,x,2)
ans =
6*x
>> diff(f,x,y)
ans =
4
```