Jared Frazier | CSCI 4350 | OLA 2 | Simulated Annealing and Gradient Ascent | Due 2021/10/07

Introduction:

A function is simply a mathematical construct that receives certain information as input and produces an output based on the rules of the construct itself. For many phenomena, a function can be used to describe (or at the very least approximate) some output given some input. For example, suppose you have some function *f(x)* that takes as input *x* the tax rate of your country and outputs the revenue. Thus, we would say the revenue is a function of the tax rate. For simplicity, we assume that the function is known, that is that the rule that describes revenues in relation to tax rate is so precisely enumerated that it is said to be deterministic: if you know the tax rate, you know the revenue. Then, we might plot this function and see how we can maximize revenue, since, after all, we want the most money because money is useful!

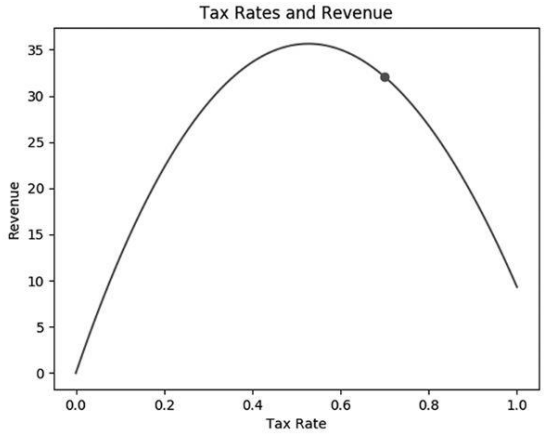

Figure 1. Arbitrary Revenue vs. Tax Rates [1].

But how do we find the tax rate that *maximizes* the revenue? Visually, it is simple to determine the maximum for Figure 1 because with only a single *x* (tax rate) and a single *y* or *f*(*x*) (revenue) we get a 2D plot (one dimension for each variable). What if we have two input variables x_0 and x_1 that we store in a list of variables called X where X = {x_0, x_1} and a single output variable that is a special function called a Gaussian Function *G*(X)? Then we have a plot where the maximum is not so obvious, see Figure 2!

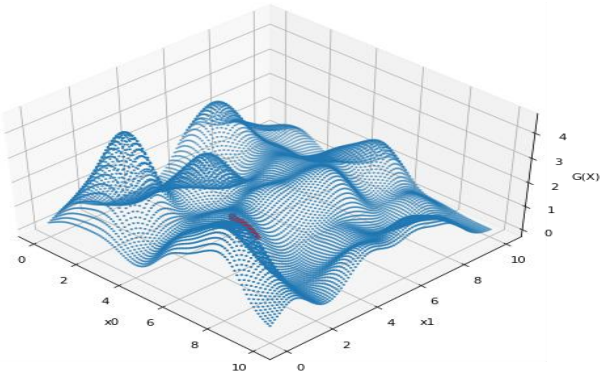

Figure 2. Plot of Gaussian function and two x-coordinates.

The Gaussian Function itself is a mixture of Gaussians Functions whose centers are at different locations in multiple dimensions. In a single dimension, this can be visualized in Figure 3.

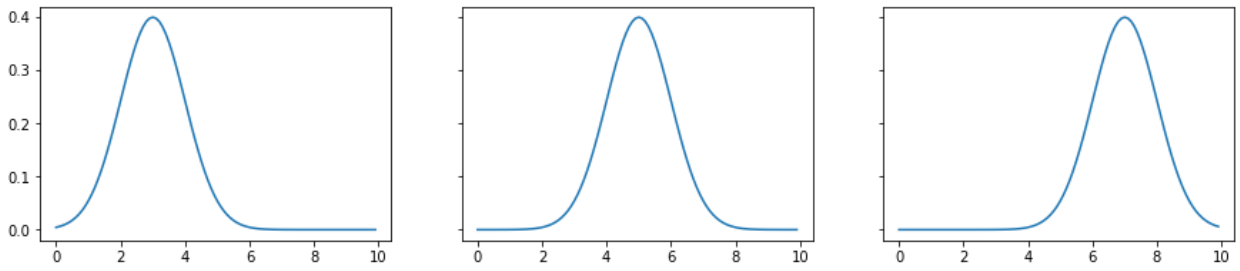

Figure 3. Gaussian functions with different centers.

If these functions are summed, then a completely new shaped function is the result, see Figure 4. Now if we were to climb up the hill (function) in Figure 4 from the left side, we might encounter a single bump, or local maximum. Likewise, climbing up the hill from the right side we would encounter another bump. In both cases if we continued hiking up the hill, we would reach the summit of the hill, also called the global maximum. Again, this problem is simple because plot is in 2D only, but if we have N-dimensions, then we will quickly be unable to visualize and determine the maximum of the function.

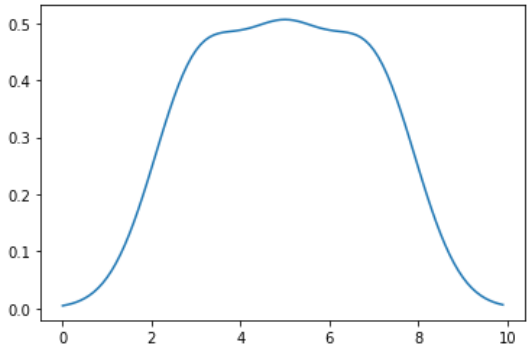

Figure 4. Sum of gaussian functions with different centers.

Methods:

Returning to Figure 2, to find the values of x_0 and x_1 such that the G(X) is maximized, two common techniques can be used: hill climbing (gradient ascent) and simulated annealing [2]–[5]. Hill climbing computes the rate of change at some location on the hill and then takes a step in a direction based on that rate of change. In other words, the location of a point in the N-dimensional hill will be updated until either the rate of change becomes very small (either in a valley or at a maximum) or some maximum number of steps is reached. This rate of change is modified by a step size which impacts how drastic the change in from the original location to the new location is. Since the rate of change is also N-dimensional, to determine whether the rate of change has become very small, the absolute value or each rate of change for the i^{th} dimension is computed and then the mean of the new rate of change vector is computed. If this value is less

than some threshold, the algorithm stops since the location in the N-dimensional space is either a summit or a valley. The pseudocode for hill climbing is shown below.

```
ascend = True
iteration = 0
while ascend and iteration < max_num_iterations:
        rate_of_change = step_size * derivative_of_sum_of_gaussian_wrt_location(location)
        location = rate_of_change + location
        position_in_gaussian_space = gaussian_as_a_function_of_location(location)
        if rate_of_change < rate_of_change_threshold:
                ascend = False
        iteration += 1
```

Simulated annealing follows a similar idea, except that rather than using rates of change to inform the next step, the next step is picked randomly and then there is some probability that the step will be taken or not taken based on where the step is in relation to the N-dimensional hill, in this case G(X). The randomness of such choices is influenced the "temperature" of an annealing schedule where higher temperatures lead to more random choices while lower temperatures lead to less random and broad choices. A linear, decreasing annealing schedule was selected due to simplicity of implementation, see equation 1.

```
def schedule: return 1 - (current_iteration+1)/(max_iteration)
```
(1)

The annealing schedule is shown in Figure 5.

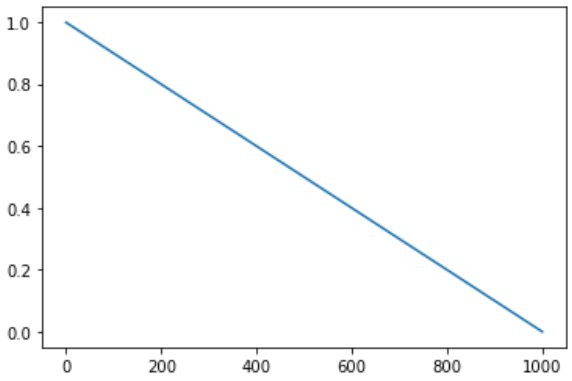

Figure 5. Linear decreasing annealing schedule.

A candidate location will be chosen if its position in the Gaussian N-dimensional hill is higher than the current position, this is called the energy. Finally, the candidate location will be selected based on the Metropolis criterion, see equation 2.

```
def metropolis_criterion: return exp((energy2-energy1)/temp)
```
(2)

The pseudocode for simulated annealing is shown below.

```
iteration = 0
while iteration < max_num_iterations:
        temp = schedule(iteration, max_num_iterations)
        candidate_location = location + randomly_choose(-0.05, 0.05)
        candidate_energy = gaussian_as_a_function_of_location(candidate_location)
        location_energy = gaussian_as_a_function_of_location(location)
        if candidate_energy > loc_energy:
                location = candidate_location
        elif metropolis_criterion(candidate_energy, location_energy, temp)>= random(0, 1):
                location = candidate_location
        iteration += 1
```

Results and Discussion:

The algorithm that performs the best is the one that reaches a higher point in the N-dimensional Gaussian space. This is because finding the maximum of such a space is the goal, and both optimization algorithms attempt to do this. The dimensions of the input can be 1, 2, 3, or 5 while the number of centers in the Gaussian space can be 5, 10, 50, or 100. Figure 6 shows the different combinations of arguments to the optimization algorithms in grey, while the white is the number of times in which simulated annealing lead to a location that corresponded to a higher point in the Gaussian space. Since there is no early stopping criteria for the simulated annealing approach, the duration of the algorithm is solely dependent on the maximum number of iterations (default at 100,000).

Dims, Centers	1, 5	1, 10	1, 50	1, 100
	24	29	10	23
Dims Centers	2, 5	2, 10	2, 50	2, 100
	7	20	24	25
Dims, Centers	3, 5	3, 10	3, 50	3, 100
	10	3	9	16
Dims, Centers	5, 5	5, 10	5, 50	5, 100
	25	27	21	11

Figure 6. Number of times simulated annealing performed better than hill climbing for different arguments. Values in white are out of a total of 100 seeds.

Surprisingly, hill climbing performed better than simulated annealing in all cases. This is particularly surprising since hill climbing is expected to get stuck in local minima due to the early stopping criteria being based solely on a small rate of change. Conversely, simulated annealing is expected to escape local minima due to random movements imposed by the annealing schedule. However, the annealing schedule I selected is likely the reason the hill climbing algorithm performs better in all cases. The annealing schedule is specific to the space that is being optimized, and while the decreasing, linear schedule I implemented was simple, it fails to escape local minima in the expected manner. With the current schedule, the simulated annealing algorithm is equal parts random walk (high temperature), stochastic gradient descent (medium temperature), and gradient descent (low temperature) [6].

The topography, that is the “hills” in the N-dimensional space, does not appear to have a consistent effect on the performance of the simulated annealing algorithm. I would expect that as the number of hills (centers) increases, the hill climbing algorithm would become consistently worse; however, the simulated annealing algorithm sometimes performs better in higher dimensions with more centers and sometimes does not. This indicates that the annealing schedule’s imposition of linearly decreasing stochasticity is substantially worse at escaping local minima than hill climbing.

Conclusion:

Two optimization algorithms were implemented for the traversal of a multidimensional climber through a Gaussian function with many centers. The annealing schedule led to significantly worse performance for the simulated annealing algorithm compared to the hill climbing algorithm. This project illustrated how parameters such as the dimensionality of data and design choices of algorithms dictate the performance of optimization algorithms.

References:

[1] B. Tuckfield, *Dive into algorithms : a pythonic adventure for the intrepid beginner*. No Starch Press, 2021.

[2] S. Russel and P. Norvig, *Artificial intelligence: A modern approach*, 3rd ed. Larsen & Keller Education, 2017.

[3] W. H. Press and S. A. Teukolsky, “Simulated Annealing Optimization over Continuous Spaces,” *Computers in Physics*, vol. 5, no. 4, p. 426, 1991, doi: 10.1063/1.4823002.

[4] Wikipedia contributors, “Simulated annealing — Wikipedia, The Free Encyclopedia.” 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Simulated_annealing&oldid=1044570865

[5] Wikipedia contributors, “Hill climbing — Wikipedia, The Free Encyclopedia.” 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hill_climbing&oldid=1021273740

[6] Y. Nourani and B. Andresen, “A comparison of simulated annealing cooling strategies,” 1998. [Online]. Available: http://iopscience.iop.org/0305-4470/31/41/011