Matthew Wells

CSCI 4350 OLA 2

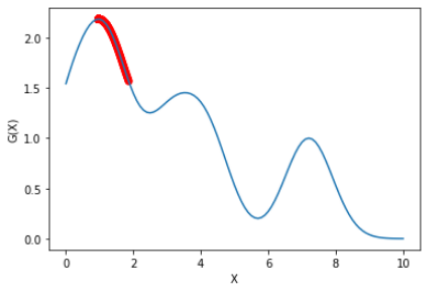
October 8, 2021

For our second lab assignment, our programs evaluated the maximum of the Sum of Gaussians (SoG) function using multi-dimensional vectors. We used 2 programs for this: one that used greedy local search, and one that used simulated annealing.

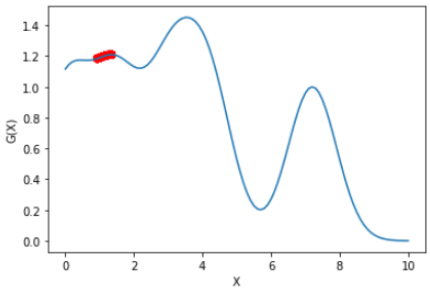
The Sum of Gaussians function is one that allows for data clustering and lets us control peaks in our programs. As a result, there are 3 main inputs for each program: the randomization seed for points, the number of X dimensions in the graph, and the number of Gaussians. The programs will take these numbers and generate data with peaks and valleys to solve.

Both programs start by randomly placing X number of dimensions of points on the problem and then attempting to find the largest maximum in the problem. Greedy local is only capable of finding a local maximum and hoping that it's the best fit (illustrated below with their respective numbers), whereas the simulated annealing function will make bad decisions on purpose a percentage of the time in order to more effectively explore the space, lowering the amount of bad decisions and "cooling off" as time goes on.

The numbers below are the arguments for the search. In order from left to right, it is the random seeding, the number of dimensions, and the number of Gaussians. As illustrated in the one dimensional tests, the greedy local search accurately finds the nearest local maximum but has no sense if what it's finding is the global maximum.

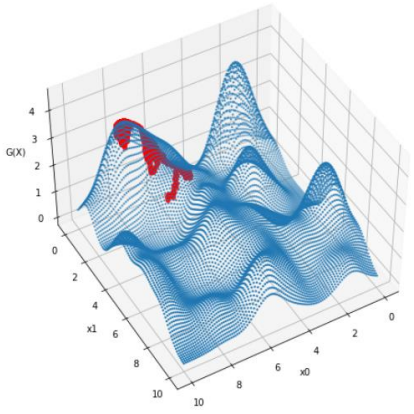


1-1-6

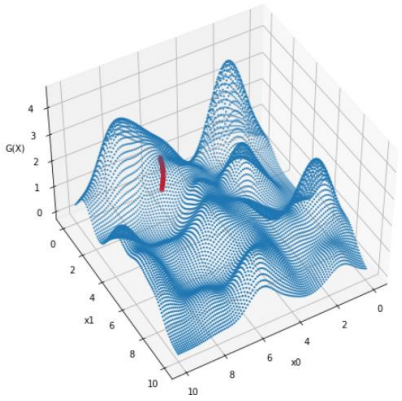


1-1-5

In this lab, my experimentation with simulated annealing happened very late into the work time, so my experimentation with the algorithm is fairly small. I began with a simple regression curve of 1000/i+1, which allows the temperature T to regress over time and allow T to hone in. This allowed a little bit of exploration across the space, but wasn't a large enough variation to explore the entire function. In the example on the next page, you can see how Simulated Annealing achieved a better result and explored more of the function overall, but stayed very close to the starting peak. The G(x) in this simulated annealing was 3.42317955, which was significantly better than the Greedy Local G(x) of 2.13401831.

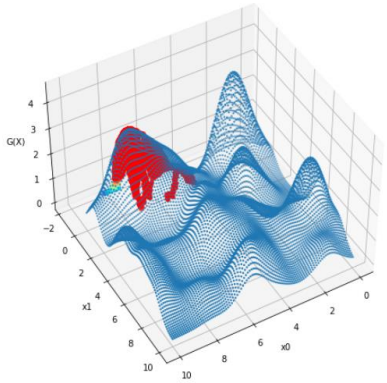


0-2-50: Simulated Annealing



0-2-50: Greedy Local

Next, I was stumbled upon an equation on the simulated annealing Wikipedia page that I tried in its base form to see how it performed: T = 1-((i+1)/100000). This function gave the solution a lot more coverage, but the final value that was settled on was worse at G(x) = 2.18972365.



0-2-50: New Simulated Annealing

In addition, due to time constraints, I was not able to run the expected tests and was only able to run a couple of refinement tests like this. If I get any closer on Saturday, I may add that data later. If I'd had the data done, I would have compared how many times SA beat the Greedy algorithm, and based on the progess I've made on SA so far, I'd imagine the SA function would overall be more successful than the Greedy function, with the exception of situations with very low Gaussian values (1 or 2).

Works Cited

"Simulated Annealing." *Wikipedia*, Wikimedia Foundation, 15 Sept. 2021, https://en.wikipedia.org/wiki/Simulated_annealing.