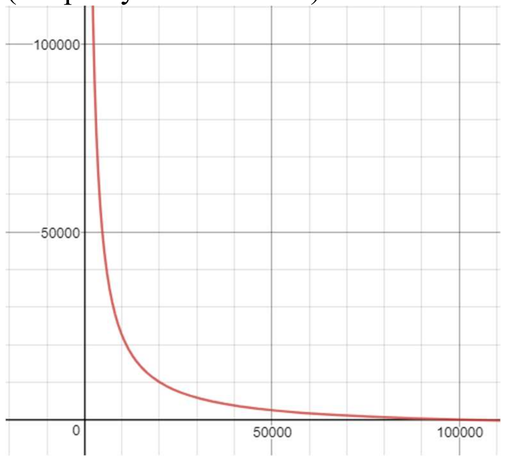
Report On Optimization Project:

Project Description and Goal:

- Project description:
  - Students are given a program that can calculate the Sum of Gaussian of different dimension the given program is also able to provided student the derivative at that point.
  - Students are asked to create programs that try to find the maximum value of the Sum of Gaussians function using two algorithm: simulated annealing and greedy local search
- Goal:
  - Understand the differences between simulated annealing algorithm and greedy local search.

Algorithm and implementation:

- Greedy local search:
  - Greedy local search algorithm will start at a random location and then simply try to follow the gradient/derivative of a function. The program will keep doing so until it reach the top. In this case it means that the algorithm will follow the gradient to a new position until the sum of Gaussian at the new position has less than 1e-8 differences
  - The only tricky part here is how much should the program move to the next position? Moving in too much distance at a time will might result in the program arrived at different local maxima. In this case step size is used as $step = 0.01 * derivative$
- Simulated annealing:
  - Greedy local search is not the perfect algorithm because it will stuck at a local maxima. Thus we have to try to find a way to get out of the current local maxima to attempt to find the global maxima. Simulated annealing achieve this by allowing bad moves so that it can try to reach other local maxima. Simulated annealing let the program explore at the start and then later on follow the path that it think is correct.
  - In this specific implementation:
    - The temperature (A factor to decide if the program will allow bad move) is determine by this equation: $T = \frac{250000000}{x+1} - 2500$
    - (Graph by Desmos.com)



- The program will loop for 100000 iterations. Each iteration will generate a new location that is -0.01 to 0.01 away from the current point in all direction. The program will check if the new location is superior.
    - If it truly is superior then the program will take it.
    - If the new location if inferior, it would have a certain percentage to take the new location. The percentage is calculate by this equation:

      $$prob = e^{\frac{Gaus_{new} - Gauss_{old}}{T}}$$

      If T is really high, the probability is close to one which means, at the start, the program will explore. While, as T gets lower and converging to 0, the probability of program making bad move will also decrease, thus the program will more likely to find the local maxima it is at.

Results and Conclusion:

This is the result that I get from running the program for all combinations of Dimension=1,2,3,4,5 and Number of Gaussian= 5,10,50,100,500,1000. And I did this tasked from seed 1 to 100 which means the sample size of my data is 3000.

Dimensions	Which Algorithm Performed?	Number of occurrence
1	Greedy	577
1	Tie	0
1	SA	23
2	Greedy	563
2	Tie	7
2	SA	30
3	Greedy	543
3	Tie	37
3	SA	20
4	Greedy	464
4	Tie	98
4	SA	38
5	Greedy	424
5	Tie	159
5	SA	17

N of Gaussian	Which Algorithm Performed?	Number of occurrence
5	Greedy	282
5	Tie	174
5	SA	44
10	Greedy	353
10	Tie	104
10	SA	43
50	Greedy	458
50	Tie	18
50	SA	24
100	Greedy	482
100	Tie	5
100	SA	13
500	Greedy	496
500	Tie	0
500	SA	4
1000	Greedy	500

1000	Tie	0
1000	SA	0

The result is surprising as greedy algorithm outperformed the simulated annealing by quite a great margin. This is not supposed to happen as simulated annealing is considered a much better algorithm as it does not tunnel vision on the local maxima it is currently at. Thus there are some implementation problem. The only think I can really think of is the fact that my max temperature is way too high as it is more than 100000 in the first loop iteration. Although the temperature schedule I used does converge to zero, T near the end is still too high. For example: at iteration 99000, T is around 25 and assuming that the differences between Gaussian is -10, the program has a 67% chance of accepting this bad move. This means that the program spend more than 99% of its life cycle acting like a random walk program. Thus it is reasonable that the greedy algorithm will outperform the random walk program. If I am going to improve the project, I will choose a much lower maximum temperature. Choosing to work with a hyperbolic function for the temperature, I think, is correct as hyperbolic function have a nice property that it approach 0 really quickly. I wish I know more about what is the expected range of the maximum between two Gaussian. This knowledge would help me optimized the tempure function even further.