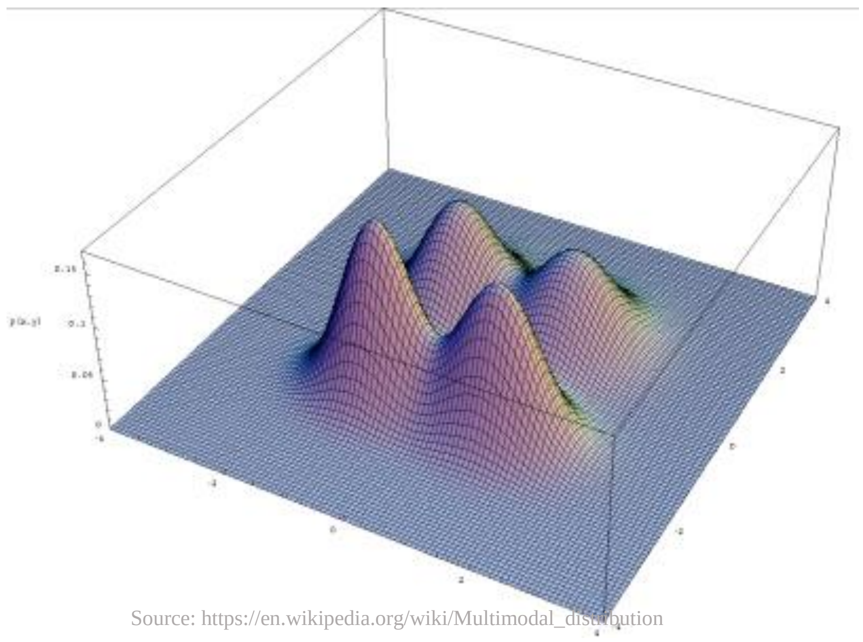
Brandon Goode


Source: https://en.wikipedia.org/wiki/Multimodal_distribution

The Sum of Gaussians function is a function that at its core consists of a space with a "hill", or center, where the function valley is the greatest at its peak. However, it can have as many dimensions and centers as you specify. The SoG function is a function that we have turned to a specific set of algorithms called local search algorithms to help and solve. Local search algorithms are algorithms that take a search space and try to find the most optimal point in the space i.e. the global maximum. One of the biggest problems that comes with local search algorithms is finding a local maximum(a high point relative to the surrounding area) instead of a global maximum. The nature of local search problems is finding the most optimal solution, and therefore trying to avoid local maximums and instead finding the global maximum of the space is imperative.

Two local search algorithms performed in this experiment are the greedy(hill-climb) algorithm, and the simulated annealing algorithm. The greedy algorithm is also known as the hill-climb algorithm due to its implementation. The hill-climb algorithm involves starting at a point in the search space, looking at the function value of the left and right side of the current point, and then choosing the move that increases the function value by the most. In practice, this ends up looking like climbing a hill by gradually making better and better moves until reaching the top of a hill.

The Simulated Annealing algorithm gets its name from the field of metallurgy. The idea for SA is to add "heat" into the system, causing the moves to become random (much like adding heat to objects in real life), and to gradually lower the temperature causing the algorithm to make better and better moves until settling on the global maximum of the function. The SA algorithm differs from the greedy algorithm because while the greedy algorithm is always trying to make the best move, the SA algorithm starts off by allowing it to make some bad moves first and then gradually starts making only good decisions.

Brandon Goode

In my implementation of the greedy algorithm, I initialized a numpy array with a random value for each dimension specified and then took the SoG function value at that random point. From that point, I added and subtracted the derivative multiplied by 0.01 from each point and then chose between those two points whichever point made the function increase most significantly. I ran this in a loop until either the function had surpassed 10,000 iterations or the function value no longer increased by a value of 1e-8 or less.

In my implementation of the Simulated Annealing algorithm, I also initialized a numpy array with a random value for each dimension and took the SoG function value from the point. The temperature(T) is then set to 1e-40. After that, I started a loop in which each point gets a random number in the range -0.1 – 0.1 added to it. That point is then checked and if its SoG function value(gY) is greater than the SoG function value of the current point(gX) then it makes the new point the current point, if the point does not have a greater function value than the current point then a probability is generated whether or not the new point should be taken or not. The probability value is equal to e^(gY-gX)/T. If gY is not greater than gX and gY does not get picked from the probability distribution, then a counter marking the fact that the function remains unchanged is incremented by 1. The counter is reset when a change happens, but if no changes occur in 100 iterations the loop stops. After each iteration of the loop the temperature is reduced by 1e-45. The loop continues for 10,000 iterations or if the function hasn't changed for 100 times in a row.

I ran experiments with each algorithm testing them with D, the number of dimensions, equal to 1, 2, 3, and 5, and with each dimension I tested them with N, the number of centers, equal to 10, 50, 100, and 1000. I ran each scenario 100 times with 100 different unique starting states(with the exception of D3-N100, D3-N1000, D5-N50, D5-N100, and D5 –N1000 in which the computational time was massive, and I ran them 50 times with 50 unique states).

With one dimension and 10 centers the results were relatively even. SA performed slightly better than Greedy and there were 2 ties, but they were nearly identical in performance. With such a small space the, it makes sense that the results would be similar.

1 Dimension, 10 centers		
Greedy	SA	Tie
46	52	2

Brandon Goode

1 Dimension, 50 centers		
Greedy	SA	Tie
69	29	2

1 Dimension, 100 centers		
Greedy	SA	Tie
70	24	6

1 Dimension, 1000 centers		
Greedy	SA	Tie
94	2	4

Starting with one dimension and 50 centers the number of wins for the Greedy algorithm increased significantly going from 46 wins in the previous test up to 69 wins in this test. The Greedy algorithm seemed to perform better and better as the number of centers increased in the first dimension going up to 70 with 100 centers and going up even more steeply to 94 wins with 1000 centers. I think the reason behind these results was the stopping point that I had set for my SA algorithm. I think while Greedy was continuously improving, my SA algorithm didn't quite have the same amount of time to slowly optimize and therefore ending up having an ever-so-slightly lower performance than the Greedy algorithm.

2 Dimension, 10 centers		
Greedy	SA	Tie
73	27	0

2 Dimension, 50 centers		
Greedy	SA	Tie
89	11	0

2 Dimension, 100 centers		
Greedy	SA	Tie
83	16	1

2 Dimension, 1000 centers		
Greedy	SA	Tie
97	3	0

Similarly to the tests in the first dimension, the tests for the second dimension were overwhelming in the Greedy algorithms favor. Once again, I believe that the reason the Greedy algorithm was able to pull out slightly more optimal solutions was simply due to me cutting the process short by prematurely stopping the SA algorithm before it was able to fully optimize the function.

Brandon Goode

3 Dimension, 10 centers		
Greedy	SA	Tie
49	50	1

3 Dimension, 50 centers		
Greedy	SA	Tie
87	13	0

3 Dimension, 100 centers		
Greedy	SA	Tie
45	5	0

3 Dimension, 1000 centers		
Greedy	SA	Tie
49	1	0

In the third dimension, the SA algorithm managed to nearly perfectly tie in regard to the number of wins versus the Greedy algorithm with 10 centers, but when the number of centers increased the Greedy algorithm once again outperformed the SA algorithm much like in the previous tests.

5 Dimension, 10 centers		
Greedy	SA	Tie
5	95	0

5 Dimension, 50 centers		
Greedy	SA	Tie
10	40	0

5 Dimension, 100 centers		
Greedy	SA	Tie
17	33	0

5 Dimension, 1000 centers		
Greedy	SA	Tie
47	3	0

The fifth dimension is when the Greedy algorithm started to underperform. The SA algorithm significantly outperformed the Greedy algorithm with 10 centers, 50 centers, and 100 centers. My hypothesis for this situation would be due to the space becoming so vast that the Greedy algorithm can't change fast enough to get out of the flat regions that entail the massive space. The tests prove my point further by showing the Greedy algorithm doing better and better as more centers get added and the space becomes more dense. Once there are 1000 centers in the space, the Greedy algorithm almost always outperformed the SA algorithm.