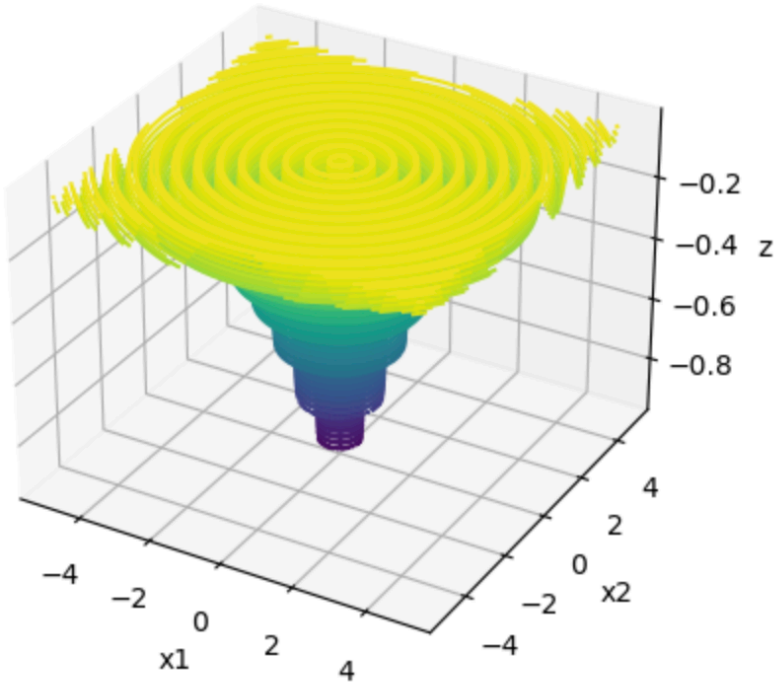
CSE 257 HW2

Q1

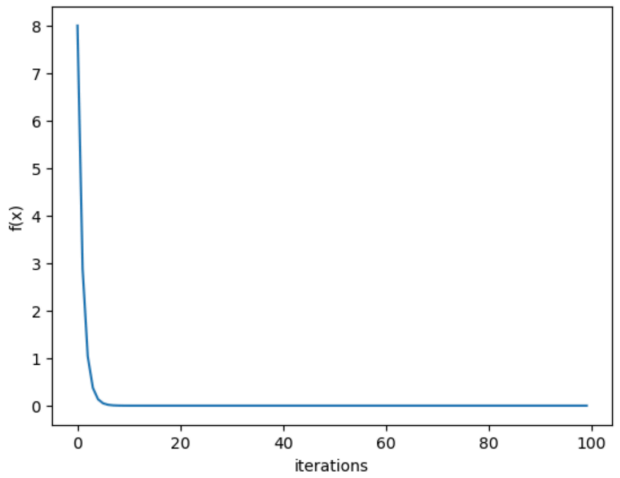
$$E_x \sim q(x)\left[\frac{p(x)}{q(x)}f(x)\right] = \int \frac{p(x)}{q(x)}f(x)\cdot q(x) = \int p(x)f(x) = E_x \sim p(x)[f(x)]$$

Q2

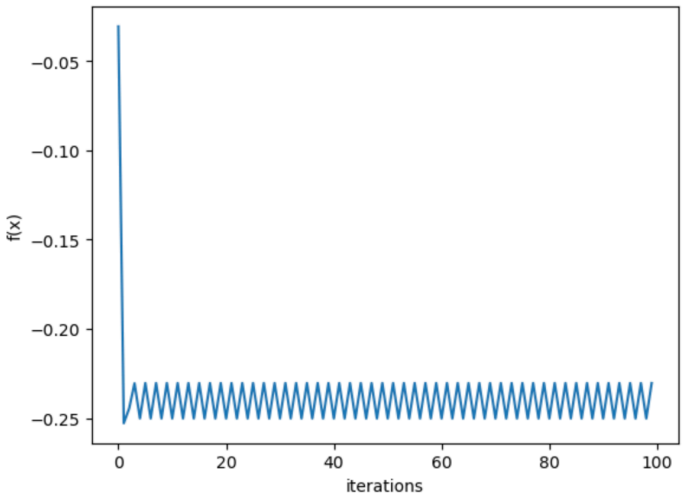
2.1 plot the 3D the drop-wave function f2



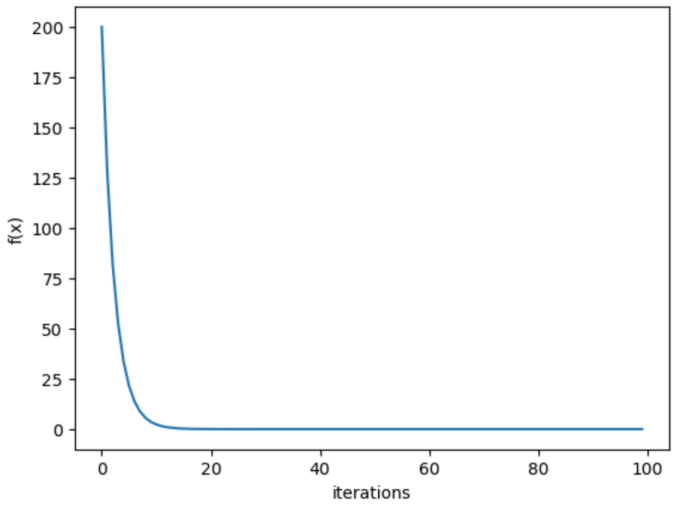
2.2 plot 1 - gradient descent for f1, alpha = 0.1



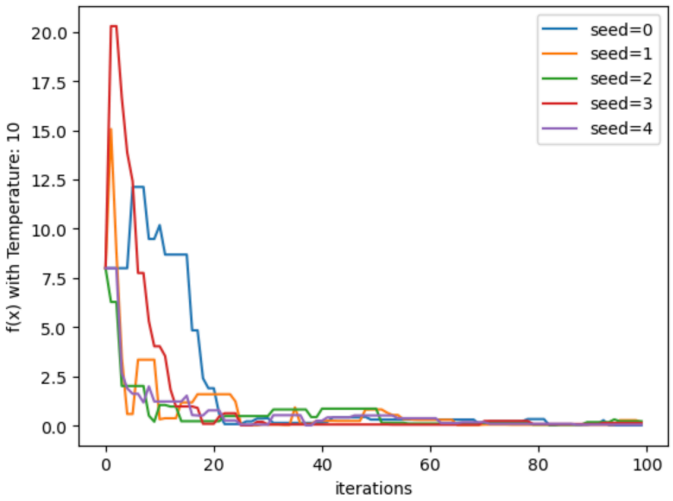
2.2 plot 2 - gradient descent for f2, alpha = 0.1



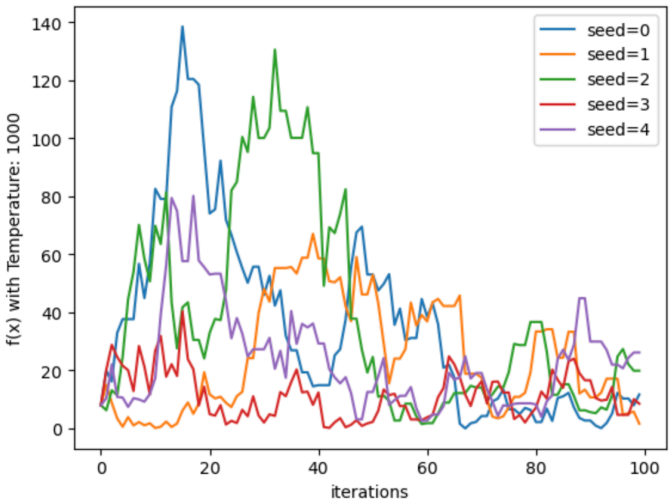
2.2 plot 3 - gradient descent for f3, alpha = 0.1



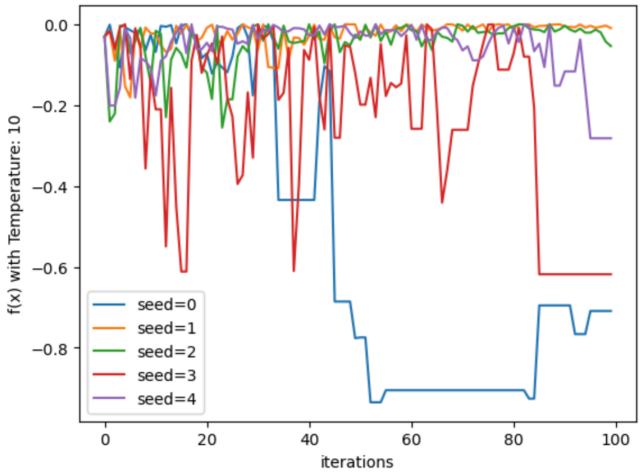
2.3 plot 1 - Simulated Annealing with T = T/k T= 10 for f1



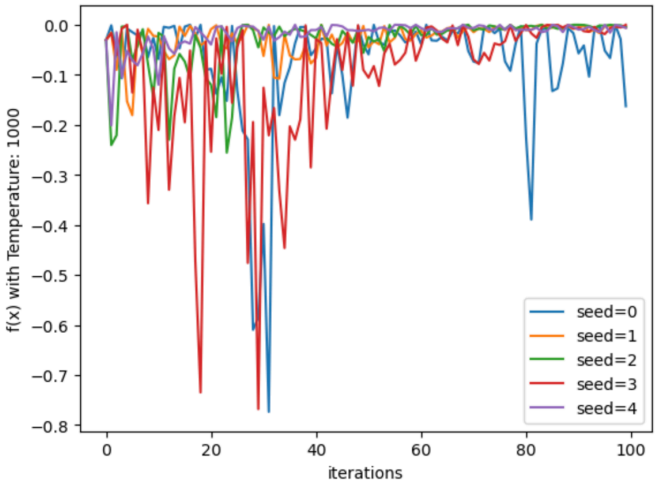
2.3 plot 2 - Simulated Annealing with T = T/k T=1000 for f1



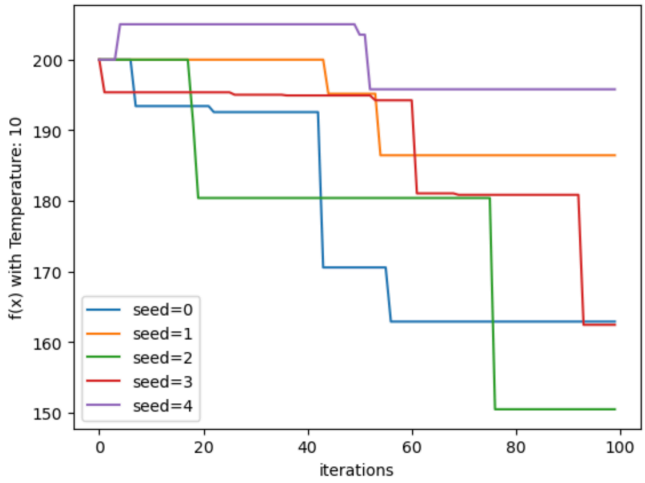
2.3 plot 3 - Simulated Annealing with T = T/k T=10 for f2



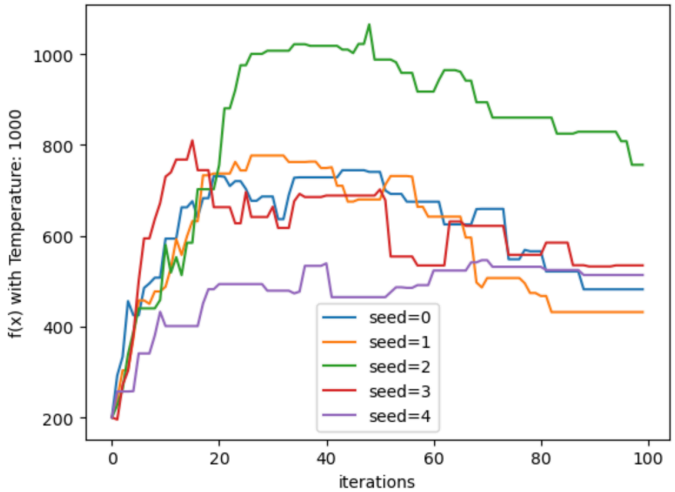
2.3 plot 4 - Simulated Annealing with T = T/k T=1000 for f2



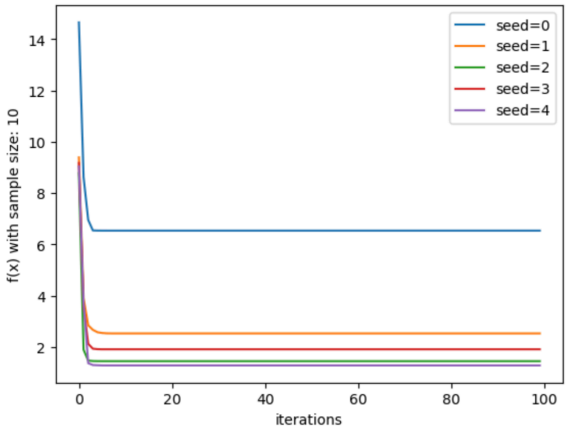
2.3 plot 5 - Simulated Annealing with T = T/k T=10 for f3



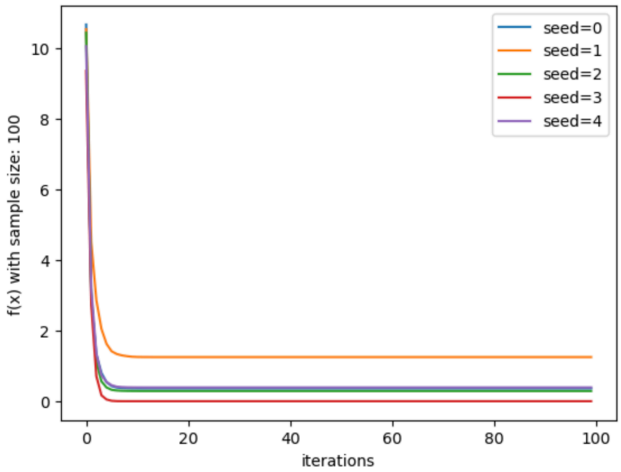
2.3 plot 6 - Simulated Annealing with T = T/k T=1000 for f3



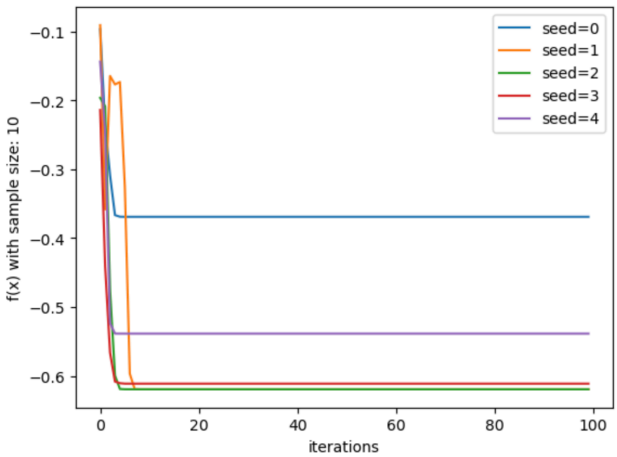
2.4 plot 1 - Cross-Entropy with k=10 for f1



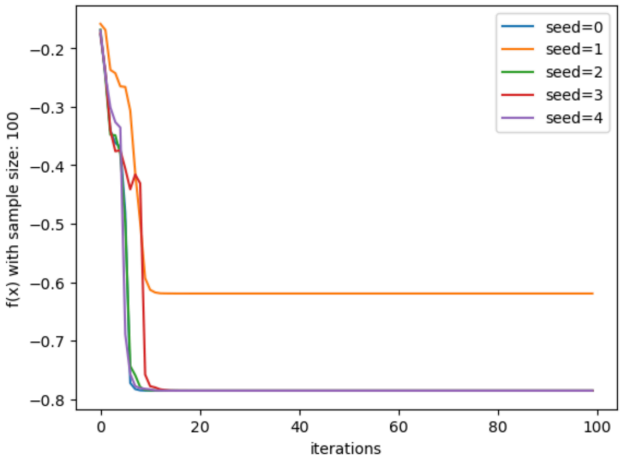
2.4 plot 2 - Cross-Entropy with k=100 for f1



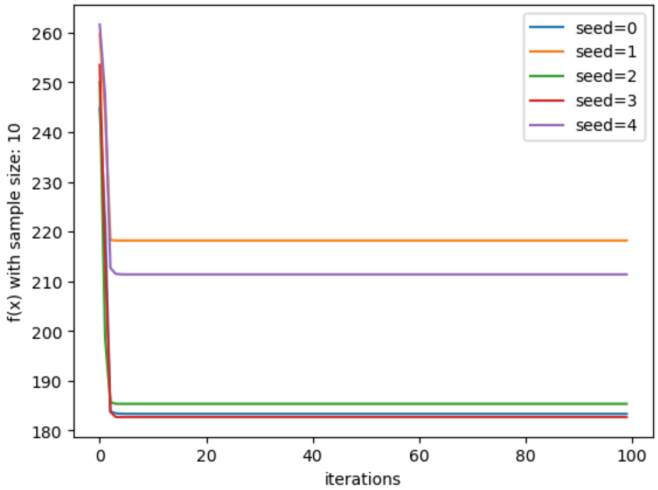
2.4 plot 3 - Cross-Entropy with k=10 for f2



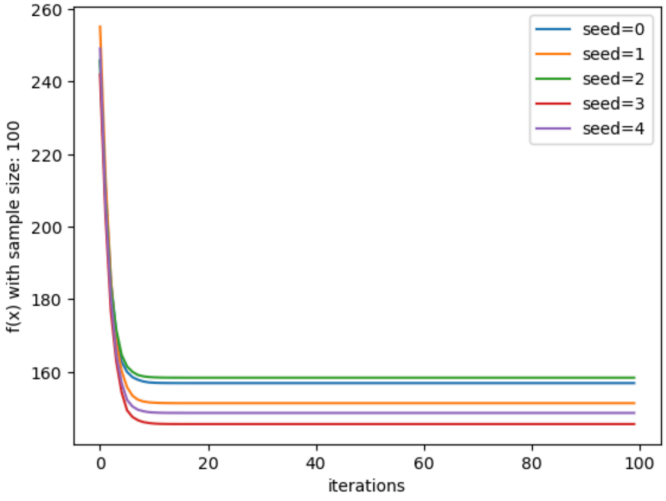
2.4 plot 4 - Cross-Entropy with k=100 for f2



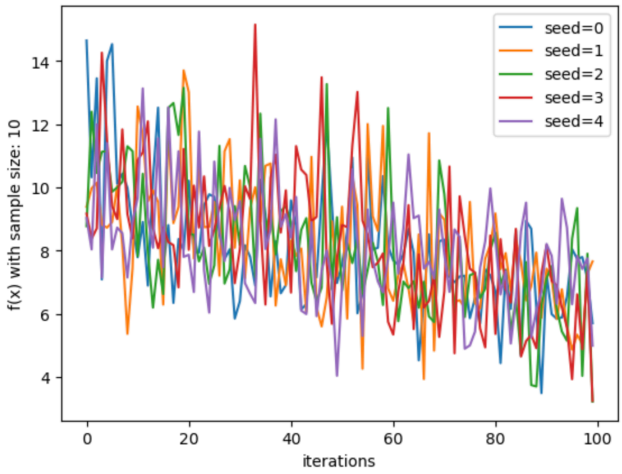
2.4 plot 5 - Cross-Entropy with k=10 for f3



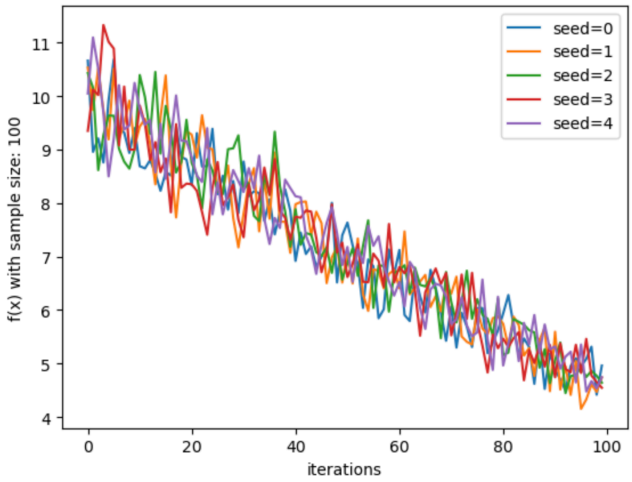
2.4 plot 6 - Cross-Entropy with k=100 for f3



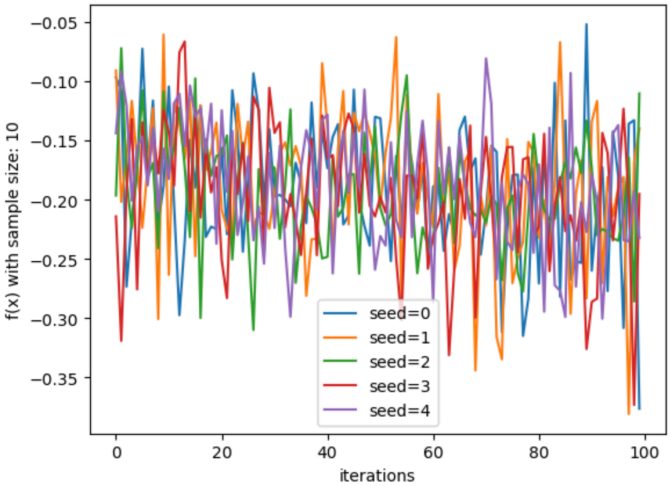
2.5 plot 1 - Search Gradient with k=10 for f1



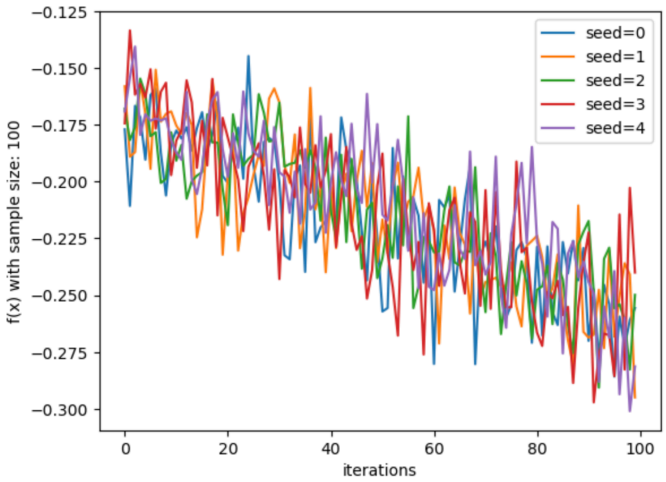
2.5 plot 2 - Search Gradient with k=100 for f1



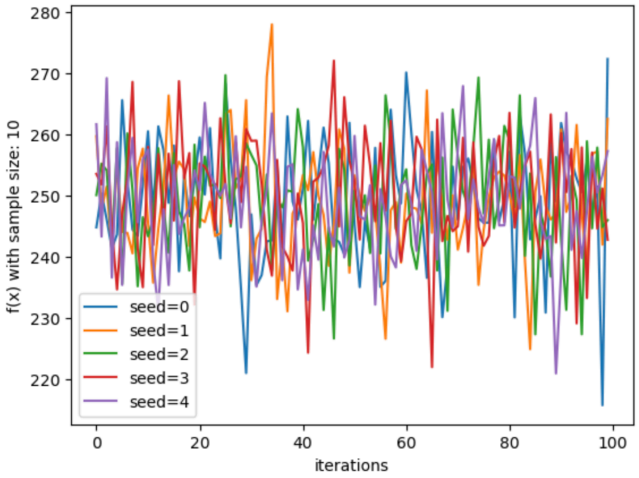
2.5 plot 3 - Search Gradient with k=10 for f2



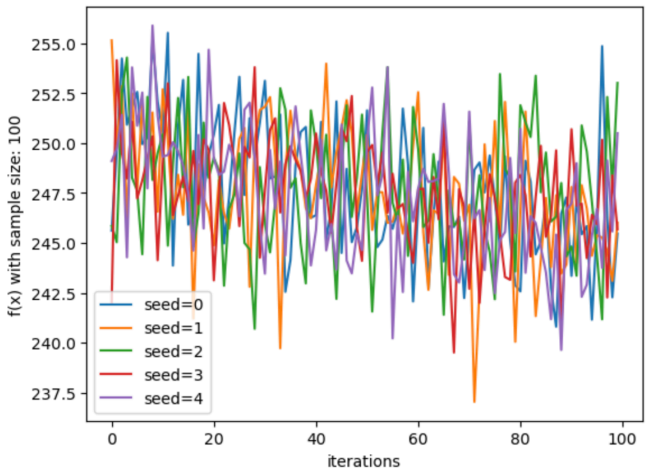
2.5 plot 4 - Search Gradient with k=100 for f2



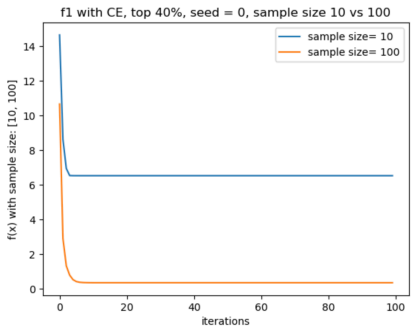
2.5 plot 5 - Search Gradient with k=10 for f3

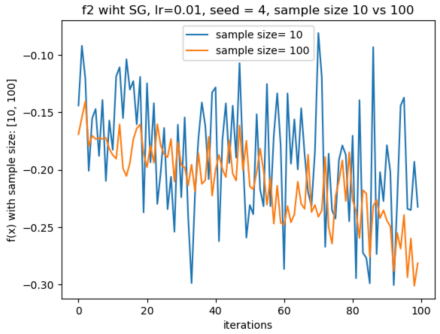


2.5 plot 2 - Search Gradient with k=100 for f3

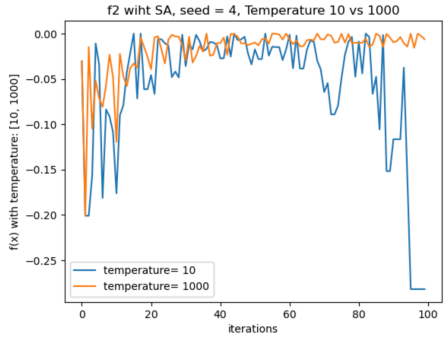


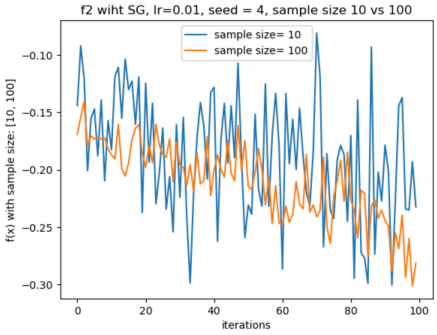
2.6



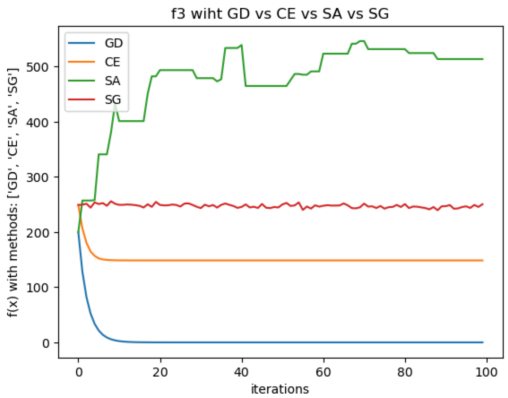


1. Sample size observation: The size of the sample has a big impact on the convergence rate for CE and SG . For CE, a larger sample size is more likely to find the global minimum compared to a smaller sample size for all functions. For SG, functions 1 and 2 have a faster descent when using a larger sample size than a smaller one. However, with f3 using SG, there was not much difference between using a larger or smaller sample size due to its high dimensionality.





2. Stability/Convergence observation: CE seems more stable than SA and SG, as it was able to converge in a few steps for all functions. The stability of SA depends on the temperature, so with higher temperatures, there was less stability at first, but after a few iterations, it became more stable due to the decrease in temperature value. SG appears to be the most unstable of the algorithms. Although it showed a general descent trend for some functions and sample sizes, it did not seem to reach absolute convergence within 100 iterations.



3. Dimension related observation: When the dimension of the function is large, it appears that gradient descent is a more effective method than sampling. With gradient descent, f3 can reach a value close to zero after 100 iterations, but the sampling methods like SA, CE, either produced high values that were not the global minimum, or they did not converge at all like SG

Q3

Proof by induction:

Base case:
At the start of the iteration i=1:
Frontier: {Start node }
Explored set: Empty

At the end of the iteration i=1:
Frontier: {All neighbors of the start node}
Explored set: {Start node }

[Separation property] Since all of the neighbors of the start node are in the frontier set, **all acyclic paths from the start node to any other node in the graph must pass through at least one node in the frontier set**
Since the start node is the only node in the explored set, means **during this case all acyclic paths from any state in the explored set to any state in the unexplored set must pass through some state in the frontier set**.
If the current state has no neighbor nodes, since there are no paths from the current state to any nodes in the unexplored set, the separation property still holds.

Given i = n-1, prove the i = n inductive step:

At start of the iteration, i = n-1:
1. Frontier and explored set are nonempty, current state at frontier
2. Current state is not the goal state

At the end of the iteration i=n-1
1. The current n-1 state is moved to the explored set
2. All of its neighbors in the unexplored set are moved to the frontier set

The nth case is proved as follow: Since we have assumed the n-1 case with induction, we only have to prove that the current state we are looking at during this iteration satisfies the separability property, which is, any acyclic path from the current state we have just moved to the explored set to any state in the unexplored set must pass through some state in the frontier set.

By the definition of the algorithm, at the end of the iteration:
1.If a neighbor state to the current state is in the unexplored set, then it is moved into the frontier set.

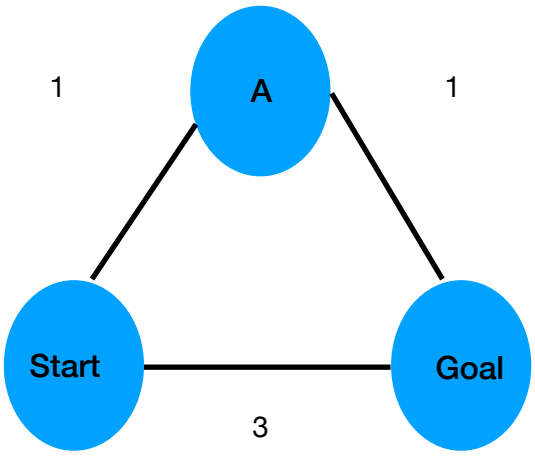
2. If a neighbor state is in the frontier set, then it will stay in the frontier set; Likewise, If in the explored set, then it will stay in the explored set.

This means none of the neighbor nodes of current state will be in the unexplored set.
If a neighbor state is in the explored set, then through our induction assumption we know that that state must satisfy the separation property. If a neighbor state is in the frontier set, then it follows that all paths from the current state to any child node of the neighbor state must pass through the frontier set. If the current state has no neighbor nodes, no paths from the current state to any nodes in the unexplored set and therefore separation property.

If there exists a path from the current state that did not satisfy the separation property, this would mean no node along this path is in the frontier set which contradicts the definition of the algorithm, since the neighbor nodes of the current state are added to the frontier set. This shows that the current state satisfies the separation property and so the nth case follows from the n-1 case.

Therefore by induction, in Dijkstra's algorithm, any acyclic path from any state in the explored set to any state in the unexplored set has to pass through some state in the frontier set.

Q4



Let h(A) = 1 and h($goal$) = 0.

1.This heuristic is consistent: h(A) ≤ c(A, $goal$) + h($goal$) 1≤1

2.The algorithm is able to find the optimal path given this heuristic. At the beginning of each iteration:

Iteration	Current state	Frontier	Explored
1	Start	{A, Goal}	{}
2	A	{Goal}	{Start}
3	Goal	{}	{Start, A}

At the start node, the algorithm considers the two paths to the goal and selects the min path: min [$c(start, goal)$ + h($goal$), $c(start, A)$ + h(A)] = min [3, 2]. So, the algorithm first explores node A with a total cost of 2 to reach it. Then, it explores the goal state through the path from node A since it has a lower cost (2) than taking the path from the start to the goal node (3). After the goal is being expanded, it returns the solution: start to A to goal is the optimal path.

When use 3h(A)

1. The heuristic becomes inconsistent: 3h(A) > $c(A, goal)$ + h($goal$) since 3>1

2. Using 3h(A) misleads the search, At the beginning of each iteration:

Iteration	Current state	Frontier	Explored
1	Start	{A, Goal}	{}
2	Goal	{A}	{Start}

At the start node, the algorithm considers the two paths to the goal and selects the min path: min [$c(start, goal)$ + h($goal$), $c(start, A)$ + h(A)] = min [3, 4]. So, the algorithm first explores the goal state with a total cost of 3 to reach it. After the goal is being expanded, it returns the solution: start to A to goal is the optimal path, which is incorrect due to the heuristic 3h(A).

Q5

Instead of minimax, a non zero sum game can be evenmax, with the following objectives:

1.Even player tries to get an even number and max player tries to get the max number.

2.The end states are positive integers with a uniformed distribution in odd/even number, and the value of the end state odd numbers has the same distribution as the value of the end state even numbers.

Since the max player's game objective is not strictly adversarial against the even player, they can both win the game, so it fits the requirement of non zero sum game.

As a result, the best strategy for max player is:

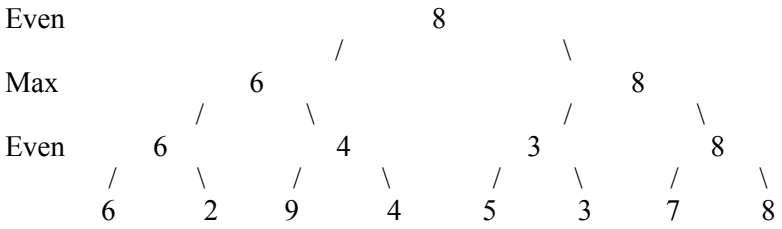
1. When both number of choice are odd/both number of choice are even, choose the max of the two
2. Else, choose the even number even if the even number of choice is smaller than the odd number of choice

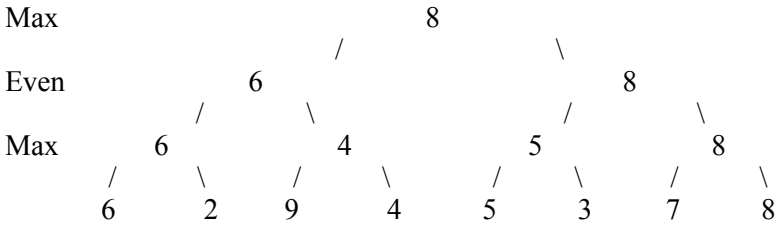
The best strategy for even player is:

1. When both number of choice are even, choose the max of the two
2. When both number of choice are odd, choose the min of the two
3. Else, choose the even number

As a result, the max player and even player will reach Nash equilibriums, which they should both return the max even number as the most optimal move.

Example:
End_states = [6,2,9,4,5,3,7,8]





Note: With a different distribution on the end state, the best strategy for max player will change. For example, with very little even number in the end state, and the mean of odd number values larger than the mean of even number values. In python file I examined one of the strategy for the situation.