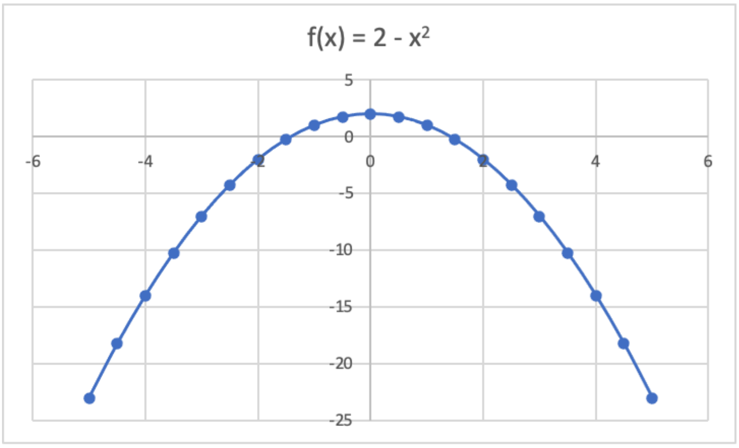
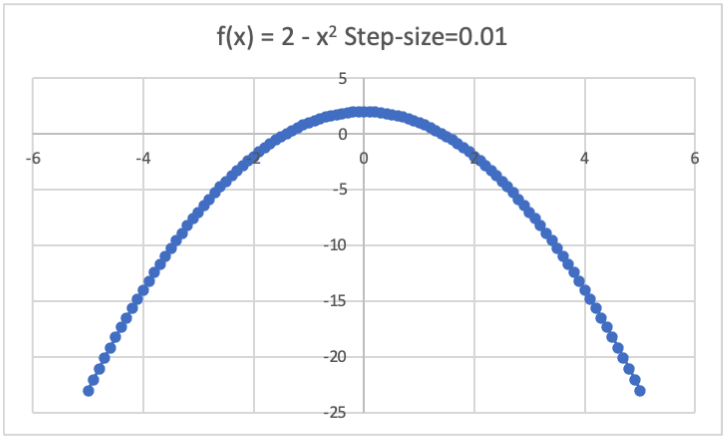
CS471: Introduction to Artificial Intelligence
Assignment 2: Hill-climbing (10 points)

1. **Hill-climbing**
    a. Consider a function f(x) = 2 - x^2 in the following discrete state-space, where x ∈ [-5, 5], step-size 0.5. Implement the hill-climbing algorithm in python to find the maximum value for the above function.



    b. Change the step-size to 0.01. Run the hill-climbing algorithm and share your observations.



Google Colab:

https://colab.research.google.com/drive/1R1oVJm28H4lbxbl-JZk_Wo-QPL99kYIA?usp=sharing

GitHub:

https://github.com/jacobalmon/CS-471/blob/main/Homework/Homework%202/hillClimb.py

Suppose we have an objective function $f(x) = 2 - x^2$. The following code represents the function:

```python
def objFunction1(x): # Function for Q1.
  return 2 - x**2
```

First, we set our current state to the initial state and use the objective function to find the value of the current state.

```python
def hillclimb(initialState, stepSize):
    curr = initialState # set curr to our initial state.
    currVal = objFunction1(initialState) # find value of state.
```

Next, we infinitely loop until the maximum has been found. To achieve this we take our step size and see the values before and after the current state. We then can find the values of those states using the objective function, and we take the maximum of the states and we update the current state with the maximum value from the two new states. If we have reached a local max, then the algorithm completes. Lastly, return the local max.

```python
while True:
      # Find the bounds of the step up and step down of current state.
      neighbors = [curr + stepSize, curr - stepSize]
      # Ensures States are within the bounds.
      neighbors = [x for x in neighbors if -5 <= x <= 5]
      # Find the values of the states we calculated.
      neighborVals = [objFunction1(x) for x in neighbors]
      # We want to use the max value since we're hillclimbing.
      bestNeighborVal = max(neighborVals)

      # Check if the curr needs to be updated.
      if bestNeighborVal > currVal:
        currVal = bestNeighborVal
        curr = neighbors[neighborVals.index(bestNeighborVal)]
      else: # If it doesn't then we ae at our max.
        break
```

During testing with both step sizes, we reach the same maximum of 2.

Testing:

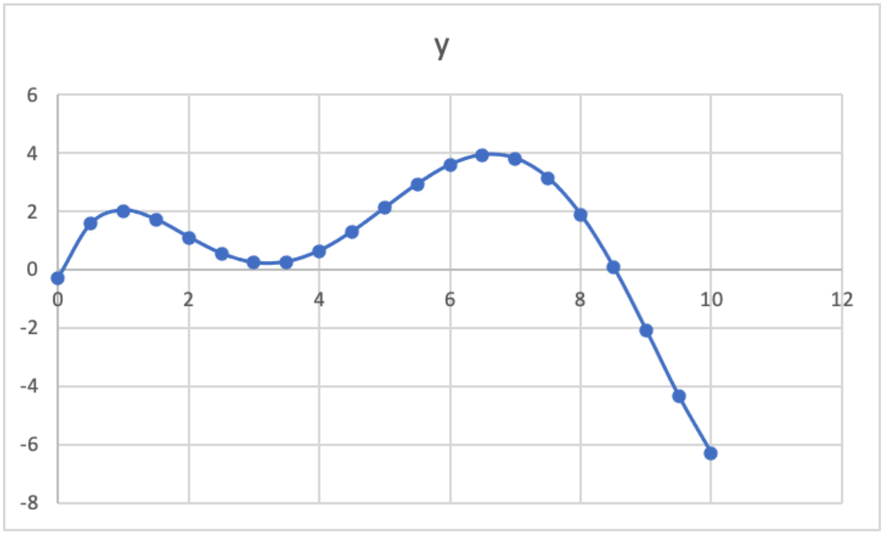
The initial state of the following tests are 3.
Note that first test uses a step size of 0.5.
Note that second test uses a step size of 0.01

```
# Testing Q1 a).
  print(hillclimb(initialState, stepSize1))
  # Testing Q1 b).
  print(hillclimb(initialState, stepSize2))
```

```
2.0
2.0
```

Through testing the second test takes exceptionally longer than the first test, since the step sizes are vastly different, but however the maximum value of 2 is still reached.

2. **Random-restart hill-climbing**
   a. Consider a function $g(x) = (0.0051x^5) - (0.1367x^4) + (1.24x^3) - (4.456x^2) + (5.66x) - 0.287$ in the following discrete state-space, where $x \in [0, 10]$, step-size 0.5. Implement the random-restart hill-climbing algorithm for 20 random restarts in python to find the global maximum value for the above function.



   b. Run the hill-climbing algorithm for the function g(x). Compare and analyze the results of hill-climbing with the random-restart hill-climbing algorithm.

Grading rubric

Implementation of hill-climbing algorithm (1a): 3 points
Observations with respect to the difference in the step-size (1b): 2 points

Implementation of random-restart hill climbing (2a): 2 points
Comparison of hill-climbing vs random-restart hill-climbing (2b): 2 points
Proper commenting and clear formatting of code: 1 point

Total score = 10 points

Google Colab:

https://colab.research.google.com/drive/1R1oVJm28H4lbxbl-JZk_Wo-QPL99kYlA?usp=sharing

GitHub:

https://github.com/jacobalmon/CS-471/blob/main/Homework/Homework%202/hillClimb.py

Now, we want to do the same thing as before but randomize 20 initial states, so we can get the global max because the issue with the regular hill climb algorithm is that it could either get stuck in a shoulder or find a local max instead of the global max. We initially set our maximum as -8, since the value -8 doesn't exist in the discrete space [0,10]. Using the previous algorithm we run the algorithm 20 times with 20 random states to find the global maximum with the associated objective function, with each local maximum, we can likely find the global maximum with bigger local maximums.

```
def hillclimb20rand(stepSize):
```

```
# Used -8 as initial since it doesn't exist within the discrete space.
maxclimb = -8
# Randomize 20 States.
for _ in range(20):
  # Make sure random states are within the range [0, 10].
  randomState = random.randint(0, 10)
  # Set curr to our initial random state.
  curr = randomState
  # Find value of curr's state with function.
  currVal = objFunction2(randomState)

  # Run infinitely, until a max value has been found.
  while True:
    # Find the bounds of the step up and step down of current state.
    neighbors = [curr + stepSize, curr - stepSize]
    # Ensures States are within the bounds.
    neighbors = [x for x in neighbors if 0 <= x <= 10]
    # Find the values of the states we calculated.
    neighborVals = [objFunction2(x) for x in neighbors]
    # We want to use the max value since we're hillclimbing.
    bestNeighborVal = max(neighborVals)

    if bestNeighborVal > currVal:
      # Check if the curr needs to be updated.
      currVal = bestNeighborVal
      curr = neighbors[neighborVals.index(bestNeighborVal)]
    else: # If it doesn't then we ae at our max.
      break

  # Keep track of the maximum value for each initial random state.
  maxclimb = max(maxclimb, currVal)

return maxclimb
```

Testing:

```
# Testing Q2.
 print(hillclimb20rand(stepSize1))
```

```
3.9287781250000213
```

Through testing the comparison between the two algorithms, they are different because one is "likely" to find the global maximum because the original hill climb algorithm runs 20 times to find that global maximum previously explained in the above code. The other one will usually find the local maximum, so it won't find the optimal solution, whereas the other one is likely to be the optimal solution.