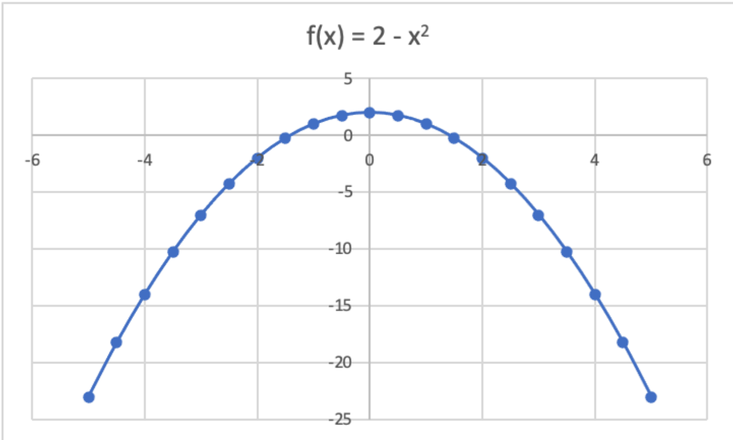
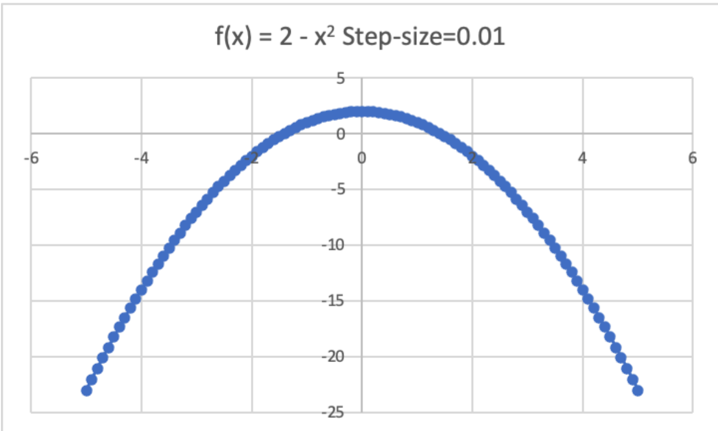
CS471: Introduction to Artificial Intelligence
Assignment 2: Hill-climbing
Malcolm Zartman
09/12/2025

Question 1: Hill-climbing

a. Consider a function $f(x) = 2 - x^2$ in the following discrete state-space, where $x \in [-5, 5]$, 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.



Colab link:
https://colab.research.google.com/drive/1DoxKoEw_1e0lsEjtlmj_z7Ihc-oWU3Eb?usp=sharing

1a)
I first am defining the function for later calculation usage:

```
def f(x):
  # This is the function we want to maximize: f(x) = 2 - x^2
  return 2 - (x * x)
```

Below is where I defined my hill_climbing function. It takes one input which is a list of x inputs or state spaces for the f(x) function that we previously defined. In general it follows this algorithm:

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

For this function, I am however going to be checking both the left and right neighbors to see which neighbor has a greater value, but I am only checking this on the first pass. I am using the variable goingLeft for the direction of the climbing. True if we are going left and decreasing the current_ind by 1, and False if we are going right and increasing the current_ind by 1.

We are starting at the 0th index for no rhyme or reason, but this does allow us to start at the lowest end of the discrete space and allows us to climb to the top over multiple iterations.

Other than tracking and increasing or decreasing the current_ind, depending on the direction of our climbing, we will also be tracking the current_value and at any point that we are not climbing anymore or we are not finding neighbors that are greater than this current_value, then we will return this current_value as our reasonable solution or local maximum.

```
def hill_climbing(problem_space):
  # Start at the beginning of the problem space
  current_ind = 0 # index of the current state
  current_x = problem_space[current_ind] # first x-value in the
state-space
  current_value = f(current_x)
  # should be -5, -23
  print(f"Starting at x = {current_x}, with f(x) = {current_value}")

  firstPass = True
  goingLeft = False
  neighbor1, neighbor2 = None, None
  while True:
    # for the first pass check both of the neighbors and only use the
greatest one
    if firstPass:
      neighborL = f(problem_space[current_ind-1]) # calculated left
neighbor's value
      neighborR = f(problem_space[current_ind+1]) # calculated right
neighbor's value
      # checking if values were calculated, if so continue - otherwise
return none
      if (neighborL != None) & (neighborR != None):
        if neighborL > neighborR: # if the left neighbor is greater then
we are going left
          print(f"Going with the Left neighbor at x =
{problem_space[current_ind-1]}, with f(x) = {neighborL}")
          print(f"Since the Right neighbor at x =
{problem_space[current_ind+1]}, with f(x) = {neighborR} is less than the
Left neighbor")
          neighbor = neighborL
          current_ind = current_ind - 1
          goingLeft = True
        else: # if the right neighbor is greater then we are going right
          print(f"Going with the Right neighbor at x =
{problem_space[current_ind+1]}, with f(x) = {neighborR}")
```

```
          print(f"Since the Left neighbor at x =
{problem_space[current_ind-1]}, with f(x) = {neighborL} is less than the
Right neighbor")
          neighbor = neighborR
          current_ind = current_ind + 1
          # goingLeft = False # already set
      else:
        print("Calculation Error")
        return None
      firstPass = False
    else: # if it's not the first pass, keep going in the chosen direction
until locl max is reached
      if goingLeft:
        neighbor = f(problem_space[current_ind-1])
        current_ind = current_ind - 1
      else:
        neighbor = f(problem_space[current_ind+1])
        current_ind = current_ind + 1

    if neighbor <= current_value: # if the neighbor is not greater, stop
traversing, we found a reasonable solution
      print(f"The neighbor at x = {problem_space[current_ind]}, with f(x)
= {neighbor}, is less than or equal to the current value.")
      print(f"We have reached a reasonable solution at x = {current_x},
with f(x) = {current_value}")
      return current_value
    else: # if the neighbor is greater than the current_value, then keep
searching for a greater one
      print(f"The neighbor at x = {problem_space[current_ind]}, with f(x)
= {neighbor}, is greator than the current value.")
      print("Evaluating next neighbor...")
      current_value = neighbor
      current_x = problem_space[current_ind]
```

In the below code, we are defining the state space or essentially the x-range of our problem, and our step size (the size of our steps in that x-range - determines how many or how little values we are checking as well as how accurate or how long we take in the algorithm). We also are using the x-range and step-size to create the state space and then send it to the hill_climbing function to find a local_maximum / reasonable solution.

```
# define the discrete state-space for x from -5 to 5 with a step-size of 0.5
x_min = -5.0
x_max = 5.0
step = 0.5

# generate the list of possible x values
states = [x_min + i * step for i in range(int((x_max - x_min) / step) + 1)]

print("Problem: Find the maximum value of f(x) = 2 - x^2")
print(f"State space for x: {states}\n")

# run the hill-climbing algorithm
max_value = hill_climbing(states)

print(f"Maximum function value = {max_value}")
```

Output:

```
Problem: Find the maximum value of f(x) = 2 - x^2
State space for x: [-5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]

Starting at x = -5.0, with f(x) = -23.0
Going with the Right neighbor at x = -4.5, with f(x) = -18.25
Since the Left neighbor at x = 5.0, with f(x) = -23.0 is less than the Right neighbor
The neighbor at x = -4.5, with f(x) = -18.25, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -4.0, with f(x) = -14.0, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -3.5, with f(x) = -10.25, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -3.0, with f(x) = -7.0, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -2.5, with f(x) = -4.25, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -2.0, with f(x) = -2.0, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -1.5, with f(x) = -0.25, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -1.0, with f(x) = 1.0, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = -0.5, with f(x) = 1.75, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = 0.0, with f(x) = 2.0, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = 0.5, with f(x) = 1.75, is less than or equal to the current value.
We have reached a reasonable solution at x = 0.0, with f(x) = 2.0
Maximum function value = 2.0
```

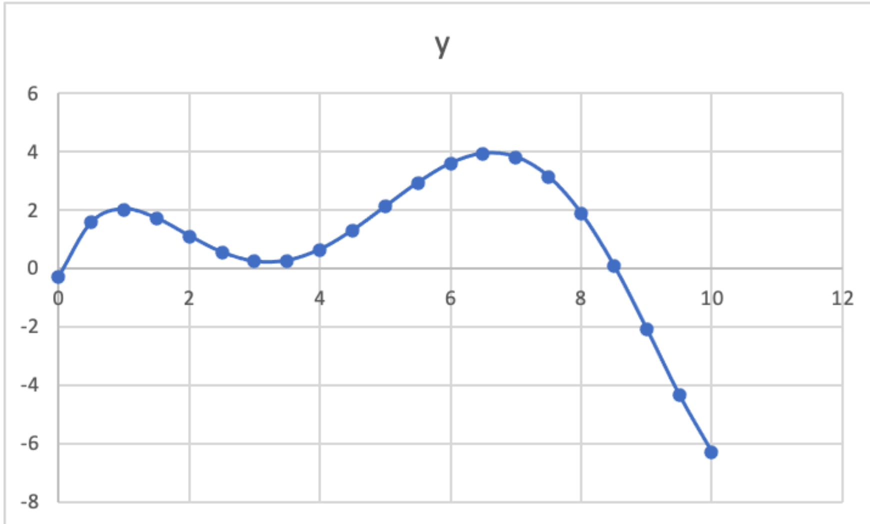
The maximum value found was 2.0

1b) By changing the step-size to 0.01, it still reached the max value of the f(x) function, but it took a whole lot longer to reach that solution. So much so, that it wouldn't make sense to paste the output of it into this document.

```
# 1b) STEP SIZE OF 0.01

# define the discrete state-space for x from -5 to 5 with a step-size of 0.5
x_minb = -5.0
x_maxb = 5.0
stepb = 0.01

# generate the list of possible x values
statesb = [x_minb + i * stepb for i in range(int((x_maxb - x_minb) / stepb) + 1)]

print("Problem: Find the maximum value of f(x) = 2 - x^2")
print(f"State space for x: {statesb}\n")

# run the hill-climbing algorithm
max_valueb = hill_climbing(statesb)

print(f"Maximum function value = {max_valueb}")
```

<u>Question 2:</u> 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.

Colab link:
https://colab.research.google.com/drive/1DoxKoEw_1e0lsEjtlmj_z7Ihc-oWU3Eb?usp=sharing

2a)
First I am defining the function for later calculation usages:

```
def g(x):
   # This is the function we want to maximize for the random_restart algorithm: f(x) = 2 - x^2
   return (0.0051 * x**5) - (0.1367 * x**4) + (1.24 * x**3) - (4.456 * x**2) + (5.66 * x) - 0.287
```

Below is the code for the random_restart function or the random-restart hill climbing algorithm. This defers from the previous algorithm because instead of starting at a predefined location, we are starting at a random location and then running the algorithm a certain amount of times. In my "main" code I define the looping logic for the iteration portion of the algorithm, but in the function below, the major difference in terms of code from the previous hill climbing function, is the random location generation and the handling for it. I use the variable space_length for generating the random locations, and to also ensure I don't have an index that is out of bounds with my problem_space list by using the modulus in order to loop back around when iterating through the neighbors.

```
import random
def random_restart(problem_space):
  # ONLY CHANGE: Start at a random state in the problem space
  space_length = len(problem_space) # will use space_length & modulos to
ensure no 'index out of range' errors
  current_ind = random.randint(0,space_length-1) # index of the current
state
  current_x = problem_space[current_ind] # first x-value in the
state-space
  current_value = g(current_x)
  print(f"Starting at x = {current_x}, with f(x) = {current_value}")

  firstPass = True
  goingLeft = False
  neighbor1, neighbor2 = None, None
  while True:
    # for the first pass check both of the neighbors and only use the
greatest one
    if firstPass:
                                                              neighborL =
f(problem_space[(current_ind-1+space_length)%space_length]) # calculated
left neighbor's value
            neighborR = f(problem_space[(current_ind+1)%space_length]) #
calculated right neighbor's value
        # checking if values were calculated, if so continue - otherwise
return none
      if (neighborL != None) & (neighborR != None):
          if neighborL > neighborR: # if the left neighbor is greater then
we are going left
                        print(f"Going with the Left neighbor at x =
{problem_space[(current_ind-1+space_length)%space_length]}, with f(x) =
{neighborL}")
                            print(f"Since the Right neighbor at x =
{problem_space[(current_ind+1)%space_length]}, with f(x) = {neighborR} is
less than the Left neighbor")
          neighbor = neighborL
```

```
          current_ind = current_ind - 1
          goingLeft = True
        else: # if the right neighbor is greater then we are going right
          print(f"Going with the Right neighbor at x = {problem_space[(current_ind+1)%space_length]}, with f(x) = {neighborR}")
          print(f"Since the Left neighbor at x = {problem_space[(current_ind-1+space_length)%space_length]}, with f(x) = {neighborL} is less than the Right neighbor")
          neighbor = neighborR
          current_ind = current_ind + 1
          # goingLeft = False # already set
      else:
        print("Calculation Error")
        return None
      firstPass = False
    else: # if it's not the first pass, keep going in the chosen direction until locl max is reached
      if goingLeft:
        neighbor = f(problem_space[(current_ind-1+space_length)%space_length])
        current_ind = current_ind - 1
      else:
        neighbor = f(problem_space[(current_ind+1)%space_length])
        current_ind = current_ind + 1

     if neighbor <= current_value: # if the neighbor is not greater, stop traversing, we found a reasonable solution
       print(f"The neighbor at x = {problem_space[current_ind%space_length]}, with f(x) = {neighbor}, is less than or equal to the current value.")
       print(f"We have reached a reasonable solution at x = {current_x}, with f(x) = {current_value}")
      return current_value
     else: # if the neighbor is greater than the current_value, then keep searching for a greater one
       print(f"The neighbor at x = {problem_space[current_ind%space_length]}, with f(x) = {neighbor}, is greator than the current value.")
      print("Evaluating next neighbor...")
      current_value = neighbor
```

```
    current_x = problem_space[current_ind%space_length]
```

Below is the logic for defining the state space as well as for the rest of the random restart hill-climbing algorithm. Here is where I handle the amount of iterations of calling this algorithm and then finding an ultimate "local-maxium" or "reasonable solution." The desired iteration amount was 20, and as I call the algorithm, I save the current local_max and then check if it was greater than the previously found max, if so, I save it as the new max in max_found, otherwise I keep looping. That process continues until I call the algorithm 20 times. At the end I display the max_found.

```
# define the discrete state-space for x from 0 to 10 with a step-size of 0.5
x_min2 = 0.0
x_max2 = 10.0
step2 = 0.5

# generate the list of possible x values
states2 = [x_min2 + i * step2 for i in range(int((x_max2 - x_min2) / step2) + 1)]

print("Problem: Find the maximum value of g(x) = (0.0051x5) - (0.1367x4) + (1.24x3) - (4.456x2) + (5.66x) - 0.287")
print(f"State space for x: {states2}\n")

# run the random restart hill-climbing algorithm
local_max = None
max_found = None
iterations = 20
count = 0
while count < iterations:
  count = count + 1
  print(f"Iteration: {count}")
  max_value2 = random_restart(states2)
  local_max = max_value2
  if (max_found == None) or (local_max > max_found):
    max_found = local_max
  print("\n")

print(f"\nMaximum function value (rounded by 4) = {round(max_found,4)}")
```

Output:

Problem: Find the maximum value of g(x) = (0.0051x5) - (0.1367x4) + (1.24x3) - (4.456x2) + (5.66x) - 0.287
State space for x: [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0]

Iteration: 1
Starting at x = 5.0, with f(x) = 2.1130000000000093
Going with the Left neighbor at x = 4.5, with f(x) = -18.25
Since the Right neighbor at x = 5.5, with f(x) = -28.25 is less than the Left neighbor
The neighbor at x = 4.5, with f(x) = -18.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 5.0, with f(x) = 2.1130000000000093

Iteration: 2
Starting at x = 2.0, with f(x) = 1.1049999999999995
Going with the Left neighbor at x = 1.5, with f(x) = -0.25
Since the Right neighbor at x = 2.5, with f(x) = -4.25 is less than the Left neighbor
The neighbor at x = 1.5, with f(x) = -0.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 2.0, with f(x) = 1.1049999999999995

Iteration: 3
Starting at x = 7.5, with f(x) = 3.1360468750000257
Going with the Left neighbor at x = 7.0, with f(x) = -47.0
Since the Right neighbor at x = 8.0, with f(x) = -62.0 is less than the Left neighbor
The neighbor at x = 7.0, with f(x) = -47.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 7.5, with f(x) = 3.1360468750000257

Iteration: 4
Starting at x = 5.0, with f(x) = 2.1130000000000093
Going with the Left neighbor at x = 4.5, with f(x) = -18.25
Since the Right neighbor at x = 5.5, with f(x) = -28.25 is less than the Left neighbor
The neighbor at x = 4.5, with f(x) = -18.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 5.0, with f(x) = 2.1130000000000093

Iteration: 5
Starting at x = 0.0, with f(x) = -0.287
Going with the Right neighbor at x = 0.5, with f(x) = 1.75
Since the Left neighbor at x = 10.0, with f(x) = -98.0 is less than the Right neighbor
The neighbor at x = 0.5, with f(x) = 1.75, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = 1.0, with f(x) = 1.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 0.5, with f(x) = 1.75

Iteration: 6
Starting at x = 5.0, with f(x) = 2.1130000000000093
Going with the Left neighbor at x = 4.5, with f(x) = -18.25
Since the Right neighbor at x = 5.5, with f(x) = -28.25 is less than the Left neighbor
The neighbor at x = 4.5, with f(x) = -18.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 5.0, with f(x) = 2.1130000000000093

Iteration: 7
Starting at x = 6.0, with f(x) = 3.5913999999999637
Going with the Left neighbor at x = 5.5, with f(x) = -28.25
Since the Right neighbor at x = 6.5, with f(x) = -40.25 is less than the Left neighbor
The neighbor at x = 5.5, with f(x) = -28.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 6.0, with f(x) = 3.5913999999999637

Iteration: 8
Starting at x = 0.0, with f(x) = -0.287
Going with the Right neighbor at x = 0.5, with f(x) = 1.75
Since the Left neighbor at x = 10.0, with f(x) = -98.0 is less than the Right neighbor
The neighbor at x = 0.5, with f(x) = 1.75, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = 1.0, with f(x) = 1.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 0.5, with f(x) = 1.75

Iteration: 9
Starting at x = 7.0, with f(x) = 3.808000000000056

Going with the Left neighbor at x = 6.5, with f(x) = -40.25
Since the Right neighbor at x = 7.5, with f(x) = -54.25 is less than the Left neighbor
The neighbor at x = 6.5, with f(x) = -40.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 7.0, with f(x) = 3.808000000000056

Iteration: 10
Starting at x = 3.0, with f(x) = 0.23559999999999354
Going with the Left neighbor at x = 2.5, with f(x) = -4.25
Since the Right neighbor at x = 3.5, with f(x) = -10.25 is less than the Left neighbor
The neighbor at x = 2.5, with f(x) = -4.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 3.0, with f(x) = 0.23559999999999354

Iteration: 11
Starting at x = 6.0, with f(x) = 3.5913999999999637
Going with the Left neighbor at x = 5.5, with f(x) = -28.25
Since the Right neighbor at x = 6.5, with f(x) = -40.25 is less than the Left neighbor
The neighbor at x = 5.5, with f(x) = -28.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 6.0, with f(x) = 3.5913999999999637

Iteration: 12
Starting at x = 1.5, with f(x) = 1.708684374999998
Going with the Left neighbor at x = 1.0, with f(x) = 1.0
Since the Right neighbor at x = 2.0, with f(x) = -2.0 is less than the Left neighbor
The neighbor at x = 1.0, with f(x) = 1.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 1.5, with f(x) = 1.708684374999998

Iteration: 13
Starting at x = 1.5, with f(x) = 1.708684374999998
Going with the Left neighbor at x = 1.0, with f(x) = 1.0
Since the Right neighbor at x = 2.0, with f(x) = -2.0 is less than the Left neighbor
The neighbor at x = 1.0, with f(x) = 1.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 1.5, with f(x) = 1.708684374999998

Iteration: 14
Starting at x = 9.0, with f(x) = -2.0617999999999705

Going with the Left neighbor at x = 8.5, with f(x) = -70.25
Since the Right neighbor at x = 9.5, with f(x) = -88.25 is less than the Left neighbor
The neighbor at x = 8.5, with f(x) = -70.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 9.0, with f(x) = -2.0617999999999705

Iteration: 15
Starting at x = 7.5, with f(x) = 3.1360468750000257
Going with the Left neighbor at x = 7.0, with f(x) = -47.0
Since the Right neighbor at x = 8.0, with f(x) = -62.0 is less than the Left neighbor
The neighbor at x = 7.0, with f(x) = -47.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 7.5, with f(x) = 3.1360468750000257

Iteration: 16
Starting at x = 9.5, with f(x) = -4.3277656249999845
Going with the Left neighbor at x = 9.0, with f(x) = -79.0
Since the Right neighbor at x = 10.0, with f(x) = -98.0 is less than the Left neighbor
The neighbor at x = 9.0, with f(x) = -79.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 9.5, with f(x) = -4.3277656249999845

Iteration: 17
Starting at x = 6.5, with f(x) = 3.9287781250000213
Going with the Left neighbor at x = 6.0, with f(x) = -34.0
Since the Right neighbor at x = 7.0, with f(x) = -47.0 is less than the Left neighbor
The neighbor at x = 6.0, with f(x) = -34.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 6.5, with f(x) = 3.9287781250000213

Iteration: 18
Starting at x = 1.5, with f(x) = 1.708684374999998
Going with the Left neighbor at x = 1.0, with f(x) = 1.0
Since the Right neighbor at x = 2.0, with f(x) = -2.0 is less than the Left neighbor
The neighbor at x = 1.0, with f(x) = 1.0, is less than or equal to the current value.
We have reached a reasonable solution at x = 1.5, with f(x) = 1.708684374999998

Iteration: 19
Starting at x = 3.0, with f(x) = 0.23559999999999354

Going with the Left neighbor at x = 2.5, with f(x) = -4.25
Since the Right neighbor at x = 3.5, with f(x) = -10.25 is less than the Left neighbor
The neighbor at x = 2.5, with f(x) = -4.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 3.0, with f(x) = 0.23559999999999354

Iteration: 20
Starting at x = 8.0, with f(x) = 1.8826000000000311
Going with the Left neighbor at x = 7.5, with f(x) = -54.25
Since the Right neighbor at x = 8.5, with f(x) = -70.25 is less than the Left neighbor
The neighbor at x = 7.5, with f(x) = -54.25, is less than or equal to the current value.
We have reached a reasonable solution at x = 8.0, with f(x) = 1.8826000000000311

Maximum function value (rounded by 4) = 3.9288

2b) By using the hill-climbing algorithm for the g(x) function, it evaluated the function rather fast, however, the result was not the global maximum of the function, rather it was a local maximum which is still a reasonable solution, just not the one we are ultimately looking for. So the random-restart hill climbing algorithm is for sure a better solution for a more complex function such as g(x), in which we run it multiple times and only keep the ultimate maximum out of the ones we find. And in this scenario, the random-restart was able to find the global maxim while the regular hill-climbing function only found the local-maxium. With this algorithm a way to optimize it further is by figuring out what would be a good amount of iterations to run that random-restart algorithm for.

```
# 2b) HILL CLIMBING WITH G(x)

# define the discrete state-space for x from -5 to 5 with a step-size of
0.5
x_min2b = -5.0
x_max2b = 5.0
step2b = 0.5

# generate the list of possible x values
states2b = [x_min2b + i * step2b for i in range(int((x_max2b - x_min2b) /
step2b) + 1)]
```

```python
print("Problem: Find the maximum value of f(x) = 2 - x^2")
print(f"State space for x: {states2b}\n")

# run the hill-climbing algorithm
max_value2b = hill_climbingG(states2b)

print(f"Maximum function value = {max_value2b}")
```

Output:

```
Problem: Find the maximum value of f(x) = 2 - x^2
State space for x: [-5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]

Starting at x = -5.0, with g(x) = -396.36199999999997
Going with the Left neighbor at x = 5.0, with g(x) = 2.1130000000000093
Since the Right neighbor at x = -4.5, with g(x) = -294.452478125 is less than the Left neighbor
The neighbor at x = 5.0, with g(x) = 2.1130000000000093, is greator than the current value.
Evaluating next neighbor...
The neighbor at x = 4.5, with g(x) = 1.299390624999993, is less than or equal to the current value.
We have reached a reasonable solution at x = 5.0, with g(x) = 2.1130000000000093
Maximum function value = 2.1130000000000093
```