Computational Intelligence 2020 **(Due: 7/04/2020)**

Submission Assignment #1: Optimization

Instructor: Jakub Tomczak *Name:* Tyron Landman*, Netid:* 2674357

1 Introduction

The purpose of this report is to determine differences and/or similarities between a gradient-based optimization method and a derivative-free optimization method. The comparisons we will be looking at will be time complexity as well as the end accuracy of the optimization method. The two optimization methods chosen for this report are the gradient descent optimization method as well as a simple hill climbing (derivative-free) optimization method. These two methods were chosen as they run along the same basic principle of simply determining which side of the current point has the fastest decline (or incline). The gradient descent method used will first take the objective function and try to find the gradients of the x variables. With these gradients you will be able determine which direction the algorithm should move in i.e. if the gradient is for example -2 you know to continue to the right in the x -axis (on a 2D graph) as the minimum (local or global) will be in that direction. Similarly the derivative free calculation also follows this same line of reasoning, however if takes a particular point, then calculates the value of the positions either side of the initial point to determine which direction to proceed to.

2 Problem statement

For our problem of determining differences/similarities, we will be using our two optimization methods to determine which algorithm is able to find the global/local minimum the fastest and most accurately. We will use the same objective function for both and is as stated below:

$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

For the gradient descent method, the first thing we do is calculate the partial derivative functions for the objective function. For the variable x_1

$$f'(x_1) = 2x_1 + 0.9\pi \sin(3\pi x_1)$$

and for the variable x_2

$$f'(x_2) = 4x_2 + 1.6\pi \sin(4\pi x_2)$$

For the derivative free method, we will simply be inputting values into the objective function and looking at which direction to go next from there.
The reason these two methods were chosen is because they both run into a similar issue. This issue is if they start on a hill or gradient that is a part of a local minimum, they will only ever find that local minimum. This means that, in their simplest form, if we apply it to some functions we may not know if the minimum (or maximum) is actually found. Another issue that may arise is that gradient descent cannot be used for discrete data. As a result for this experiment we will be using continuous data.
Lastly, as we will be using a simple version of both gradient descent and hill climbing, there comes the issue of plateaus. Because of the code used, the algorithm moves by looking at the change occurring in the points. If the gradient reaches 0, it might not be a local minimum but a flat plateau. And for the hill climbing, if either side if the same value as it's current point a plateau has been found. The best way for us to remedy this is to use the same input values for testing each function and run the function multiple times over.

3 Methodology

To begin the experiment I first created the code required for both the gradient descent method, and the hill climbing algorithm. For the gradient descent method, the code begins as above, with finding the partial derivatives for each variables x_1, and x_2. Because we have already calculated the partial derivatives behind, all we have to do with them now is create a function that takes in x_1, and x_2 as arguments and returns the values produced.
Next step is to define define the hyper-parameters; the alpha and a variable that would act as measure of change to determine if the algorithm should move or finish and the maximum amount of iterations we will allow the algorithm to have. The alpha allows us to fine tune the algorithm increasing the accuracy by lowering the alpha, or decreasing it by increasing the alpha value. Unfortunately, as the alpha decreases in size (allowing for greater accuracy) the maximum iteration count will need to be increased as well as movement along the curve will take place at a much slower pace.
When we begin the code for gradient descend, we find a random starting location as our starting point. The difference variable will then be used to assess the positions next to the starting point and determine whether or not the algorithm should adjust the current point or to end. It will end if the difference between the current position and the next position is below a predetermined (difference variable) threshold. The final algorithm used to calculate the next point therefore becomes:
x - $alpha$ * gradient function(x_1,x_2).
Where x represents the current position, and the gradient function is as discussed above.
Finally, the max iteration variable will be adjusted to limit the gradient descent method from taking too long to find the local minimum as it continuously loops until it means it's break condition (difference variable threshold).

The next method used will be the hill climbing method. As previously mentioned, this method will be used because of it's similarity to how the gradient descent method. This method requires us to begin the same as the gradient descent method by defining a function for the objective function.
This means we will have a function that takes variables x_1, and x_2 and runs it, not through the partial derivatives but the original equation.
$(f(x) = x_1^2 \ + \ 2x_2^2 \ - 0.3\cos(3\pi x_1) \ - \ 0.4\cos(4\pi x_2) \ + \ 0.7)$

Next, randomly select a starting point. From there, we create a function that finds the positions on all sides of the starting point to determine the x_1, and x_2 variables. With these new values we are able to send them as parameters to the original function created to ascertain a score and assess whether the algorithm should move, and if moved, where too. This code is slightly easier than the gradient descent method as you don't have to adjust as many hyper parameters. It is simply looping through the function and adjusting the current position to a different one if a better evaluation is returned.

4 Experiments

For the experimentation for our problem statement, our goal was to determine which hyper-parameters would be best to use for each method given the objective function. For gradient-descent, we wanted to know the effects of adjusting the maximum iteration count, the alpha, and the difference variable values. For the hill climbing method, we adjusted the maximum iteration, as well as the alpha as well.
For both methods a timer was added to determine how long each method took to complete it's individual results. Alongside the time, the mean and standard deviation will be calculated. The timer will help to determine the efficiency of the method. This, taking into account the small amount of iterations and relative simplicity will most likely lead to small differences between the two but will give us an idea for larger amounts of data. As for the mean and standard deviation, having these will allow us to gain a better insight into the accuracy of the methods themselves.

5 Results and discussion

5.1 Gradient Descent

Table 1: As you can see from the table, we took 3, predefined values for the variables x_1, and x_2. We then

x_1, x_2	Alpha	Difference	Iterations	Time taken (seconds)	Results
[-43.527, -6.511]	0.01	0.0001	100/100	0.00448	[-5.689 -0.933]
[-43.527, -6.511]	0.01	0.0001	215/1000	0.00398	[-1.223,-0.933]
[-43.527, -6.511]	0.001	0.0001	501/1000	0.0156	[-5.928, -0.935]
[-43.527, -6.511]	0.001	0.0001	2040/10000	0.0156	[-1.228, -0.935]
[-43.5273, -6.511]	0.001	0.000001	2332/10000	0.0156	[-1.222, -0.933]
[-84.760 -87.862]	0.01	0.0001	100/100	0	[-10.955, -1.694]
[-84.760 -87.862]	0.01	0.0001	248/1000	0.0039	[-1.223, -0.933]
[-84.760 -87.862]	0.001	0.0001	1000/1000	0.156	[-11.464, -1.860]
[-84.760 -87.862]	0.001	0.0001	2373/10000	0.0963	[-1.228, -0.935]
[-84.760 -87.862]	0.001	0.000001	2665/1000	0.04787	[-1.222, -0.933]
[98.037, 38.258]	0.01	0.0001	100/100	0	[12.627 1.283]
[98.037, 38.258]	0.01	0.0001	255/1000	0.003989	[1.223, 0.933]
[98.037, 38.258]	0.001	0.0001	1000/1000	0.01795	[13.242, 1.339]
[98.037, 38.258]	0.001	0.0001	2446/10000	0.04288	[1.228, 0.935]
[98.037, 38.258]	0.001	0.000001	2738/10000	0.06248	[1.222, 0.933]

Table 1: Gradient Descent Results

adjusted one of either the Alpha, the difference, or the iteration values (left is iterations completed, right side is max iterations allowed). Right away we can see from the first trial in each group that having a max iteration hyper-parameter set to 100 leads to less than optimal results, however as per the time it is almost instant in most cases.
The next thing to notice is that with enough iterations the results do converge to a point around the vector [-1.222, -0.935]. This occurs in all cases so is most likely a global minimum for this function.
Another thing to note is that when the alpha is increased the results did get worse, before becoming even more accurate as the difference threshold and max iterations were increased.

Some further testing, with 10 random starting vectors all shower that when ran, the gradient descent converges with a mean of 1.022 for the x_1 variable, and -0.093 for the x_2 variable. This then leads to a standard deviation of 1.223 for the x_1 variable, and 0.881 for the x_2 variable.

5.2 Hill Climbing

Below are the results for the hill climbing method.
Table 2: Just like with the Gradient descent, I used 3 different vectors and evaluated them to see how the Hill

x_1, x_2	Alpha	iterations/ Max iterations	Time taken	Results
[-43.527, -6.511]	0.01	100/100	0.003	[-42.527, -5.511]
[-43.527, -6.511]	0.01	1000/1000	0.598	[-33.527, -0.931]
[-43.527, -6.511]	0.1	423/1000	0.1596	[-1.227, -0.411]
[-43.527, -6.511]	1	43/1000	0.00099	[-0.527, 0.488]
[-43.527, -6.511]	1.15	38/1000	0.001	[0.172, 0.388]
[-84.760, -87.862]	0.01	100/100	0.00396	[-83.760, -86.862]
[-84.760, -87.862]	0.01	1000/1000	0.03789	[-74.760, -77.862]
[-84.760, -87.862]	0.1	869/1000	0.0319	[-1.260, -0.964]
[-84.760, -87.862]	1	88/1000	0.003	[-0.760, 0.137]
[-84.760, -87.862]	1.15	76/1000	0.003	[0.339, -0.462]
[98.0377, 38.258]	0.01	100/100	0.004	[97.0377, 37.2581]
[98.0377, 38.258]	0.01	1000/1000	0.039	[88.0377, 28.258]
[98.0377, 38.258]	0.1	969/1000	0.0359	[1.137, 0.958]
[98.0377, 38.258]	1	98/1000	0.004	[0.0377, 0.258]
[98.0377, 38.258]	1.15	85/1000	0.004	[0.287, 0.308]

Table 2: Hill Climbing Results

climbing method performs.
The first thing to note is that the alpha performs more poorly when lower (in comparison to gradient descent where the lower the more accurate as we found). Some things to take into account for this however is that from the first two tests, we can see that the resulting vectors do appear to be converging, the max iterations would simply require to be increased substantially. This is fine for such a function however as more variable are introduced this may become implausible. The reason being, the average of the time for iterations over 2,000 comes to 0.0468 seconds. However, for hill climbing we can see, even just at around 1000 iterations the average time is 0.178 (with the possibility of the 0.598 time being an outlier). Upon removing the potential outlier however we still achieve a time of 0.0281975 which is slower then the gradient descent.
What is interesting to note however is as the *alpha* is increased, not only does the hill climbing method reach it's lowest point almost instantly (revealing a minimum that the gradient descent was unable to find) and doing so in an iteration count much lower than the best performing gradient descent test.

With respect to mean and standard deviation, the alpha of 1.15 was chosen as it had the best results. From the results we can see that the mean for the hill climbing method, for variable x_1 is 0.011, and for x_2 is -0.1. Following this, the standard deviation can be calculated and stands at 0.85 and 0.278 respectively.

5.3 Comparison

When comparing the two together, it makes sense that the hill climbing method is slightly slower as it's calculation is larger, however as we saw we can get around that by simply increasing the alpha, offering us a much faster result, and even closer to the global minimum.
Whats also interesting is that when run, the results for the mean and standard deviation show that the hill climbing method is a lot more consistent in it's results.
From what we have seen so far, it is interesting to find that hill climbing is producing not only more accurate results, but a lot more consistency. It has the drawback of being slower, however this can be worked around simply by increasing the alpha and going from there.

6 Conclusion

In conclusion, both algorithms are fairly consistent upon converging on the minimum found in the function. Hill climbing seems to be faster with a higher alpha, but gradient descent would be more Consistenly faster (as a higher alpha for gradient descent would also increase it's speed). Both can be implemented fairly simply, however I did find implementing gradient descent method much simpler as the hill climbing took longer to implement.