Assignment #5

This assignment is due on Tuesday, November 28th 23:59:59 class via email to christian.wallraven+AMF2023@gmail.com.

Important: You need to name your file properly. If you do not adhere to this naming convention, I may not be able to properly grade you!!!

If you are done with the assignment, make one zip-file of the assignment1 directory and call this zip-file `STUDENTID_A5.zip` (e.g.: `2016010001_A5.zip`). The correctness of the IDs matters! **Please double-check that the name of the file is correct!!**

Also: Please make sure to comment all code, so that I can understand what it does. Uncommented code will reduce your points!

Finally: please read the assignment text carefully and make sure to implement EVERYTHING that is written here – if you forget to address something I wrote, this will also reduce your points! Precision is key ☺!

All code should be submitted in one ipython-notebook called "gradient_descent.ipynb".

Part1 Gradient descent (30 points):

Your task is to implement Gradient Descent as discussed in class, which looks for the minimum of a function with a known derivative.

Implement the following function

```
xoptimal,foptimal,niterations = gradient_descent(f,g,xstart,
lambda,tolerance,maxiterations,doplot=True)
```

where `f` is a pointer to a function, `g` is a pointer to the gradient of the function, `xstart` is the starting point, `lambda` is the step size, `tolerance` is the squared norm of the gradient at which you accept that the minimum is found (it should be $||g||^2 = 0$, but for numerical stability, usually a small, non-zero value is chosen), `maxiterations` is the maximum number of iterations that you allow to take place, and `doplot` controls whether to plot the surface plot (see below).

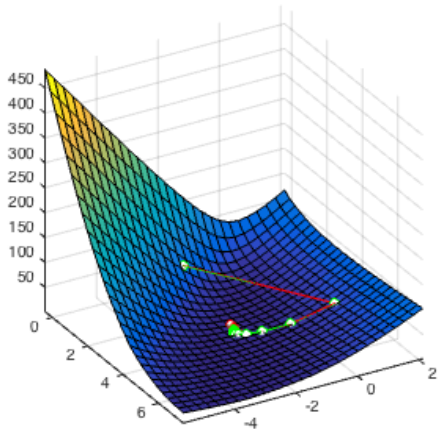
In each iteration step k of the gradient descent iteration you will of course need to change the current evaluation point like so:

$$x_k = x_{k-1} - \lambda g(x_{k-1})$$

The function should return `xoptimal`, which is the optimal point (a numpy array of course), `foptimal`, which is the function value at `xoptimal`, and `niterations`, which is the number of iterations the algorithm made to reach the stopping criterion.

For our purposes we will use a function with two parameters for optimization. This means that the gradient function will return two values (one for d/dx and one for d/dy).

When you call the function, it should also create a surface plot and plot the successive points of the optimization like so (you will need to look up how to do this in matplotlib ^^):



Part2 Example function (10 points):

Test your gradient descent function with f(x,y) = x**2 + x*cos(x*y/3) + 3*y**2, `xstart = (10,10), lambda = .03, tolerance = 1e-8, maxiter = 5000, doplot=True`

You will need to implement `g` yourself by hand and then supply this to the function. What is `xoptimal, foptimal, niterations`?

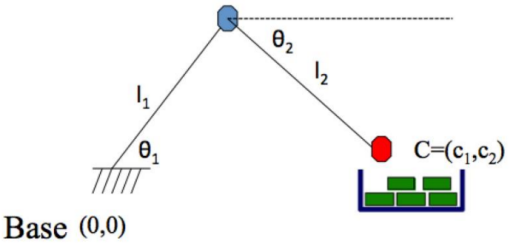
What is the value of `lambda` for which this will NOT converge? Test this out in your notebook and find the smallest `lambda` for which this happens. Likewise, what is the largest value of `lambda` for which this converges? Insert all values into your notebook.

Part3 Robot arm (20 points):

Next, we solve the problem from class, trying to optimize the position of a robot arm, based on an initial starting point.

Find the joint angles θ that minimizes the distance between the character position and user specified position

$$\arg\min_{\theta_1,\theta_2}(l_1\cos\theta_1 + l_2\cos(\theta_2) - c_1)^2 + (l_1\sin\theta_1 - l_2\sin(\theta_2) - c_2)^2$$



In all following parts, make sure to use the degree versions of sine and cosine functions in numpy!

Define the function and gradient so that you can call `gradient_descent`.

Draw the configuration of the arms into a plot, using Θ_1 = 45 deg, Θ_2 = 45 deg, l_1 = 20, l_2 = 30 . The first joint should be located at [0,0] – the x- and y-limits of the plot should be [-50 50], [-50 50].

Select a point the arm has to be able to reach and run gradient descent. Plot this point, and the end position into another plot to verify that your setup worked.

Next, make a grid of desired end points covering x and y in steps of 2 from -50:50:2. Run this with `doplot=False`, recording all outputs.

Make three 2D plots that show the outputs `xoptimal`, `foptimal`, `niterations` in a nice format in the x-y plane. **Think about a nice way to visualize this**.

Discuss the outputs - what can you say, for example, for the starting points that the arm cannot reach? How did the gradient_descent optimize then? Does this make "sense"?