Assignment #5

This assignment is due on **May 26th, 11:59:59PM** via email to mailto:christian.wallraven+AMS2019@gmail.com.

If you are done with the assignment, make one zip-file of the `assignment5` directory and call this zip-file `STUDENTID1_STUDENTID2_STUDENTID3_A5.zip` (e.g.: `2016010000_2017010001_A5.zip` for a team consisting of two students or `2016010000_2017010001_2017010002_A5.zip` for a three-student team). The order of the IDs does not matter, but the correctness of the IDs does! **Please double-check that the name of the file is correct!!**

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

ALSO: I can also surf on the internet for code. Downloading and copying and pasting other peoples' code is plagiarism and will NOT be tolerated. Please also clearly state whom you worked with (if applicable)!

Part1 Gradient descent (60 points):

Your task is to implement and test the gradient descent algorithm that looks for the minimum of a function $f:\Re^n \rightarrow \Re$, given its gradient $\vec{g} = \nabla f = \begin{pmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \dots \\ \partial f/\partial x_n \end{pmatrix}$. Note that the function f takes a vector as an input and returns a number as an output, whereas the gradient takes a vector as input and returns a vector as output! Please take a look at the lecture slides on gradient descent again to familiarize yourself with the idea and the algorithm.

Note that gradient descent is an iterative algorithm. You need to choose a starting value for optimization x_0. Then you're trying to shoot in the direction of the current gradient of the function. That is, your next x-value is given by

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

Obviously, you need to choose λ carefully, otherwise strange things might happen. In addition, the algorithm needs a stopping criterion. We will be supplying two criteria here. In addition, the function $f:\Re^2 \rightarrow \Re$, so that the gradient will contain two parts. Hence, the function definition in Matlab should look like the following:

```
function [xoptimal,foptimal,niterations] = gradient_descent
(f,g1,g2,xstart,lambda,tolerance,maxiterations)
```

where `f` is a **function handle** to the definition of the function, `g1,g2` are **function handles** to the definition of the partial derivatives of the function f, `xstart` is an array of starting values for the optimization (you will need 2 coordinates of course), `lambda` is the update rate in the gradient descent step, `tolerance` is the value of the **squared norm of the gradient** at which you accept that the minimum is found (it should be $||g||^2_{L2} = 0$, but for numerical stability, usually a small, non-zero value is chosen), and `maxiterations` is the maximum number of iterations that you allow to take place.

If you don't remember what function handles are, please look them up in Matlab help. Briefly, you can define a new function `f` in Matlab by writing:

```
f = @(x1,x2) x1.^2 + x2.^2;
```

If you want to call this function, you need to write:

```
f(1,2)

ans =

     5
```

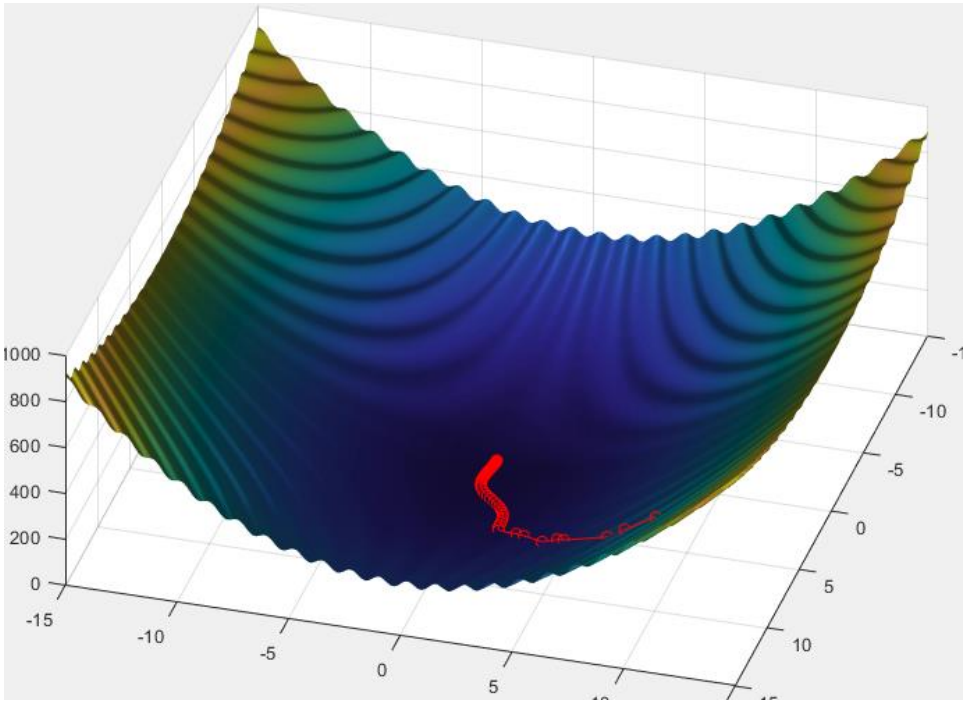
Similarly, you can pass this new function as a parameter to another function as a function handle. You can therefore simply call the above-defined function by writing

```
gradient_descent(f,…)
```

Now let's go back to our function definition. The function should return `xoptimal`, which is the optimal point (an array of course), `foptimal`, which is the function value at `xoptimal`, and `niterations`, which is the number of iterations the algorithm made to reach criterion. When you call the function, it should create a surface plot (use `ezsurf` or `surf`) and plot the successive points of the optimization as shown during class. For plotting the point you need to save the previous and current point in variables, and use `plot3` to draw a line like so:

```
plot3([prevx currx],[prevy curry],[f(prevx,prevy)
f(currx,curry)],'ro-');
```

It should look (roughly) like this (for the function defined below):



a) Now we're going to test your algorithm.

Use

```
f = @(x1,x2) x1.^2 + x1.*cos(x1.*x2/3) + 3*x2.^2;

xstart = [10 10]'; lambda = 0.03; tolerance = 1e-8, maxiter =
1000;
```

Now define the suitable gradient function. **Do NOT use the numerical gradient, but use your brain to actually derive the function!!**

This gives you two partial derivatives that you will use in your gradient descent function:

```
g1 = @(x1,x2) …

g2 = @(x1,x2) …
```

Write a script `simpleTest.m` that calls the function and outputs the results for `[xoptimal,foptimal,niterations]`

We chose for updating each step `lambda = 0.03`.

In class I showed you that if you make too large steps, the optimization will not converge. What is the value of `lambda` **up to two digits** that will result in non-convergence? Try this out yourself and insert the value of `lambda` into the script.

Obviously, if you make larger steps, you could reach the minimum faster. What is the value of lambda that has the minimum number of steps in order to reach the minimum point? Try this out yourself and insert the value of `lambda` into the script.

------ See next page for part b) -------

b) Solve the problem from class shown in the picture below. For this, you need to write a new script that defines a new function and new partial derivatives. **Note that you will need to use `sind` and `cosd` if you want angles to be calculated in degrees in Matlab!!**

Draw the configuration of the arms into a plot, using $\theta_1 = 45\,\text{deg}, \theta_2 = 45\,\text{deg}, l_1 = 20, l_2 = 30$. The first joint should be located at [0,0] – the x- and y-limits of the plot should be [-50 50], [-50 50].

Then get the user's input for the desired point `[c1,c2]` by using `[c1,c2]=ginput(1),` which will allow the user to click one point in the plot.

Then run your optimization and plot the final configuration of the arm into the plot.

Make another surf-plot of the optimization function f and see how different this looks from the previous surface of part a).

Does you arm always react as it should? For example, when you click on a point it CANNOT reach?

Insert your observations as comment into the script.

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

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