Assignment #6

This assignment is due on **May 12th one hour before class** via email to mailto:wallraven@korea.ac.kr. Please observe the directory structure in this zip-file: solutions for `part1` should go into the `part1` directory, etc.

If you are done with the assignment, make one zip-file of the `assignment5` directory and call this `<LASTNAME_FIRSTNAME_A5.zip>` (e.g.: `HONG_GILDONG_A5.zip`).

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 who 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.

a) Implement the gradient descent 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. Hence, the function definition in Matlab should look like the following:

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

where `f` is a function handle to the definition of the function, `g` is a **function handle** to the definition of the gradient of the function f, `xstart` is an array of starting values for the optimization, `lambda` is the update rate in the gradient descent step, `tolerance` is the value of the gradient at which you accept that the minimum is found (it should be g=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, `foptimal`, which is the function value at `xoptimal`, and `niterations`, which is the number of iterations the algorithm made to reach criterion.

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

Use

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

xstart = [10 10]'; lambda = 0.2; tolerance = 1e-6, maxiter = 1000;
```

Now define the suitable gradient function.

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

Now use `lambda = 0.4;`

What happens now? Why? Insert your observations as comments into the script.

c) Make a new function gradient_descent2 based on gradient_descent, which will work only for $f:\Re^2 \to \Re$ and which will plot the result of the optimization process. For this, use

```
f = @(x,y) -y.*exp(-x.^2-y.^2);
```

define a suitable gradient function and plot the result of the optimization into a figure of the function generated with the command `ezsurf(f,[-3 3 -3 3])`.

At each step, the algorithm should plot the current **x**,f(**x**) into the figure, and connect the points with a red line, such that you can follow the path to the optimization point.

Using

```
lambda = 0.2; tolerance = 1e-6, maxiter = 1000;
```

find a good starting point, and save the resulting plot as a PNG-file.

(If you want, you can change the code to also use the function from part b), such that you can follow the steps there ☺)