Coursework 1
Mathematics for Machine Learning (70015)

This coursework is a coding assignment. The python code you submit must compile on a standard CSG Linux installation.

You are not permitted to use any symbolic manipulation libraries (e.g. sympy) or automatic differentiation tools (e.g. pytorch, tensorflow) for your submitted code (though, of course, you may find these useful for checking your answers). Your code will be checked for imports. Note that if you use python you should not need to import anything other than the packages that are already imported in the template code (numpy) for the submitted code for this assignment.

You are given a framework containing the following three files:

`solution.py`	Provide your solution to the coursework questions by filling in the missing bits in this python file.
`test.py`	Run `python test.py` to check if your solutions are correct*.
`plot.py`	This file can be useful to visualize results. For example run: `python plot.py --function 2 --gradient fd` to plot gradient-descent on function f_2 using finite-difference gradients. Or run: `python plot.py --function 3 --gradient exact` to plot gradient descent on function f_3 using exact gradients.

In summary, you are required to submit the python file named `solution.py` including your changes.

1 Differentiation

In this question, we define the following constants:

$$\boldsymbol{B} = \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix}, \qquad \boldsymbol{a} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

We define also the following functions, which are all $\mathbb{R}^2 \to \mathbb{R}$

$$\begin{aligned}
f_1(\boldsymbol{x}) &= \boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{a}^T\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} \\
f_2(\boldsymbol{x}) &= \cos\big((\boldsymbol{x}-\boldsymbol{b})^T(\boldsymbol{x}-\boldsymbol{b})\big) + (\boldsymbol{x}-\boldsymbol{a})^T\boldsymbol{B}(\boldsymbol{x}-\boldsymbol{a}) \\
f_3(\boldsymbol{x}) &= 1 - \big(\exp\big(-(\boldsymbol{x}-\boldsymbol{a})^T(\boldsymbol{x}-\boldsymbol{a})\big) + \exp\big(-(\boldsymbol{x}-\boldsymbol{b})^T\boldsymbol{B}(\boldsymbol{x}-\boldsymbol{b})\big) - \frac{1}{10}\log|\frac{1}{100}\boldsymbol{I} + \boldsymbol{x}\boldsymbol{x}^T|\big)
\end{aligned}$$

Throughout this exercise, we remain consistent in our convention of using row vectors for our gradients (ie. $\frac{\partial f_1}{\partial x}, \frac{\partial f_2}{\partial x}, \frac{\partial f_3}{\partial x} \in \mathbb{R}^{1\times 2}$).

a) **[2 marks]** Write a function `f1_check_minimum(B, a, b)` that checks whether function f_1 has a minimum given certain values $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{B}$.
Hint: you may not be required to use all three arguments.

b) Write functions to calculate gradients:

b.1) **[4 marks]** Remember (animation in lectures) that a gradient is found by taking

$$\lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{1}$$

We can approximate this by calculating the expression for a small but finite Δx, which is known as the *finite-differences* approximation.
Write a function `grad_fd(fn, x)` that returns the gradients of function `fn` at point `x`, using finite-differences. Use `delta=1e-5`.
The function should take a column numpy (2, 1) vector for 'x' as input, and output a numpy (1, 2) row vector for the gradient.

b.2) **[6 marks]** Write the functions `f1_grad(x)`, `f2_grad(x)` and `f3_grad(x)` that return gradients of f_1, f_2 and f_3, using your own derivations.
The functions should take a column numpy (2, 1) vector for 'x' as input, and output a numpy (1, 2) row vector for the gradient.

c) **[4 marks]** Use your gradients to implement a gradient descent algorithm with 50 iterations to find a local minimum for both f_2 and f_3, by finishing the function `grad_descent(fn, grad_fn)`.

Try plotting the optimization trajectories for the different functions. You can use `plot.py`, but you may need to alter the plotting range on the x- and y-axis. Experiment with different default values `start_x`, `start_y` and `lr` of the `gradient_descent` function. Can you find converging and diverging behaviour?

d) **[3 marks]** Find a good initial location and learning rate that finds good minima on both f_2 and f_3. Use the `plot.py` file to plot trajectories for both functions. Set the default values for *start_x*, *start_y* and *lr* of the `gradient_descent` function to your found values.

*It can be helpful to use the public unit tests and check your solution by running `python test.py`. A separate more extensive set of tests will be used for grading. The public tests do not cover *every* aspect of the question that will be grated. However, if you thought through your answers carefully, rather than develop to the unittest, you should pass the private tests as well.

Details on how to hand in answers will be communicated soon. This will be done via LabTS and CATE, to allow for automatic grading. Only a completed `solutions.py` will be needed for grading.