TME 2 - Gradient Descent

The goal of this TME is to experiment with gradient descent in the context of linear regression and logistic regression. Throughout the session, the input space has d dimensions, the labels are real numbers (for linear regression) or in $\{-1, 1\}$ (for logistic regression), the functional search space is parameterized by a weight vector $\mathbf{w} \in \mathbb{R}^d$, and n denotes the number of examples. We will not consider bias in this TME (no weight $\mathbf{w}_0$).

Attention !
All your functions in this session should be able to take matrices of examples and label vectors as input—not just a single example and label. We will follow these conventions : $X \in \mathbb{R}^{n,d}$, $\mathbf{w} \in \mathbb{R}^{d,1}$, $Y \in \mathbb{R}^{n,1}$.
There are some pitfalls when manipulating matrices with `numpy` :

- The operator $*$ allows element-wise multiplication for matrices of the same dimensions, but sometimes it performs matrix multiplication when the matrices have compatible sizes (e.g., $(1, d)$ and $(d, 1)$).
- The operator `ndarray.dot()` performs matrix multiplication.
- Avoid loops as much as possible (no loops are required in the implementation of cost functions and gradients !). Python is very slow in such cases...
- Sometimes you will pass a matrix as input, sometimes a vector (e.g., when selecting a single row of examples), and the operators will behave differently depending on the case... Be sure to transform your inputs at the beginning of all your functions to avoid bugs (e.g., `y = y.reshape(-1,1); w = w.reshape(-1,1); x = x.reshape(y.shape[0],w.shape[0])`).
- Use `np.sign` and `np.maximum`.

Implementation of Cost Functions

Implement (without using any loops ! Each function should take 1-2 lines) :

- A function `mse(w,x,y)` that returns the mean squared error cost for a linear function parameterized by `w` on the data `x` (size n, d) and the labels `y`. Your function should output the cost as a matrix of size $n, 1$ (the cost for each example).
- A function `reglog(w,x,y)` that returns the logistic regression cost.
- The functions `mse_grad(w,x,y)` and `reglog_grad(w,x,y)` that return the gradients of the mean squared error and logistic regression as matrices of size n, d.

You can test your functions with the function `check_fonctions()`.

Bonus : Recall that the first-order Taylor expansion of a multivariate function is given by :

$$f(\mathbf{x}) = f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)\nabla f(\mathbf{x}_0) + O(\|\mathbf{x} - \mathbf{x}_0\|^2)$$

Using this expansion, write a function `grad_check(f,f_grad,N=100)` to test the accuracy of your gradient functions. This function should randomly draw N points and verify the gradient computation for these N points on average. Start with the 1-dimensional case, and if time permits, extend it to d-dimensional cases.

Gradient Descent

Implement a function `descente_gradient(datax,datay,f_loss,f_grad,eps,iter)` that performs gradient descent to optimize the cost `f_loss` (whose gradient is given by `f_grad`) on the data `datax` and labels `datay`, with a learning rate of `eps` and `iter` iterations. Your function should return the optimal parameter `w` found, the list of `w`, and the cost function values over the iterations.

Experiments

In the file `mltools.py`, you will find a function `gen_arti(nbex=1000,data_type=0,epsilon=0.02)` that generates datasets with `nbex` points of types : 2 Gaussians (`data_type=0`), 4 Gaussians (`data_type=1`), and a checkerboard (`data_type=2`) with noise `epsilon`. Additional functions include :

- `plot_data(data,labels)` to visualize the data ;
- `plot_frontiere(data,f,step)` to plot the decision boundaries of function `f` for a 2D problem by discretizing the space into `step` intervals ;
- `make_grid(data,xmin,xmax,ymin,ymax,step)` to construct a discretized grid of a 2D space (with boundaries defined by the minimum and maximum values of `data` or the provided parameters).

Examples of usage are provided in the TME code skeleton.
Test your gradient descent implementation on the two-Gaussian problem. Compare the results between linear regression and logistic regression. Visualize the decision boundaries and plot the cost evolution over iterations. What happens when the learning rate is increased or decreased significantly ? Analyze the behavior for a separable problem and for a non-separable problem (e.g., with significantly increased noise).
Visualize the cost function in the weight space for the two dimensions of the problem (see the TME code skeleton). Plot the trajectory followed by the optimization algorithm (the `w` values over iterations) on the same graph.
Experiment with other types of artificial data.