Exercise 1: Risk Minimization and Gradient Descent

You want to estimate the relationship between a continuous response variable $\boldsymbol{y} \in \mathbb{R}^n$ and some feature $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ using the linear model with an appropriate loss function L.

(a) Describe the model f used in this case, its hypothesis space $\mathcal{H}$ and the theoretical risk function.

(b) Given $f \in \mathcal{H}$, explain the different parts of the Bayes regret if (i) $f^* \in \mathcal{H}$; if (ii) $f^* \notin \mathcal{H}$.

(c) Define the empirical risk and derive the gradients of the empirical risk.

(d) Show that the empirical risk is convex in the model coefficients. Why is convexity a desirable property? Hint: Compute the Hessian matrix $\boldsymbol{H} \in \mathbb{R}^{p \times p}$ and show that $\boldsymbol{z}^\top \boldsymbol{H} \boldsymbol{z} \geq 0 \, \forall \boldsymbol{z} \in \mathbb{R}^p$, i.e., show that the Hessian is positive semi-definite (psd).

(e) Write a function implementing a gradient descent routine for the optimization of this linear model. Start with:

```r
#' @param step_size the step_size in each iteration
#' @param X the feature input matrix X
#' @param y the outcome vector y
#' @param beta a starting value for the coefficients
#' @param eps a small constant measuring the changes in each update step.
#' Stop the algorithm if the estimated model parameters do not change
#' more than \code{eps}.

#' @return a set of optimal coefficients beta
gradient_descent <- function(step_size, X, y, beta, eps = 1e-8){

  # >>> do something <<<

  return(beta)

}
```

(f) Run a small simulation study by creating 20 data sets as indicated below and test different step sizes α (fixed across iterations) against each other and against the state-of-the-art routine for linear models (in R, using the function `lm`, in Python, e.g., `sklearn.linear_model.LinearRegression`).

- Compare the difference in estimated coefficients $\beta_j, j = 1, \ldots, p$ using the mean squared error, i.e.

$$p^{-1} \sum_{j=1}^{p} (\beta_j^{truth} - \hat{\beta}_j)^2$$

  and summarize the difference over all 100 simulation repetitions.

- Compare the run times of your implementation and the one given by the state-of-the-art method by wrapping the function calls into a timer (e.g., `system.time()` in R).

```r
# settings
n <- 10000
p <- 100
nr_sims <- 20
```

```r
# create data (only once)
X <- matrix(rnorm(n*p), ncol=p)
beta_truth <- runif(p, -2, 2)
f_truth <- X%*%beta_truth

# create result object
result_list <- vector("list", nr_sims)

for(sim_nr in nr_sims)
{

  # create response
  y <- f_truth + rnorm(n, sd = 2)

  # >>> do something <<<


  # save results in list (performance, time)
  result_list[[sim_nr]] <- add_something_meaningful_here

}
```

(g) Why is gradient descent maybe not the best option for optimization of a linear model with $L2$ loss? What other options exist? Name at least one and describe how you would apply this for the above problem.

(h) Can we say something about the algorithm's consistency w.r.t. $\mathbb{P}_{xy}$, if $f^* \notin \mathcal{H}$?