Optimisation Methods: Assignment 5

Deborah Sulem
deborah.sulem@usi.ch

Spring 2025

The purpose of this assignment is to familiarise yourself with the stochastic gradient algorithm and further exercise your skills in unconstrained optimisation.

It is due for **Wednesday 30th April at 2 pm**. Only the second part (Python implementation) will be graded but you can also submit your answers for the first part and the latter will also be corrected (though not graded). For the first part, you need to submit a Python notebook on iCorsi and follow these instructions:

- Put the answers for each part of the question into separate cells.
- Before each cell, put a markdown header that says which part of the question comes in the following cell.
- Good coding style is part of the grade: add clear comments throughout the code when it is necessary.
- Before you submit your notebook, make sure it runs.

Part 0: reading

The book "Numerical Optimisation" (Nocedal & Wright) does not cover the SGD algorithm. However I recommend re-reading Chapters 1-3 on line search algorithms, which should read easier now.

Part 1: written exercises

Exercise 1
Consider the affine function

$$f(x) = c^T x + b = \sum_{i=1}^{n} c_i x_i + b.$$

with $x \in \mathbb{R}^n, c \in \mathbb{R}^n, b \in \mathbb{R}$. For which values of x, b and c are the necessary optimality conditions verified?

Exercise 2
Consider the bivariate function

$$f(x_1, x_2) = \frac{1}{2}x_1^2 + x_1 \cos x_2.$$

Use the optimality conditions to identify the minima of this function.

Exercise 3
Consider the function

$$f(x) = \frac{9}{2}x_1^2 + \frac{1}{2}x_2^2.$$

and the point $x^{(0)} = (1, 1)^T$.

1. *Write the second-order Taylor approximation at* $x^{(0)}$.
2. *Calculate Newton's direction at* $x^{(0)}$.
3. *Perform one iteration of Newton's algorithm at* $x^{(0)}$ *and find the new value of* f.

Part 2: programming problems

Problem 1 (Stochastic Gradient Descent)
In this problem we will consider training (fitting) a linear regression model to predict a person's weight from their height. The linear regression model is the following:

$$\textit{weight} \approx x_1 + x_2 \times \textit{height}.$$

with $x_1, x_2 \in \mathbb{R}$*, respectively called the intercept and slope of the linear model. Denoting the weight (outcome) variable* y*,* h *the height variable and* $z = (1, h)^T$*, the linear regression model can be re-written as*

$$y \approx x^T z$$

with parameter $x = (x_1, x_2)^T$*. To fit this model we will use a dataset* `height_weight_genders.csv`*, which contains* N *vectors corresponding to the gender, height, and weight of individuals. Given this data, we try to find* x *that minimises the prediction error of the linear model:*

$$f(x) = \frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - x^T z^{(i)})^2 = \frac{1}{2N}\sum_{i=1}^{N} f_i(x)$$

with $f_i(x) = (y^{(i)} - x^T z^{(i)})^2$*. Note that here we used the standard squared loss and the factor* $\frac{1}{2}$ *is simply used to have a simpler expression of the gradient. We can stack all data vectors* $z^{(i)}$ *row-wise into a* $N \times 2$ *matrix* Z*, i.e.,*

$$Z = \begin{pmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(N)} \end{pmatrix}$$

Note that the first column of Z contains only 1 values. We call Z the data matrix. Moreover, we denote by $Y = (y^{(1)}, y^{(2)}, \dots, y^{(N)}$ the vector of length N containing all the outcome variables. With this notation we can re-write $f(x)$ as

$$f(x) = \frac{1}{2N}\|Y - Zx\|^2,$$

where $\|\cdot\|$ is the Euclidean norm. To minimiser f, we will use the Gradient Descent (GD) algorithm, the Stochastic Gradient Descent (SGD) algorithm and its variant with mini-batching.

1. *Import the dataset from the file* `height_weight_genders.csv` *in an array (dimension $N \times 3$) where each data vector corresponds to a row.* Hint: since the data contains string values (gender), you may first import the data into a Pandas dataframe from the Pandas library with the function `pandas.read_csv`, then create a Numpy array where the first column contains the gender encoded into 0/1 values (1 = Female, 0 = Male), the second column contains the height and the third column contains the weight.

2. *Compute the mean μ and standard deviation σ of the height column, then replace each height value $h^{(i)}$ in the array by its standardised version:*

$$\frac{h^{(i)} - \mu}{\sigma}.$$

3. *Create a new array corresponding the data matrix Z (recall that its first column contains 1 while the second column contains the height values) and the vector Y containing all the weight values.*

4. *Write a function* `objective` *which computes the value of the prediction error f for any input (y, Z, x) where $x \in \mathbb{R}^2$.*

5. *Write a function* `gradient` *which computes the gradient vector of f for any input (y, Z, x). Check that*

$$\nabla f(x) = \frac{1}{N} Z^T(Zx - Y) = \frac{1}{N}\sum_{i=1}^{N}(x^T z^{(i)} - y^{(i)})z^{(i)} = \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(x).$$

6. *Show that f is μ-strongly convex and find such μ for our data set.* Hint: $\nabla^2 f(x) = \frac{1}{N} Z^T Z$ and you may use the function `numpy.linalg.eigh` to find eigenvalues.

7. *When the matrix $Z^T Z$ is invertible (which is the case here), the global minimiser x^* of f can be obtained by solving the linear equations:*

$$Z^T Z x^* = Z^T y.$$

   Compute the global minimiser x^* and the optimal value of the objective $f(x^*)$ and print it.

8. *We now consider the problem of computing a stochastic gradient estimate at some point $x^{(0)} = (10, 2)$. Compute one such estimate:*

$$g^{(0)} = \nabla f_{i^{(0)}}(x^{(0)})$$

   where $i^{(0)}$ is a randomly chosen index in $\{1, \dots, N\}$.

9. *We want to empirically check that* $g^{(0)}$ *is an unbiased estimate of* $\nabla f_{i^{(0)}}(x^{(0)})$*. For this, compute 10000 times a gradient estimate as in the previous question, average the obtained estimates, and check that the average is close to* $\nabla f_{i^{(0)}}(x^{(0)})$ *(with an even higher number of estimates you should see the error decreasing).*

10. *We consider computing a stochastic gradient estimate with mini-batching*

$$\bar{g}^{(0)} = \frac{1}{m}\sum_{j=1}^{m} \nabla f_{i^{(j)}}(x^{(0)})$$

where m *is the mini-batch size and* $i^{(1)}, i^{(2)}, \ldots, i^{(m)}$ *are random indices sampled uniformly from* $\{1, \ldots, N\}$ *without replacement. Write a function* `stochastic_gradient` *which compute a stochastic gradient estimate with mini-batching for any input* (y, Z, x, m)*.*

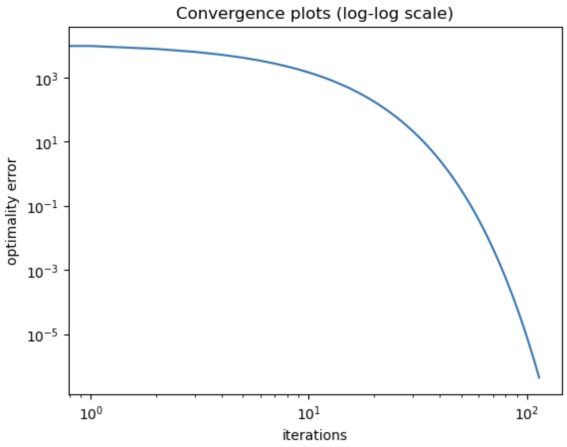
11. *Write (or re-use) a function that runs GD with a fixed step size and saves the values of* $f(x^{(k)})$ *at each iteration, and apply it to minimise* f

    - *starting from* $x^{(0)} = (10, 2)$*,*
    - *for a maximum number of iterations of* 10^4 *and tolerance level* 10^{-4}*,*
    - *with step size (learning rate)* $\alpha = 1$*. Compute the running time of GD and print the final optimality error*

$$f(x^{(K)}) - f(x^*)$$

      at the last iteration.

    - *Plot the optimality error* $f(x^{(k)}) - f(x^*)$ *versus the iteration index* k *in log-log scale (to better visualise the convergence) for GD. Your plot should look like this:*



12. *Write a function* `stochastic_gradient_descent` *that runs SGD with mini-batching and a fixed learning rate and saves the values of* $f(x^{(k)})$ *at each iteration.*

13. *Apply the function* `stochastic_gradient_descent` *to minimise* f *with the same value of initial point, learning rate, max number of iterations, tolerance level, and*

with $m = 1$ (plain SGD). As for GD, compute its running time and print the final optimality error

$$f(x^{(K)}) - f(x^*)$$

at the last iteration.

14. *Plot the optimality error $f(x^{(k)}) - f(x^*)$ versus the iteration index k in log-log scale (to better visualise the convergence) for plain SGD and comment the results.*

15. *Repeat the 2 previous questions with $m = 10$ (SGD with minibatching) and comment the results. Remark: the computational time of SGD might be higher than for GD here because we ask to compute the objective value at each iteration, which we would not need to do in practical settings.*

16. *To improve the convergence of SGD, we consider using vanishing step sizes. More precisely we consider using*

$$\alpha^{(k)} = \frac{1}{k+1}, \qquad k \geq 0. \tag{1}$$

Write a function `better_stochastic_gradient_descent` *that implements SGD with mini-batching and learning rates as in* (1).

17. *Run the function* `better_stochastic_gradient_descent` *with the same hyperparameters as before, compute its running time, print the final optimality error, and plot the optimality error $f(x^{(k)}) - f(x^*)$ versus the iteration index k in log-log scale. Comment the results.*

18. *Finally we want to check the theoretical (sublinear) convergence rate:*

$$f(x^{(k)}) - f(x^*) = \mathcal{O}(\frac{1}{k}).$$

Plot (in dotted line) the line $\frac{10}{k}$ on top of the previous plot. Your plot should look like this: