Mathematics for Machine Learning - Coursework 1

Constantin Eulenstein

October 2020

1 Exercise 1a)

The idea is to expand $f_1(x) = \mathbf{x}^\top\mathbf{B}\mathbf{x} - \mathbf{x}^\top\mathbf{x} + \mathbf{a}^\top\mathbf{x} - \mathbf{b}^\top\mathbf{x}$ as well as $(\mathbf{x}-\mathbf{c})^\top\mathbf{C}(\mathbf{x}-\mathbf{c}) + c_0$ to find $\mathbf{C}$, $\mathbf{c}$ and c_0 through the comparison of coefficients.
$f_1(x) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} =$
$3x_1^2 - 4x_1x_2 + 3x_2^2 + 2x_1$

The completed square form is equal to the following:
$\begin{bmatrix} x_1 - c_1 & x_2 - c_2 \end{bmatrix} \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} x_1 - c_1 \\ x_2 - c_2 \end{bmatrix} + c_0 = C_{11}x_1^2 + (C_{12}+C_{21})x_1x_2 + C_{22}x_2^2 - (2C_{11}c_1 + (C_{12} + C_{21})c_2)x_1 - ((C_{12} + C_{21})c_1 + 2C_{22}c_2)x_2 + C_{11}c_1^2 + (C_{12} + C_{21})c_2c_1 + C_{22}c_2^2 + c_0$

Coefficient comparison gives the following equations:

$$3 = C_{11} \tag{1}$$

$$-4 = (C_{12} + C_{21}) \tag{2}$$

$$3 = C_{22} \tag{3}$$

$$2 = -(2C_{11}c_1 + (C_{12} + C_{21})c_2) \tag{4}$$

$$0 = -((C_{12} + C_{21})c_1 + 2C_{22}c_2) \tag{5}$$

$$0 = C_{11}c_1^2 + (C_{12} + C_{21})c_2c_1 + C_{22}c_2^2 + c_0 \tag{6}$$

Solving this system of equations gives: $\mathbf{C} = \begin{bmatrix} 3 & C_{12} \\ C_{21} & 3 \end{bmatrix}$ where $(C_{12} + C_{21}) = -4$, $\mathbf{c} = \begin{bmatrix} -3/5 \\ -2/5 \end{bmatrix}$

and $c_0 = -3/5$. To obtain a $\mathbf{C}$ that is symmetric, one can choose C_{12} and C_{21} to be the following:
$\mathbf{C} = \begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix}$

2 Exercise 1b)

The Hessian matrix is $\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f_1}{\partial x_1^2} & \frac{\partial^2 f_1}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f_1}{\partial x_2 \partial x_1} & \frac{\partial^2 f_1}{\partial x_2^2} \end{bmatrix}$. Let's first compute $\frac{df_1}{d\mathbf{x}}$. From before we know that $f_1(x) = 3x_1^2 - 4x_1x_2 + 3x_2^2 + 2x_1$. Consequently, $\frac{df_1}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 6x_1 - 4x_2 + 2 & 6x_2 - 4x_1 \end{bmatrix}$ and $\mathbf{H} = \begin{bmatrix} 6 & -4 \\ -4 & 6 \end{bmatrix}$. Calculating the eigenvalues of the Hessian matrix, one obtains $\lambda_1 = 2$ and $\lambda_2 = 10$. Those are both larger than 0 and, consequently the point where $\frac{df_1}{d\mathbf{x}} = \mathbf{0}^\top$ holds, is going to be a minimum. Solving this equation one obtains a minimum at $\mathbf{x_{min}} = \begin{bmatrix} -\frac{3}{5} \\ -\frac{2}{5} \end{bmatrix}$. Using this point we get the minimum value for f_1: $f_1(\mathbf{x_{min}}) = 1.32$

3 Exercise 1c)

For the gradients the derivative of the completed square form is very useful:
$\frac{\partial \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial x_k} = \sum_{j=1}^n \frac{\partial x_j}{\partial x_k} \sum_{i=1}^n x_i A_{ji} + \sum_{j=1}^n x_k \sum_{i=1}^n \frac{\partial x_i}{\partial x_k} A_{ji} = \sum_{i=1}^n x_i A_{ki} + \sum_{j=1}^n x_j A_{jk}$
Arranging these partial derivatives in a row leads to: $\frac{d\mathbf{x}^\top \mathbf{A}\mathbf{x}}{d\mathbf{x}} = \mathbf{x}^\top(\mathbf{A} + \mathbf{A}^\top)$
I will use this property multiple times to compute the derivatives of the completed square forms in f_2 and f_3.

$f_2(x) = cos(z(\mathbf{x})) + (\mathbf{x} - \mathbf{a})^\top \mathbf{B}(\mathbf{x} - \mathbf{a})$

, where $z(\mathbf{x}) = (\mathbf{x} - \mathbf{b})^\top(\mathbf{x} - \mathbf{b})$

Therefore,

$\frac{df_2}{d\mathbf{x}} = \frac{\partial f_2}{\partial z}\frac{\partial z}{\partial \mathbf{x}} + (\mathbf{x} - \mathbf{a})^\top(\mathbf{B} + \mathbf{B}^\top) = -sin(z(\mathbf{x}))2(\mathbf{x} - \mathbf{b})^\top + (\mathbf{x} - \mathbf{a})^\top(\mathbf{B} + \mathbf{B}^\top)$

This, in turn leads to the following partial derivatives:

$\frac{\partial f_2}{\partial x_1} = -(2x_1 + 4)sin((x_1 + 2)^2 + (x_2 - 2)^2) + 8x_1 - 4(x_2 - 1)$

$\frac{\partial f_2}{\partial x_2} = -(2x_2 - 2)sin((x_1 + 2)^2 + (x_2 - 2)^2) + 8x_2 - 4(x_1 + 2)$

The following for f_3:

$f_3(x) = 1 - (exp(d(\mathbf{x})) + exp(g(\mathbf{x}))) - \frac{1}{10}log(h(\mathbf{x}))$

, where $d(\mathbf{x}) = -(\mathbf{x} - \mathbf{a})^\top(\mathbf{x} - \mathbf{a})$

and $g(\mathbf{x}) = -(\mathbf{x} - \mathbf{b})^\top \mathbf{B}(\mathbf{x} - \mathbf{b})$

and $h(\mathbf{x}) = det(\begin{bmatrix} 0.01 + x_1^2 & x_1 x_2 \\ x_2 x_1 & 0.01 + x_2^2 \end{bmatrix}) = \frac{x_1^2}{100} + \frac{x_2^2}{100} + \frac{1}{100^2}$

Consequently,

$\frac{df_3}{d\mathbf{x}} = \frac{\partial f_3}{\partial d}\frac{\partial d}{\partial \mathbf{x}} + \frac{\partial f_3}{\partial g}\frac{\partial g}{\partial \mathbf{x}} + \frac{\partial f_3}{\partial h}\frac{\partial h}{\partial \mathbf{x}} = -exp(d(\mathbf{x}))(-2(\mathbf{x} - \mathbf{a})^\top) - exp(g(\mathbf{x}))(-(\mathbf{x} - \mathbf{b})^\top(\mathbf{B} + \mathbf{B}^\top)) - \frac{1}{10}log(h(\mathbf{x}))\frac{1}{h(\mathbf{x})}\begin{bmatrix} \frac{2x_1}{100} & \frac{2x_2}{100} \end{bmatrix}$

This, in turn leads to the following partial derivatives:

$\frac{\partial f_3}{\partial x_1} = 2x_1 exp(-x_1^2 - (x_2 - 1)^2) + (8(x_1 + 2) - 4(x_2 - 1))exp(-4((x_1 + 2)^2 + (x_2 - 1)^2 - (x_1 + 2)(x_2 - 1))) - \frac{x_1}{5(0.01 + (x_1^2 + x_2^2))}$

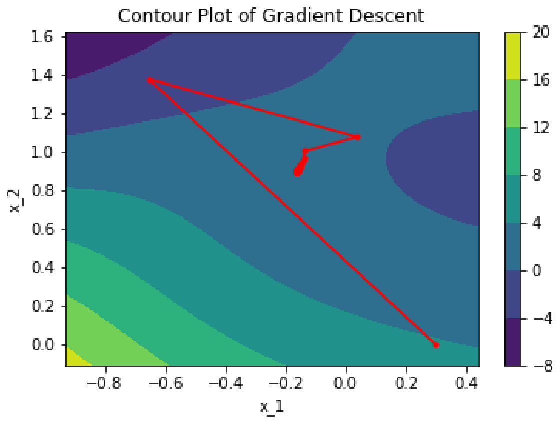
$\frac{\partial f_3}{\partial x_2} = (2x_2 - 1)exp(-x_1^2 - (x_2 - 1)^2) + (8(x_2 - 1) - 4(x_1 + 2))exp(-4((x_1 + 2)^2 + (x_2 - 1)^2 - (x_1 + 2)(x_2 - 1))) - \frac{x_2}{5(0.01 + (x_1^2 + x_2^2))}$

These partial derivatives functions both for f_1 and f_2 are calculated in the corresponding python functions gradf_2 and def gradf_3 in the file coding_answers.py. They both accept numpy (2,) array inputs and return numpy (2,) outputs.

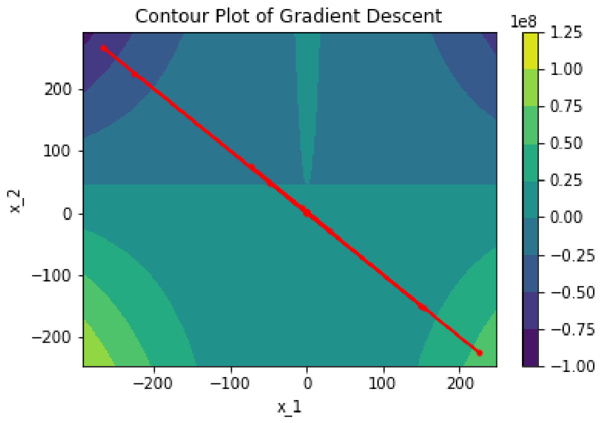
4 Exercise 1d)

First, let's examine the contour plot for function f_2. The contour plots depict the first component of $\mathbf{x}$ on the x axis and the second on the y axis. The red lines (and the dots) show the gradient steps of my algorithm over 50 iterations. For my step size, I chose a step size that reduces with each iteration, s.t. $\gamma = a\frac{1}{step}$, where step is equal to the iteration step and a is a constant factor. Also in every contour plot the gradient descent algorithm starts from point (0.3, 0).

In Figure 1 we can see how the contour plots vary depending on the step sizes. Using a step size of $\gamma = 0.15\frac{1}{step}$ results in minimum value at $\mathbf{x_{min}} = \begin{bmatrix} -0.16201 \\ 0.89819 \end{bmatrix}$ and a gradient value $\frac{df_2}{d\mathbf{x}}(\mathbf{x_{min}}) = \begin{bmatrix} 0.00974 & 0.02829 \end{bmatrix}$. A step size of $\gamma = 1\frac{1}{step}$ yields a minimum value at $\mathbf{x_{min}} = \begin{bmatrix} -0.16325 \\ 0.89462 \end{bmatrix}$ and a gradient value $\frac{df_2}{d\mathbf{x}}(\mathbf{x_{min}}) = \begin{bmatrix} 1.55136e - 07 & 4.46369e - 07 \end{bmatrix}$. Consequently, one can observe that in figure (b) the results of the algorithm steps are oscillating way more, but achieving a better



(a) $f_2, \gamma = 0.15\frac{1}{step}$



(b) $f_2.\gamma = 1\frac{1}{step}$

Figure 1: Contour plots with different step sizes γ of the gradient descent algorithm applied on f_2

example. On the other hand in figure (a), one can better observe how the algorithm is changing after each step and how it is converging towards the optimum.

For f_3 we will now do something very similar. I chose again a step size that reduces with each iteration, s.t. again $\gamma = a\frac{1}{step}$. Trying out multiple step sizes, I came to the conclusion that step size $\gamma = 1\frac{1}{step}$ achieves a very good and well observable result (in the contour plot). We do not observe the same oscillation as we did with a similar step size in f_2. In Figure 2 we observe a minimum value at $\mathbf{x_{min}} = \begin{bmatrix} 0.00091 \\ 1.09151 \end{bmatrix}$ and a gradient value $\frac{df_2}{\mathbf{dx}}(\mathbf{x_{min}}) = \begin{bmatrix} 0.00166 & -0.00022 \end{bmatrix}$.

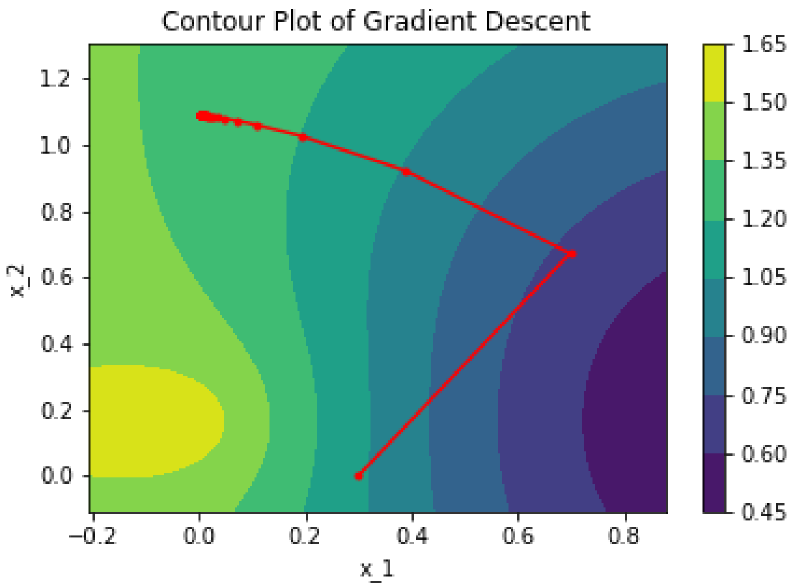


Figure 2: $f_3, \gamma = 1\frac{1}{step}$

5 Exercise 1e)

In this exercise we look at the gradient descent algorithm with constant step sizes γ, that not vary with iterations. Let's first have a look at f_2: In figure 3 I plot different values of gamma (starting from 0.01 until 0.1) to the mean absolute error (MAE) of our gradient descent algorithm (For calculating the MAE I used the optimal value of our gradient function, which is the null vector and our found gradient value $\frac{df_2}{d\mathbf{x}}(\mathbf{x_{min}})$). One can observe that our algorithm works very well for small values until approximately 0.09. Here the gradients point roughly in the same direction, while the objective improves. With γ >0.09 the algorithm performs abruptly worse and keeps worsening with rising γ. This means that a small gamma is able to approximate very well, while at some γ around 0.09 the algorithm starts oscillating widely and, consequently, overshooting. Therefore it computes a worse result with every step. It seems as if, starting at point (0.3,0) and $\gamma > 0.09$, we hit something like a resonance, which means that it starts oscillating and overshooting more and more with rising gamma. The consecutive gradients steps point always in opposite directions, which is why this rising overshoot occurs.

One can also see in the figure that there is a small rise in the beginning for very small γ. When calculating the gradient descent with very small $\gamma < 0.02$ one observes that the algorithm performs worse because the step size is too small. So it is just not enough to reach the optimum in only 50 iterations.

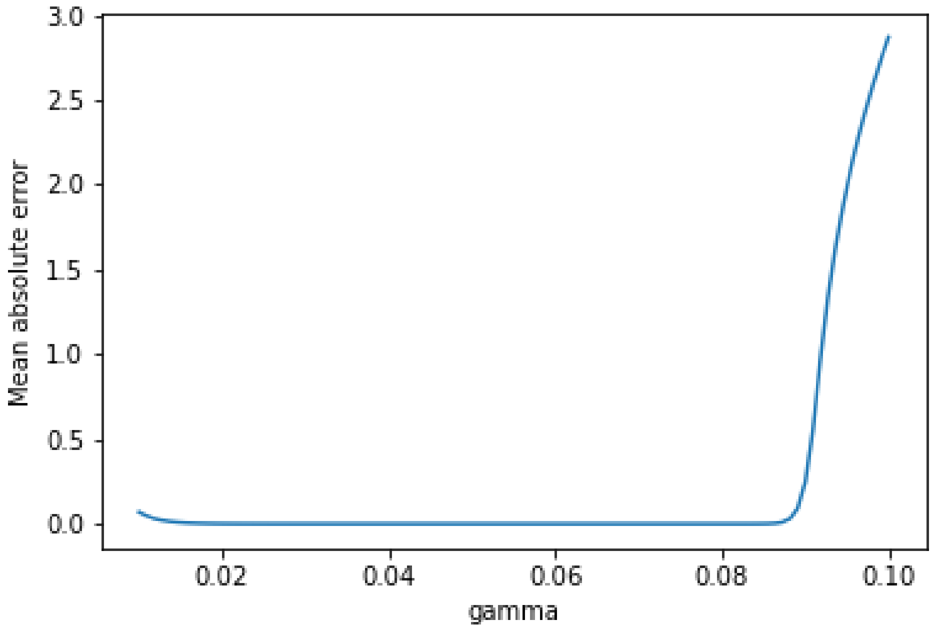


Figure 3: f_2: The absolute error of the gradient descent algorithm for f_2 dependent on γ

For function f_3 we can make a similar observation. In figure 4, one can observe, how for too small

γ and for too large γ the algorithm performs worse. For very small γ it performs even worse than for very large (however, by far not as bad as for function 2). Again, the reason is that, if the step size is too small ($\gamma < 0.06$) the algorithm can not reach towards the optimum in 50 iterations, while, if it's too large ($\gamma > 0.9$), oscillations will grow and the objective may become worse with each iteration. However, the gradient steps don't form a straight line as with f_2 since consecutive gradients don't always point in exactly alternating directions. For the right gamma

One can conclude from this exercise that γ is problem dependent and one has to perform an analysis to obtain the best value for γ.

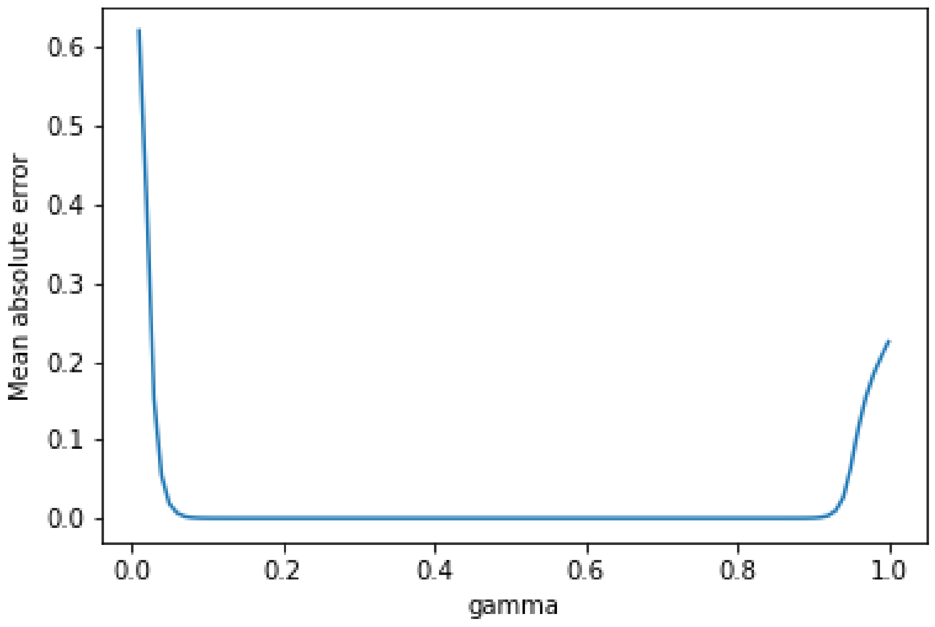


Figure 4: f_2: The absolue error of the gradient descent algorithm for f_3 dependent on γ

If we now use grossly mis-specified $\gamma > 1$, f_2 will start oscillating even crazier, and the calculated gradient of our algorithm is completely off by a factor of thousands. For f_3, the gradients will also start oscillating and overshooting widely, as one can see in figure 5 (for a γ of 1.5), where it is performing not too good ($\frac{df_3}{d\mathbf{x}}(\mathbf{x_{min}}) = \begin{bmatrix}4.22442e-05 & 0.48738\end{bmatrix}$). But somehow it converges again for very large $\gamma > 2$.

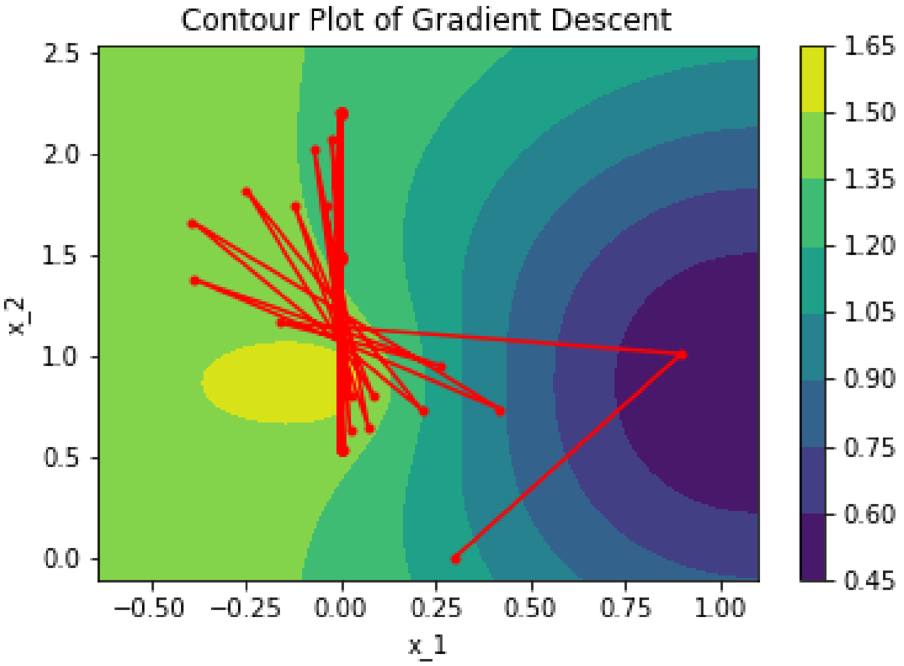


Figure 5: $f_3, \gamma = 1.5$