477 - Computational Optisimation CW 1

Adrian Catană

November 9, 2018

Part 1.1. Let x be a vector with $K = 10$ elements and let f be a function such that $f(x) = \log \sum_k e^{x_k}$. Then, we get $\nabla f = \frac{1}{\sum_k e^{x_k}} \cdot [e^{x_1}, e^{x_2}, ..., e^{x_{10}}]^T$ and
$\nabla^2 f = -\frac{1}{(\sum_k e^{x_k})^2}[e^{x_1}, e^{x_2}, ..., e^{x_{10}}]^T[e^{x_1}, e^{x_2}, ..., e^{x_{10}}] + \frac{1}{\sum_k e^{x_k}}[E_1, E_2, ..., E_{10}]$, where E_i is a column vector with 10 lines where the ith line is e^{x_i} and the rest are 0. Now, take arbitrary $z \in \mathbb{R}^{10}$ and we want to prove $z^T(\nabla^2 f)z \geq 0$. Thus,
LHS $= z^T \cdot \frac{1}{\sum_k e^{x_k}}[E_1, E_2, ..., E_{10}] \cdot z - z^T \cdot (\frac{1}{(\sum_k e^{x_k})^2}[e^{x_1}, e^{x_2}, ..., e^{x_{10}}]^T[e^{x_1}, e^{x_2}, ..., e^{x_{10}}]) \cdot z = \frac{1}{(\sum_k e^{x_k})^2}(z^T \cdot \sum_k e^{x_k} \cdot [E_1, E_2, ..., E_{10}] \cdot z - z^T \cdot ([e^{x_1}, e^{x_2}, ..., e^{x_{10}}]^T[e^{x_1}, e^{x_2}, ..., e^{x_{10}}]) \cdot z) = \frac{1}{(\sum_k e^{x_k})^2}(\sum_k e^{x_k} \cdot (\sum_k e^{x_k} z_k^2) - (\sum_k e^{x_k} z_k)^2)$. Now, by using CBS (Cauchy-Schwarz), we get $\sum_k e^{x_k} \cdot (\sum_k e^{x_k} z_k^2) \geq (\sum_k e^{x_k} z_k)^2$. But since $\frac{1}{(\sum_k e^{x_k})^2} \geq 0$, we get that $z^T(\nabla^2 f)z \geq 0$, $\forall z \in \mathbb{R}^{10}$, so our function is convex.

Part 1.2. Let f be a convex function, $f : \mathbb{R}^{1x10} \to \mathbb{R}$ and let g an affine mapping s.t. $g_i(B) = x_i^T B$ an affine mapping (easily provable by showing that $g_i(X)+g_i(Y) = g_i(X+Y)$), $g : \mathbb{R}^{nx10} \to \mathbb{R}^{10}$, $x_i \in \mathbb{R}^n$. Let $h = f \circ g_i$, $\lambda \in (0, 1)$, then $\lambda h(V) + (1 - \lambda)h(U) = \lambda(f \circ g_i(V)) + (1 - \lambda)(f \circ g_i(U)) \geq f(\lambda g_i(V) + (1 - \lambda)g_i(U)) = f(g_i(\lambda V) + g_i((1 - \lambda)U)) = f(g_i(\lambda V+(1-\lambda)U)) = h(\lambda V+(1-\lambda)U)$. Here, I first used the convexity of f, then I used that, for $\lambda \in \mathbb{R}$, $\lambda g(U) = g(\lambda U)$ (since g is an affine mapping) and the affine mapping property that I talked about above. Thus, h is convex. Also, note that I have defined g_i without adding a constant term, since we want to use the property I just proved to show that if f is the function defined at 1 on vectors and g_i is defined as needed for $\left(\log \sum_{k=1}^{10} \exp\left(x_i{}^\top \beta_k\right)\right)$.

Part 1.3. Let $\lambda \in (0, 1)$, x_i a column vector with K elements and let $f_i(U) = -x_i^T U$, f defined on column vectors with K elements. Now, we choose β, γ two column vectors with K elements and then $-\lambda f_i(\beta) - (1-\lambda)f(\gamma) = -\lambda x_i^T \beta - (1-\lambda)x_i^T \gamma = -x_i^T(\lambda\beta + (1-\lambda)\gamma) \geq -x_i^T(\lambda\beta + (1 - \lambda)\gamma)$, thus f_i is convex.

Part 1.4. Let $\lambda \in (0, 1)$ and x_i, y_i column vectors with K elements. Then, $\lambda||x_i||_1 + (1 - \lambda)||y_i||_1 = ||\lambda x_i|| + ||(1 - \lambda)y_i||_1 \geq ||\lambda x_i + (1 - \lambda)y_i||_1$ (by triangle inequality for $||x||_1$ so regularisation is convex.

Part 1.5. We simply use that a sum of convex functions is convex (easy proof, take f, g convex functions, then $\lambda(f + g)(x) + (1 - \lambda)(f + g)(y) = \lambda f(x) + (1 - \lambda)f(y) + \lambda g(x) + (1 - \lambda)g(y) \geq f(\lambda x + (1 - \lambda)y) + g(\lambda x + (1 - \lambda)y) = (f + g)(\lambda x + (1 - \lambda)y))$, $\forall \lambda \in (0, 1)$. Now, $\log \sum_{k=1}^{10} \exp\left(x_i{}^\top \beta_k\right)$ is convex by using 1.2, $-x_i{}^\top \beta_{y_i+1}$ is convex by using 1.3, thus

$\log \sum_{k=1}^{10} \exp\left({x_i}^\top \beta_k\right) - {x_i}^\top \beta_{y_i+1}$ is convex by summing them. Thus, summing by i gives us that $\sum_{i=1}^{m} \left(\log \sum_{k=1}^{10} \exp\left({x_i}^\top \beta_k\right) - {x_i}^\top \beta_{y_i+1} \right)$ is convex; we finish by using 1.4, summing after k and multypling by $\lambda > 0$ (which still preserves convexity, since the inequality signs are going to be the same) that $\lambda \sum_{k=1}^{10} \|\beta_k\|_1$ is convex. Thus, again, sum of convexes is convex, so $g(B) + h(B)$ is convex.

Part 2.1. Let's start by noting that $h(0) = 0$. Also, norm 1 means that for each column we are summing after the modulus of each entry, so $h(B) = \lambda \sum_{k=1}^{10} ||\beta_k||_1 = \lambda(\sum_{k=1}^{10}(\sum_{i=1}^{n} |B_{ik}|))$. Now, let's take B such that all entries are 0, excepting $B_{1,1}$, which we will denote for now as variable x and define B_x as this matrix. Now, $\lim_{x\to 0} \frac{||B_x||_1 - 0}{x-0} = \lim_{x\to 0} \frac{|x|}{x}$. However, $\lim_{x\to 0_+} \frac{|x|}{x} = 1$, while $\lim_{x\to 0_-} \frac{|x|}{x} = -1$, so our function is not differentiable in $x = 0$. That means that our main function h cannot be differentiable, since we found a neighborhood around $B = 0$ for which we do not have the same derivative values. Thus, we conclude that $h \notin \mathcal{C}^1$.

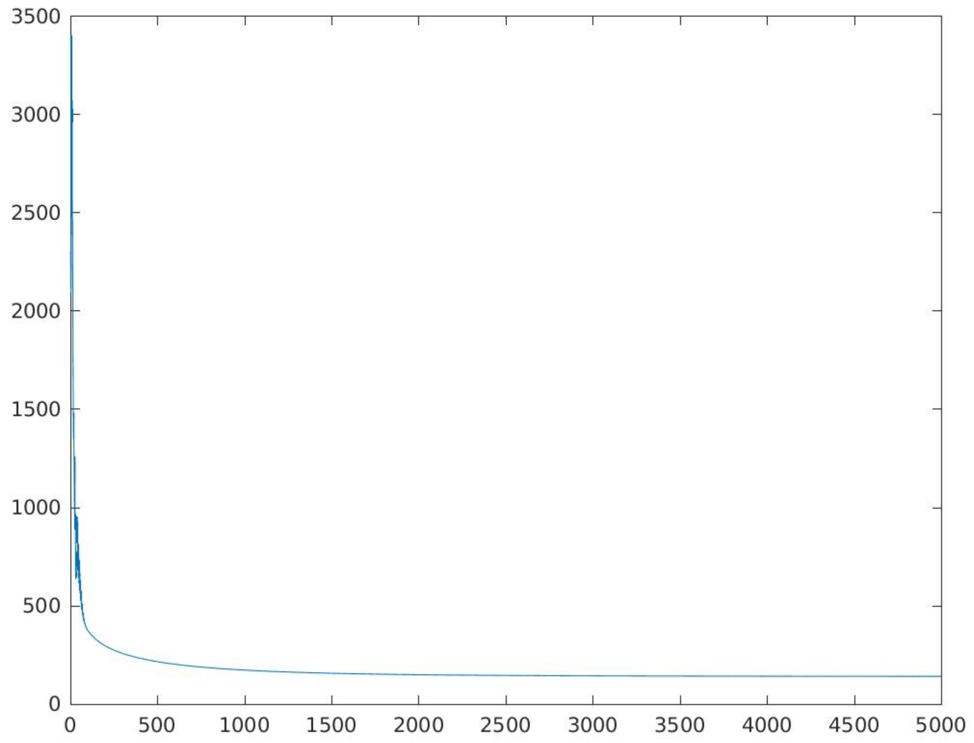
Part 2.2. Let's prove that l_2 regularization is convex. By that, let's choose X, Y two column vectors with n elements. Then we want to prove that $\lambda||X||_2^2 + (1-\lambda)||Y||_2^2 \geq ||\lambda X + (1-\lambda)Y||_2^2$, $\forall \lambda \in (0,1)$. But $||U||_2^2 = \sum_{i=1}^{n} U_i^2$, where U_i are the elements of the n length vector U. Then, we must prove that $\lambda \sum_{i=1}^{n} X_i^2 + (1-\lambda) \sum_{i=1}^{n} Y_i^2 \geq \sum_{i=1}^{n} (\lambda X_i + (1-\lambda)Y_i)^2$, which we are going to prove by summing up inequalities like $\lambda X_i^2 + (1-\lambda)Y_i^2 \geq (\lambda X_i + (1-\lambda)Y_i)^2$. But this is equivalent to proving that $\lambda X_i^2 + (1-\lambda)Y_i^2 - (\lambda X_i + (1-\lambda)Y_i)^2 \geq 0$. After expanding the squares and grouping the terms together after X_i and Y_i we get that $(\lambda - \lambda^2)(X_i^2 - 2X_iY_i + Y_i^2) \geq 0 \iff (\lambda - \lambda^2)(X_i - Y_i)^2 \geq 0$, which is true since $\lambda \in (0,1) \Rightarrow \lambda > \lambda^2$ and the square is always postitive. Thus, l_2 regularisation is convex. But we already know that $g(B)$ is convex and thus summing up with the regularisation, which is convex, will give us a convex problem.

We also know that $g(B) \in C^1$, so we only need to show that the regularisation term $\in C^1$. For this part, let's just split the regularisation function into the u_i functions, $i = \overline{1,n}$, such that $u_i(U) = U_i^2$, where U is a column vector with n elements. But these functions are differentiable and their derivatives are continuous at all points (square function has this property), so we get that their sum $\sum_{i=1}^{n} u_i(U)$ is C^1. But this is exactly $||U||_2^2$, i.e. 2-norm is C^1. By summing up this property for β_i we get that $\sum_{i=1}^{10} ||\beta_i||_2^2 \in C^1$. Also, the constant, strictly positive term λ does not influence the differentiability neither the first's derivative being continuous, so we can safely get that $\lambda \sum_{i=1}^{10} ||\beta_i||_2^2 \in C^1$. Thus, since $f = g + u$ and $g, u \in C^1$, $f \in C^1$.

Part 2.4. $\|\nabla f(B^{(j)})\|_2 < \epsilon$ corresponds to FONC, since we are checking over the whole $\mathbb{R}^n$. The ϵ provides the computational error, and we assume that once the gradient is close to 0 within a tolerance then we are really close to the minimum. Also, the other two conditions,

$\|B^{(j+1)} - B^{(j)}\|_2 < \epsilon$ and $|f(B^{(j+1)}) - f(B^{(j)})| < \epsilon$ both correspond to FONC. The first one makes the program stop when we got two consecutive values too close to one another, avoiding thus the risk of not being able to compute the gradient for the next steps. Also, these last 2 points are important to stop, since maybe FONC conditions are not satisfied, but at least we can figure out what is wrong and diagnose the problem. Also, none of them is regarding second order conditions, since that would require computing the Hessian, a really costly operation to be done in all iterations.

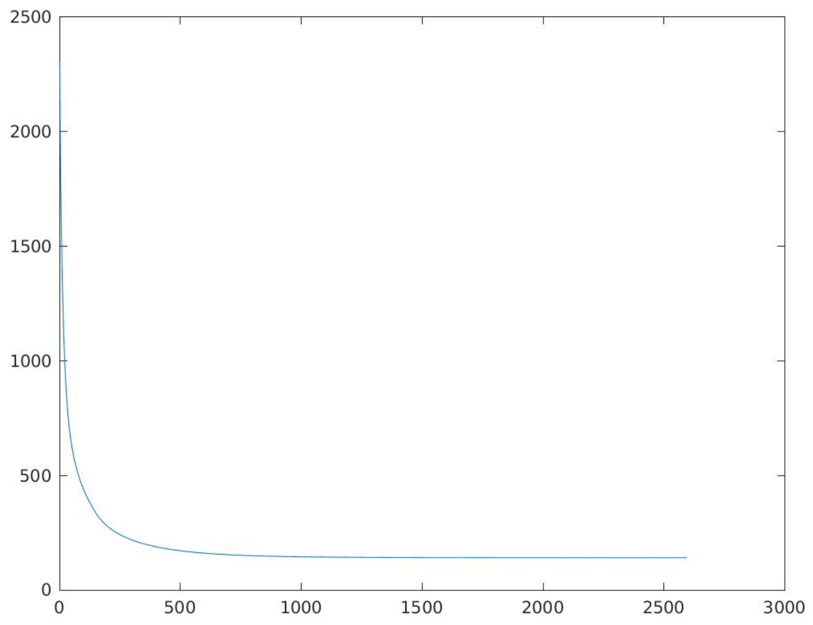
Part 2.5.



Part 2.6. Instead of using a constant step-size every iteration, we could just take the one that is giving us the smaller value for the function in the next step, such that the next function value is as small as possible when taking the current gradient direction. This will give the maximum improvement, i.e. at each iteration k, select $\alpha_k \in \arg\min_{\alpha\geq 0} f(x^{(k)} - \alpha(\nabla f(x^{(k)})))$, this giving us the optimisation problem we seek for. This is nothing but a one dimensional optimisation problem, that can be solved using classic exact line search algorithms.

Part 2.7. For this part it is very interesting to see the tradeoff between finding the right step-size (and thus adding more computing overhead for the optimum step, while making a fewer number of iterations) and keeping a constant step-size that would go through much more iterations, but each iteration taking a smaller amount of time.
Talking in terms of iterations first, it is clear that having a step-size that is optimized for each iteration would get us to a minimum faster, since this gives us the chance of jumping over many iterations of smaller steps (if going in the same direction) or going just enough for the minimum, without having a step-size that is too big (and thus would miss the next optimum value).
My algorithm that finds an optimum step-size is based on the Secant Method, with first two starting points as 0.0001 and 0.00005 (I have tested several values and these helped me converge to the minimum, close to 141.25). However, for computing the optimum step-size, I have tried many different thresholds, in order to stop our algorithm for taking too long on some part that is different from what we actually seek, i.e. the actual global minimum of the function. The first plot below shows how many iterations it takes, if we compute the optimum step-size every time for a number of $k = 100$ iterations. That is, at every "true" iteration that is computing a new value for our matrix B, we also have a number of k more steps to compute the optimal step-size, based on the Secant Method - this proved to be pretty expensive. However, this lead to better results, since the algorithm did not stop now because the squared difference was under the threshold (as it happened in the constant step size at 2.5), but because the value of the function reached our set tolerance. This made me think of minimizing the threshold for the number of iterations in the Secant Method Algorithm from 100 to 10, and, indeed, the results were the same in terms of function values, but computed faster than both the 100 steps case and than the constant step-size one. Note that the minimum obtained was, in both cases when using the Secant Method, equal to 141.273478.

2600 iterations for varying-step size


2500
2000
1500
1000
500
0
0
500
1000
1500
2000
2500
3000