Numerical Optimization - Hand-In 2

Isabela Blucher

February 15, 2018

Exercise 2.6

To prove that all isolated local minimizers are strict let x^* be an isolated local minimizer. By definition, there is a neighborhood $\mathcal{N}$ of x^* such that x^* is the only local minimizer in $\mathcal{N}$. Let x be an arbitrary point in that same neighborhood and $x \neq x^*$, which means $x \in \mathcal{N}$. If $f(x^*) \geq f(x)$,it would mean that x is also a local minimizer in the neighborhood $\mathcal{N}$, which would contradict the statement that x^* is an isolated local minimizer. So, we have that $f(x^*) < f(x)$, for all $x \in \mathcal{N}$, and by definition, x^* is a strict local minimizer.

Exercise 2.8

Let $\mathcal{S}$ be the set of global minimizers of the convex function f. By the properties of convex sets we know that if $\mathcal{S}$ is empty or only has one element, it is convex. To prove this for more elements we choose $x, y \in \mathcal{S}$ and $\alpha \in [0, 1]$. Since f is convex, the following property is satisfied

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$$

And because x and y are both global minimizers of f, $f(x) = f(y)$. Replacing that in the inequality we have

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(x) = \alpha f(x) + f(x) - \alpha f(x) = f(x)$$

But because x is a global minimizer of f

$$f(\alpha x + (1-\alpha)y) \geq f(x) \implies f(\alpha x + (1-\alpha)y) = f(x)$$

Therefore, we can affirm that $\alpha x + (1-\alpha)y \in \mathcal{S}$, which is the definition of a convex set. So, the set $\mathcal{S}$ of global minimizers of f is a convex set.

Exercise 2.9

Since $-\nabla f$ represents steepest descent, then $(p_k)(-\nabla f) = \|p_k\| \cdot \|\nabla f\| cos(\theta)$, and p_k is a descent direction if $cos(\theta) > 0$. Which means that, to show that p_k is a descent direction we need to verify $p_k \cdot \nabla f < 0$ for $f(x_1, x_2) = (x_1 + x_2^2)^2$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2(x_1 + x_2^2)(1) \\ 2(x_1 + x_2^2)(2x_2) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2^2 \\ 4x_1x_2 + 2x_2^3 \end{bmatrix}$$

For $p^T = (-1, 1)$ and $x^T = (1, 0)$ we have

$$p_k \nabla f_k = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = -2$$

Since -2 < 0, p_k is a descent direction.
Now we want to find all minimizers for $\min_{\alpha > 0} f(x_k + \alpha p_k)$.

$$(x_k + \alpha p_k)^T = (1, 0)^T + \alpha(-1, 1)^T = (1 - \alpha, \alpha)^T$$

Applying this result to the original function we have

$$f((x_k + \alpha p_k)^T) = f((1 - \alpha, \alpha)^T) = (1 - \alpha + \alpha^2)^2$$

To find the minimizers, we find the stationary points and then a second derivative test

$$\frac{d}{d\alpha} f(x_k + \alpha p_k) = 2(1 - \alpha + \alpha^2)(-1 + 2\alpha) = 0 \implies \alpha = \frac{1}{2}$$

$$\frac{d^2}{d\alpha^2} f(x_k + \alpha p_k) = 12\alpha^2 - 12\alpha + 6$$

When $\alpha = \frac{1}{2}$, the second derivative is

$$\frac{d^2}{d\alpha^2} f(x_k + \alpha p_k) = 3 - 6 + 6 = 3$$

Since $3 > 0$, $\alpha = \frac{1}{2}$ is a minimizer for f.

Programming Assignment

Rosenbrock Function

With the new interface implementation of the Rosenbrock function, which takes a 2D-point and returns a scalar, the matlab function fminunc is able to be used. The new interface was tested, and works correctly.

For the function fminunc, firstly without gradient or Hessian information, it finds the local minimizer (1, 1) if the starting point is not so far away from the minimizer. For example, for a starting point of (5,5), it doesn't get to (1,1) but instead to (0.9995, 0.9991) and for a starting point of (-5, -5), it gets to (1.0003, 1.0007).

As soon as more information is given through the options setting of the function, it gets better at finding the local minimizer from points further away. After the gradient information was included in the function, it found a possible local minimizer (1, 1) for the coordinates (5,5) and (-5, -5). The stopping criteria seems to get more precise the more information the function has. With some research, it seems that without providing the gradient, the solver estimates it by finite differences. So, when providing this derivative, we are saving computational time and increasing the accuracy of the results.

After including the Hessian, the convergence speed and precision seem to get better. After some research on the function, it seems that providing a Hessian makes the solver run more reliably and with fewer iterations.

But the further away from the minimizer a starting iterate is, the less precise about the result the function is. Instead of reporting that a local minimum was found, it prints out that a local minimum is possible, with an estimate of the value. That means that the solver may have found a local minimum, but cannot be certain due to the first-order optimality measure being greater than the optimality tolerance.

Logistic Regression

The result of the computation of the gradient of the Logistic Regression function is

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} & \frac{\partial L}{\partial \theta_1} & \cdots & \frac{\partial L}{\partial \theta_d} \end{bmatrix}$$

where

$$\frac{\partial L}{\partial \theta_i} = \sum_{i=1}^{n} \frac{-y(i)}{(e^{y(i) f_\theta(x^{(i)})} + 1)} \frac{\partial f}{\partial \theta_i}(x^{(i)})$$

and

$$\frac{\partial f}{\partial \theta_i}(x^{(i)}) = x_c^{(i)}$$

which means that this partial derivative is always an element of the dataset, where i is the row index and c the column index.

And for the computation of the Hessian

$$\nabla^2 L(\theta) = \begin{bmatrix} \frac{\partial L^2}{\partial \theta_0^2} & \frac{\partial L^2}{\partial \theta_1 \theta_0} & \cdots & \frac{\partial L^2}{\partial \theta_d \theta_0} \\ \vdots & \vdots & & \vdots \\ \frac{\partial L^2}{\partial \theta_0 \theta_d} & \frac{\partial L^2}{\partial \theta_1 \theta_d} & \cdots & \frac{\partial L^2}{\partial \theta_d^2} \end{bmatrix}$$

where

$$\frac{\partial L^2}{\partial \theta_j \theta_k} = \sum_{i=1}^{n} (y^{(i)})^2 x_i^{(j)} x_i^{(k)} \left(\frac{e^{y(i) f_\theta(x^{(i)})}}{(e^{y(i) f_\theta(x^{(i)})} + 1)^2}\right)$$

.

The test implemented for verifying the correctness of the gradient and Hessian of the Logistic Regression function was based on the finite differences method. Essentially, we want to verify for the gradient if $L(\theta+\varepsilon) - L(\theta) \approx \varepsilon \nabla L(\theta)$, and after we guarantee the gradient works, we can verify the Hessian with a similar approach $\nabla f(x+\varepsilon) - \nabla f(x) \approx \varepsilon \nabla^2 f(x)$. The error is a scalar value calculated as $err = \frac{\|\text{estimated gradient or Hessian}\|}{\|\text{computed gradient or Hessian}\|}$.

For testing the gradient, the starting arbitrary ε was $10^{(-4)}$, and the error between the estimated and the computed gradients was 8.7119e-05, for $\varepsilon = 10^{(-5)}$ the error was 8.8073e-06, and the error starts growing again for $\varepsilon = 10^{(-8)}$, where the precision on the floating point arithmetic starts getting too small. We can see in the plot below how the smaller the epsilon, the smaller the error, and also the turn-off point where the floating point arithmetic gets too small for computing precisely.

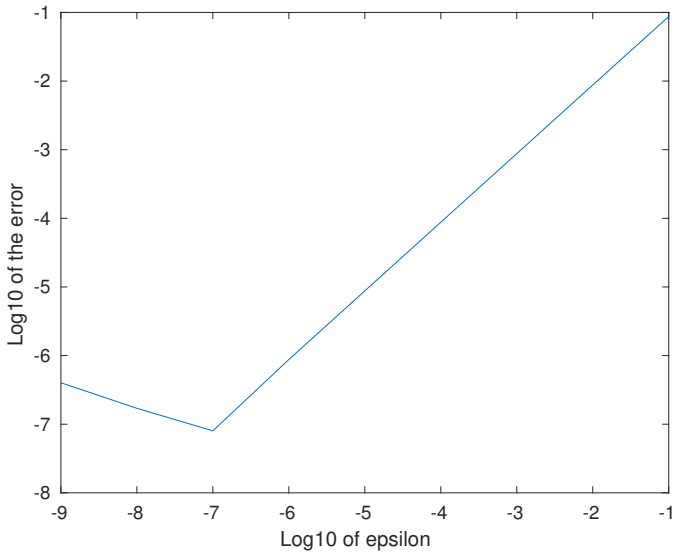

Figure 1: Plot of the computed gradient error versus the value of epsilon

The bigger the epsilon, the bigger the error, so for a sufficiently small epsilon, the error is also very small, which means the computation made approximates well the gradient of $L(\theta)$.

For testing the Hessian, starting with the same arbitrary $\varepsilon = 10^{(-4)}$, the error between the estimated and the computed Hessian is 2.9381e-05, for $\varepsilon = 10^{(-5)}$ the error is 2.1917e-06. As for the gradient, the error for the estimation of the Hessian, also grows when epsilon gets too small. The plot below illustrates again that the smaller the epsilon, the smaller the error.

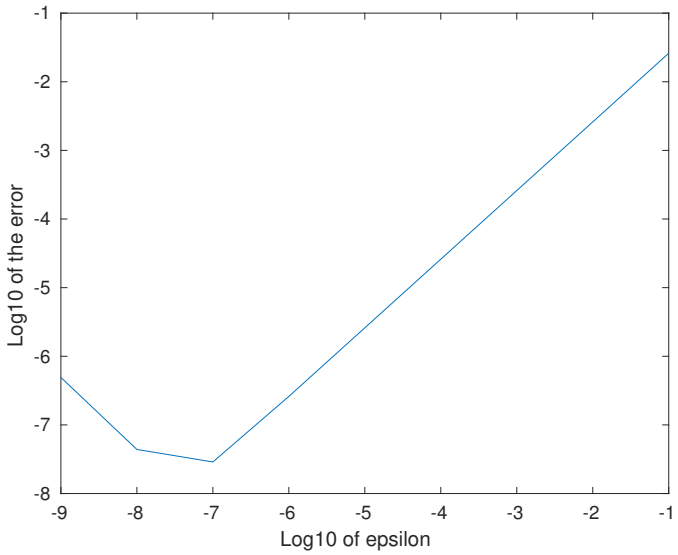

Figure 2: Plot of the computed Hessian error versus the value of epsilon

Again, like in the gradient estimation, the bigger the epsilon, the bigger the error, so for a sufficiently small epsilon, the error also becomes very small, which means we have successfully computed the Hessian of the Rosenbrock function.