IOE 511/MATH 562: Homework № 2

Due: 2/10 (by 11:59PM EST)

1. Consider the function (note, this is the same function as Homework 1, Question 9),
$$f(x) = x_1 e^{-\frac{1}{2}(x_1^2+x_2^2)}.$$
Determine the two stationary points of $f(x)$. Using your knowledge of the first and second order optimality conditions, determine if they are (local/global?) minimizers or maximizers, or only saddle points.

2. Consider the function
$$f(x) = x^4\left(2+\cos\left(\tfrac{1}{x}\right)\right).$$
Prove that $x^* = 0$ is a strict local minimizer, but not an isolated minimizer.

3. Show that if f is strictly convex, then there exists at most one local minimum of f.

4. Consider a convex twice continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ that has L-Lipschitz continuous gradients. In class we saw that this implies that $\nabla^2 f(x) \preceq LI$, for all $x \in \mathbb{R}^n$.

   (a) Show that for any $x, y \in \mathbb{R}^n$ the above implies,
$$f(y) \le f(x) + \nabla f(x)^T(y-x) + \frac{L}{2}\|y-x\|_2^2. \tag{1}$$

   (b) Show that the following conditions are equivalent:
$$f(y) \le f(x) + \nabla f(x)^T(y-x) + \frac{L}{2}\|y-x\|_2^2$$
$$f(y) \ge f(x) + \nabla f(x)^T(y-x) + \frac{1}{2L}\|\nabla f(y) - \nabla f(x)\|_2^2$$
$$(\nabla f(x) - \nabla f(y))^T(x-y) \ge \tfrac{1}{L}\|\nabla f(x) - \nabla f(y)\|_2^2$$

   (c) **EXTRA CREDIT 1:** If $f : \mathbb{R}^n \to \mathbb{R}$ is a convex continuously differentiable function (**not** twice continuously differentiable) with L-Lipschitz continuous gradients, one can show (1), i.e., the twice continuously differentiable assumption is **not required**. Prove the result.

   (d) **EXTRA CREDIT 2:** Is convexity important? Necessary? Why?

5. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function. Suppose that a point x^* is a local minimum of f along every direction that passes through x^*. That is, for any $d \neq 0$, the function
$$g_d(\alpha) := f(x^* + \alpha d),$$
is minimized at $\alpha = 0$.

   (a) Prove that there are no descent directions of f at x^* and that $\nabla f(x^*) = 0$.

(b) Show by example that x^* need not be a local minimum of f.

(c) In class, we discussed that absence of descent directions is a necessary condition for a local minimum, but we left the question of whether it is a sufficient condition unanswered. What can you say about this issue, in light of the above discussion?

6. Consider a stongly convex quadratic function of the form: $f(x) = \frac{1}{2}x^TQx - q^Tx$. Show that the one dimensional minimizer of f along the ray $x_k + \alpha d_k$ is given by:

$$\alpha_k = -\frac{\nabla f(x_k)^T d_k}{d_k^T Q d_k}.$$

7. Prove the following theorem.

Theorem: Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex and differentiable function with L-Lipschitz continuous gradients. Let $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$, where $\alpha_k = \alpha \in (0, \frac{1}{L}]$. Then,

$$f(x_k) - f^* \leq \frac{\|x_0 - x^*\|_2^2}{2\alpha k},$$

where $f(x^*)$ is the optimal value and x^* is a minimizer.

We will guide you through the proof since it is more involved than what we saw in class. Here are some hints/guides:

[1] By Lipschitz continuity and the step size condition, we have (this is from class)

$$f(x_{k+1}) \leq f(x_k) - \frac{\alpha}{2}\|\nabla f(x_k)\|_2^2.$$

(Note, the derivation of this step is in `gd.pdf`.)

[2] Derive an inequality relating $f(x_k)$ and f^* (Hint 1: I do not mean $f^* \leq f(x_k)$. Hint 2: Use properties of convexity and derive an upper bound for $f(x_k)$ in terms of f^*.)

[3] Plug in the inequality derived in step [2] to the inequality in [1].

[4] Re-arrange the expression such that it is of the form: $f(x_{k+1}) - f^* \leq \frac{1}{2\alpha}(\cdots)$.

[5] Add and subtract $\|x_k - x^\star\|_2^2$ inside the expression $\frac{1}{2\alpha}(\cdots)$.

[6] Derive an expression for $\|x_{k+1} - x^*\|_2^2$ that is independent of x_{k+1}. Note, $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$.

[7] Use the expression derived in [6] within the inequality from [5].

[8] The final step is to use the telescopic sum trick!

MATLAB SECTION OF HOMEWORK

This is the first homework where a Matlab implementation of an optimization algorithm is required. In this and all future exercises that involve coding, you will need to upload your Matlab files to Canvas, while continuing to upload the pdf file with your written answers. In addition to reading your pdf file and grading your answers to the questions posed in the exercise, we may check correctness of your code by running it, possibly on a different set of inputs than you were asked to explore in the homework. **Comments below apply to this and all future homeworks that ask for Matlab implementations, so keep this cover page handy**.

There will be **precise instructions given for how to name your Matlab files and the functions implemented in them, and what functionality they should have**. You must follow these instructions exactly to receive credit. The implementation part of your work will be graded by a computer script; if you deviate from the instructions even in a small way, the script code will fail and you will not receive points for this part of the problem. A `framework` Matlab file can be found on Canvas under the `Code` folder. You are encouraged to use this template code for the homeworks and project, however, this is not required. **What is required is that the inputs and outputs of your main functions are consistent with those required, and that the main function is named appropriately;** see below for details.

Many people use *print statements, pauses and other mechanisms* in execution as they debug and test their code. Once you have the version of your code you are ready to submit, remove all of them from your code. When executed, your Matlab function should only print to the screen or otherwise display what you have been asked to print/display/output, and should not pause mid-execution.

Please create a zip file and upload all your code and written assignments in one file for each Homework (File name: `LastName_FirstName_Homework#.zip`). Note also that, in addition to writing and executing some code, each implementation exercise will ask questions and request that you report specific results based on your computation. These results and answers to these questions **must be included in the pdf file of your solutions**; it is not sufficient for you to submit code that, when executed, produces those results (i.e., the grader of the homework should not have to run your code to produce the results you were asked to report yourself).

One final comment... **make sure you comment your code!**

Some tips/suggestions for coding/debugging

- Comment your code!
- Printing output is one of the best ways to debug code
- Use unit test to check all components (no matter how small) of your code
- Start small and simple
- *Why are bugs/issues so hard to find?* Usually, it bugs/issues are small/tiny (e.g., a plus sign should have been a minus sign), and hard to find since we do not expect such a simple part of the code to be wrong. From personal experience, bugs are either where you least expect to find them or right in front of you in the simplest operations of the code.
- And, of course, comment your code!

8. In this section of the homework, you will *investigate the performance of Gradient Descent and Newton's method on two unconstrained optimization problems*. **The goal is to understand the merits and limitations of each method**.

   Algorithms:

   - Gradient Descent (with constant step size);
   - Gradient Descent (with a backtracking line search[1]);
   - Newton's Methods (with a backtracking line search)[2].

   Problems:

   - Quadratic (2 problems):

     $$f(x) = \tfrac{1}{2}x^T Ax + b^T x + c,$$

     where $x \in \mathbb{R}^n$ ($n \in \{2, 10\}$) and A is positive definite. The data for this problem (A, b, c, x_0 and x^*) is on the course website; files:

     – `Homeworks/HW_2/Data/quadratic2.mat`;
     – `Homeworks/HW_2/Data/quadratic10.mat`.

     Note, you can load the data in Matlab using: `load('quadratic10.mat')`.

   - Rosenbrock (1 problem):

     $$f(x) = (1 - w)^2 + 100(z - w^2)^2, \text{ where } x = [w\ z]^T \in \mathbb{R}^2.$$

     Starting point $x_0 = [1.2\ 1.2]^T$. Note: $x^* = [w^*\ z^*]^T = [1\ 1]^T$.

   Constants: Use the following set of constants: $\bar{\alpha} = 1$; $c_1 = 10^{-4}$; $\rho = 0.5$; max_iters $= 100$; $\epsilon = 10^{-6}$.

   Termination Conditions: You will implement two types of termination conditions. The first is guided by the theory (i.e., gradient is small at minimizers)

   $$\|\nabla f(x_k)\|_{\text{inf}} \le \epsilon \max\{\|\nabla f(x_0)\|_{\text{inf}}, 1\}.$$

   The second condition is to make sure that your code does not run forever: $k <$ max_iters, where k is the iteration counter.

   Presenting Results/Comparing Algorithms: There are many ways to present numerical results and compare the performance of algorithms. For this homework we will focus on **comparing the performance of the algorithms in terms of iterations**.

   Deliverables:

[1] See the end of this document for a detailed description of the backtracking line search.

[2] Important Note: When implementing Newton's method in Matlab avoid computing the inverse of the Hessian matrix. Instead, use the \ command. Namely, instead of $x = inv(A) * b$, use $x = A \backslash b$.

[1] Matlab implementation of the three methods. Similar to the code framework provided (but not necessarily using the framework) your main code (`optSolver` in the provided) should take as input: *problem*, *method*, *options*, and ouput: *final iterate* and *function value* of a given method on a given problem, i.e.,

```
[x,f]=optSolver_LastName_FirstName(problem,method,options).
```

It is of paramount importance that you name your Matlab function as described above. Note, the inputs *problem*, *method*, *options* are `structs` in Matlab. See framework code and https://www.mathworks.com/help/matlab/ref/struct.html for more information.

[2] A short report of your findings. Guidance provided below.

Tasks & Guiding questions:

[1] Derive the formulae for the gradient and Hessian of the two problems.

[2] Implement functions that given x (and possibly other inputs, e.g., A, b, c) computes the function value, gradient and Hessian of the objective functions.

[3] Implement the three algorithms. As mentioned above, the code should be implemented such that each method can be run from a main function (e.g., a function similar to `optSolver`) with **exactly the same inputs and outputs as the framework code, and named as specified above**.

[4] For each problem, make a plot showing $f(x_k) - f^*$ vs. iterations (k) for each of the method.

- Use a different color for each line (i.e., different color for each method).
- Use a log scale on the y axis.
- For Gradient Descent with constant step size, experiment with a few different constant values (only show 3 in your plot).
- Comment on the performance of the methods. Does the performance match the theory we saw in class? Any strange behavior?

[5] **Quadratic Problems:**

- How does dimension effect the performance of the methods?
- Is the behavior of Newton's method a coincidence? Can you show analytically why this is happening? (Remember A is positive definite.)

[6] **Rosenbrock Problem:**

- How does the starting point affect the performance of the methods? Try the following starting points: $[1.2 \ 1.2]^T$, $[-0.001 \ \ -0.001]^T$, $[0.99 \ 0.99]^T$ and $[-1.2 \ 1]^T$. For each starting point create a $f(x_k) - f^*$ vs. iterations (k) plot showing the performance of the methods (only run and show the variants that use the backtracking line search).

[7] **Other Questions:**

- What step sizes are selected by the backtracking variants? How do these compare with the constant step size variants?
- What are the advantages and limitations of the methods?

- Is our comparison of the algorithms fair? Justify your answer for or against, and if appropriate provide an alternative.

Algorithm 1 Backtracking Line Search (*Numerical Optimization*, Chapter 3)

Inputs: $\bar{\alpha} > 0$, $\tau \in (0, 1)$, $c_1 \in (0, 1)$

```
1: Set $\alpha \leftarrow \bar{\alpha}$
2: repeat until $f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k$
3:         $\alpha \leftarrow \tau\alpha$
4: end (repeat)
5: Terminate with $\alpha_k = \alpha$
```

9. *(Bonus, 10 points)* What would be helpful to improve your learning in IOE 511/MATH 562?