Stony Brook University
CSE592 – Convex Optimization – Spring 18
Homework 1, Due: Feb, 20, 2018, 11:59PM

February 5, 2018

Instructions

- The homework is due on February 20, 2018. Anything that is received after the deadline will not be considered.
- The write-up **must** be prepared in Latex or Word and converted to pdf. No scanned hand-written notes!
- We can use any Latex class you like, just report question number and your answer.
- If the question requires you to implement a Python function, you must also submit a file with the implementation. Make sure it is sufficiently well documented that the TAs can understand what it is happening.

1 Gradient Decent without Linesearch

In this problem we will consider gradient descent with predetermined step sizes. That is, instead of determining η_t by a linesearch method using the objective function, the current iterate x_t and the descent direction Δ_t, it will be set to some pre-determined sequence.

1. For a strongly convex twice-continuously differentiable function $f(x)$ with bounded Hessian, $mI \preccurlyeq \nabla^2 f(x) \preccurlyeq MI$, $\kappa = M/m$, consider gradient descent with a fixed step size $\eta_t = \frac{1}{M}$. Prove that with this step size, after

$$T = \frac{1}{\log(\frac{\kappa}{\kappa-1})} \log\left(\frac{f(x^{(0)}) - p^*}{\epsilon}\right) \tag{1}$$

   iterations, x_t will be ϵ-suboptimal.

   How many gradient evaluations are performed to reach an ϵ-suboptimal solution? How many function evaluations?

2. The above choice of step-size requires knowing beforehand a bound on the Hessian. You will now show that the choice of a fixed (equal for all iterations) stepsize *must* depend on the function (or at least on the magnitude of its Hessian).

   For any η, find a twice-continuously differentiable strongly convex function $f(x)$ with bounded Hessian and an initial point x_0 such that gradient descent with fixed stepsize $\eta_t = \eta$ starting at x_0 yields a sequence of iterates that does *not* converge to the optimum.

   Note that we require the Hessian be bounded, i.e. there exists some M s.t. $\nabla^2 f(x) \preccurlyeq MI$ for all x, but the bound M you have to assume will of course depend on η.

   Hint: it is enough to consider scalar (one-dimensional) quadratic functions.

2 Newton's Method

In this question we will formally prove the affine invariance of Newton's method. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$ and an affine transform $y \in \mathbb{R}^m \mapsto Ay + b$ where $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. Define $g(y) = f(Ay + b)$.

1. For $x = Ay + b$, let Δx and Δy be the Newton steps for $f(x)$ and $g(y)$ respectively. Prove that $\Delta x = A\Delta y$.

2. Prove that for any $\eta > 0$, the exit condition for backtracking linesearch on $f(x)$ in direction Δx will hold if and only if the exit condition holds for $g(y)$ in direction Δy.

3. Consider running Newton's method on $g(\cdot)$ starting at some $y^{(0)}$ and on $f(\cdot)$ starting at $x^{(0)} = Ay^{(0)} + b$. Use the above to prove that the sequences of iterates obeys $x^{(k)} = Ay^{(k)} + b$ and $f(x^{(k)}) = g(y^{(k)})$.

4. Prove that Newton's decrement for $f(\cdot)$ at x is equal to Newton's decrement for $g(\cdot)$ at y, and so the stopping conditions are also identical.

3 Programming Exercise

In this exercise, we will experiment with bisection search algorithm and get familiar with ways to implement the oracle access. The included archive contains partial Python code. In order to compile and run the code, you need to install Python3 and you will need NumPy package for the future exercises. One easy way to do set up both of them is to install Anaconda which is a free Python distribution that includes many Python packages for science, math, engineering, data analysis.

In this assignment, you will experiment with bisection, gradient descent, and Newtons method. The included archive contains partial python code, which you must complete. Areas that you will fill in are marked with "TODO" comments. **For this section, you should turn in *only* the file algorithms.py.**

3.1 Algorithms

All algorithms are implemented in `algorithms.py`. The "exact" line search is the only complete function, even if it calls the incomplete bisection method.

Bisection. Complete the implementation of the bisection method in `algorithms.py`. See your notes for the implementation. Remember that the algorithm has to have a stopping criterion that guarantees that the solution is ϵ-suboptimal.

Backtracking line search. Complete the implementation of the backtracking line search in `algorithms.py`. This is algorithm 9.2 of Boyd and Vandenberghe.

Gradient descent and Newtons method. Complete the implementations of gradient descent and Newtons method in `algorithms.py`. Gradient descent is algorithm 9.3 of Boyd and Vandenberghe, and Newtons method is algorithm 9.5. Your implementation of gradient descent should terminate once $\|\nabla f(x)\|_2^2 \leq \epsilon$, and your implementation of Newtons method should terminate once the squared Newton decrement $\lambda^2(x) = (\nabla f(x))^\top (\nabla^2 f(x))^{-1} \nabla f(x)$ satisfies $\lambda^2(x) \leq \epsilon$.

3.2 Example Functions

The file `main.py` and `hw1_function.py` contains a number of functions on which you can try your algorithms. Also, there is complete code to plot the iterates of the algorithms and level curves of the function: you can use it for debugging and for understanding what the algorithms do. Experiment with the algorithms and the functions, get a sense of how they behave. The familiarity you'll get with these algorithms is probably the most important thing to learn in this assignment.

3.3 Implementation Tips

For simplicity, I suggest to define matrix and vectors as NumPy matrices, so that you can use the "*" for matrix-vector multiplication, ".T" for the transpose, and "$.I$" for the matrix inverse.