CS 726: Homework #5

Posted: Apr 13, 2025. Due: Apr 25, 2025 on Canvas

Please typeset your solutions.

You should provide sufficient justification for the steps of your solution. The level of detail should be such that your fellow students can understand your solution without asking you for further explanation.

Note: You can use the results we have proved in class – no need to prove them again.

Q1.1	Q1.2	Q1.3	Q1.4	Q1.5	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q4	Total
8	7	8	4	8	8	7	15	10	5	20	100 pts

Q 1 Damped Newton's Method

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a twice continuously differentiable m-strongly convex function, with L_1-Lipschitz gradients and L_2-Lipschitz Hessians. Let $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$. Consider the damped Newton's method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k), \tag{1}$$

where α_k is determined using the backtracking line search with parameter $\bar{\alpha} = 1$ and $\rho < 1$. That is:

- Set $\alpha_k \leftarrow \bar{\alpha} = 1, \mathbf{d}_k = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$, $c_1 = \frac{1}{4}$;
- **repeat** until $f(x_k + \alpha_k \mathbf{d}_k) \le f(x_k) + c_1 \alpha_k \langle \nabla f(x_k), \mathbf{d}_k \rangle$ (first Weak Wolfe Conditoin)
  $\alpha_k \leftarrow \rho \alpha_k$
- **end (repeat)**
- **return** α_k

In this question, we will walk through the global convergence analysis of damped Newton's method.

Q 1.1 Descent Lemma

With α_k chosen by backtracking line search as above, show that i) $\alpha_k \ge \rho \frac{3m^2}{2L_1^2}$, ii) the worst-case number of backtracking line search iterations per iteration of damped Newton method is at most $\max\left(\log_\rho(\frac{3m^2}{2L_1^2}) + 1, 1\right)$, and iii) the following analogue of the descent lemma holds for damped Newton method:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \le -\frac{\beta}{2} \|\nabla f(\mathbf{x}_k)\|_2^2, \tag{2}$$

where $\beta = \rho \frac{3m^2}{4L_1^3}$.

Hint: By the definition of the backtracking line search, the stepsize α_k/ρ does *not* satisfy the first Weak Wolfe condition. It is fine if you get some other constants than $\frac{3}{2}$ and $\frac{3}{4}$ above.

Solution:

Q 1.2 Damped Newton Phase

Let $\eta > 0$ be given. Use Q 1.1 to show that it takes at most $k = O\left(\frac{L_1^3}{m^3} \log\left(\frac{L_1(f(\mathbf{x}_0)-f(\mathbf{x}^*))}{\eta}\right)\right)$ iterations to find a point satisfying $\|\nabla f(\mathbf{x}_k)\|_2 \leq \eta$.

Hint: The analysis in Lecture 7–8, Section 3.3 may be useful.

Solution:

Q 1.3 Unit Stepsize

Let $\bar{\eta} := m^2/(2L_2)$. Prove that if $\eta_k := \|\nabla f(\mathbf{x}_k)\|_2 \leq \bar{\eta}$, then the first Weak Wolfe Condition (with $c_1 = 1/4$) holds for $\alpha_k = 1$, and thus the backtracking line search returns $\alpha_k = 1$.

Hint: Compare the first Weak Wolfe Condtiion with Taylor's theorem (Lecture 3, Theorem 1, Part 4).

Solution:

Q 1.4 Quadratic Convergence Phase

Recall Theorem 1 in Lecture 19 on the convergence of basic Newton's method. Adapt the proof of that theorem to show that, once we have $\alpha_k = 1$, we also get:

$$\|\nabla f(\mathbf{x}_{k+1})\|_2 \leq \frac{L_2}{2m^2}\|\nabla f(\mathbf{x}_k)\|_2^2 \tag{3}$$

Solution:

Q 1.5 Overall Iteration Complexity

Recall the definition of $\bar{\eta}$ from Q 1.3. Assume that $\frac{1}{4\bar{\eta}} < 1$.

1. Suppose $\|\nabla f(\mathbf{x}_k)\|_2 \leq \bar{\eta}$ holds. Use Q 1.3 and Q 1.4 to show that $\|\nabla f(\mathbf{x}_{k+1})\|_2 \leq \bar{\eta}$ also holds and that we get quadratic convergence in $\|\nabla f(\cdot)\|_2$ for $j \geq k$.
2. Use Part Q 1.2 to bound the number of iterations k until $\|\nabla f(\mathbf{x}_k)\|_2 \leq \bar{\eta}$.
3. Use quadratic convergence to bound the number of iterations k' from the first iteration k in which it holds $\|\nabla f(\mathbf{x}_k)\|_2 \leq \bar{\eta}$ until the first iteration $k + k'$ for which $\|\nabla f(\mathbf{x}_{k+k'})\|_2 \leq \epsilon$, where $\epsilon > 0$.
4. Combine the two bounds above to show that damped Newton method takes a total of at most

$$K = O\left(\frac{L_1^3}{m^3} \log\left(\frac{L_1 L_2 (f(\mathbf{x}_0) - f(\mathbf{x}^*))}{m}\right) + \log\log(\bar{\eta}/\epsilon)\right)$$

iterations to achieve $\|\nabla f(\mathbf{x}_K)\|_2 \leq \epsilon$.

Note: For the first term on the RHS above, It's OK if you have something like $O\left(\frac{L_1^3}{m^3} \log\left(\frac{L_1^2 L_2^4 (f(\mathbf{x}_0)-f(\mathbf{x}^*))^3}{m^3}\right)\right)$ or similar. That is, you may ignore constant exponents inside the log and assume that they can be absorbed by the big-Oh notation.

Solution:

Q 2 SGD for Nonconvex Functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be an L-smooth (possibly nonconvex) function. Given $\mathbf{x} \in \mathbb{R}^d$, let $\tilde{\mathbf{g}}(\mathbf{x};\xi)$ be an unbiased stochastic estimate of the gradient $\nabla f(\mathbf{x})$ whose variance is bounded by σ^2, that is:

$$(\forall \mathbf{x} \in \mathbb{R}^d): \quad \mathbb{E}_\xi[\tilde{\mathbf{g}}(\mathbf{x};\xi)] = \nabla f(\mathbf{x}) \quad \text{and} \quad \mathbb{E}_\xi[\|\tilde{\mathbf{g}}(\mathbf{x};\xi) - \nabla f(\mathbf{x})\|_2^2] \le \sigma^2.$$

Consider stochastic gradient descent algorithm defined by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \tilde{\mathbf{g}}(\mathbf{x}_k;\xi_k), \tag{SGD}$$

where the step size α_k is strictly positive and $\xi_0, \xi_1, \ldots$ are sampled i.i.d. from an unknown distribution.

(Note that we cannot directly apply the results from Lecture 18, which assumes convexity of f.)

Q 2.1 Expected Descent

Prove that as long as $\alpha_k < \frac{1}{L}$ and $\|\nabla f(\mathbf{x}_k)\|_2^2 \ge \sigma^2$, it holds that

$$\mathbb{E}_{\xi_k}[f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)] \le -\beta_k \|\nabla f(\mathbf{x}_k)\|_2^2,$$

where $\beta_k = \alpha_k - L\alpha_k^2 > 0$.

Hint: for a random vector Z, it holds that $\mathbb{E}[\|Z\|_2^2] = \|\mathbb{E}Z\|_2^2 + \mathbb{E}[\|Z - \mathbb{E}Z\|_2^2]$.

Solution:

Q 2.2 Constant Stepsize

Consider running (SGD) with a constant stepsize $\alpha_k = \frac{1}{2L}, \forall k$. Use Q2.1 to show that after at most $\frac{8L(f(\mathbf{x}_0)-f(\mathbf{x}^*))}{5\sigma^2}$ iterations, we have that $\mathbb{E}[\min_{0\le i\le k} \|\nabla f(\mathbf{x}_i)\|_2^2] \le 2\sigma^2$, where the expectation is taken w.r.t. all randomness in the algorithm (all $\xi_0, \xi_1, \ldots, \xi_k$).

Solution:

Q 2.3 Diminishing Stepsize

Consider now running (SGD) with a diminishing stepsize $\alpha_k = \frac{1}{2L\sqrt{k+1}}$. Given $\epsilon > 0$ satisfying $\epsilon \le \min(1, \sigma^2)$, use Q2.1 to show that after at most

$$k = O\Big((\frac{\sigma^2}{\epsilon}\ln(\frac{\sigma^2}{\epsilon}))^2 + \frac{L^2(\mathbb{E}[f(\mathbf{x}_0)] - f(\mathbf{x}^*))^2}{\epsilon^2}\Big)$$

iterations, we have $\mathbb{E}[\min_{0\le i\le k} \|\nabla f(\mathbf{x}_i)\|_2^2] \le \epsilon$, where the expectation is taken w.r.t. all randomness in the algorithm?

Hint: For any $k_0 \ge 2$, use Q2.1 to show that

$$\mathbb{E}\big[\min_{0\le i\le 2k_0} \|\nabla f(\mathbf{x}_i)\|_2^2\big] \le \frac{8L(\mathbb{E}[f(\mathbf{x}_{k_0})] - f(\mathbf{x}^*))}{\sqrt{2k_0+1}}. \tag{4}$$

Then use Q2.1 to get a coarse upper bound on $\mathbb{E}[f(\mathbf{x}_{k_0})]$ and thereby obtain

$$\mathbb{E}\big[\min_{0\le i\le 2k_0} \|\nabla f(\mathbf{x}_i)\|_2^2\big] \le \max\left\{\frac{2\sigma^2\ln(k_0)}{\sqrt{2k_0+1}}, \frac{8L(\mathbb{E}[f(\mathbf{x}_0)] - f(\mathbf{x}^*))}{\sqrt{2k_0+1}}\right\}. \tag{5}$$

Finally, choose k_0 to make the RHS of (5) smaller than ϵ.

Solution:

Coding Assignment

You should code in Python 3.7+ and your code needs to compile/run without any errors to receive any points for the coding assignment.

Your submission for Homework #5 should be two files: **hw5.ipynb** (or **hw5.py**) and **hw5.pdf**, and do **NOT** archive into a zip file. We **strongly recommend** submitting a .ipynb file rather than a .py file.

- The .ipynb or .py file should implement the algorithms and produce the required figures.
- In the .pdf, you should include the answers to the questions below **AND the produced figures.**

To implement Euclidean projections onto simplex and ℓ_1 ball, you are allowed to use any code you find online (for example, take a look at https://gist.github.com/daien/1272551/edd95a6154106f8e28209a1c7964623ef8397246).

Q 3 Comparing Frank-Wolfe and PGD

In this question, you are asked to compare the Frank-Wolfe method with Projected Gradient Descent (PGD), on the following problem instance:

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}),$$

where $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_1 \leq 1\}$ is the ℓ_1 ball and $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{B}\mathbf{x} - \mathbf{b}\|_2^2$ is a quadratic function defined by: (i) $\mathbf{B}$, a $p \times d$ matrix whose entries are i.i.d. Gaussian with mean zero and variance one, where $d = 200$ and $p = 10$; and (ii) $\mathbf{b} \in \mathbb{R}^p$ is the vector of all 1's.

Q 3.1

For Frank-Wolfe: What are the vertices of $\mathcal{X}$? What is the closed form expression for the minimizer $\mathbf{v}_k = \text{argmin}_{\mathbf{u}\in\mathcal{X}}\{\langle\nabla f(\mathbf{x}_k), \mathbf{u} - \mathbf{x}_k\rangle\}$?

For PGD: You can use the code available at the provided link (at the beginning of the coding assignment) to implement projections onto $\mathcal{X}$.

Run both algorithms 30 times for 450 iterations (each time, you should regenerate the random matrix $\mathbf{B}$). You should initialize both algorithms at $\mathbf{x}_0 = \mathbf{e}_1$, where $\mathbf{e}_1$ is the first standard basis vector. Your code should output a figure that shows the ratio $\frac{f(\mathbf{x}_k)}{f(\mathbf{x}_0)}$ against the iteration count for PGD and the Frank-Wolfe algorithm *averaged over these 30 independent runs*. Discuss the results. Which algorithm is faster? Do you see what you expect from the theoretical results we derived in class?

Q 3.2

You should also consider, for each of the algorithms, the iterate $\mathbf{x}_k$ for which $\frac{f(\mathbf{x}_k)}{f(\mathbf{x}_0)} \leq 10^{-1}$ holds for the first time (i.e., the first iteration at which the ratio drops below 10^{-1}). For each algorithm, your code should output the average number of nonzero elements of these points, where the average is over 30 independent runs. You should also mention these two average values in your report. Which of the two algorithms constructs sparser solutions (solutions with fewer nonzero elements)? Can you explain why?

A practical consideration: In the second part of the question, it might happen that in some of the random runs PGD does not find a solution such that $\frac{f(\mathbf{x}_k)}{f(\mathbf{x}_0)} \leq 0.1$ within 450 iterations. To account for that, you can modify your code so that there are 30 repeats in which you get $\frac{f(\mathbf{x}_k)}{f(\mathbf{x}_0)} \leq 0.1$ within 450 iterations (use a while loop instead of a for loop).

Solution:

Q 4 First-order methods in the presence of gradient noise

In this question, you should implement three algorithms: (i) Nesterov's AGD for L-smooth convex functions (Lecture 9–10, Algorithm 2), (ii) standard gradient descent with a constant step size $1/L$, and (iii) stochastic gradient descent from Lecture 18 with a diminishing step size $a_k = 1/(L\sqrt{k})$ **and** iterate averaging (i.e., your algorithm output $\mathbf{x}_k^{\text{out}}$ rather than $\mathbf{x}_k$). All three algorithms should **not** work with the exact gradients; instead, you should implement them with "noisy" gradient oracles $\tilde{\mathbf{g}}(\mathbf{x}, \boldsymbol{\xi}) = \nabla f(\mathbf{x}) + \varepsilon\boldsymbol{\xi}$, where $\boldsymbol{\xi}$ are vectors with i.i.d. mean-zero variance-one Gaussian elements, and $\varepsilon > 0$ determines the noise level. (In this case, algorithms (i) and (ii) become stochastic, but we will still call them AGD and GD.)

The problem instance is $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{M}\mathbf{x} - \mathbf{b}^T\mathbf{x}$, where $\mathbf{M} \in \mathbb{R}^{d\times d}$ is a tridiagonal matrix with 2's on the main diagonal and -1's on the remaining two diagonals and $\mathbf{b} = \mathbf{e}_1$ (same function as in HW3 Q6). You should use $d = 200$. Recall that f is $L = 4$ smooth.

Your algorithms should all be initialized at $\mathbf{x} = \mathbf{0}$ and run for at least 1000 iterations.

Your code should output three figures, produced for three different values of ε : (i) $\varepsilon = 10^{-5}$, (ii) $\varepsilon = 10^{-3}$, and (iii) $\varepsilon = 10^{-1}$. Each figure should show the optimality gap $f(\mathbf{x}_k) - f^*$ (on a log scale) against the iteration count k, for all three algorithms.

Discuss your results. What do you observe? Do the algorithms converge? Which algorithm is fastest? Which one is the most stable? Which one would you use in practice, under what circumstances, and why?

Solution: