CS 726: Homework #3

Posted: 10/14/2022, due: 10/25/2022 by 8pm on Canvas

Please typeset your solutions.

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

Q 1. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function.

(i) Prove that if f is L-smooth, then:

$$(\forall \mathbf{x} \in \mathbb{R}^d) : \quad \frac{1}{2L}\|\nabla f(\mathbf{x})\|_2^2 \le f(\mathbf{x}) - f(\mathbf{x}^*) \le \frac{L}{2}\|\mathbf{x} - \mathbf{x}^*\|_2^2. \qquad [5\text{pts}]$$

(ii) Prove that if f is convex and L-smooth, then the following holds:

$$(\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d) : \quad \frac{1}{2L}\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2^2 \le f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle . \qquad (1)$$

Hint: Consider the function $f_{\mathbf{x}}(\mathbf{z}) = f(\mathbf{z}) - \langle \nabla f(\mathbf{x}), \mathbf{z}\rangle$. What is its minimizer? Is $f_{\mathbf{x}}(\mathbf{z})$ smooth? [10pts]

(iii) Prove the converse to Part (ii): If the inequality in Eq. (2) holds, then f is convex and L-smooth. [10pts]

Solution:

(i) Prove that if f is L-smooth, then:

$$(\forall \mathbf{x} \in \mathbb{R}^d) : \quad \frac{1}{2L}\|\nabla f(\mathbf{x})\|_2^2 \le f(\mathbf{x}) - f(\mathbf{x}^*) \le \frac{L}{2}\|\mathbf{x} - \mathbf{x}^*\|_2^2. \qquad [5\text{pts}]$$

Recall that L smooth gives:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \le L\|\mathbf{x} - \mathbf{y}\|$$

and we also have a lemma:

$$f(\mathbf{y}) \le f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$$

thus, to prove the desired claim. Using the above lemma, with $\mathbf{x}$ and $\mathbf{x}^*$, we have that :

$$f(\mathbf{x}) - f(\mathbf{x}^*) \le \langle \nabla f(\mathbf{x}^*), (\mathbf{x} - \mathbf{x}^*)\rangle + \frac{L}{2}\|\mathbf{x} - \mathbf{x}^*\|^2 = \frac{L}{2}\|\mathbf{x} - \mathbf{x}^*\|^2$$

The last equality comes from $\mathbf{x}^*$ is the minimizer, then $\nabla f(\mathbf{x}^*) = 0$.
Secondly, for the left-hand side of the inequality, assume again that $\mathbf{x}^*$ is the minimizer. Then we have:

$$f(\mathbf{x}^*) = \min_{x' \in \mathcal{X}} f(\mathbf{x}') \le \min_{\mathbf{x}' \in \mathcal{X}} (f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{x}' - \mathbf{x}\rangle + \frac{L}{2}\|\mathbf{x}' - \mathbf{x}\|^2 = f(\mathbf{x}) + \min_{\mathbf{x}' \in \mathcal{X}} (\langle \nabla f(\mathbf{x}), (\mathbf{x}' - \mathbf{x})\rangle + \frac{L}{2}\|\mathbf{x} - \mathbf{x}'\|^2)$$

Similar to the concept in class that we assume we know $f(\mathbf{x})$ and we desire to bound $\langle \nabla f(\mathbf{x}), (\mathbf{x}' - \mathbf{x})\rangle + \frac{L}{2}\|\mathbf{x} - \mathbf{x}'\|^2$. Recall that we have:

$$\frac{q^2}{2} - pq \ge \frac{-p^2}{2}$$

take $q = \sqrt{L}(\mathbf{x} - \mathbf{x}')$, $p = \frac{1}{\sqrt{L}}\nabla f(\mathbf{x})$, gives:

$$\frac{L\|\mathbf{x}' - \mathbf{x}\|^2}{2} - \langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{x}' \rangle \geq \frac{-\|\nabla f(\mathbf{x})\|^2}{2}$$

implies:

$$\frac{L|\mathbf{x}' - \mathbf{x}\|^2}{2} + \langle \nabla f(\mathbf{x}), \mathbf{x}' - \mathbf{x} \rangle \geq \frac{-\|\nabla f(\mathbf{x})\|^2}{2L}$$

implies:

$$f(\mathbf{x}^*) - f(\mathbf{x}) \leq \frac{-\|\nabla f(\mathbf{x})\|^2}{2L}$$

(ii) Prove that if f is convex and L-smooth, then the following holds:

$$(\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d): \quad \frac{1}{2L}\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2^2 \leq f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle . \tag{2}$$

Based on on the hint, we want to consider the function $f_x(\mathbf{z}) = f(\mathbf{z}) - \langle \nabla f(\mathbf{x}), \mathbf{z} \rangle$. Now we see $\mathbf{x}$ as fixed and want to minimize by choosing $\mathbf{z}$. Notice that $f(\mathbf{z}) - \langle \nabla f(\mathbf{x}), \mathbf{z} \rangle = f(\mathbf{z}) - \nabla f(\mathbf{x})^T \mathbf{z}$. Take the first order derivative with respect to $\mathbf{z}$, and set to zero,

$$\nabla f(\mathbf{z}) - \nabla f(\mathbf{x})^T = 0$$

thus, the minimizer is when $\mathbf{x} = \mathbf{z}$. Recall from L-smooth,

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$$

thus:

$$f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x})^T, \mathbf{y} - \mathbf{x} \rangle \leq \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$$

Consider:

$$\begin{aligned} f_x(\mathbf{z}) &= f(\mathbf{z} - \langle \nabla f(\mathbf{x}), \mathbf{z} \rangle) \\ f_x(\mathbf{y}) &= f(\mathbf{y}) - \langle \nabla f(\mathbf{x}), \mathbf{y} \rangle \\ f_x(\mathbf{x}) &= f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{x} \rangle \\ f_y(\mathbf{z}) &= f(\mathbf{z}) - \langle \nabla f(\mathbf{y}), \mathbf{z} \rangle \end{aligned}$$

they are smooth, so we have:

$$f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle = f(\mathbf{y}) - \langle \nabla f(\mathbf{x}), \mathbf{y} \rangle - (f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{x} \rangle) = f_x(\mathbf{y}) - f_x(\mathbf{x}) \leq \frac{1}{2L}\|\nabla f_x(\mathbf{y})\|^2$$

the last inequality comes from the left-hand side of part (i), but we replace f with f_x. And then, recover the last term with the definition:

$$\begin{aligned} f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle &= f(\mathbf{y}) - \langle \nabla f(\mathbf{x}), \mathbf{y} \rangle - (f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{x} \rangle) \\ &= f_x(\mathbf{y}) - f_x(\mathbf{x}) \leq \frac{1}{2L}\|\nabla f_x(\mathbf{y})\|^2 \\ &= \frac{1}{2L}\|\nabla(f(\mathbf{y}) - \langle \nabla f(\mathbf{x}), \mathbf{y} \rangle)\|^2 \\ &= \frac{1}{2L}\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\|^2 \end{aligned}$$

(iii) Prove the converse to Part (ii): If the inequality in Eq. (2) holds, then f is convex and L-smooth. Notice from (ii), we have:

$$f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \geq \frac{1}{2L} \|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\|^2$$

By symmetry, we also have:

$$f(\mathbf{x}) - f(\mathbf{y}) - \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \frac{1}{2L} \|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\|^2$$

If we add these two inequalities together, we then have:

$$-\langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle - \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \frac{1}{L} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|^2$$

implies:

$$\langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{y} \rangle - \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \frac{1}{L} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|^2$$

Implies:

$$\langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \frac{1}{L} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|^2$$

Now we use this to show that f is L smooth. And notice that by Cauchy Schwarz inequality, that we have:

$$|\mathbf{z}^T \mathbf{x}| \leq \|\mathbf{z}\|_* \|\mathbf{x}\|$$

and the dual norm of Euclidean is still the Euclidean norm. As a result,

$$\begin{aligned}(\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}))^2 (\mathbf{x} - \mathbf{y})^2 &\geq \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} = \mathbf{y} \rangle^2 \\ &\geq \frac{1}{L^2} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2^4\end{aligned}$$

Implies that:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$$

To show convexity, recall that there are equivalent conditions of μ - strong convexity. In particular, the following are equivalent,

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) = \frac{\mu}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

$$g(\mathbf{x}) = f(\mathbf{x}) - \frac{\mu}{2} \|\mathbf{x}\|^2 \text{ is convex}$$

For this question, I would try to show that eq.(1) implies that $g(\mathbf{x}) = f(\mathbf{x}) - \frac{L}{2}\|\mathbf{x}\|$ is convex, thus f is L strong convex thus convex. Notice that we have:

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

implies:

$$\frac{L}{2} \|\mathbf{y}\|^2 - f(\mathbf{y}) \leq \frac{L}{2} \|\mathbf{x}\|^2 - f(\mathbf{x}) + \langle \mathbf{x} - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle$$

implies that:

$$g(\mathbf{x}) \geq \langle \nabla g(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle$$

implies that g is convex. Then since g is convex, then f is L strongly convex, then convex.

Q 2. Use (2) to prove that gradient descent when applied to an L-smooth convex function with step size $\alpha_k = \frac{1}{L}$ guarantees that $\|\nabla f(\mathbf{x}_{k+1})\|_2 \leq \|\nabla f(\mathbf{x}_k)\|_2$, for all $k \geq 0$. [10pts]

Solution:

By the previous question, we have:

$$\frac{1}{2L}\|\nabla f(\mathbf{x})\|^2 \leq f(\mathbf{x}) - f(\mathbf{x}^*)$$

In other words,

$$\|\nabla f(\mathbf{x})\|^2 \leq 2L(f(\mathbf{x}) - f(x^*)) = 2Lf(\mathbf{x}) - 2Lf(\mathbf{x}^*)$$

Implies:

$$\|\nabla f(\mathbf{x})\|^2 \leq 2L(f(\mathbf{x}) - f(\mathbf{x}^*))$$

Implies:

$$\|\nabla f(\mathbf{x}_{k+1}\|^2 \leq 2Lf(\mathbf{x}_{k+1}) - 2Lf(\mathbf{x}*)$$

implies:

$$\|\nabla f(\mathbf{x}_k\|^2 \leq 2Lf(\mathbf{x}_k) - 2Lf(\mathbf{x}*)$$

In another sight, the upper bound for $\|\nabla f(\mathbf{x}_k)\|$, the gradient, is an increasing function of $f(\mathbf{x}_k)$, as long as we could show $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$, then we could have $\|\nabla f(\mathbf{x}_{k+1})\|\|\nabla f(x_k)\|$ and recall that for an L-smooth function, we have:

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - vx\|^2$$

Thus,

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla f(x_k)^T(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{L}{2}\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2$$

The basic descent updates:

$$\mathbf{x}_{k+1} = \mathbf{x} - \alpha_k \nabla f(\mathbf{x}_k) = \mathbf{x}_K - \frac{1}{L}\nabla f(\mathbf{x}_k)$$

then:

$$\begin{aligned} f(\mathbf{x}_{k+1}) &\leq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(-\frac{1}{L}\nabla f(\mathbf{x}_k)) + \frac{L}{2}\|(\frac{1}{L})^2\nabla f^2(\mathbf{x}_k)\| \\ &= f(\mathbf{x}_k) - \frac{1}{L}\|\nabla^2 f(\mathbf{x}_k)\| + \frac{1}{2L}\|\nabla^2 f(\mathbf{x}_k)\| \\ &= f(\mathbf{x}_k) - \frac{1}{2L}\|\nabla^2 f(\mathbf{x}_k)\| \\ &< f(\mathbf{x}_k) \end{aligned}$$

Thus, $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$ as desired.

Q 3. Consider the unconstrained optimization problem $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$, where f is an L-smooth convex function. Assume that $\|\mathbf{x}_0 - \mathbf{x}^*\|_2 \leq R$, for some $R \in (0, \infty)$, and let $f_\epsilon(\mathbf{x}) = f(\mathbf{x}) + \frac{\epsilon}{2R^2}\|\mathbf{x} - \mathbf{x}_0\|_2^2$. Let $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$ and $\mathbf{x}_\epsilon^* = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^d} f_\epsilon(\mathbf{x})$. Prove that:

$$(\forall \mathbf{x} \in \mathbb{R}^d) : f(\mathbf{x}) - f(\mathbf{x}^*) \leq f_\epsilon(\mathbf{x}) - f_\epsilon(\mathbf{x}_\epsilon^*) + \frac{\epsilon}{2}.$$

Prove that standard gradient descent applied to function f_ϵ finds a point $\mathbf{x}_k$ with $f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \epsilon$ in $O(\frac{L}{\epsilon}R^2\log(\frac{LR^2}{\epsilon}))$ iterations. How does this compare to applying gradient descent directly to f? [15pts]

Solution:

Recall that if $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - vx\|^2$ then we say f is μ strongly convex. Consider replacing $\frac{\epsilon}{R^2} = \mu$ define $f_\mu(\mathbf{x}) = f(\mathbf{x}) + \frac{\mu}{2}\|\mathbf{x}-\mathbf{x}_0\|^2$, and we recall that a function is μ strongly convex when $g(\mathbf{x}) = f(\mathbf{x}) - \frac{\mu}{2}\|\mathbf{x}\|^2$ is convex. Thus, for $f_\mu(\mathbf{x}) = f(\mathbf{x}) + \frac{\mu}{2}\|\mathbf{x}-\mathbf{x}_0\|^2$, if f is convex, $f_\mu(\mathbf{x})$ is μ strongly convex. Recall from the chapter 2 handout, 3.3 rate comparison, we have that if the gradient descent takes the step size of $\frac{1}{L}$, f is L-smooth, and μ strongly convex, we have that:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \epsilon$$

after $k = O(\frac{L}{\mu}\log(\frac{L\|\mathbf{x}^*-\mathbf{x}_0\|^2}{\epsilon}))$ iterations. Thus, in our case, we have $f_\epsilon(\mathbf{x})$ to be $\frac{\epsilon}{R^2}$ strongly convex, we have:

$$f_\epsilon(\mathbf{x}) - f_\epsilon(\mathbf{x}^*_\epsilon) \leq \frac{\epsilon}{2}$$

in $O(\frac{L}{\frac{\epsilon}{R^2}}\log(\frac{L\|\mathbf{x}^*-\mathbf{x}_0\|^2}{\epsilon})) = O(\frac{L}{\epsilon}R^2\log(\frac{LR^2}{\epsilon}))$.

As for how this compares to applying gradient descent directly to f? From 3.3 rate comparison, we have if f is L-smooth and convex, then $f(\mathbf{x}) - f(\mathbf{x}^*) \leq \epsilon$ in at most $O(\frac{LR^2}{\epsilon})$ iterations. Compare $O(\frac{L}{\epsilon}R^2\log(\frac{LR^2}{\epsilon}))$ and $O(\frac{L}{\epsilon}R^2)$. We let $\frac{L}{\epsilon}R^2 = Z$, we are comparing $Z\log Z$ and Z. and $Z\log Z \geq Z$ when $Z < e \approx 2.718$. That is, we prefer applying gradient descent directly to f when Z is small, or say when f behaves well in that L is small; however, when L is large, we would rather bear a little error coming from adding the quadratic term, that $\mathbf{x}^*_\epsilon \neq \mathbf{x}^*$, and we want to make use of the strong convexity.

Q 4. Recall that gradient descent when applied to an L-smooth convex function f with step size $\alpha_k = \frac{1}{L}$ guarantees that at iteration $k \geq 1$,

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{L\|\mathbf{x}_0 - \mathbf{x}^*\|^2}{2k}, \tag{3}$$

where $\mathbf{x}^*$ minimizes f.

Suppose that our function f turns out to be, in addition, $(2,\mu)$-sharp. That is,

$$(\forall \mathbf{x} \in \mathbb{R}^d): \quad f(\mathbf{x}) - f(\mathbf{x}^*) \geq \frac{\mu}{2}\text{dist}(\mathbf{x}, \mathcal{X}^*)^2, \tag{4}$$

where $\mathcal{X}^*$ is the set of all minimizers of f and $\text{dist}(\mathbf{x},\mathcal{X}^*) = \inf_{\mathbf{y}\in\mathcal{X}^*}\|\mathbf{x}-\mathbf{y}\|$.

Prove using (3) and (4) that after at most $k = \lceil\frac{2L}{\mu}\rceil$ iterations, we have

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{2}. \tag{5}$$

Argue using (5) that for any $\epsilon > 0$, gradient descent guarantees

$$f(\mathbf{x}_K) - f(\mathbf{x}^*) \leq \epsilon$$

after at most $K = O\big(\frac{L}{\mu}\log\big(\frac{f(\mathbf{x}_0)-f(\mathbf{x}^*)}{\epsilon}\big)\big)$ iterations. [10pts]

Solution:

$$
\begin{aligned}
&f(\mathbf{x}_k) - f(\mathbf{x}^*) \le \frac{L\|\mathbf{x}_0 - \mathbf{x}^*\|^2}{2k} \\
&f(\mathbf{x}) - f(\mathbf{x}^*) \ge \frac{\mu}{2}\mathrm{dist}(\mathbf{x}, \mathbf{x}^*)^2 \\
&\to f(\mathbf{x}_0) - f(\mathbf{x}^*) \ge \frac{\mu}{2}\mathrm{dist}(\mathbf{x}_0, \mathbf{x}^*)^2 \\
&= \frac{\mu}{2}\|\mathbf{x}_0 - \mathbf{x}^*\|^2 \\
&\to f(\mathbf{x}_0) - f(\mathbf{x}^*)\frac{2}{\mu} \ge \|\mathbf{x}_0 - \mathbf{x}^*\|^2 \\
&\to f(\mathbf{x}_k) - f(\mathbf{x}^*) \le \frac{(f(\mathbf{x}_0) - f(\mathbf{x}^*))\frac{2}{\mu}}{2k} \\
&= \frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)L}{k\mu} \\
&\le \frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{2} \text{ if } k = \lceil \frac{2L}{\mu} \rceil
\end{aligned}
$$

And then we argue using Eq.(4) that for any $\epsilon > 0$, gradient descent gurantees $f(\mathbf{x}_k) - f(\mathbf{x}^*) \le \epsilon$ at most $k = O(\frac{L}{\mu}\log\frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{\epsilon})$ iterations. from the above argument, we have that:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \le (\frac{1}{2})^r(f(\mathbf{x}_0) - f(\mathbf{x}^*))$$

r stands for how many runs of $k = \lceil \frac{2L}{\mu} \rceil$ iterations. Using the fact that $1 + x \le e^x$, thus:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \le (\frac{1}{2})^r(f(\mathbf{x}_0) - f(\mathbf{x}^*)) = (1 - \frac{1}{2})^r(f(\mathbf{x}_0) - f(\mathbf{x}^*)) \le e^{\frac{-1}{2}r}(f(\mathbf{x}_0) - f(\mathbf{x}^*))$$

if we take $r = 2\log\frac{f(\mathbf{x}_0) - f(^*)}{\epsilon}$

Then the last term above is less than $e^{\frac{-1}{2}2\log(\frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{\epsilon})}(f(\mathbf{x}_0) - f(\mathbf{x}^*)) = \epsilon$. For us to have $2\log(\frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{\epsilon})$ runs of $\lceil \frac{2L}{\mu} \rceil$ iterations is the same as claiming that $k = O(\frac{L}{\mu}\log(\frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{\epsilon}))$.

Q 5. In class, we have analyzed the following variant of Nesterov's method for L-smooth convex optimization:

$$
\begin{aligned}
\mathbf{x}_k &= \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_{k-1} \\
\mathbf{v}_k &= \mathbf{v}_{k-1} - a_k\nabla f(\mathbf{x}_k) \\
\mathbf{y}_k &= \mathbf{x}_k - \frac{1}{L}\nabla f(\mathbf{x}_k),
\end{aligned}
$$

where L is the smoothness constant of f, $a_0 = A_0 = \frac{1}{L}$, $\frac{{a_k}^2}{A_k} = \frac{1}{L}$, $A_k = \sum_{i=0}^k a_i$. We take $\mathbf{x}_0 \in \mathbb{R}^d$ to be an arbitrary initial point and $\mathbf{y}_0 = \mathbf{v}_0 = \mathbf{x}_0 - \nabla f(\mathbf{x}_0)/L$.

Prove that we can state Nesterov's method in the following equivalent form:

$$
\begin{aligned}
\mathbf{x}_k &= \mathbf{y}_{k-1} + \frac{a_k}{A_k}\Big(\frac{A_{k-1}}{a_{k-1}} - 1\Big)(\mathbf{y}_{k-1} - \mathbf{y}_{k-2}), \\
\mathbf{y}_k &= \mathbf{x}_k - \frac{1}{L}\nabla f(\mathbf{x}_k).
\end{aligned}
\tag{6}
$$

Hint: It is helpful to first prove that $\mathbf{y}_k = \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_k$. [10pts]

Solution:

Notice that we want to show that the method determined by the rule:

$$\begin{gathered}\mathbf{x}_k = \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_{k-1}(1)\\ \mathbf{v}_k = \mathbf{v}_{k-1} - a_k\nabla f(\mathbf{x}_k)(2)\\ \mathbf{y}_k = \mathbf{x}_k - \frac{1}{L}\nabla f(\mathbf{x}_k)(3)\end{gathered}$$

implies:

$$\begin{gathered}\mathbf{x}_k = \mathbf{y}_{k-1} + \frac{a_k}{A_k}\Big(\frac{A_{k-1}}{a_{k-1}} - 1\Big)(\mathbf{y}_{k-1} - \mathbf{y}_{k-2})(4)\\ \mathbf{y}_k = \mathbf{x}_k - \frac{1}{L}\nabla f(\mathbf{x}_k)(5)\end{gathered} \tag{7}$$

We want to first show that (1), (2), and (3) implies:

$$\mathbf{y}_k = \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_k$$

Notice that

Thus we establish that from the first group of rules, they imply $\mathbf{y}_k = \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{y}_k$.

Secondly, we want to prove that:

$$\mathbf{y}_k = \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_k$$

implies:

$$\mathbf{x}_k = \mathbf{y}_{k-1} + \frac{a_k}{A_k}(\frac{A_{k-1}}{a_{k-1}} - 1)(\mathbf{y}_{k-1} - \mathbf{y}_{k-2})$$

since from (1)

$$\begin{aligned}\mathbf{x}_k &= \frac{A_{k-1}}{A_k}\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_{k-1}\\ &= \mathbf{y}_{k-1} + (\frac{A_{k-1}}{A_k} - 1)\mathbf{y}_{k-1} + \frac{a_k}{A_k}\mathbf{v}_{k-1}\\ &= \mathbf{y}_{k-1} + (\frac{A_{k-1}}{A_k} - 1)\mathbf{y}_{k-1} + \frac{a_k}{A_k}\frac{A_{k-1}}{a_{k-1}}(\mathbf{y}_{k-1} - \frac{A_{k-2}}{A_{k-1}}\mathbf{y}_{k-2})\\ &= \mathbf{y}_{k-1} + (\frac{-a_k}{A_k})\mathbf{y}_{k-1} = \frac{a_k}{A_k}\frac{A_{k-1}}{a_{k-1}}(\mathbf{y}_{k-1} - \frac{A_{k-2}}{A_{k-1}}\mathbf{y}_{k-2})\\ &= \mathbf{y}_{k-1} + \frac{a_k}{A_k}(\frac{A_{k-1}}{a_{k-1}} - 1)\mathbf{y}_{k-1} - (\frac{a_k}{A_k}\frac{A_{k-1}}{a_{k-1}}\frac{A_{k-2}}{A_{k-1}})\mathbf{y}_{k-2}\\ &= \mathbf{y}_{k-1} + \frac{a_k}{A_k}(\frac{A_{k-1}}{a_{k-1}} - 1)\mathbf{y}_{k-1} - \frac{a_k}{A_k}(\frac{A_{k-1}}{a_{k-1}} - 1)\mathbf{y}_{k-2}\\ &= \mathbf{y}_{k-1} + \frac{a_k}{A_k}(\frac{A_{k-1}}{a_{k-1}} - 1)(\mathbf{y}_{k-1} - \mathbf{y}_{k-2})\end{aligned}$$

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. Jupyter notebook is accepted and you can include the discussion and figures in the Jupyter notebook. You may only use modules from the Python standard library plus NumPy and Matplotlib. Do **not** archive your code file(s) into a zip file.

You should turn in both the code (as text file(s)) and a pdf with the figures produced by your code together with the appropriate answers to the questions below.

Q 6. In this question, you are asked to implement the following algorithms:

- Gradient descent with the constant step size $\alpha_k = 1/L$.
- Gradient descent with the exact line search. In every step, this method sets the step size α_k as

$$\alpha_k = \operatorname*{argmin}_{\alpha \in \mathbb{R}} f(\mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)).$$

- Lagged steepest descent, defined as follows: Let α_k be the exact line search gradient descent step size corresponding to the point $\mathbf{x}_k$. Lagged steepest descent updates the iterates as: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_{k-1}\nabla f(\mathbf{x}_k)$ (i.e., the step size "lags" by one iteration).
- Nesterov's method for smooth convex minimization.

The problem instance we will consider here is $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$, where $d = 200$ and $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{M}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ is characterized by:

$$\mathbf{M} = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 \\ 0 & 0 & -1 & 2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

$\mathbf{M}$ and $\mathbf{b}$ can be generated in Python using:

```
import numpy as np
A = np.zeros((d, d))
for i in range(1, d-1):
   A[i, i-1] = 1
   A[i, i+1] = 1
A[0, 1] = 1
A[n-1, n-2] = 1
M = 2*np.eye(d) - A
b = np.zeros(d)
b[0] = 1
```

Observe that you can compute the minimizer $\mathbf{x}^*$ of f given $\mathbf{M}$ and $\mathbf{b}$, and thus you can also compute $f(\mathbf{x}^*)$. It is possible to show that the top eigenvalue of $\mathbf{M}$ is $L = 4$.

Initialize all algorithms at $\mathbf{x}_0 = \mathbf{0}$. All your plots should be showing the optimality gap $f(\mathbf{x}) - f(\mathbf{x}^*)$ (with $\mathbf{x} = \mathbf{y}_k$ for Nesterov and $\mathbf{x} = \mathbf{x}_k$ for all other methods) on the y-axis and the iteration count on the x-axis. The y-axis should be shown on a logarithmic scale (use `set(gca, 'YScale', 'log')` after the figure command in Matlab).

(i) Use a single plot to compare gradient descent with constant step size, gradient descent with the exact line search, and Nesterov's algorithm. Use different colors for different algorithms and show a legend with descriptive labels (e.g., GD:constant, GD:exact, and Nesterov). Discuss the results. Do you see what you expect from the analysis we saw in class?

(ii) Use a single plot to compare Nesterov's algorithm to lagged steepest descent. You should, again, use different colors and a legend. What can you say about lagged steepest descent? How does it compare to Nesterov's algorithm?

(iii) Modify the output of Nesterov's algorithm and lagged steepest descent: you should still run the same algorithms, but now your plot at each iteration k should show the lowest function value up to iteration k for each of the two algorithms. Discuss the results. [30pts]

Solution:

In the following, I am going to provide the graphs corresponding to each question. The codes will be attached in this file as an appendix, and also in Jupiter Notebook on Canvas.

(i) Use a single plot to compare gradient descent with constant step size, gradient descent with the exact line search, and Nesterov's algorithm. Use different colors for different algorithms and show a legend with descriptive labels (e.g., GD:constant, GD:exact, and Nesterov). Discuss the results. Do you see what you expect from the analysis we saw in class?

The following is the graph comparing three algorithms, with the y-axis telling the distance the current stage function value is from the minimum.What we could notice is the convergence rate is greatest from Nesterov's algorithm, then the exact line search, and then the gradient descent with a constant step size. All of them guarantee a non-increasing function value in each iteration.

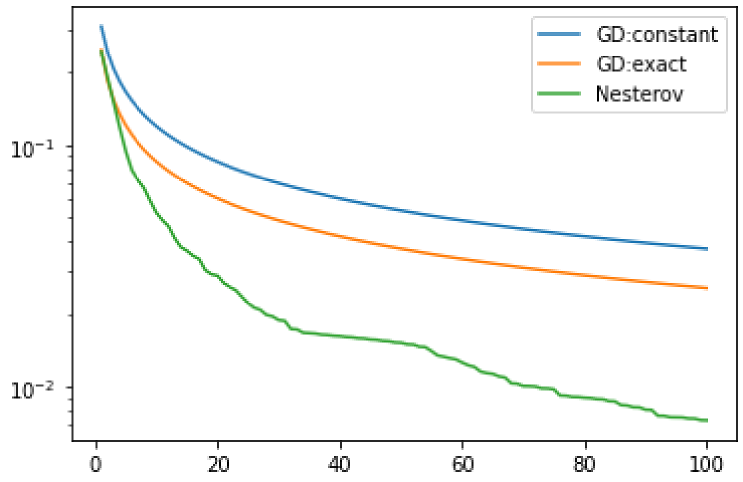


Figure 1: Q1 Graph

(ii) Use a single plot to compare Nesterov's algorithm to lagged steepest descent. You should, again, use different colors and a legend. What can you say about lagged steepest descent? How does it compare to Nesterov's algorithm?

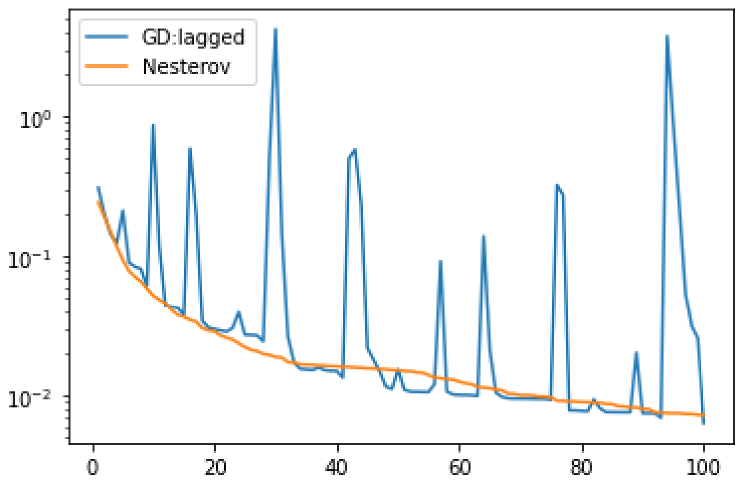


Figure 2: Q2 Graph

The lagged steepest descent compared with Nesterov's algorithm, does not gurantee the function value keeps on decreasing.

(iii) Modify the output of Nesterov's algorithm and lagged steepest descent: you should still run the same algorithms, but now your plot at each iteration k should show the lowest function value up to iteration k for each of the two algorithms. Discuss the results.

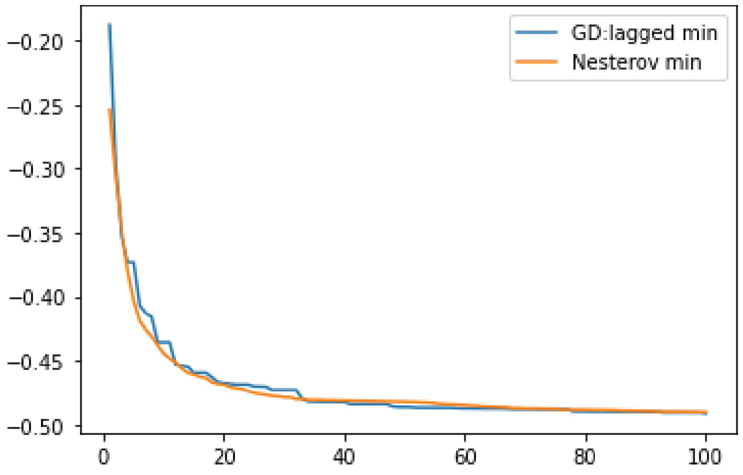


Figure 3: Q3 Graph

Contrary to the previous question, if we compare the lowest function value up to the current iteration, Nesterov's algorithm does not necessarily dominate the lagged steepest descent.

Appendix

```
# first check the python version
!python --version
```

```
     Python 3.7.15
```

```
# create the matrix.
d = 200
import numpy as np
import math as math
A = np.zeros((d, d))
for i in range(1, d-1):
  A[i, i-1] = 1
  A[i, i+1] = 1
  A[0, 1] = 1
  A[i-1, i-2] = 1
  M = 2*np.eye(d) - A
  b = np.zeros(d)
  b[0] = 1
```

Check a few basic matrix operation syntax

```
L = 4
arr1 = np.array([[1, 2],
                 [3, 4]])
arr2 = np.array([[5, 6],
                 [7, 8]])

arr3 = np.array([[1, 6],
                 [7, 8]])




# matrix multiplication
arr_result =arr1@arr2@arr3

# scalar multplication
arr_result_2= 0.5*arr1

# transpose
arr_result_3 = np.transpose(arr1)
print(np.matmul(arr1,arr2))
print(arr_result)

# inverse
np.linalg.inv(M)

# x* is M^{-1}b
minimizer = np.linalg.inv(M)@b
print(minimizer)
```

```
[[19 22]
 [43 50]]
[[173 290]
 [393 658]]
[0.995 0.99  0.985 0.98  0.975 0.97  0.965 0.96  0.955 0.95  0.945 0.94
 0.935 0.93  0.925 0.92  0.915 0.91  0.905 0.9   0.895 0.89  0.885 0.88
 0.875 0.87  0.865 0.86  0.855 0.85  0.845 0.84  0.835 0.83  0.825 0.82
 0.815 0.81  0.805 0.8   0.795 0.79  0.785 0.78  0.775 0.77  0.765 0.76
 0.755 0.75  0.745 0.74  0.735 0.73  0.725 0.72  0.715 0.71  0.705 0.7
 0.695 0.69  0.685 0.68  0.675 0.67  0.665 0.66  0.655 0.65  0.645 0.64
 0.635 0.63  0.625 0.62  0.615 0.61  0.605 0.6   0.595 0.59  0.585 0.58
 0.575 0.57  0.565 0.56  0.555 0.55  0.545 0.54  0.535 0.53  0.525 0.52
 0.515 0.51  0.505 0.5   0.495 0.49  0.485 0.48  0.475 0.47  0.465 0.46
 0.455 0.45  0.445 0.44  0.435 0.43  0.425 0.42  0.415 0.41  0.405 0.4
 0.395 0.39  0.385 0.38  0.375 0.37  0.365 0.36  0.355 0.35  0.345 0.34
 0.335 0.33  0.325 0.32  0.315 0.31  0.305 0.3   0.295 0.29  0.285 0.28
 0.275 0.27  0.265 0.26  0.255 0.25  0.245 0.24  0.235 0.23  0.225 0.22
 0.215 0.21  0.205 0.2   0.195 0.19  0.185 0.18  0.175 0.17  0.165 0.16
 0.155 0.15  0.145 0.14  0.135 0.13  0.125 0.12  0.115 0.11  0.105 0.1
 0.095 0.09  0.085 0.08  0.075 0.07  0.065 0.06  0.055 0.05  0.045 0.04
 0.035 0.03  0.025 0.02  0.015 0.01  0.005 0.   ]
```

cost_f is a function which inputs x and outouts the function value $\frac{1}{2}x^T MX - b^T x$. Furthermore, we can direcltly compute the minimizer to this function is $M^{-1}b$, and thus we have the minimum to this function. In my codes, they are called minimizer and minimum.

```
def cost_f(x):
  result = 0.5*np.transpose(x)@M@x - np.transpose(b)@x
  return(result)

# create the x_0 all zero in 200 dimensions
x_0 = np.zeros(200)
print(x_0)

## compute the gradient at x_k
def gradient(x):
  return(M@x - b)
```

```
## the manunally computed miminum
minimum = cost_f(minimizer)
print(minimum)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
-0.4975000000000001
```

The following implements the gradient descent with constant step size. We use the array store_f_basic to store the function value after each iteration.

```
## The basic gradient descent
store_f_basic = np.zeros(100)

## initialization
x= np.zeros(200)
cost = cost_f(x)
for i in range(100):
  x = x - 0.25*gradient(x)
  store_f_basic[i] = cost_f(x)

print(store_f_basic)
```

```
[-0.1875     -0.25390625 -0.29052734 -0.31452942 -0.3318119  -0.34501898
 -0.35553555 -0.36416624 -0.37141468 -0.37761433 -0.38299591 -0.38772483
 -0.39192315 -0.39568321 -0.39907631 -0.4021585  -0.40497453 -0.40756062
 -0.40994645 -0.41215661 -0.41421164 -0.41612888 -0.41792307 -0.41960683
 -0.42119105 -0.42268516 -0.42409739 -0.42543496 -0.42670422 -0.4279108
 -0.42905969 -0.43015534 -0.43120175 -0.43220249 -0.43316079 -0.43407958
 -0.43496149 -0.43580892 -0.43662407 -0.43740893 -0.43816534 -0.43889497
 -0.43959937 -0.44027996 -0.44093805 -0.44157485 -0.44219148 -0.442789
 -0.44336836 -0.44393047 -0.44447617 -0.44500624 -0.44552141 -0.44602237
 -0.44650977 -0.44698419 -0.4474462  -0.44789635 -0.44833512 -0.448763
 -0.44918041 -0.44958779 -0.44998552 -0.45037399 -0.45075354 -0.45112452
 -0.45148723 -0.45184199 -0.45218908 -0.45252877 -0.45286132 -0.45318698
 -0.45350598 -0.45381855 -0.45412489 -0.45442523 -0.45471973 -0.4550086
 -0.45529202 -0.45557014 -0.45584314 -0.45611116 -0.45637437 -0.45663289
 -0.45688687 -0.45713644 -0.45738173 -0.45762285 -0.45785993 -0.45809307
 -0.45832238 -0.45854797 -0.45876994 -0.45898838 -0.45920338 -0.45941504
 -0.45962344 -0.45982865 -0.46003077 -0.46022988]
```

The Exact line search that we need to have different step size,
Based on my computation, the exact step size is:
α_k = $\frac{\nabla f(x_k)^T M x_k - b^T \nabla f(x_k)}{(\nabla f(x_k)^T M f(x_k))}$

step_size_exact_line is a function takes in x and compute α. We use store_f_exact to store the function value after each iteration.

```
# this is a function which outputs the above alpha_k by inputing x_k
def step_size_exact_line(x):
  numerator = -np.transpose(b)@gradient(x) + np.transpose(gradient(x))@M@x
  denominator = np.transpose(gradient(x))@M@gradient(x)
  value = numerator/denominator
  return(value)

x= np.zeros(200)
## example for myself
print(step_size_exact_line(x))

## the exact line search
# initialize the starting point
x = np.zeros(200)

# get a array to store the cost at each iteration
store_f_exact = np.zeros(100)

for i in range(100):
  # first get step size
  step_size = step_size_exact_line(x)
  x = x - step_size*gradient(x)
  store_f_exact[i] = cost_f(x)



print(store_f_exact)
```

```
0.5
[-0.25       -0.3125     -0.34375    -0.36328125 -0.37695312 -0.38720703
 -0.39526367 -0.40180969 -0.40726471 -0.41190147 -0.41590595 -0.41940987
 -0.42250949 -0.42527701 -0.42776778 -0.43002503 -0.43208312 -0.4339697
 -0.43570734 -0.43731466 -0.43880716 -0.44019791 -0.44149796 -0.44271675
 -0.44386241 -0.44494198 -0.44596158 -0.44692655 -0.44784161 -0.44871091
 -0.44953816 -0.45032662 -0.45107925 -0.45179867 -0.45248726 -0.45314716
 -0.45378031 -0.45438846 -0.45497323 -0.45553606 -0.4560783  -0.45660118
 -0.45710582 -0.45759325 -0.45806444 -0.45852026 -0.45896153 -0.45938902
 -0.45980342 -0.46020538 -0.46059552 -0.46097441 -0.46134258 -0.46170052
 -0.4620487  -0.46238755 -0.46271748 -0.46303888 -0.46335211 -0.46365751
 -0.4639554  -0.46424608 -0.46452984 -0.46480695 -0.46507767 -0.46534223
 -0.46560087 -0.46585381 -0.46610124 -0.46634338 -0.4665804  -0.46681248
 -0.46703979 -0.46726249 -0.46748074 -0.46769469 -0.46790446 -0.4681102
 -0.46831203 -0.46851008 -0.46870447 -0.46889529 -0.46908267 -0.4692667
 -0.46944749 -0.46962512 -0.46979969 -0.46997128 -0.47013998 -0.47030587
 -0.47046902 -0.47062952 -0.47078742 -0.47094281 -0.47109574 -0.47124628
 -0.4713945  -0.47154045 -0.47168418 -0.47182576]
```

This part we perform lagged steepest descent. store_f_lagged is an array to store the function value after each iteration of lagged descent algorithm.

```
store_f_lagged = np.zeros(100)
x_old = np.zeros(200)
x_new = x_old - 0.25*gradient(x_old)
store_f_lagged[0] = cost_f(x_new)
for i in (range(1, 100)):
  x_new_update = x_new - step_size_exact_line(x_old)*gradient(x_new)
  x_old = x_new
  x_new = x_new_update
  store_f_lagged[i] = cost_f(x_new_update)
print(store_f_lagged)
```

```
[-0.1875     -0.296875   -0.35329861 -0.37315538 -0.28480457 -0.40721423
 -0.41285505 -0.41580772 -0.43600432  0.37152334 -0.37836103 -0.45345818
 -0.45391402 -0.4549562  -0.45964747  0.09427691 -0.29824042 -0.46326884
 -0.4668073  -0.467576   -0.4682471  -0.46882537 -0.46711448 -0.45784423
 -0.47027239 -0.47039393 -0.47057794 -0.4730827   0.02161616  3.76685807
 -0.34779156 -0.47111618 -0.48025915 -0.4819522  -0.48209294 -0.48224112
 -0.48160989 -0.48225197 -0.48246963 -0.48248958 -0.48400833  0.00464923
  0.08414461 -0.26243698 -0.47555822 -0.47943218 -0.48259289 -0.48587656
 -0.48631697 -0.48216209 -0.48644905 -0.48680589 -0.48683181 -0.48685981
 -0.48689638 -0.48545144 -0.4049009  -0.48677477 -0.48728702 -0.48737051
 -0.48739052 -0.48741557 -0.48753431 -0.3578661  -0.47631906 -0.48700641
 -0.48770119 -0.4879242  -0.48795688 -0.48796991 -0.48796583 -0.48798966
 -0.48800721 -0.48801364 -0.48813266 -0.17401111 -0.22209439 -0.48963202
 -0.48964436 -0.48970252 -0.48975157 -0.48808676 -0.48937156 -0.48984835
 -0.48986271 -0.48986788 -0.48987567 -0.48988519 -0.47726053 -0.48999671
 -0.49000653 -0.49001027 -0.49054123  3.30726528  0.39099761 -0.27531838
 -0.44386299 -0.46599211 -0.4718566  -0.49113549]
```

In the following we perform the Nesterov accelerated method. collect_f_Nesterov is an array to store the function value after each iteration of Nesterov's algorithm.

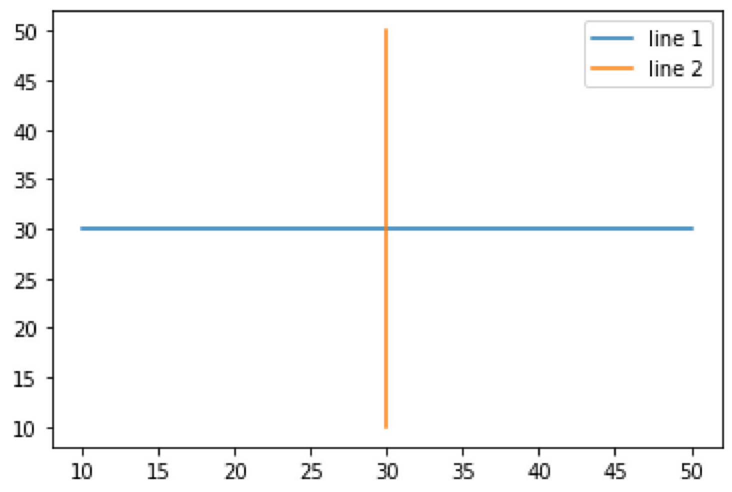
Double-click (or enter) to edit

```
# first initialize v, y, x, and a_0 = A_0
L = 4
A_old = 1/L
x = np.zeros(200)
y = x - gradient(x)/L
v = y
collect_f_Nesterov = np.zeros(100)
for i in range(100):
  a = step_size_exact_line(x)
  A_new = A_old+a
  x = A_old/A_new*y + a/A_new*v
  v = v - a*gradient(x)
  y = x - 0.25*gradient(x)
  A_old = A_new
  collect_f_Nesterov[i] = cost_f(y)

print(collect_f_Nesterov)
```

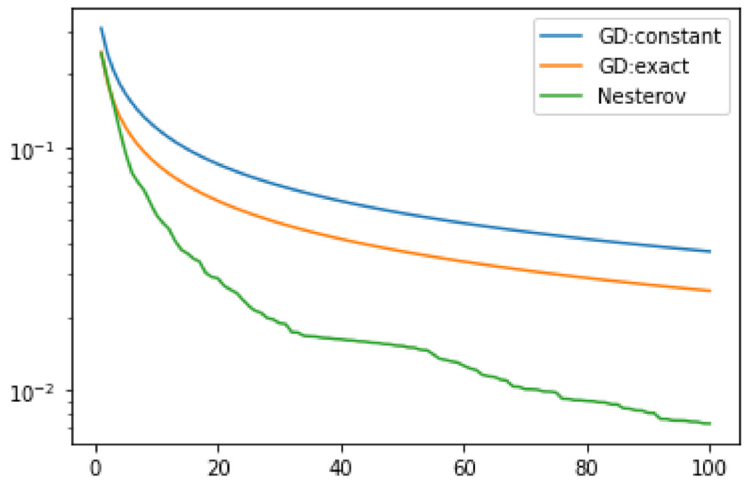
```
[-0.25390625 -0.30627137 -0.34727054 -0.37959937 -0.40336475 -0.41893139
 -0.42575358 -0.43101745 -0.43854854 -0.44486316 -0.44854403 -0.45129453
 -0.45619725 -0.45958953 -0.46090312 -0.46263099 -0.46369327 -0.46707362
 -0.46823656 -0.4686442  -0.47058732 -0.47154896 -0.4723232  -0.4738642
 -0.47526702 -0.47624525 -0.47668317 -0.47767046 -0.47795578 -0.47863084
 -0.47877196 -0.48017145 -0.48023317 -0.48077932 -0.48083439 -0.48090264
 -0.48107866 -0.48112415 -0.48124754 -0.48130168 -0.48142923 -0.48148157
 -0.48161602 -0.48166644 -0.48181093 -0.48185954 -0.48201843 -0.48206557
 -0.48224604 -0.48229236 -0.48250889 -0.48255595 -0.48284658 -0.48290137
 -0.48343099 -0.48401254 -0.48417948 -0.48436341 -0.48451037 -0.48489004
 -0.48521635 -0.4854073  -0.48592857 -0.4860956  -0.48617869 -0.48645218
 -0.48656893 -0.48715574 -0.48718725 -0.48740764 -0.48742158 -0.48745969
 -0.48762981 -0.48764404 -0.4877208  -0.48827891 -0.48830312 -0.48840573
 -0.48841693 -0.48847616 -0.48850459 -0.4885901  -0.48861239 -0.48875943
 -0.48877907 -0.48908138 -0.48911301 -0.48925211 -0.48926322 -0.48944405
 -0.48946203 -0.48989537 -0.48990205 -0.48999691 -0.49000402 -0.49001328
 -0.49008715 -0.49009668 -0.49020396 -0.49021653]
```

We import the matplotlib here and try the graphing syntax.

```
import matplotlib.pyplot as plt

# create data
x = [10,20,30,40,50]
y = [30,30,30,30,30]

# plot lines
plt.plot(x, y, label = "line 1")
plt.plot(y, x, label = "line 2")
plt.legend()
plt.show()
```



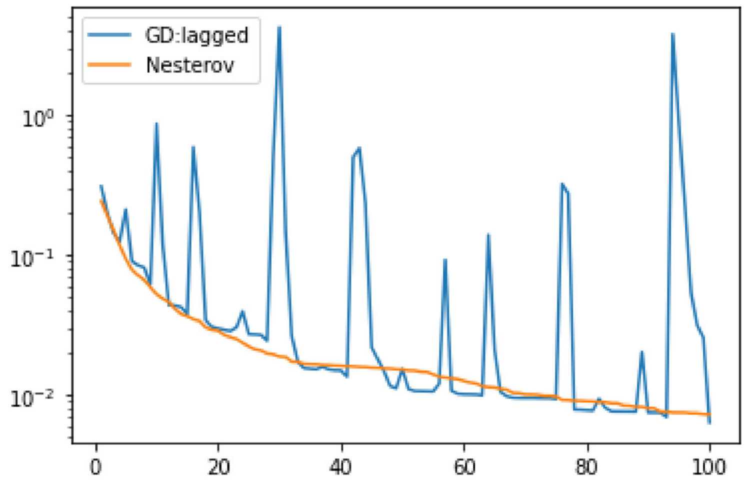
```
# Compute the distance between each stage and the true minimum
opt_gap_basic_GD = store_f_basic - minimum
opt_gap_exact_line_GD = store_f_exact - minimum
opt_gap_NAG= collect_f_Nesterov - minimum

step = np.arange(1, 101, 1, dtype=int)
# plot lines
plt.yscale("log")
plt.plot(step, opt_gap_basic_GD, label = "GD:constant")
plt.plot(step, opt_gap_exact_line_GD, label = "GD:exact")
plt.plot(step,opt_gap_NAG, label = "Nesterov")
plt.legend()
plt.show()
```



Question 2: compare Nesterov's algorithm to lagged steepest descent.

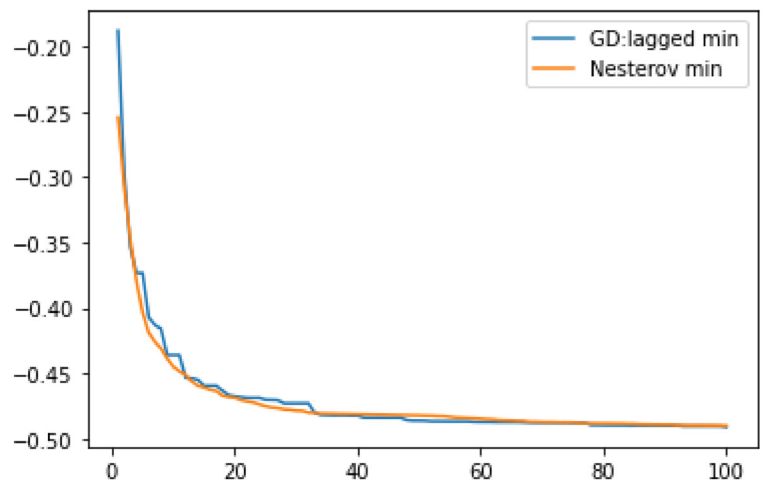
```
# Compute the distance between each stage and the true minimum
opt_gap_lagged = store_f_lagged - minimum
```

```
opt_gap_NAG= collect_f_Nesterov - minimum

step = np.arange(1, 101, 1, dtype=int)
# plot lines
plt.yscale("log")
plt.plot(step, opt_gap_lagged, label = "GD:lagged")
plt.plot(step,opt_gap_NAG, label = "Nesterov")
plt.legend()
plt.show()
```



Question 3 check the minimum up to the current iteration

```
lagged_min = np.zeros(100)
min = store_f_lagged[0]
for i in range(100):
  if(min>store_f_lagged[i]):
    min = store_f_lagged[i]
  lagged_min[i] = min

NAG_min = np.zeros(100)
min = collect_f_Nesterov[0]
for i in range(100):
  if(min>collect_f_Nesterov[i]):
    min = collect_f_Nesterov[i]
  NAG_min[i] = min

# plot lines
plt.plot(step, lagged_min, label = "GD:lagged min")
plt.plot(step,NAG_min, label = "Nesterov min")
plt.legend()
plt.show()
```


GD:lagged min
Nesterov min
−0.20
−0.25
−0.30
−0.35
−0.40
−0.45
−0.50
0
20
40
60
80
100