Princess Tara Zamani
Adv Opt for Machine Learning
Homework 6

<u>Problem 1 - Hard Thresholding</u>

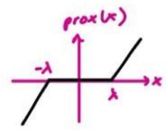
Given the following,

$$prox_h(x) := arg\,min_{z_i \in R} \{ \lambda 1\{z_i \neq 0\} + \frac{1}{2}(z_i - x_i)^2 \}, \forall i = 1, ..., n$$

derive the solution formula for the 1-dimensional optimization problem, and then obtain the formula for the proximal mapping, $prox_{\lambda\|\cdot\|_0}(x)$.

Solution:

$prox_{\lambda\|\cdot\|_1}$ - soft thresholding
$= (|x| - \lambda)_+ \cdot sgn(x)$



$prox_{\lambda\|\cdot\|_0}$ - look at $\underset{z \in \mathbb{R}}{argmin} \underbrace{\{\lambda \mathbb{1}\{z \neq 0\} + \frac{1}{2}(z-x)^2\}}_{F_x(z)}$

- derive analytical form using cases

$$F_x(z) = \begin{cases} \lambda(1) + \frac{1}{2}(z-x)^2, & z \neq 0 \\ \cancel{\lambda(0)} + \frac{1}{2}(z-x)^2, & z = 0 \end{cases} = \begin{cases} \lambda + \frac{1}{2}(z-x)^2, & z \neq 0 \\ \frac{1}{2}x^2, & z = 0 \end{cases}$$

↳ constant

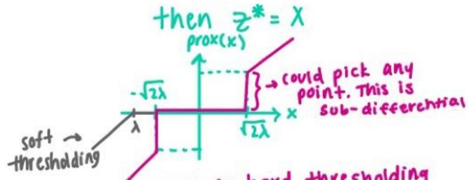
- find minimizer for each part

$$F_x(z) = \begin{cases} \text{min val is } \lambda, & z = x \\ \text{min val is } \frac{x^2}{2}, & z = 0 \end{cases}$$

compare 2 min values
↳ go with what is lower

→ If $\frac{x^2}{2} \leq \lambda \Leftrightarrow |x| \leq \sqrt{2\lambda}$
then $z^* = 0$

If $\lambda < \frac{x^2}{2} \Leftrightarrow |x| > \sqrt{2\lambda}$
then $z^* = x$



→ this is hard thresholding
↳ discontinuous
↳ may lead to local min

$$prox_{\lambda\|\cdot\|_0}(x) = (\mathbb{1}\{|x| > \sqrt{2\lambda}\}) \cdot x$$

Simulation I - Proximal Gradient Descent for Compressed Sensing

Gradient Descent Implementation

The first objective of this assignment was to apply gradient descent to solve the least-mean-square problem defined as:

$$min_{x \in R^n} f(x) = \frac{1}{2}\|Ax - b\|_2^2, \text{ where } x \in R^n, A \in R^{m\times n}, b \in R^m.$$

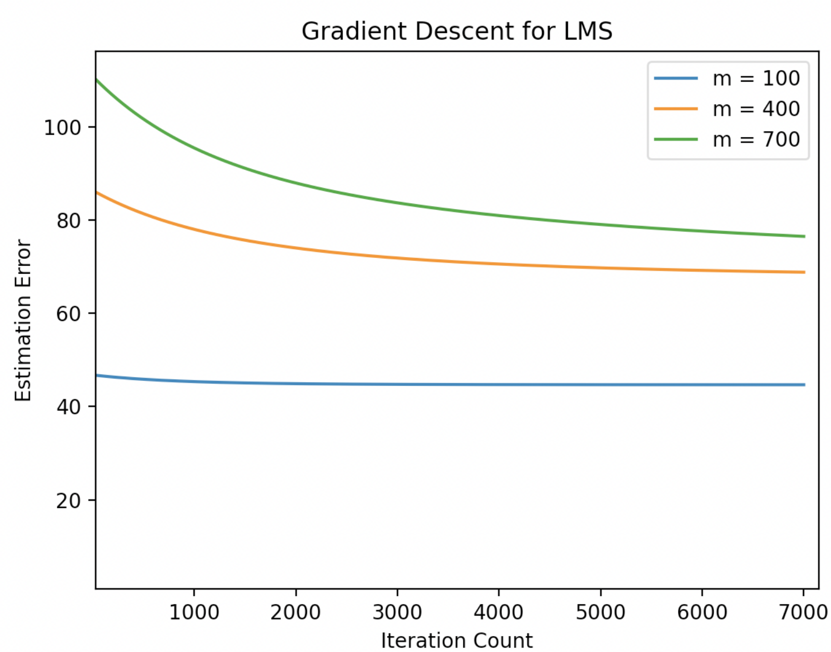
The gradient for the definition above is given as $\nabla f(x) = A^T(Ax - b)$.

The underlying signal (*z*), and the initialization of x_0 were generated randomly using the numpy.random.randn function. The underlying signal was made sparse by randomly replacing 95% of its values with zeros. The initial noise variance was chosen to be 0.1. The variables *A* and *b* were generated in the same way as in HW2.

The gradient algorithm implementation is iterative and takes the following form:

$$x_{k+1} = x_k - \eta\nabla f(x_k), \text{ where } \eta \text{ is the learning rate.}$$

The results were obtained using the following parameters: n = 1000, learning rate = 1e-3, variance = 0.1, iteration number k = 7000. The algorithm was run three times setting the value of m to 100, 400, 700.
The results of the gradient descent implementation estimation error are shown below.



We can see that as m increases, the longer it takes for the error to converge. This is logical because a larger m means that more samples are provided and thus, more gradients need to be updated and obtained.

Proximal Gradient Descent Implementation

The second objective of this assignment was to apply proximal gradient descent to solve the compressed sensing problem defined as:

$$min_{x \in R^n} f(x) \ = \frac{1}{2}\|Ax - b\|_2^2 \ + \ \lambda||x||_1$$, where x ∈ R^n , A ∈ $R^{m\times n}$, b ∈ R^m.

The proximal operator of $\lambda||x||_1$ is given as:

$$prox_{\lambda\|\cdot\|_1}(x) \ = \ (|x| - \lambda)_+ * sign(x) \ , \ \ where(u)_+ := \ max\{u, 0\}$$

The underlying signal (*z*), and the initialization of x_0 were generated randomly using the numpy.random.randn function. The underlying signal was made sparse by randomly replacing 95% of its values with zeros. The initial noise variance was chosen to be 0.1. The variables *A* and *b* were generated in the same way as in HW2.

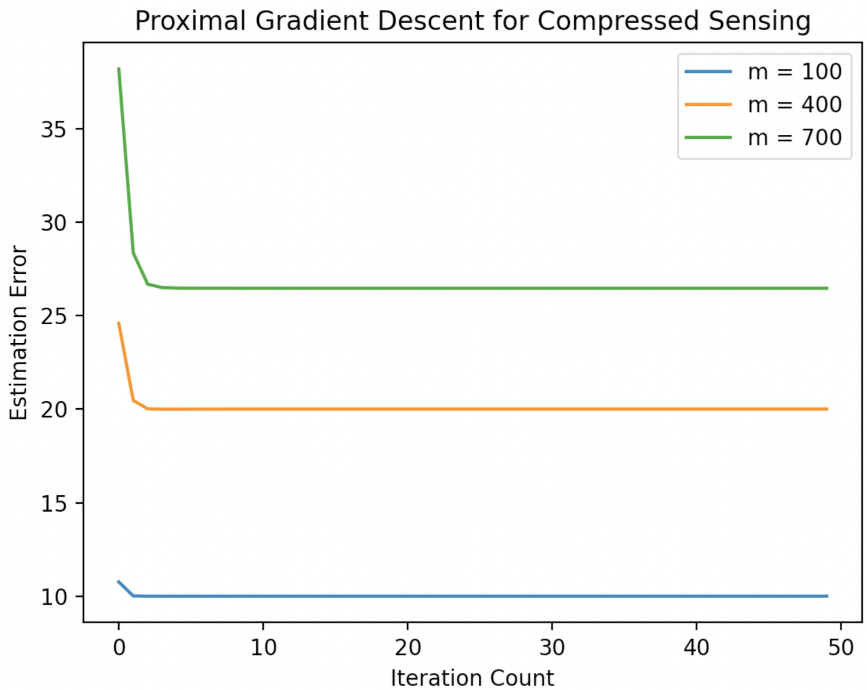
The proximal gradient algorithm implementation is iterative and takes the following update rule:

$$x_{k+1} = prox_{\lambda\|\cdot\|_1}(x_k - \eta\nabla f(x_k))$$, where η is the learning rate.

The λ was fined tuned by experimentally running the model with the values of $\lambda = [10, \ 1, \ 0.1, \ 0.01, \ 0.001, \ 0.0001]$ and seeing what the lowest error was. The final λ was determined to be 0.1.

The results were obtained using the following parameters: n = 1000, learning rate = 1e-3, λ = 0.1, variance = 0.1, iteration number k = 50. The algorithm was run three times setting the value of m to 100, 400, 700.

The results of the proximal gradient descent implementation estimation error are shown below.



We see that as m increases, the slope of the estimation error decreases at a steeper rate and although it is a little tough to see, it seems that it might take the larger m values a few more iterations to converge.

When comparing LMS and CS, CS is clearly a better choice. It takes far fewer iterations to converge and the convergence happens a lot faster, for all m values, than in the LMS model.