最优化理论与算法大作业

Jingwei Liang*

November 21, 2021

1 Introduction

In this project, we will investigate how to use *gradient descent* to solve signal and image processing problem.

Consider the following observation model

$$\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}} + \epsilon \tag{1.1}$$

where we have

- $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ is the ground truth signal/image;
- $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is our observation matrix;
- $\epsilon \in \mathbb{R}^m$ is additive white Gaussian noise, and
- $\boldsymbol{b} \in \mathbb{R}^m$ is our observation.

Our goal is to recover $\boldsymbol{x}$ from the noise contaminated observation $\boldsymbol{b}$. To this end, we can consider the following model

$$\min_{\boldsymbol{x}} \ \mu h(\nabla \boldsymbol{x}) + f(\boldsymbol{x})$$

where

- $\mu > 0$ is the tradeoff parameter to balance the two terms;
- $f(\boldsymbol{x})$ is the data fidelity term, typical choice of f takes
$$f(\boldsymbol{x}) = \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2.$$
- ∇ is the discrete gradient operator to be discussed.
- $h(\nabla \boldsymbol{x})$ is the regularization term whose form will be specified later.

1.1 Discrete gradient operator

In this part, we present the matrix form of the discrete gradient operator. In 1D case, the form of ∇ is rather simple, which reads

$$\nabla_{1d} = \begin{bmatrix} -1 & 1 & & & & \\ & -1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 1 & \\ & & & & -1 & 1 \\ & & & & & 0 \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

*Institute of Natural Sciences, Shanghai Jiao Tong University. E-mail: jingwei.liang@sjtu.edu.cn

In Figure 1 below we provide an example of piecewise constant 1D signal. Its discrete gradient is provided in Figure 2. It is also easy to compute the transpose of ∇_{1d}, which is

$$\nabla_{1d}^T = \begin{bmatrix} -1 & & & & & \\ 1 & -1 & & & & \\ & 1 & \ddots & & & \\ & & \ddots & -1 & & \\ & & & 1 & -1 & \\ & & & & 1 & 0 \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

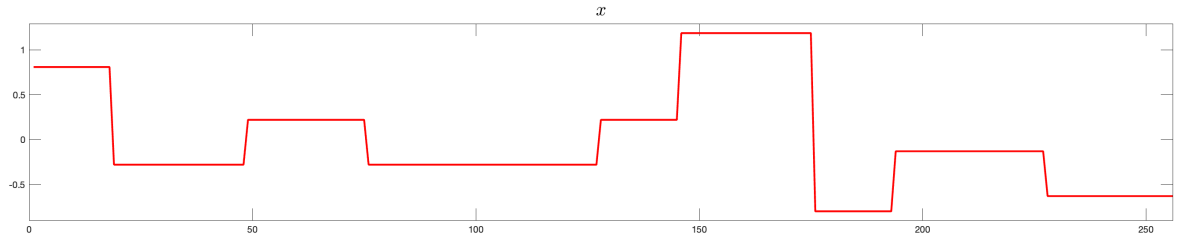

Figure 1: Example of 1D signal.

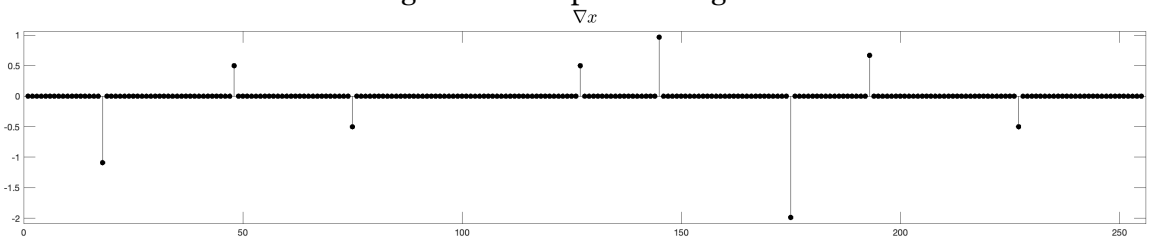

Figure 2: Gradient of the 1D signal.

The expression of ∇ becomes complicated for the 2D cases, since in this case we have two directions: vertical and horizontal. As we can see from Figure 3, the horizontal gradient and vertical gradient of an given image.

Figure 3: Gradient of the 1D signal.

Before the discussion, we need the following setups. First we have that ∇ can be written as

$$\nabla_{2d} = \begin{bmatrix} \nabla_h \\ \nabla_v \end{bmatrix},$$

where ∇_h corresponds to horizontal discrete gradient, and ∇_v for the vertical one. Apparently, we have

$$\nabla_{2d}^T = \begin{bmatrix} \nabla_h^T & \nabla_v^T \end{bmatrix}$$

Then given an square matrix $\boldsymbol{x} \in \mathbb{R}^{n \times n}$, let $\boldsymbol{c}_i \in \mathbb{R}^n, i = 1, ..., n$ be its columns and $\boldsymbol{r}_i \in \mathbb{R}^{1 \times n}, i = 1, ..., n$

be its rows, that is

$$\boldsymbol{x} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n] = \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \vdots \\ \boldsymbol{r}_n \end{bmatrix}$$

In general, there two approaches to represent the 2D discrete gradient operators — explicitly writing down the matrix, and implicitly implement the matrix.

- The first one is the same as the 1D case, that we explicitly store the matrix. For this approach, we store the image as an column vector, that is

  $$\mathrm{vec}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \vdots \\ \boldsymbol{c}_n \end{bmatrix} \in \mathbb{R}^{nn}.$$

  Let $\mathrm{Id} \in \mathbb{R}^{n\times n}$ be the identity matrix, then we have

  $$\nabla_h = \nabla_{1d} \otimes \mathrm{Id} \quad \text{and} \quad \nabla_v = \mathrm{Id} \otimes \nabla_{1d},$$

  where $\otimes$ stands for the "Kronecker product" of two matrices whose definition is provided below in the appendix.

  Use the above expression, we need to solve the problem in the space of $\mathbb{R}^{nn}$ and back to $\mathbb{R}^{n\times n}$ once the problem is solved.

  One problem of this approach is the dimension of ∇_{2d}, which is $2nn \times nn$. Though it is a sparse matrix, this approach is not effective in practice.
- The second approach is that, no need to store ∇_h and ∇_v, only need to know the output once we apply them to an image. More precisely, we have

  $$\nabla_h \boldsymbol{x} = \begin{bmatrix} \boldsymbol{c}_2 - \boldsymbol{c}_1 & \boldsymbol{c}_3 - \boldsymbol{c}_2 & \cdots & \boldsymbol{c}_n - \boldsymbol{c}_{n-1} & \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

  and

  $$\nabla_v \boldsymbol{x} = \begin{bmatrix} \boldsymbol{r}_2 - \boldsymbol{r}_1 \\ \boldsymbol{r}_3 - \boldsymbol{r}_2 \\ \vdots \\ \boldsymbol{r}_n - \boldsymbol{r}_{n-1} \\ \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

  Then for the transpose,

  $$\nabla_h^T \boldsymbol{x} = \begin{bmatrix} -\boldsymbol{c}_1 & \boldsymbol{c}_1 - \boldsymbol{c}_2 & \boldsymbol{c}_2 - \boldsymbol{c}_3 & \cdots & \boldsymbol{c}_{n-2} - \boldsymbol{c}_{n-1} & \boldsymbol{c}_{n-1} \end{bmatrix}$$

  and

  $$\nabla_v^T \boldsymbol{x} = \begin{bmatrix} -\boldsymbol{r}_1 \\ \boldsymbol{r}_1 - \boldsymbol{r}_2 \\ \boldsymbol{r}_2 - \boldsymbol{r}_3 \\ \vdots \\ \boldsymbol{r}_{n-2} - \boldsymbol{r}_{n-1} \\ \boldsymbol{r}_{n-1} \end{bmatrix}.$$

  All the above computation can be easily realized in MATLAB using `diff` function.

1.2 The choice of h

Now we discuss the choice of $h(\boldsymbol{x})$. Recall that the absolute value function is not differentiable, below we discuss a smoothed version of it. Let $\eta > 0$ be a strictly positive constant, and consider

$$\phi_\eta(x) = \begin{cases} |x| - \frac{\eta}{2} : & |x| \geq \eta, \\ \frac{1}{2\eta} x^2 : & |x| < \eta. \end{cases}$$

It can be easily verified that ϕ_η is smooth differentiable. Back to h, we take

$$h(\boldsymbol{x}) = \sum_{i=1}^{n} \phi_\eta(x_i). \tag{1.2}$$

Note that $\eta > 0$ is a free parameter here.

We can also consider the following choice of h,

$$h(\boldsymbol{x}) = \|\nabla \boldsymbol{x}\|^2, \tag{1.3}$$

which is smooth differentiable.

2 Signal reconstruction

The first problem we consider is the 1D signal reconstruction. For this problem, consider the following settings for (1.1) which can be changed at your will

- $(m, n) = (64, 256)$ [you can also consider other choices of (m, n), but please make sure $m < n/2$].
- $\bar{\boldsymbol{x}}$ is a piecewise constant signal with $\kappa = 8$ jumps [you can also consider other choices of κ, with $\kappa < n/10$ and $m > 5\kappa$]. `A pre-generated` $\bar{x}$ `is available at ``x.txt'' file, or you can generate it using MATLAB function ``barx = func_piecewise_constant(n, kappa);''.`
- $\boldsymbol{A}$ is a random Gaussian matrix with variance $1/\sqrt{m}$. `In MATLAB, you can generate such an` $\boldsymbol{A}$ `using ``A = randn(m,n) /sqrt(m);''.`
- ϵ is random Gaussian noise with variance 0.5. `In MATLAB, you can generate such an` ϵ `using ``epsilon = randn(m,1) /2;''.`

With the above settings, we can generate our observation $\boldsymbol{b}$ of (1.1).

Now consider reconstruction $\bar{\boldsymbol{x}}$ using the following model

$$\min_{\boldsymbol{x}} \ \Phi(\boldsymbol{x}) = \mu h(\nabla \boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \tag{2.1}$$

where we have the following tasks:

1. For h of (1.2):
   - Write down the gradient of the objective function $\Phi(\boldsymbol{x})$, and consequently the gradient descent scheme to solve (3.1).
   - What is the Lipschitz constant of the gradient of $\Phi(\boldsymbol{x})$.
   - Implement the gradient descent to solve (3.1), and observe the convergence behaviors of the algorithms in terms of
     $$\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\| \quad \text{and} \quad \Phi(\boldsymbol{x}_k) - \Phi(\boldsymbol{x}_{k-1}).$$
     Use 10^{-8} as the stopping criterion of the algorithm.
   - Compare the effect of different μ, and try to find or approximate the optimal μ if there is. If there is such an optimal μ, describe your approach of approximating it.
   - Compare the effect of different η. How should we choose η, the larger the better or the smaller the better?
2. Compare the difference between h of (1.2) and h of (1.3), which one is better?
3. Suppose that the noise ϵ is not Gaussian distributed, and rather uniformly distributed in $[-0.1, 0.1]$,

will (3.1) produce satisfactory result?

3 Optional: image denoising

Let $\boldsymbol{A}$ be the identity operator in (1.1), the problem becomes image denoising. Consider the following settings for (1.1) which can be changed at your will

- Find a gray scale image (using the provided ones or find one yourself), better be a square image with resolution smaller than 256×256.
- After loading the image, the range of each pixel value should be in $[0, 255]$ whichever platform you are suing.
- ϵ is random Gaussian noise with variance 20. `In MATLAB, you can generate such an` ϵ `using ``epsilon = 20* randn(nn);''.`

With the above settings, we can generate our observation $\boldsymbol{b}$ of (1.1).

Consider the following denoising model

$$\min_{\boldsymbol{x}} \Phi(\boldsymbol{x}) = \mu h(\nabla \boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}\|^2, \tag{3.1}$$

where we have the following tasks: for h of (1.2):

- Implement the gradient descent to solve (3.1), and observe the convergence behaviors of the algorithms in terms of

  $$\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\| \quad \text{and} \quad \Phi(\boldsymbol{x}_k) - \Phi(\boldsymbol{x}_{k-1}).$$

  Use 10^{-8} as the stopping criterion of the algorithm.
- Compare the effect of different μ, and try to find or approximate the optimal μ if there is. If there is such an optimal μ, describe your approach of approximating it.
- Compare the effect of different η. How should we choose η, the larger the better or the smaller the better?

A Appendix

Kronecker product Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{p\times q}$ be two matrices, with

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix} \quad \text{and} \quad \boldsymbol{B} = \begin{bmatrix} b_{1,1} & \cdots & b_{1,q} \\ \vdots & \ddots & \vdots \\ b_{p,1} & \cdots & b_{p,q} \end{bmatrix}.$$

Their Kronecker product, denoted by $\boldsymbol{A} \otimes \boldsymbol{B}$, is an $mp \times nq$ matrix with the block structure

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{1,1}\boldsymbol{B} & \cdots & a_{1,n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m,1}\boldsymbol{B} & \cdots & a_{m,n}\boldsymbol{B} \end{bmatrix} \in \mathbb{R}^{mp\times nq}.$$

Moreover, we have

$$(\boldsymbol{A} \otimes \boldsymbol{B})^T = \boldsymbol{A}^T \otimes \boldsymbol{B}^T.$$