Modeling and Algorithms for Prof Shi's Project

Hongda Li, Yining Zhou, Xiaoping Shi

February 5, 2024

Abstract

We propose some better algorithm for a problem in detecting structure of probability transition matrices from data.

1 Introduction

We describe an optimization problem introduced by Prof Shi and his student Yining. To start we define the following quantities for the optimization problem.

1. $n \in \mathbb{N}$. It denotes the numer of states for a Markov Chain.
2. $p \in \mathbb{R}^{n \times n}$ denotes the probability transition matrix. It's in small case because it's also the variable for the optimization problem. It supports two types of indexing, $p_{i,j}$ for $i, j \in \{1, \cdots, n\}$, or p_j with $j \in \{1, \cdots, n^2\}$. More on this later.
3. $\eta_{ij} \geq 0$ for $i, j \in \{1, \cdots, n\}$ is a parameter of the problem.
4. $\hat{p}$ is the empirically measured probability transition matrix. They are the maximal likelihood estimators for the transition probability in the transition probability matrix.
5. λ is the regularization parameter.
6. $\mathbf{C}_n^m$ is the combinatoric term that counts all possible subset of size $n, n < m$ from a super set of size m.

When p is referred to as a vector we may say $p \in \mathbb{R}^{n^2}$, if it's referred to as the matrix, we will use $p \in \mathbb{R}^{n \times n}$. When indexing p using a tuple, or a single number, it's possible to translate between the two type of indexing scheme using the following bijective map:

$$\begin{aligned}(i, j) &\mapsto k := i \times n + j \\ k &\mapsto (i, j) := (\lfloor k/n \rfloor, \mathrm{mod}(k, n) + 1).\end{aligned}$$

We emphasize, in different programming languages and development environments, the convention of indexing a muti-array using different kind of tuples can be very different. For now we use the above indexing, which is a row major index convention (Like Python).

1.1 Some Mathematical Entities

We introduce some mathematical entities for more context.

1.2 The Optimization Problem

The optimization problem is posed as

$$\underset{p\in\mathbb{R}^{n^2}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} -\eta_{ij}\log(p_{i,j}) + \lambda \sum_{i=1}^{n^2}\sum_{\substack{j=1\\ j\neq i}}^{n^2} \frac{1}{2}\frac{|p_i - p_j|}{|\hat{p}_i - \hat{p}_j|} \right\} \tag{1.2.1}$$

$$\text{s.t: } \left\{ \begin{aligned} &\sum_{i=1}^{n} p_{i,j} = 1 \quad \forall i = 1,\cdots,n \\ &\quad p_{i,j} \geq 0 \quad \forall i = 1,\cdots,n, j = 1,\cdots,n \end{aligned} \right\} \tag{1.2.2}$$

There are 3 parts to the optimization problem posed above. It has a smooth differentiable function, the sume of $\eta_{ij}\log(p_{i,j})$ but its gradient is not Lipschitz. The ias a non-smooth part with $p_i - p_j$ for all indices $1,\cdots,n^2$ and $i \neq j$. If $\hat{p}_i - \hat{p}_j = 0$, then we would ignore the term. Finally, it has a linear constraints on all n^2 variables. $p \in \mathbb{R}^{n^2}$ is a vector with the structure $\Delta_n \oplus \Delta_n \oplus \cdots \oplus \Delta_n$. Each Δ_n is a probability simplex. It's defined as $\Delta_n = \{x \in \mathbb{R}^n_+ : \sum_{i=1}^n x_n = 1\}$. For simplicity we just denote using notation $\Delta_{n\times n} = \Delta_n \oplus \Delta_n \oplus \cdots \oplus \Delta_n$.

2 Modeling

The non-smooth part with the absolute value requires some amount of creativity if we were to use common optimization algorithms.

2.1 Representation of the Non-smooth Part

Claim 2.1.1. The nonsmooth objective can be model as

$$\lambda \sum_{i=1}^{n^2}\sum_{j>i}^{n^2} \frac{|p_i - p_j|}{|\hat{p}_i - \hat{p}_j|} = \|Cp\|_1 \qquad \text{where } C \text{ is } \mathbf{C}_2^{n^2} \text{ by } n^2.$$

The one norm represents the summation of absolute values. The transformation $\mathbf{C}p$ is linear transformation and it's a vector of length $\mathbf{C}_2^{n^2}$. The vector is long and it has a dimension of $(1/2)(1+n^2)n^2$. Each term inside of the summation is a row of the matrix C. Each row of matrix C has exactly 2 non-zero elements in it. Suppose that $i \in \{1,\cdots,\mathbf{C}_2^{n^2}\}$ denoting the index for a specific row of matrix C denotes the index of a specific row, and $j \in \{1,\cdots,n^2\}$ denotes a specific column of matrix C. Mathematically describing the matrix is difficult, but it can be algorithmically defined. A sparse matrix format can be described by as a mapping from (i,j), the set of all indices to the element in the vector. The following algorithm 1 construct such a mapping.

Algorithm 1 Matrix Make Algorithm

Let C be a $\mathbf{C}_2^{n^2}$ by n^2 zero matrix.
for $i = 1, \cdots, n^2$ **do**
  for $j = 1, \cdots, n^2$ **do**
    if $|\hat{p}_i - \hat{p}_j| == 0$ **then**
      continue
    end if
    $C[i \times n^2 + j, i] := \lambda / |\hat{p}_i - \hat{p}_j|$
    $C[i \times n^2 + j, j] := -\lambda / |\hat{p}_i - \hat{p}_j|$
  end for
end for

Remark 2.1.1. In practice, we should use sparse matrix data format such as SCR, SCC in programming languages.

2.2 Modeling it for Sequential Quadratic Programming

To start, refer to Seqential Qaudratic Programming Wikipedia for brief overview about what sequential quadratic programming problem is. To use sequential programming, we model the non-smooth $\|Cp\|_1$ parts of the objective function as a linear constraints. Define $u \in \mathbb{R}_+^{\mathbf{C}_2^{n^2}}$ then the below problem is equivalent to the original formulation:

$$\underset{p,u}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} -\eta_{ij} \log(p_{i,j}) + \sum_{i=1}^{\mathbf{C}_2^{n^2}} u_k \right\} \tag{2.2.1}$$

$$\text{s.t: } \left\{ \begin{array}{c} -u \leq Cp \leq u \\ p \in \Delta_{n \times n} \\ u \in \mathbb{R}_+^{\mathbf{C}_2^{n^2}} \end{array} \right\} \tag{2.2.2}$$

This is a Non-linear programming problem and it has a convex objective. Common NLP packages in programming languages can solve this efficiently. However it's potentially possible that these solvers are not adapated to huge sparse matrix C that has special structure to it.

Remark 2.2.1. To formulate into a linear programming with some relaxations, consider that the non-linear objective $-\eta_{ij} \log(p_{i,j})$ can be discretized.

2.3 Modeling it for Operator Splitting

Operator splitting method aims for objective function of the type $f + g$ where f, g are both convex function and they are proximal friendly. And $\text{ri.dom}(f) \cap \text{ri.dom}(g) \neq \emptyset$. Recall that proximal operator of function f is the resolvent operator on the subgradient $(I + \partial f)^{-1}$. Subgradient is just a generalize type of gradient that can handle continuous function that is not necessarily differentiable. To model the original formulation in such a form, we introduce these quantities:

1. $f_1(x_1) : \mathbb{R}^{n^2} \mapsto \mathbb{R} := \sum_{i=1}^{n^2} -\eta_{i,j}(\log(p_{i,j}))$
2. $f_2(x_2) : \mathbb{R}^{\mathbf{C}_2^{n^2}} \mapsto \mathbb{R} := \|x_2\|_1$.
3. $f_3(x_3) : \mathbb{R}^{n^2} \mapsto \bar{\mathbb{R}} := \delta_{\Delta_{n\times n}}(x_3)$

The above three functions represent the three parts of the summed objective.
Let $f(x_1, x_2, x_3) := f_1(x_1)+f_2(x_2)+f_3(x_3)$. However, they share different variables x_1, x_2, x_3, from different dimension. We want g to represents the constraints that $x_2 = Cp$, and $x_1 = x_3 = p$. Simplifying: $x_2 = Cx_1, x_1 = x_3$. This is a linear system of the form

$$\begin{bmatrix} C & -I & \mathbf{0} \\ I & \mathbf{0} & -I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \mathbf{0}.$$

Simply denote the above as $Ax = \mathbf{0}$, then g is just a non-smooth function that happens to be an indicator function of a convex set. The convex set is the set of all solutions to the linear system. Therefore setting $g = \delta_{Ax=\mathbf{0}}(x)$, to be the indicator of the linear constraints, we had a complete equivalent representation of the original form of the problem.

The matrix A is a $(\mathbf{C}_2^{n^2} + n^2) \times (3n^2)$ matrix. It's still a sparse matrix. The matrix would have to be encoded carefully by translating copies of the same matrix to 3 different locations.

2.4 Another Method that is not Cursed by the Dimension

...

2.5 A Quick Tour into Operator Splitting Method

Operator splitting method is relatively new and doesn't have a mature toolings in software. Using the method requires the use of the Proximal Mapping, which can be non-trivial to implement for even simple functions. Read section [1, section 2.7] of Ryu's Textbook for more information about the algorithm and operator splitting in general.

3 Implementation, Software, and Toolings

The software for Squential Quadratic programming is relatively mature. Whether they support sparse matrix inversion depends on the developers. The software operator splitting method is less common. To use Operator splitting method such as Douglas Rachford (Equivalently the ADMM via a the Dual), Peachman Rachford, it's required to compute the proximal operator of functions. The following theorem is relevant to efficiently compute the proximal mapping for our model.

Theorem 1 (Separable Proximal Theorem)**.** Suppose that function $f(x)$ can be written as the Euclidean cross product $(f_1(x_1), f_2(x_2), \cdots, f_m(x_m))$, and we assume that each of

$x_1, x_2, \cdots, x_n$ are vector in $\mathbb{R}^{k_1}, \mathbb{R}^{k_2}, \cdots, \mathbb{R}^{k_m}$ such that $x = \bigoplus i = 1^m x_i$,then we can write the proximal gradient operator of f in parallel form which is given as:

$$\text{prox}_{f,\lambda}((x_1, x_2, \cdots, x_m)) = (\text{prox}_{f_1,\lambda}(x_1), \text{prox}_{f_2,\lambda}(x_2), \cdots \text{prox}_{f_m,\lambda}(x_m)).$$

In general, if there exists any permutation of the list of $x_1, x_2, \cdots, x_m$, the parallelization property of prox will be come applicable.

Recall objective $f(x_1, x_2, x_3) = f_1(x_1) + f_2(x_2) + f_3(x_3)$ from section 2.1. It has the applicable form and hence we have:

$$\underset{\lambda f}{\text{prox}}((x_1, x_2, x_3)) = \underset{\lambda f_1}{\text{prox}}(x_1) \oplus \underset{\lambda f_2}{\text{prox}}(x_2) \oplus \underset{\lambda f_3}{\text{prox}}(x_3).$$

Next, we proceed to write each f_i for $i = 1, 2, 3$ into separable sum as well. f_1, f_2 are trivial to do however $\delta_{\Delta_{n\times n}}$ requires some thought. Consider $x_3 = \bigoplus_{i=1}^n x_{3,i}$, so that the vector is the direct product, then

$$\delta_{\Delta_{n\times n}}(x_3) = \sum_{i=1}^{n} \delta_{\Delta_n}(x_{3,i}).$$

The indictor function $\delta_{\Delta_{n\times n}}$ acts on groups of n elements in the vector x_3 individually and separately, hence it admits the above separable summation form. We now list the proximal mapping for each of the atomic operatons: $\lambda \log(\cdot), \lambda|\cdot|, \delta_{\Delta_n}$ and $\delta_{Ax=\mathbf{0}}(\cdot)$. Most of then can be found in the literatures.

$$\begin{aligned}
\underset{-\lambda\log(\cdot)}{\text{prox}}\ (x) &= \frac{x + \sqrt{x^2 + 4\lambda}}{2} \\
\underset{\lambda|\cdot|}{\text{prox}}(x) &= \text{sign}(x)\max(|x| - \lambda, 0) \\
\underset{\delta_{\Delta_n}}{\text{prox}} &= \Pi_{\Delta_n} \\
\underset{\delta_{\{Ax=\mathbf{0}\}}}{\text{prox}} &= \Pi_{Ax=\mathbf{0}}.
\end{aligned}$$

3.1 Projection onto Probability Simplex and Sparse Linear System

Projecting onto a probability simplex and a linar system that is sparse requires a bit more attention for software implementation sake. We introduce the following theorem from Beck textbook to assist with the projection onto a probability simplex.

Theorem 2 (Projecting onto Hyperplane Box Intersections)**.** Let $C \subseteq \mathbb{R}^n$ to be non-empty and defined as $C := \text{Box}[l, u] \cap H^{=}_{a,b}$, where $\text{Box}[l, u] = \{x \in \mathbb{R}^n : l \leq x \leq u\}$. We assume that $l \in [-\infty, \infty)$ and $u \in (-\infty, \infty]$. And $H^{=}_{a,b} = \{x \in \mathbb{R}^n : \langle a, x\rangle = b\}$, where $a \in \mathbb{R}^n \setminus \{\mathbf{0}\}$. Then the projection onto the set is equivalent to the following conditions

$$\begin{aligned}
&\Pi_C(x) = \Pi_{\text{box}[l,u]}(x - \mu a) \\
&\text{where } u \in \mathbb{R} \text{ solves: } \langle a, \Pi_{\text{Box}[l,u]}(x - ua)\rangle = b.
\end{aligned}$$

Proof. See [2, thereom 6.27] by Beck for more information. □

Remark 3.1.1. The equation for u has a continuous mapping $\mathbb{R} \mapsto \mathbb{R}$, it can be solved efficiently using method such as the bisection method.

Theorem 3 (Projecting onto the Probability Simplex)**.** The projection onto the simplex Δ_n can be computed via

$$\Pi_{\Delta_n}(x) = \Pi_{\mathbb{R}_+}(x - \mu a)$$
$$\text{where } \mu \in \mathbb{R} \text{ solves: } \langle \mathbf{1}, \Pi_{\mathbb{R}_+}(x - \mu \mathbf{1}) \rangle = 1.$$

Proof. Use the previous theorem 2, observe that u can be a vector of infinity and hence setting $\mathbb{R}_+$ to be the box, and the probability simplex consrants $\sum_{i=1}^{n} x_i = 1$ is the hyperplane $H_{a,b}^{=}$ with $a = \mathbf{1}, b = 1$. □

Theorem 4 (Projection onto Affine Spaces)**.** Define the set $C = \{x \in \mathbb{R}^n : Ax = b\}$, assuming that $A \in \mathbb{R}^{m \times n}$ is full rank, then the projection onto the set is compted via

$$\Pi_C(x) = x - (A^T A)^{-1} A(Ax - b).$$

Remark 3.1.2. In the case when A is not full rank, do a reduce QR decomposition of $A = QR$ where R is $k \times n$ with $k < m$. In the case of our application, the matrix A is a full rank matrix. The matrix $(A^T A)^{-1}$ in our case can be very slow to invert since it's $\mathbf{C}_2^{n^2} \times \mathbf{C}_2^{n^2}$. One way to make it easier is to use sparse matrix invert method. In our case the matrix we are inverting is sparse and symmetric, it can be solved efficiently say for example,

4 The Software APIs for the Programmers

Option SLSQP in "scipy.optimize.minimize" performs sequential quadratic programming. I am not sure whether they would support efficient sparse matrix inverse. Interior points method (the engine behind many constraint optimization solver) is also a good way of solving the problem, it's also more specialized than the SLSQP solver. However their availability in python seems to be limited. See here for tutorials about how to use "scipy.optimize" module. Visits pyprox for a package that implements various type of proximal operators in python.

4.1 Scipy.optimize.SLSQP

scipy optimization documentations with examples. The "scipy.SLSQP" programming API accepts programming problem of the following form

$$\min_{l \le x \le u} f(x) \text{ s.t:}$$
$$\begin{cases} c_j(x) = 0, & j \in \Upsilon \\ g_j(x) \ge 0, & j \in \mathcal{I} \end{cases}$$

For each element $i = n, \cdots, N$, we have the constraints in the form of $l_i \in [-\infty, \infty)$ and $u_i \in (-\infty, \infty]$. The programmer is required to provide

1. $\nabla f(x)$, the gradient of the objective and the objective function in the form of a function handle in python.

2.

Representing the sequence of constraints $c_j(x)$ for $J \in \Upsilon$ as a vector function $C(x)$. Similarly, we represent inequality constraints as the vector funtion $G(x)$. Recall formulation from section 2.2. The inequality constraints are $-u \leq Cp \leq u$, $p \in \Delta_{n\times n}$. The box constraint is $u \in \mathbb{R}_+^{\mathbf{C}_2^{n^2}}$. The equality and inequality constriants can be modeled by the linear system:

$$\underbrace{\begin{bmatrix} -C & I \\ C & I \end{bmatrix}}_{=:\mathcal{C}} \begin{bmatrix} p \\ u \end{bmatrix} \geq \mathbf{0}$$

$$\underbrace{\begin{bmatrix} \mathbf{1}_n^T & & & \\ & \mathbf{1}_n^T & & \\ & & \ddots & \\ & & & \mathbf{1}_n^T \end{bmatrix}}_{\mathcal{G}} p - \mathbf{1} = \mathbf{0}.$$

Here $\mathbf{1}_n \in \mathbb{R}^n$ is a vector of 1. Observe that, $\mathcal{C}$ is a $2\mathbf{C}_2^{n^2}$ by $2n + \mathbf{C}_2^{n^2}$ matrix. $\mathcal{G}$ is a $n \times n^2$ matrix. The matrix $\mathcal{C}, \mathcal{G}$ denoted above are the Jacobi of the inequality and equality constraints respectively. Finally, the gradient of the objective function would be a vector with dimension $n^2 + \mathbf{C}_2^{n^2}$. Denote $\div$, to indicate that this is an operator that does element-wise division. For example, in below, we have $-\eta \div p$, which means dividing each element of the vector μ by element of the vector p, resulting in an element of the same dimension.

$$\nabla \left[\sum_{i=1}^{n^2} -\eta_i \log(p_i) + \sum_{i=1}^{\mathbf{C}_2^{n^2}} u_k \right] = \begin{bmatrix} -\eta \div p \\ \mathbf{1}_{\mathbf{C}_2^{n^2}} \end{bmatrix}. \tag{4.1.1}$$

At this point, we have everything ready for coding it up in python.

References

[1] E. K. Ryu and W. Yin, *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge: Cambridge University Press, 2022. [Online]. Available: https://large-scale-book.mathopt.com/

[2] A. Beck, *First-Order Methods in Optimization | SIAM Publications Library*, ser. MOS-SIAM Series in Optimization. SIAM. [Online]. Available: https://epubs.siam.org/doi/book/10.1137/1.9781611974997