Dauphine | PSL
UNIVERSITÉ PARIS

Lecture notes on regularized, large-scale and distributed optimization

Clément W. Royer

M2 IASD & M2 MASH - 2023/2024

- The last version of these notes can be found at:
  https://www.lamsade.dauphine.fr/∼croyer/ensdocs/RLD/LectureNotesOML-RLD.pdf.
- Comments, typos, etc, can be sent to `clement.royer@lamsade.dauphine.fr`.
- **Major updates of the document**
  - 2023.12.12: Full version.
  - 2023.11.08: First version of the notes.

Foreword

The purpose of these lecture notes is to discuss several characteristics of optimization problems arising in machine learning, that preclude from applying (stochastic) gradient methods as seen in the previous parts of the course. We are particularly interested in the use of regularization to bring structure into an optimization problem, and in techniques that allow to deploy algorithms at scale. More precisely, the learning goals are the following:

- Understand the key principles of proximal algorithms, and why those are useful for regularized problems.
- Understand the purpose of regularization terms in data science problems, with a focus on sparsity-inducing regularizers.
- Understand the main principle behind coordinate descent methods, and their interest in a distributed environment.
- Apply duality to design algorithms for distributed and decentralized optimization.

Notations

- Scalars (i.e. reals) are denoted by lowercase letters: $a, b, c, \alpha, \beta, \gamma$.
- Vectors are denoted by **bold** lowercase letters: $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$.
- Matrices are denoted by **bold** uppercase letters: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$.
- Sets are denoted by **bold** uppercase cursive letters : $\mathcal{A}, \mathcal{B}, \mathcal{C}$.
- The set of natural numbers (nonnegative integers) is denoted by $\mathbb{N}$; the set of integers is denoted by $\mathbb{Z}$.
- The set of real numbers is denoted by $\mathbb{R}$. Our notations for the subset of nonnegative real numbers and the set of positive real numbers are $\mathbb{R}_+$ and $\mathbb{R}_{++}$, respectively.
- The notation $\mathbb{R}^d$ is used for the set of vectors with $d \in \mathbb{N}$ real components; although we may not explicitly indicate it in the rest of these notes, we always assume that $d \geq 1$.
- A vector $\boldsymbol{x} \in \mathbb{R}^d$ is thought as a column vector, with $x_i \in \mathbb{R}$ denoting its i-th coordinate in the canonical basis of $\mathbb{R}^d$. We thus write $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$, or, in a compact form, $\boldsymbol{x} = [x_i]_{1 \leq 1 \leq d}$.
- Given a column vector $\boldsymbol{x} \in \mathbb{R}^d$, the corresponding row vector is denoted by $\boldsymbol{x}^{\mathrm{T}}$, so that $\boldsymbol{x}^{\mathrm{T}} = [x_1 \ \cdots \ x_d]$ and $[\boldsymbol{x}^{\mathrm{T}}]^{\mathrm{T}} = \boldsymbol{x}$. The scalar product between two vectors in $\mathbb{R}^d$ is defined as $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^{d} x_i y_i$.
- The Euclidean norm of a vector $\boldsymbol{x} \in \mathbb{R}^d$ is defined by $\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}$.
- We use $\mathbb{R}^{n \times d}$ to denote the set of real rectangular matrices with n rows and d columns, where n and d will always be assumed to be at least 1. If $n = d$, $\mathbb{R}^{d \times d}$ refers to the set of square matrices of size d.
- We identify a matrix in $\mathbb{R}^{d \times 1}$ with its corresponding column vector in $\mathbb{R}^d$.
- Given a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$, $\boldsymbol{A}_{ij}$ refers to the coefficient from the i-th row and the j-th column of $\boldsymbol{A}$: the diagonal of $\boldsymbol{A}$ is given by the coefficients $\boldsymbol{A}_{ii}$. Provided this notation is not ambiguous, we use the notations $\boldsymbol{A}$, $[\boldsymbol{A}_{ij}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq d}}$ and $[\boldsymbol{A}_{ij}]$ interchangeably.
- For every $d \geq 1$, $\mathbf{I}_d$ refers to the identity matrix in $\mathbb{R}^{d \times d}$ (with 1s on the diagonal and 0s elsewhere).

Contents

Chapter 1

Proximal methods and regularization

1.1 Regularized problems

A common practice in machine learning problems consists in enforcing a specific structure of the machine learning model through the objective function rather than by using constraints on the model parameters. Such regularized problems have the following form :

$$\underset{\boldsymbol{x}\in R^d}{\text{minimize}} \quad \underbrace{f(\boldsymbol{x})}_{data-fitting\ term} + \underbrace{\lambda\Omega(\boldsymbol{x})}_{regularization\ term} . \tag{1.1.1}$$

where $\lambda > 0$ is called a regularization parameter. This parameter controls the weight of the regularization term compared to the data-fitting term. It is common for one of the terms in (1.1.1) (and sometimes both) to be nonsmooth, in the sense that the associated functions may not possess a gradient at every point. Such problems are called composite optimization problems in general, and regularized problems in specific settings such as data science and inverse problems.

When $\lambda \to 0$, the problem becomes essentially equivalent to minimizing the data-fitting term $f(\boldsymbol{x})$ only, and regularization no longer matters (numerically, this occurs for small values of λ). When $\lambda \to \infty$, the problem amounts to minimizing the regularization term, without accounting for the data-dependent term. A key issue in regularization consists in finding the right balance between data fitting and regularization.

Example 1.1.1 *The following functions are common choices for regularization terms:*

- *ℓ_2/ridge regularization:* $\Omega(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$;
- *ℓ_1/LASSO regularization:* $\Omega(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$;
- *Elastic net:* $\Omega(\boldsymbol{x}) = \frac{\gamma}{2}\|\boldsymbol{x}\|_2^2 + \|\boldsymbol{x}\|_1$ *for* $\gamma > 0$;
- *Group LASSO: Given a partition $\mathcal{G}$ of $\{1, \ldots, d\}$, set* $\Omega(\boldsymbol{x}) = \sum_{g\in\mathcal{G}} \|[\boldsymbol{x}]_g\|_2$, *where* $[\boldsymbol{x}]_g \in \mathbb{R}^{|g|}$ *is the vector formed by the subset of coordinates of $\boldsymbol{x}$ corresponding to g.*

At a broader level, regularization terms can be used to promote generalization capabilities of the solution, by reducing the sensitivity of the solution with respect to the problem data. They are also used to enforce sparsity in the problem solution. Finally, they can be used to improve the optimization landscape of a problem, for instance by guaranteeing that the regularized problem has a unique solution.

Constrained optimization and regularization The regularization approach bears a connection with the introduction of constraints on the problem variables. In fact, any set of constraints $\boldsymbol{x} \in \mathcal{X}$ can be encoded as a regularization function using the so-called indicator function

$$\Omega(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{X} \\ +\infty & \text{otherwise.} \end{cases}$$

Conversely, it is possible to describe the solution of a regularized problem as that of a constrained optimization problem. However, the philosophy behind regularization is to penalize points that do not satisfy a certain property instead of excluding those points like in a constrained formulation. The interest of regularization lies in the possibility of weighing the regularization term relatively to the data-fitting term.

In this chapter, we review the main algorithmic approaches that can be used to tackle problems involving regularized optimization problems.

1.2 Proximal methods

Proximal methods are not restricted to nonsmooth optimization problems, but have proven particularly useful in the presence of nonsmoothness. Those techniques rely on the proximal operator, a mathematical object that is both defined as the solution to an optimization problem and used as a step in other optimization algorithms.

1.2.1 Proximal operator

The proximal operator is a fundamental tool used to both analyze problems with regularization and build algorithms for these problems.

Definition 1.2.1 *Let $\varphi : \mathbb{R}^d \to \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^d$. The* **proximal operator of φ at $\boldsymbol{x}$**, *denoted by $\mathrm{prox}\,[\varphi]\,[\boldsymbol{x}]$, is given by*

$$\mathrm{prox}_{\varphi}\,[\boldsymbol{x}] := \underset{\boldsymbol{u}\in\mathbb{R}^d}{\mathrm{argmin}} \left\{ \varphi(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{x}\|^2 \right\}. \tag{1.2.1}$$

As stated, the proximal operator is a set-valued mapping, i.e. its value is a set of vectors (possibly empty or infinite!). However, for certain classes of functions, the value of the proximal operator is a singleton. It is then convenient to identify it with the corresponding vector. We make this identification below in the case of convex functions, which will be our focus throughout these notes.

Definition 1.2.2 *Let $\varphi : \mathbb{R}^d \to \mathbb{R}$ be a convex function. Then the proximal operator of φ is defined as the function $\mathrm{prox}_{\varphi}\,[\cdot] : \mathbb{R}^d \to \mathbb{R}^d$ such that*

$$\forall \boldsymbol{x} \in \mathbb{R}^d, \mathrm{prox}_{\varphi}\,[\boldsymbol{x}] := \boldsymbol{u}^*, \{\boldsymbol{u}^*\} = \underset{\boldsymbol{u}\in\mathbb{R}^d}{\mathrm{argmin}} \left\{ \varphi(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{x}\|^2 \right\}. \tag{1.2.2}$$

Example 1.2.1 *For any $\boldsymbol{x} \in \mathbb{R}^d$ and any $\lambda > 0$, the following properties hold:*

- $\mathrm{prox}_{\mathbf{0}}\,[\boldsymbol{x}] = \boldsymbol{x}$.

- $\mathit{prox}_{\frac{\lambda}{2}\|\cdot\|^2}[\boldsymbol{x}] = \frac{\boldsymbol{x}}{1+\lambda}$.
- $\mathit{prox}_{\lambda\|\cdot\|_1}[\boldsymbol{x}]$ *is defined componentwise by*

$$\forall j = 1, \dots, d, \quad \left[\mathit{prox}_{\lambda\|\cdot\|_1}[\boldsymbol{x}]\right]_j = \begin{cases} x_j - \lambda & \text{if } x_j > \lambda \\ 0 & \text{if } x_j \in [-\lambda, \lambda] \\ x_j + \lambda & \text{if } x_j < -\lambda \end{cases}$$

- *If h is the indicator function of a convex set $\mathcal{X}$ (i.e. $h(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{X}$ and $h(\boldsymbol{x}) = \infty$ otherwise), then $\mathit{prox}_h[\boldsymbol{x}]$ corresponds to the projection on $\boldsymbol{x}$ onto $\mathcal{X}$.*

In general, proximal operators are useful when they are uniquely defined (which is always the case when the original function is convex) and when they are easy to compute numerically. The next two sections describe two algorithms built on proximal operator calculations.

1.2.2 Proximal point method

In this section, we consider

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, h(\boldsymbol{x}) \tag{1.2.3}$$

where $h : \mathbb{R}^d \to \mathbb{R}$ is a convex function. Without assumptions on the differentiability of h, we cannot rely on a gradient-type method for optimizing this problem. However, a clever algorithm based on proximal operators was proposed in the 1970s, that builds a sequence of approximate solutions of problem (1.2.3).

Algorithm 1: Proximal point method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \dots$ **do**

1. Compute a steplength $\alpha_k > 0$.
2. Compute $\boldsymbol{x}_{k+1}$ such that

$$\boldsymbol{x}_{k+1} \in \underset{\boldsymbol{z}\in\mathbb{R}^d}{\text{argmin}} \left\{ h(\boldsymbol{z}) + \tfrac{1}{2\alpha_k}\|\boldsymbol{z} - \boldsymbol{x}_k\|_2^2 \right\}. \tag{1.2.4}$$

end

The process is given in Algorithm 1. At every iteration, the next iterate $\boldsymbol{x}_{k+1}$ is computed by solving

$$\underset{\boldsymbol{z}\in\mathbb{R}^d}{\text{minimize}} \left\{ h(\boldsymbol{z}) + \tfrac{1}{2\alpha_k}\|\boldsymbol{z} - \boldsymbol{x}_k\|_2^2 \right\}. \tag{1.2.5}$$

At first glance, this problem is not easier than the original one, because it involves the original objective h. However, the presence of the **proximal term** $\frac{1}{2\alpha_k}\|\boldsymbol{z} - \boldsymbol{x}_k\|_2^2$ actually makes the problem easier to solve. In particular, the subproblem (1.2.5) is strongly convex (and thus the update (1.2.4) is uniquely defined), and solving this subproblem always improves the objective.

Lemma 1.2.1 *At the kth iteration of Algorithm 1, we have*

$$h(\boldsymbol{x}_{k+1}) \leq h(\boldsymbol{x}_k) - \frac{1}{2\alpha_k}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2^2. \tag{1.2.6}$$

Using the definition of the proximal operator, note that an iteration of Algorithm 1 can be rewritten as

$$\boldsymbol{x}_{k+1} = \text{prox}_{\alpha_k h}\left[\boldsymbol{x}_k\right].$$

1.2.3 Proximal gradient methods

In this section, we consider a specific class of regularized optimization problems, often termed as **composite optimization** problems.

Definition 1.2.3 (Composite optimization) *A composite optimization problem is of the form:*

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}) + \lambda\Omega(\boldsymbol{x}), \tag{1.2.7}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a smooth, $\mathcal{C}^1$ function, $\lambda > 0$ and $\Omega : \mathbb{R}^d \to \mathbb{R}$ is a convex, nonsmooth regularizer.

Because of the nonsmoothness of Ω, the overall objective of problem (1.2.7) is nonsmooth, and one may then consider applying a subgradient method to tackle this problem. However, knowledge about the particular problem structure, and the fact that f is a smooth function, allows for deriving better methods for this problem, that do not require direct calculations of subgradients.

Proximal gradient techniques, that are described in Algorithm 2, exploit the smoothness of f to iteratively solve a sequence of subproblems approximating the original one.

Algorithm 2: Proximal gradient method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**

1. Compute the gradient of the smooth part $\nabla f(\boldsymbol{x}_k)$.
2. Compute a steplength $\alpha_k > 0$.
3. Compute $\boldsymbol{x}_{k+1}$ such that

$$\boldsymbol{x}_{k+1} \in \underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{argmin}} \left\{ f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_k) + \tfrac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 + \lambda\Omega(\boldsymbol{x}) \right\}. \tag{1.2.8}$$

end

The cost of an iteration of Algorithm 2 includes a gradient calculation as well as solving an auxiliary optimization problem (1.2.8), called the proximal subproblem. Such a method is only practical if the subproblems are easier to solve than the original problem.

Connection with the proximal operator If $\Omega \equiv 0$ (i. e. Ω is the zero function and the problem is un-regularized), one can show that the solution of (1.2.8) is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k).$$

We thus recognize the gradient descent iteration. For arbitrary Ω, the following formula applies:

$$\boldsymbol{x}_{k+1} = \mathrm{prox}_{\alpha_k \lambda \Omega} \left[\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \right].$$

As such, proximal gradient bears a close connection with the proximal operator.

However, note that applying the proximal point to problem (1.2.7) yields a different iteration, even in the case $\Omega \equiv 0$! Indeed, in that case, and assuming convexity of f, the proximal point iteration can be written as

$$\boldsymbol{x}_{k+1} = \mathrm{prox}_{\alpha_k f} \left[\boldsymbol{x}_k \right] = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_{k+1}).$$

The proximal point method is thus an implicit method, in the sense that its iteration is defined implicitly by the above equation to be solved for $\boldsymbol{x}_{k+1}$.

Remark 1.2.1 *Proximal gradient methods can be designed using most of the tools that can be applied to gradient descent : this includes stepsize choices, acceleration as well as stochastic aspects. Moreover, theoretical guarantees have been established for both nonconvex and convex f, though the latter has attracted more attention in the optimization literature. Note that the analysis of such methods is performed by looking at the vector*

$$\frac{1}{\alpha_k} \left(\boldsymbol{x}_k - \mathit{prox}_{\alpha_k \lambda \Omega} \left[\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \right] \right),$$

that plays a similar role than the gradient in analyzing gradient descent (and reduces to the gradient when $\Omega \equiv 0$).

1.3 The case of ℓ_2 regularization

A **regularized problem with ℓ_2 regularization**, also known as **ridge regularized problem**, has the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}} \, f(\boldsymbol{x}) + \frac{\lambda}{2} \|\boldsymbol{x}\|^2. \tag{1.3.1}$$

The ridge regularizer $\boldsymbol{x} \mapsto \frac{1}{2}\|\boldsymbol{x}\|^2$ has several interpretations. It effectively penalizes $\boldsymbol{x}$s with large components, and can be shown to be equivalent to a constraint on the squared norm $\|\boldsymbol{w}\|^2$. In addition, a ridge regularizer has the effect to reduce the variance of the problem solution with respect to the data. Finally, when the regularizer $\lambda > 0$ is big enough, this often turns the objective function into a strongly convex one, with the positive implications in terms of convergence speed and uniqueness of the (global) minimum.

1.3.1 General principles

Suppose that the function f in (1.3.1) is $\mathcal{C}^1$. Then, the kth iteration of gradient descent applied to (1.3.1) is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \left(\nabla f(\boldsymbol{x}_k) + \lambda \boldsymbol{x}_k\right) = (1 - \lambda\alpha_k)\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \tag{1.3.2}$$

where $\alpha_k > 0$ is a positive stepsize. When $\lambda = 0$, we recover the classical gradient descent iteration. When $\lambda > 0$, however, the algorithm will modify the coefficients of the vector $\boldsymbol{x}_k$. In this case, one typically chooses α_k so that $1 - \lambda\alpha_k \in (0, 1)$, and thus the coefficients of $\boldsymbol{x}_k$ are reduced in magnitude at every iteration in the update formula. This process corresponds to **weight decay** in deep learning.

Suppose now that we apply proximal gradient to problem (1.3.1). The corresponding iteration is

$$\boldsymbol{x}_{k+1} = \mathrm{prox}_{\alpha_k \lambda \Omega}\left[\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)\right] = \frac{1}{1 + \lambda\alpha_k}\left[\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)\right] = \frac{1}{1 + \lambda\alpha_k}\boldsymbol{x}_k - \frac{\alpha_k}{1 + \lambda\alpha_k}\nabla f(\boldsymbol{x}_k). \tag{1.3.3}$$

As λ increases, this iteration decays both the coordinates of the iterate $\boldsymbol{x}_k$ and that of the gradient step. In fact, it can be shown that the coordinates of the iterates decrease uniformly in a smooth fashion as $\lambda \to 0$.

1.3.2 Linear least squares

To illustrate further the use of proximal methods, we consider a simple linear least-squares problem. Given data under the form of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and a vector $\boldsymbol{y} \in \mathbb{R}^n$, we seek to solve

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{2n}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\lambda}{2}\|\boldsymbol{x}\|^2 \tag{1.3.4}$$

for $\lambda \geq 0$.

Gradient descent VS proximal point The iteration of gradient descent applied to this problem gives

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \left[\frac{1}{n}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}) + \lambda \boldsymbol{x}_k\right], \tag{1.3.5}$$

where $\alpha_k > 0$ and $\frac{1}{n}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y})$ is the gradient of the data fitting term at $\boldsymbol{x}_k$. On the other hand, the iteration of the proximal point method gives

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \left[\frac{1}{n}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}) + \lambda \boldsymbol{x}_{k+1}\right],$$

which can be rewritten as

$$\boldsymbol{x}_{k+1} = \left[(1 + \lambda\alpha_k)\boldsymbol{I} + \frac{1}{n}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right]^{-1}\left(\boldsymbol{x}_k + \alpha_k \frac{1}{n}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}\right). \tag{1.3.6}$$

One then sees that each iteration of the proximal point method requires to solve a linear system, which is more expensive than the cost of one gradient descent iteration. However, it can be shown that (1.3.6) has good stability properties. Note also that the linear system solve becomes easier as λ increases.

1.4 Conclusion

Proximal methods are a class of algorithms that rely on proximal operators to perform optimization steps on possibly complex functions. They are particularly useful in the context of regularized optimization, in which

The most classical regularization techniques in optimization and beyond are ℓ_2 and ℓ_1 regularization. In a data-related setting, the former is typically used to reduce the dependency of the solution with respect to the data, while the latter promotes solutions that have zero coordinates.

In both cases, the proximal gradient method can then be written in closed form, leading to algorithms that are implementable quite efficiently. In particular, the case of regularized linear least squares has been widely studied, and the resulting problems are now part of numerous statistics and signal processing toolboxes.

Chapter 2

Sparse optimization

In this chapter, we investigate regularized problems in which the goal is to produce a solution vector with a significant number of zero coefficients that still provide a good model with respect to the data-fitting term in the regularized problem. Such vectors, called **sparse**, are often thought as a way to simplify the original model. For instance, in the case of linear models, zero coefficients means that some features are ignored by the model, and thus learning can be performed using less features. This feature selection process can be done *a posteriori* (i.e. once the learning problem has been solved), but it can also be encoded into the optimization problem through the use of regularization terms.

2.1 Sparse regularization terms

2.1.1 Motivation : Sparse models and the ℓ_0 norm

While computing a model to explain some data, we might want to compute a model that explains the data using as few features as possible[1]. Mathematically speaking, if our model is parameterized by a vector $\boldsymbol{x} \in \mathbb{R}^d$, our goal is to compute a vector that explains the data with as few nonzero coordinates as possible.

There exists a regularizer that penalized vectors with nonzero components (not just large as opposed to the ridge regularizer), called the ℓ_0 norm [2]. An ℓ_0-regularized problem has the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}) + \lambda\|\boldsymbol{x}\|_0, \quad \|\boldsymbol{v}\|_0 = |\{i|[\boldsymbol{v}]_i \neq 0\}|.$$

However, this function is nonsmooth and discontinuous; its combinatorial nature also introduces more complexity to the original problem.

2.1.2 Convex sparse regularizers

Researchers have investigated numerous alternatives to the ℓ_0. The main surrogate for the ℓ_0 norm is the ℓ_1 norm, defined by

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{d} |[\boldsymbol{x}]_i|. \tag{2.1.1}$$

[1] The goal of this process is *feature selection*.

[2] Though technically this function defines a semi-norm.

It can be shown that this function is the closest convex upper bound to the ℓ_0 norm. This function is continuous and convex; moreover, it is actually a norm function, which endows it with many desirable properties. As a result, the class of problems of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}) + \lambda\|\boldsymbol{x}\|_1, \tag{2.1.2}$$

called ℓ_1 **regularized problems** or **lasso regularized problems**, has emerged as a tractable alternative to the ℓ_0 regularized formulation.

Variations on ℓ_1 A number of regularizing terms have been proposed based on the ℓ_1 norm, as well as the ℓ_2 norm

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{d} [\boldsymbol{x}]_i^2}.$$

The later is also a convex approximation to the ℓ_0 norm, but it is not as tight as the ℓ_1 norm. Nevertheless, one can combine the ℓ_1 and ℓ_2 norms to build special regularizers, such as the so-called group LASSO regularizer

$$\Omega(\boldsymbol{x}) = \sum_{g\in\mathcal{G}} \|\boldsymbol{x}_g\|_2,$$

where $\mathcal{G}$ is a partition of $\{1,\dots,d\}$ and $\boldsymbol{x}_g$ denotes the vector in $\mathbb{R}^{|g|}$ formed by the coordinates of $\boldsymbol{x}$ corresponding to g. Other variations on this concept are presented in Figure 2.1 via their level sets.

2.2 The case of ℓ_1 regularization

2.2.1 Proximal gradient and ℓ_1 regularization

A natural way to tackle problems of the form 2.1.2 is through the proximal gradient framework of Algorithm 2. Indeed, unlike for general regularizers, one can obtain a closed-form solution of the subproblem (1.2.8). Indeed, the proximal subproblem, given by

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}} \left\{ f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}_k) + \tfrac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{x}_k\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \right\},$$

has a unique solution. To obtain it, one computes the usual gradient step $\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)$, then one applies the **soft-thresholding function** $s_{\alpha_k\lambda}(\bullet)$ to each component, where this function is given by

$$\forall \mu > 0, \forall t \in \mathbb{R}, \qquad s_\mu(t) = \begin{cases} t+\mu & \text{if } t < -\mu \\ t-\mu & \text{if } t > \mu \\ 0 & \text{otherwise.} \end{cases}$$

(Note that this function encodes the proximal operator for the ℓ_1 norm.) As a result, the solution of the proximal subproblem is defined component-wise according to the components of the gradient step. The resulting update is at the heart of the corresponding proximal algorithm, popularized in signal and image processing under the name ISTA (Iterative Soft-Thresholding Algorithm). A description of ISTA is given in Algorithm 3.

It can be shown that the use of the soft-thresholding function does promote zero components in the new iterates, which results in sparser solutions at the end of the algorithmic run.

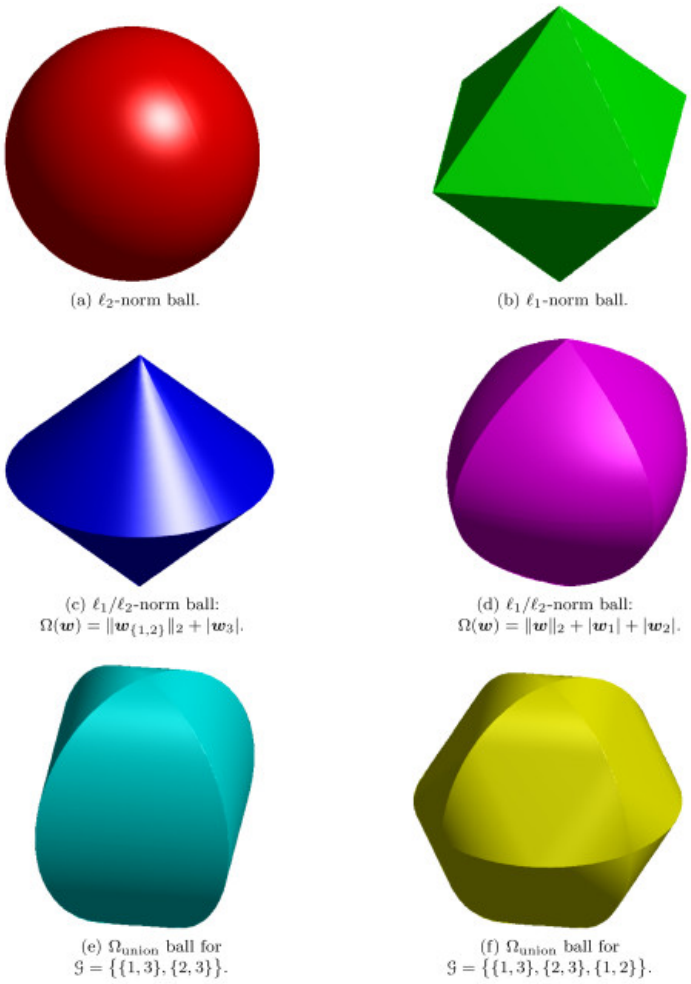

Figure 2.1: Level sets (balls of fixed radius) for sparse regularizers. *Source:[1]*.

Remark 2.2.1 *A notable improvement on ISTA was the inclusion of momentum, which resulted in a new algorithm called FISTA (Fast ISTA): this method is now the most widely used instance of ISTA.*

2.2.2 Subgradient approach

In smooth optimization, the gradient is a vector that quantifies the rate of local change in the function. Generalized notions of gradient have been defined, especially in the case of convex, nonsmooth functions. We provide below the most common definition of a subgradient.

Definition 2.2.1 (Subgradient and subdifferential) *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function. A vector $\boldsymbol{g} \in \mathbb{R}^d$ is called a subgradient of f at $\boldsymbol{x} \in \mathbb{R}^d$ if*

$$\forall \boldsymbol{z} \in \mathbb{R}^n, \qquad f(\boldsymbol{z}) \;\geq\; f(\boldsymbol{x}) + \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}).$$

The set of all subgradients of f at $\boldsymbol{x}$ is called the subdifferential of f at $\boldsymbol{x}$, and denoted by $\partial f(\boldsymbol{x})$.

Algorithm 3: ISTA: Iterative Soft-Thresholding Algorithm.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**

1. Compute the gradient of the smooth par $\nabla f(\boldsymbol{x}_k)$.
2. Compute a steplength $\alpha_k > 0$.
3. Compute $\boldsymbol{x}_{k+1}$ component-wise through the following rule

$$[\boldsymbol{x}_{k+1}]_i = \begin{cases} [\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)]_i + \alpha_k \lambda & \text{if } [\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)]_i < -\alpha_k \lambda \\ [\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)]_i - \alpha_k \lambda & \text{if } [\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)]_i > \alpha_k \lambda \\ 0 & \text{if } [\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)]_i \in [-\alpha_k \lambda, \alpha_k \lambda]. \end{cases} \tag{2.2.1}$$

end

Note that when the function f is differentiable at $\boldsymbol{x}$, we have $\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$, thus the notion of subdifferential matches that of the gradient for differentiable functions. In addition, the subdifferential provides a characterization of the global minima of f, as shown by the following result.

Theorem 2.2.1 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function, and $\bar{\boldsymbol{x}} \in \mathbb{R}^d$.*

$$\mathbf{0} \in \partial f(\bar{\boldsymbol{x}}) \quad \Leftrightarrow \quad \bar{\boldsymbol{x}} \in \operatorname*{argmin}_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}).$$

Again, the result of Theorem 2.2.1 generalizes that of the smooth setting, since in that case the subdifferential consists in a single element.

Example 2.2.1 *Let $f : \mathbb{R} \to \mathbb{R}$, $f(x) = |x|$.*

$$\partial f(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ [-1, 1] & \text{if } x = 0. \end{cases}$$

The set $[-1, 1]$ contains 0, which confirms that $x^* = 0$ is the minimum of f.

Note that the subdifferential of the ℓ_1 norm is defined componentwise according to the example above.

By analogy with gradient descent, we can design a subgradient method, as shown by Algorithm 4.

Such a method offers a flexibility in choosing the subgradient, which can be an issue. Moreover, choosing the stepsize is more difficult than for gradient descent, due to the nonsmooth nature of the problem. In fact, a subgradient can lead to increase in the function value for any stepsize, hence the choice of subgradient is critical to the success of this method. A convenient choice consists in setting $\boldsymbol{g}_k$ such that

$$\boldsymbol{g}_k \in \operatorname*{argmin}_{\boldsymbol{g} \in \mathbb{R}^d} \{ \, \|\boldsymbol{g}\| \mid \boldsymbol{g} \in \partial f(\boldsymbol{x}_k) \, \}. \tag{2.2.2}$$

Algorithm 4: Generic subgradient method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**

1. Compute a subgradient $\boldsymbol{g}_k \in \partial f(\boldsymbol{x}_k)$.
2. Compute a steplength $\alpha_k > 0$.
3. Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k$.

end

With this choice, Algorithm 4 will not change the iterate once it reaches a point with $\mathbf{0}_{\mathbb{R}^d}$ in its subdifferential (i.e. a minimum for a convex function). Computing $\boldsymbol{g}_k$ according to (2.2.2) may however be an expensive procedure as it could require to compute the entire subdifferential.

In the context of ℓ_1 norm regularization, it is possible to define the subdifferential of $f + \lambda\|\cdot\|_1$ explicitly when f is $\mathcal{C}^1$. Otherwise, calculus rules about subdifferentials come into play.

2.2.3 The LASSO estimator

As for ℓ_2 regularization, using ℓ_1 regularization together with a linear least-squares loss is an important class of problems. Such problems have the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{x}\|_1, \tag{2.2.3}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{y} \in \mathbb{R}^n$ form the problem data, and $\lambda > 0$. This problem is often referred to as the LASSO (Least Absolute Shrinkage and Selection Operator) problem, though other terminologies such as basis pursuit can be found in the literature.

The properties of the solution of problem (2.2.3) can be described according to the columns of $\boldsymbol{A}$. Every column contains all the observed values for a specific feature captured by the linear model.

Theorem 2.2.2 *Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ denote the columns of $\boldsymbol{A}$, and let $\boldsymbol{x}^*$ be a solution of problem (2.2.3). Then, for any $j = 1, \ldots, d$, we have*

$$\begin{cases} |\boldsymbol{b}_j^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{y})| & \leq & \lambda n & \text{if } [\boldsymbol{x}^*]_j = 0 \\ \boldsymbol{b}_j^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{y}) & = & -\lambda n \mathit{sgn}([\boldsymbol{x}^*]_j) & \text{otherwise,} \end{cases} \tag{2.2.4}$$

where $\mathit{sgn}(t) = 1$ if $t > 0$ and $\mathit{sgn}(t) = -1$ if $t < 0$.

The result of Theorem 2.2.2 implies that the larger λ is, the more components of $\boldsymbol{x}^*$ will be zero. In that sense, the use of ℓ_1 regularization leads to sparser solutions than in absence of regularization.

Chapter 3

Coordinate descent methods

In this chapter, we address the treatment of large-scale optimization problems, where the number of parameters to be optimized over is extremely large. As we witness a growth in both the model complexity (i.e. the number of parameters) and the amount of data available (i.e. the size of the dataset), standard optimization techniques may suffer from the curse of dimensionality and their performance may deteriorate as dimensions grow. As a result, the practical difficulty of the problem increases with the dimension, simply because there are more variables to consider. However, on structured problems such as those arising in data science, there often exists a low-dimensional or separable structure that allows for optimization steps to be taken over a subset of variables. This is the underlying idea of **coordinate descent methods**, that have regained interest in the early 2000s due to their applicability in certain data science settings.

3.1 Basics of coordinate descent

3.1.1 Algorithm

Consider the unconstrained optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}), \tag{3.1.1}$$

where $f \in \mathcal{C}^1(\mathbb{R}^d)$. The idea of coordinate descent methods consist in taking a gradient step with respect to a single decision variable at every iteration. To this end, we observe that for every $\boldsymbol{x} \in \mathbb{R}^d$, the gradient of f at $\boldsymbol{x}$ can be decomposed as

$$\nabla f(\boldsymbol{x}) = \sum_{j=1}^{d} \nabla_j f(\boldsymbol{x}) \boldsymbol{e}_j,$$

where ∇_j denotes the partial derivative with respect to the j-th variable of the function f (that is, the jth coordinate of f) and $\boldsymbol{e}_j \in \mathbb{R}^d$ is the jth coordinate vector of the canonical basis in $\mathbb{R}^d$. The coordinate descent approach replaces the full gradient by a step along a coordinate gradient, as formalized in Algorithm 5.

The variants of coordinate descent are mainly identified by the way they select the coordinate sequence $\{j_k\}$. There exist numerous rules for choosing the coordinate index, among which:

Algorithm 5: Coordinate descent method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**

1. Select a coordinate index $j_k \in \{1, \ldots, d\}$.
2. Compute a steplength $\alpha_k > 0$.
3. Set

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla_{j_k} f(\boldsymbol{x}_k) \boldsymbol{e}_{j_k}. \tag{3.1.2}$$

end

- Cyclic: Select the indices by cycling over $\{1, \ldots, d\}$ in that order. After d iterations, all indices have been selected.
- Randomized cyclic: Cycle through a random ordering of $\{1, \ldots, d\}$, that changes every d steps.
- Randomized: Draw j_k at random in $\{1, \ldots, d\}$ at every iteration.

The last two strategies are those for which the strongest results can be obtained.

Block coordinate descent Rather than using a single index, it is possible to select a subset of the coordinates (called "block" in the literature). The kth iteration of such a *block coordinate descent* algorithm thus is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \sum_{j \in \mathcal{B}_k} \nabla_j f(\boldsymbol{x}_k) \boldsymbol{e}_j, \tag{3.1.3}$$

where $\mathcal{B}_k \subset \{1, \ldots, d\}$.

3.1.2 Coordinate descent and stochastic gradient

Our description of coordinate descent, and particularly the randomized variant, is reminiscent of the stochastic gradient algorithm. In fact, randomized coordinate descent can be viewed as a special case of stochastic gradient, in which the formula

$$\nabla f(\boldsymbol{x}) = \frac{1}{d} \sum_{j=1}^{d} d \nabla_j f(\boldsymbol{x}) \boldsymbol{e}_j$$

is used to define a finite sum with d gradients that can be sampled using any distribution that one would use in a stochastic gradient setting. Note that, unlike in a stochastic gradient framework, any coordinate descent step (even randomized ones) uses a descent direction.

Consider a finite-sum problem of the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{x}), \quad f_i(\boldsymbol{x}) := \ell_i(\boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x}), \tag{3.1.4}$$

where $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x}$ is a linear model of the data vector $\boldsymbol{a}_i$, and $\ell_i : \mathbb{R} \to \mathbb{R}$ is a convex loss function specific to the ith data point (such as $\ell_i(h) = \frac{1}{2}(h - y_i)^2$ for linear least squares). If the number of data points is large, it is natural to think of applying stochastic gradient to this problem. Another approach consists in considering an equivalent formulation of (3.1.4) through (Fenchel) duality, given by

$$\underset{\boldsymbol{v}\in\mathbb{R}^n}{\text{maximize}}\, g(\boldsymbol{v}) := -\frac{1}{n}\sum_{i=1}^{n} f_i^*(v_i) \tag{3.1.5}$$

where for any convex function $\phi : \mathbb{R}^m \to \mathbb{R}$, the convex conjugate function ϕ^* is defined by

$$\phi^*(\boldsymbol{a}) = \sup_{\boldsymbol{b}\in\mathbb{R}^m} \left\{\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b} - \phi(\boldsymbol{b})\right\}.$$

The so-called dual problem (3.1.5) has a finite-sum, separable form. It can thus be tackled using (dual) *coordinate ascent*, the counterpart of coordinate descent for minimization: the iteration of this method is given by

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha_k \nabla_i g(\boldsymbol{v}_k), \tag{3.1.6}$$

leading to updating the iterate one coordinate at a time. Under appropriate assumptions on the problem, the iteration (3.1.6) is equivalent to the original stochastic gradient iteration, with $\boldsymbol{x}_k = \frac{1}{\lambda n}\sum_{i=1}^{n}[\boldsymbol{v}_k]_i \boldsymbol{a}_i$. For this reason, stochastic gradient is sometimes viewed as applying coordinate ascent to the dual problem.

3.2 Theoretical guarantees of coordinate descent methods

A famous 3-dimensional example designed by M. J. D. Powell in 1973 shows that coordinate descent methods do not necessarily converge.

Nevertheless, it is possibly to provide guarantees on coordinate descent methods under appropriate assumptions. In particular, a linear rate of convergence can be obtained for coordinate descent methods on strongly convex problems: we provide below the necessary assumptions to arrive at such a result.

Assumption 3.2.1 *The objective function f in (3.1.1) is $\mathcal{C}^1$ and μ-strongly convex, with $f^* = \min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$. Moreover, for every $j = 1, \ldots, d$, the partial derivative $\nabla_i f$ is L_i-Lipschitz continuous, i.e.*

$$\forall \boldsymbol{x} \in \mathbb{R}^d,\ \forall h \in \mathbb{R}, \quad |\nabla_j f(\boldsymbol{x} + h\boldsymbol{e}_j) - \nabla_j f(\boldsymbol{x})| \leq L_j |h|. \tag{3.2.1}$$

We let $L_{\max} = \max_{1\leq j\leq d} L_j$.

Theorem 3.2.1 *Suppose that Assumption 3.2.1 holds, and that Algorithm 5 is applied to problem (3.1.1) with $\alpha_k = \frac{1}{L_{\max}}$ for all k and j_k being drawn uniformly at random in $\{1, \ldots, d\}$. Then, for any $K \in \mathbb{N}$, we have*

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \left(1 - \frac{\mu}{dL_{\max}}\right)^K \left(f(\boldsymbol{x}_0) - f^*\right). \tag{3.2.2}$$

Other results have been established in the convex and nonconvex settings, under additional assumptions. In all cases, properties on the partial derivatives are required.

Acceleration In a convex optimization setting, it is possible to combine randomized coordinate descent with the accelerated gradient paradigm [1]. Starting from $\boldsymbol{x}_0 \in \mathbb{R}^d$ et $\boldsymbol{v}_0 = \boldsymbol{x}_0$, every iteration k draws i_k uniformly at random between 1 and d, then performs the following calculations:

$$\left\{ \begin{array}{lcl} \boldsymbol{u}_k & := & \lambda_k \boldsymbol{v}_k + (1-\lambda_k)\boldsymbol{x}_k \\ \boldsymbol{x}_{k+1} & := & \boldsymbol{u}_k - \frac{1}{L_{j_k}} \nabla_{j_k} f(\boldsymbol{u}_k) \boldsymbol{e}_{j_k} \\ \boldsymbol{v}_{k+1} & := & \mu_k \boldsymbol{v}_k + (1-\mu_k)\boldsymbol{u}_k - \frac{\gamma_k}{L_{j_k}} \nabla_{j_k} f(\boldsymbol{u}) \boldsymbol{e}_{j_k}, \end{array} \right.$$

where $\{\lambda_k, \mu_k, \gamma_k\}$ are sequences that depend on the dimension d, and possibly on a strong convexity constant if f happens to be strongly convex. Although this method possesses better complexity guarantees than randomized coordinate descent, it requires to maintain additional vector sequences. In terms of accesses to the coefficients, the cost of this accelerated method is thus higher than that of a basic coordinate descent iteration.

3.3 Applications of coordinate descent methods

Coordinate descent techniques are particularly useful for large-scale sparse optimization. Consider a regularized problem of the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \tilde{f}_i(\boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x}) + \sum_{j=1}^{d} \Omega(w_j), \tag{3.3.1}$$

where $\tilde{f}_i : \mathbb{R} \to \mathbb{R}$ is (possibly) data-dependent, $\boldsymbol{x}_i \in \mathbb{R}^d$ is a sparse data vector, and $\Omega : \mathbb{R} \to \mathbb{R}$ is a regularization function applied componentwise to the vector $\boldsymbol{x}$.

Example 3.3.1 (Regularized least squares with sparse data) *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ with sparse rows and $\boldsymbol{y} \in \mathbb{R}^n$, consider the problem*

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\textit{minimize}}\, f(\boldsymbol{x}) := \frac{1}{2n} \sum_{i=1}^{n} \|\boldsymbol{X}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda \sum_{j=1}^{d} w_i^2.$$

Apply Algorithm 5 to this problem. For any iteration k, if we move along the j_kth coordinate, the partial derivative under consideration is

$$\nabla_{j_k} f(\boldsymbol{x}_k) = \boldsymbol{x}_{j_k}^{\mathrm{T}} (\boldsymbol{X}\boldsymbol{x}_k - \boldsymbol{y}) + 2\lambda [\boldsymbol{x}_k]_{j_k}.$$

By storing the vector $\{\boldsymbol{X}\boldsymbol{x}_k\}$ across all iterations, the calculation of $\nabla_{j_k} f(\boldsymbol{x}_k)$ can be greatly reduced when $\boldsymbol{x}_{j_k}$ is sparse, to the point that the cost of a coordinate descent iteration will be of the order of the number of nonzero elements in $\boldsymbol{x}_{j_k}$.

Coordinate descent techniques are quite prominent in parallel optimization algorithms. In this setting, several cores are cooperating to solve problem (3.1.1): each core can then run *its own coordinate descent method* and all cores update the same shared iterate vector. The most efficient parallel coordinate descent techniques perform these iterations in an asynchronous fashion, which does not prevent from guaranteeing convergence of this framework!

[1] See Irène Waldspurger's lecture on this topic.

Link with stochastic gradient

3.4 Conclusion

Large-scale problems have always pushed optimization algorithms to their limits, and have lead to reconsidering certain algorithms in light of their applicability to large-scale settings. Coordinate descent methods are the perfect example of classical techniques that regained popularity because of their efficiency in data science settings. On some instances, randomized coordinate descent techniques bear a close connection with stochastic gradient methods. More globally, coordinate descent methods are quite efficient on large-dimensional problems that have a separable structure. Finally, the use of coordinate descent methods in parallel environments has also contributed to their revival in optimization.

Chapter 4

Distributed and constrained optimization

In this chapter, we describe the theoretical insights behind distributed optimization formulations, in which several agents collaborate to solve an optimization problem. This paradigm can be modeled using a linearly constrained optimization formulation, and handling such formulations requires dedicated algorithms. We first set the mathematical foundations of these methods via a brief introduction to duality, then present our algorithms of interest.

4.1 Linear constraints and dual problem

Consider the following optimization problem with linear equality constraints:

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{4.1.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$. For simplicity, we will assume that the feasible set $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ is not empty.

Duality theory consists in handling constraints formulations by reformulating the problem into an unconstrained optimization problem. We present the theoretical arguments for the special case of problem (4.1.1), which yields a much simpler analysis.

Definition 4.1.1 *The* Lagrangian function *of problem* (4.1.1) *is given by*

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) := f(\boldsymbol{x}) + \boldsymbol{y}^{\mathrm{T}} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) . \tag{4.1.2}$$

The Lagrangian function combines the objective function and the constraints, and allows to restate the original problem as an unconstrained one, called the primal problem:

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}} \max_{\boldsymbol{y}\in\mathbb{R}^m} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}). \tag{4.1.3}$$

The solutions of the primal problem are identical to that of problem (4.1.3) in our case. The difficulty of solving problem (4.1.3) lies in the definition of its objective function as the optimal value of a maximization problem.

Definition 4.1.2 *The* **dual problem** *of* (4.1.1) *is the maximization problem*

$$\underset{\boldsymbol{y}\in\mathbb{R}^m}{\text{maximize}} \min_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}), \tag{4.1.4}$$

where the function $\boldsymbol{y} \mapsto \min_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y})$ *is called the dual function of the problem.*

Unlike the primal problem, the dual problem is always concave (i.e. the opposite of the dual function is convex), which facilitates its resolution by standard optimization techniques. The goal is then to solve the dual problem in order to get the solution of the primal problem, thanks to properties such as the one below.

Assumption 4.1.1 *We suppose that* **strong duality** *holds between problem* (4.1.1) *and its dual, that is,*

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} \max_{\boldsymbol{y}\in\mathbb{R}^m} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}) = \max_{\boldsymbol{y}\in\mathbb{R}^m} \min_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}).$$

Assumption 4.1.1 is typically satisfied when f is convex, but strong duality may hold even on non-convex problems.

4.2 Dual algorithms

We are now concerned with solving the dual problem (4.1.4), and we will present three methods for this purpose.

4.2.1 Dual ascent

The **dual ascent** method is implicitly a subgradient method applied to the dual problem (which we recall is a maximimization problem). At every iteration, it starts from a primal-dual pair $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ and performs the following iteration:

$$\begin{cases} \boldsymbol{x}_{k+1} & \in & \operatorname{argmin}_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}_k) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \alpha_k(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}), \end{cases} \tag{4.2.1}$$

where $\alpha_k > 0$ is a stepsize for the dual ascent step, and $\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$ is a subgradient for the dual function $\boldsymbol{y} \mapsto \min_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y})$ at $\boldsymbol{y}_k$.

4.2.2 Augmented Lagrangian

The dual ascent method generally has weak convergence guarantees. For this reason, the optimization literature has introduced other frameworks based on a regularized version of the Lagrangian function.

Definition 4.2.1 *The* **augmented Lagrangian** *of problem* (4.1.1) *is the function on* $\mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}_{++}$ *by*

$$\mathcal{L}^a(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\lambda}) := f(\boldsymbol{x}) + \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \frac{\lambda}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2. \tag{4.2.2}$$

Augmented Lagrangians thus are a family of functions parameterized by $\lambda > 0$, that put more emphasis on the constraint violation as λ grows.

The augmented Lagrangian algorithm, also called method of multipliers, performs the following iteration:

$$\left\{ \begin{array}{lcl} \boldsymbol{x}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}^a(\boldsymbol{x}, \boldsymbol{y}_k; \lambda) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \lambda(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}). \end{array} \right. \tag{4.2.3}$$

In this algorithm, λ is constant and used as a constant stepsize: many more sophisticated choices of both the augmented Lagrangian function and the stepsizes have been proposed. In general, the advantages of augmented Lagrangian techniques are that the subproblems defining $\boldsymbol{x}_{k+1}$ become easier to solve (thanks to regularization) and that the overall guarantees on the primal-dual pair are stronger.

4.3 Dual methods and decomposition

A key idea in modern optimization, that has resulted in numerous theoretical and numerical improvements, consists in exploiting the structure of a given problem as much as possible. The underlying idea is that a decomposition of a large, complex problem can lead to many smaller and simpler (sub)problems that will be easier and cheaper to solve than the original one. We describe below how this idea can be carried out in the context of dual algorithms.

4.3.1 Dual decomposition

Suppose that we consider a linearly constrained problem with a separable form:

$$\left\{ \begin{array}{ll} \mathrm{minimize}_{\boldsymbol{u}\in\mathbb{R}^{d_1}, \boldsymbol{v}\in\mathbb{R}^{d_2}} & f(\boldsymbol{u}) + g(\boldsymbol{v}) \\ \mathrm{subject\ to} & \boldsymbol{A}\boldsymbol{u} + \boldsymbol{B}\boldsymbol{v} = \boldsymbol{c}, \end{array} \right. \tag{4.3.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{d_1 \times m}$, $\boldsymbol{B} \in \mathbb{R}^{d_2 \times m}$ and $\boldsymbol{c} \in \mathbb{R}^m$. Given the particular structure (sometimes called splitting) between the variables $\boldsymbol{u}$ and $\boldsymbol{v}$, one may want to update those variables separately rather than gathering them into a single vector $\boldsymbol{x}$ and performing a single update.

This idea is precisely that of **dual decomposition**. At iteration k, the dual decomposition method applied to problem (4.3.1) computes

$$\left\{ \begin{array}{lcl} \boldsymbol{u}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{u}\in\mathbb{R}^{d_1}} \mathcal{L}(\boldsymbol{u}, \boldsymbol{v}_k, \boldsymbol{y}_k) \\ \boldsymbol{v}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{v}\in\mathbb{R}^{d_2}} \mathcal{L}(\boldsymbol{u}_k, \boldsymbol{v}, \boldsymbol{y}_k) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \alpha_k(\boldsymbol{A}\boldsymbol{u}_{k+1} + \boldsymbol{B}\boldsymbol{v}_{k+1} - \boldsymbol{c}), \end{array} \right. \tag{4.3.2}$$

where $\alpha_k > 0$. Interestingly, the calculations for $\boldsymbol{u}_{k+1}$ and $\boldsymbol{v}_{k+1}$ are completely independent, and can be carried out in parallel. This observation and its practical realization have lead to successful applications of dual decomposition in several fields.

4.3.2 ADMM

The **Alternated Direction Method of Multipliers**, or **ADMM**, is an increasingly popular variation on the augmented Lagrangian paradigm that bears some connection with coordinate descent approaches, in that it splits the problem in two sets of variables.

Recall problem (4.3.1) above. For any $\lambda > 0$, the augmented Lagrangian of problem (4.3.1) has the form

$$\mathcal{L}^a(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{y}; \lambda) \quad = \quad f(\boldsymbol{u}) + g(\boldsymbol{v}) + \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{u} + \boldsymbol{B}\boldsymbol{v} - \boldsymbol{c}) + \frac{\lambda}{2} \|\boldsymbol{A}\boldsymbol{u} + \boldsymbol{B}\boldsymbol{v} - \boldsymbol{c}\|^2 .$$

The ADMM iteration exploits the separable nature of the problem by computing the values $\boldsymbol{u}$ and $\boldsymbol{v}$ independently. Starting from $(\boldsymbol{u}_k, \boldsymbol{v}_k, \boldsymbol{y}_k)$, the ADMM counterpart to iteration (4.2.3) is

$$\left\{ \begin{array}{lcl} \boldsymbol{u}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{u} \in \mathbb{R}^{d_1}} \mathcal{L}^a(\boldsymbol{u}, \boldsymbol{v}_k, \boldsymbol{y}_k; \lambda) \\ \boldsymbol{v}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{v} \in \mathbb{R}^{d_2}} \mathcal{L}^a(\boldsymbol{u}_{k+1}, \boldsymbol{v}, \boldsymbol{y}_k; \lambda) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \lambda(\boldsymbol{A}\boldsymbol{u}_{k+1} + \boldsymbol{B}\boldsymbol{v}_{k+1} - \boldsymbol{c}). \end{array} \right. \tag{4.3.3}$$

The first two updates of the iteration (4.3.3) cannot be run in parallel, but the philosophy is slightly different from that of the dual decomposition method. Indeed, in ADMM, it is common that solving for a subset of the variables will be much easier than solving for all variables at once (see our example in the next section). The ADMM framework gives the possibility to exploit this property within an augmented Lagrangian method.

Remark 4.3.1 *The idea of splitting the objective and the constraints across two groups of variables can be declined into as many groups of variables as possible, depending on the structure of the problem.*

To end this section, we briefly mention that there exist convergence results for ADMM-type frameworks, typically under convexity assumptions on the problem [3]. A typical result consist in showing that

$$\left\{ \begin{array}{lcl} \|\boldsymbol{A}\boldsymbol{u}_k + \boldsymbol{B}\boldsymbol{v}_k - \boldsymbol{c}\| & \to & 0 \\ f(\boldsymbol{u}_k) + g(\boldsymbol{v}_k) & \to & \min_{\boldsymbol{u},\boldsymbol{v}} f(\boldsymbol{u}) + g(\boldsymbol{v}) \\ \boldsymbol{y}_k & \to & \boldsymbol{y}^*, \end{array} \right.$$

where $\boldsymbol{y}^*$ is a solution of the dual problem.

4.4 Decentralized optimization

We end this chapter by describing an increasingly common setup in optimization over large datasets, often termed consensus optimization or decentralized optimization. In this setup, we consider a dataset that is split across m entities called *agents*. Every agent uses its own data to train a certain learning model parameterized by a vector in $\mathbb{R}^d$. To this end, each agent not only has its own function $f^{(i)}$, but also its own copy of the model parameters $\boldsymbol{x}^{(i)}$. The optimization problem at hand considers a master iterate $\boldsymbol{x}$, and attempts to reach consensus between all the agents. This leads to the following formulation:

$$\begin{array}{ll} \mathrm{minimize}_{\boldsymbol{x}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)} \in \mathbb{R}^d} & \sum_{i=1}^m f^{(i)}(\boldsymbol{x}^{(i)}) \\ \mathrm{subject\ to} & \boldsymbol{x} = \boldsymbol{x}^{(i)} \quad \forall i = 1, \ldots, m. \end{array} \tag{4.4.1}$$

This problem is a proxy for $\mathrm{minimize}_{\boldsymbol{x} \in \mathbb{R}^d} \sum_{i=1}^m f^{(i)}(\boldsymbol{x})$, but the latter problem cannot be solved by a single agent since every agent has exclusive access to its data by design. The formulation (4.4.1)

models the fact that all agents are involved in computing $\boldsymbol{x}$ by acting on $\boldsymbol{x}_i$. It is possible to apply ADMM to problem (4.4.1) by setting

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \vdots \\ \boldsymbol{x}^{(m)} \end{bmatrix} \in \mathbb{R}^{md}, \quad \boldsymbol{v} = \boldsymbol{x} \in \mathbb{R}^d.$$

Generalization The idea behind the formulation (4.4.1) can be extended to the case of data spread over a network, represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: every vertex $s \in \mathcal{V}$ of the graph represents an agent, while every edge $(s, s') \in \mathcal{E}$ represents a channel of communication between two agents in the graph. Letting $\boldsymbol{x}^{(s)} \in \mathbb{R}^d$ and $f^{(s)} : \mathbb{R}^d \to \mathbb{R}$ represent the parameter copy and objective function for agent $s \in \mathcal{V}$, respectively, the consensus optimization problem can be written as:

$$\begin{array}{ll} \text{minimize}_{\{\boldsymbol{x}^{(s)}\}_{s\in\mathcal{V}} \in (\mathbb{R}^d)^{|\mathcal{V}|}} & \sum_{s\in\mathcal{V}} f^{(s)}(\boldsymbol{x}^{(s)}) \\ \text{subject to} & \boldsymbol{x}^{(s)} = \boldsymbol{x}^{(s')} \quad \forall (s, s') \in \mathcal{E}. \end{array} \tag{4.4.2}$$

When the graph is fully connected, i.e. all agents communicate, this problem reduces to an unconstrained problem. However, in general, the solutions of this problem are much difficult to identify, and one must work through minimizing the objective and satisfying the so-called consensus constraints.

Decentralized gradient methods To wrap up this chapter, we describe an increasingly popular class of algorithms that extends gradient descent to the decentralized setting. The decentralized gradient framework is designed for problems of the form

$$\underset{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(m)}\in\mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{m} f_i(\boldsymbol{x}^{(i)}),$$

without consensus constraints but with an implicit graph structure $(\mathcal{V}, \mathcal{E})$ connecting the agents. Given a matrix $\boldsymbol{W} \in \mathbb{R}^{m\times m}$ that is doubly stochastic (i.e. with nonnegative coefficients such that the sum of all rows and all columns are 1) and satisfies $[\boldsymbol{W}]_{ij} \neq 0$ if and only if $i = j$ or $(i, j) \in \mathcal{E}$, the kth iteration of the decentralized gradient method at agent i reads

$$\boldsymbol{x}_{k+1}^{(i)} = \sum_{j=1}^{m} [\boldsymbol{W}]_{ij} \boldsymbol{x}_k^{(j)} - \alpha_k \nabla f_i(\boldsymbol{x}_k^{(i)}). \tag{4.4.3}$$

The iteration (4.4.3) thus combines a gradient step for agent i together with a so-called **mixing step** (or consensus step) in which the current value of the iterate for agent i is combined with that of its neighbors. This framework is steadily gaining popularity in the machine learning community.

4.5 Conclusion

In modern data science tasks, the amount of data available requires distributed storage, and possibly agents cooperating in order to solve the optimization problem at hand. Linearly constrained formulations can capture this behavior, and such constraints can be handled in a efficient manner using dual variables. Augmented Lagrangian techniques are among the most popular methods in this category, and these methods can be further specialized to account for structure in the problem. The ADMM framework has emerged as one of the most interesting formulations used to split the calculation into (presumably) cheaper subproblems. It is also very well suited for operating in a distributed or decentralized environment.

Chapter 5

Exercises

Exercise 1: Proximal gradient and regularization

In this exercise, we revisit the proximal operator and the proximal gradient method on a specific problem. Given a point $w \in \mathbb{R}^d$, we consider

$$\text{minimize}_{x \in \mathbb{R}^d} \ \|x\|_1 + \frac{1}{2\alpha}\|x - w\|_2^2, \tag{5.0.1}$$

where $\|x\|_1 = \sum_{i=1}^d |[x]_i|$, $\|x\|_2^2 = \sum_{i=1}^d [x]_i^2$ and $\alpha > 0$.

a) Using the properties of $x \mapsto \|x\|_1$, explain why the objective function of (5.0.1) cannot be optimized by gradient-type techniques.

b) Justify that problem (5.0.1) and

$$\text{minimize}_{x \in \mathbb{R}^d} \ \alpha\|x\|_1 + \frac{1}{2}\|x - w\|_2^2 \tag{5.0.2}$$

have the same solution set (argmin). Since both functions are strongly convex, what can be said about this solution set?

c) Write down an optimality condition for problem (5.0.1).

d) In this question, we view problem (5.0.1) as computing a proximal operator.

   i) Using the definition of the proximal operator, write down the solution of problem (5.0.1) as the value of a proximal operator of a certain function.

   ii) By repeatedly solving instances of problem (5.0.1) using the last solution find as w, what algorithm do we obtain?

e) In this question, we view problem (5.0.2) in a composite form, where $\frac{1}{2}\|x - w\|_2^2$ is a data-fitting term and $\alpha\|x\|_1$ is a regularization term.

   i) What is the purpose of such a regularization term? Why is it computationally worth using?

   ii) Write down an iteration of proximal gradient applied to this problem with $x_k = w$ and stepsize α. What do you observe then? Is it to be expected?

Exercise 2: Second-difference regularization

In this exercise, we consider a regularized optimization problem of the form

$$\text{minimize}_{x\in\mathbb{R}^d}\ f(x) + \frac{\lambda}{2}\|Lx\|_2^2, \tag{5.0.3}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, $\|v\|_2^2 = \sum_{j=1}^d [v]_j^2$ for any $v \in \mathbb{R}^d$ and $L \in \mathbb{R}^{d\times d}$ is the second-difference matrix defined by

$$L_{ij} = \begin{cases} 1 & \text{if } j = i+1 \text{ or } j = i-1 \\ -2 & \text{if } j = i \\ 0 & \text{otherwise.} \end{cases}$$

Such problems arise when the vector x represents a discretization of a real-valued function. Using the proposed regularization promotes solutions whose components vary continuously.

a) In this question, we assume that $d \gg 1$ and that the function f is separable, in that it can be written as

$$f(x) = \sum_{j=1}^d f^j([x]_j),$$

where every $f^j : \mathbb{R} \to \mathbb{R}$ only depends on the jth coordinate of the output vector x.

i) Justify that the second term in the objective is partially separable (which implies that its gradient also is).

ii) Recall that the gradient of the second term is given by λLx for any $x \in \mathbb{R}^d$. Suppose that we apply a basic coordinate descent method to problem (5.0.3). Justify how an iteration of such a method can be considered cheaper than an iteration of gradient descent.

iii) Suggest a block variant of that method that makes use of the structure of the second term in the objective.

b) In this question, we suppose that f is strongly convex. We modify problem (5.0.3) by introducing an auxiliary variable $z \in \mathbb{R}^d$, leading to

$$\begin{array}{ll} \text{minimize}_{\substack{x\in\mathbb{R}^d \\ z\in\mathbb{R}^d}} & f(x) + \frac{\lambda}{2}\|z\|_2^2 \\ \text{s.t.} & Lx - z = 0. \end{array} \tag{5.0.4}$$

i) Write down the Lagrangian for problem (5.0.4). Using this function, how can we rewrite problem (5.0.4)?

ii) What is the difference between a Lagrangian and an augmented Lagrangian?

iii) How does the introduction of the variable z allows for applying ADMM to this problem? What is the advantage of such an approach here?

c) Finally, we suppose that the objective function f can be expressed as a finite sum

$$f(x) = \sum_{i=1}^n f_i(x),$$

where every f_i is strongly convex and continuously differentiable. We consider that all f_i are spread across different agents, but that all agents know the regularization term.

i) Consider first the formulation (5.0.3). Rewrite this problem under the assumption that every agent has its own copy of the problem variable, and is using its own function f_i instead of f.

ii) Using the same idea as for obtaining (5.0.4), explain how the problem from the previous question can be reformulated as a linearly-constrained problem in order to apply ADMM.

Bibliography

[1] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4:1–106, 2012.

[2] A. Beck. *First-Order Methods in Optimization*. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, 2017.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, 2010.

[4] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1:123–231, 2013.

[5] S. J. Wright. Coordinate descent algorithms. *Math. Program.*, 151:3–34, 2015.

[6] S. J. Wright and B. Recht. *Optimization for Data Analysis*. Cambridge University Press, 2022.