Dauphine | PSL
UNIVERSITÉ PARIS

Lecture notes on large-scale and distributed optimization

Clément W. Royer

M2 IASD & M2 MASH - 2022/2023

- The last version of these notes can be found at:
  https://www.lamsade.dauphine.fr/∼croyer/ensdocs/LSD/LectureNotesOML-LSD.pdf.
- Comments, typos, etc, can be sent to `clement.royer@lamsade.dauphine.fr`.

- **Major updates of the document**
  - 2022.12.11: Full version of the notes released.
  - 2022.11.27: First version of the notes.

Foreword

The purpose of these lectures is to review optimization frameworks that allow to deploy the methods seen in other parts of the course at scale. The learning goals are as follows:

- Understand the main principle behind coordinate descent methods, and their interest in a distributed environment.
- Apply duality to design algorithms for constrained and distributed optimization.
- Propose algorithms tailored to a decentralized optimization setting.

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 $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}$.
- 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

Introduction

In these lectures, we dive into a increasingly important area of focus in optimization methods for data science. 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. The goal of these notes is to present some algorithmic ideas that can reduce the impact of large dimensions, either in terms of parameters or data points.

Chapter 2 focuses on a setting in which the number of parameters to optimize over is extremely large, and describes how coordinate descent approaches can be used in this setting. Chapter 3 considers the case in which the data itself is not available in a centralized fashion. Using tools from duality theory, efficient algorithms that can exploit this particular problem structure can be applied in that setting.

Chapter 2

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. In general, due to the curse of dimensionality, the 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.

2.1 Basics of coordinate descent

2.1.1 Algorithm

Consider the unconstrained optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}), \tag{2.1.1}$$

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 1.

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:

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

Algorithm 1: 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
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla_{j_k} f(\boldsymbol{x}_k) \boldsymbol{e}_{j_k}. \tag{2.1.2}$$

end

- 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
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \sum_{j \in \mathcal{B}_k} \nabla_j f(\boldsymbol{x}_k) \boldsymbol{e}_j, \tag{2.1.3}$$
where $\mathcal{B}_k \subset \{1, \ldots, d\}$. A block typically consists in coordinates drawn without replacement.

Remark 2.1.1 *As described in the lab session, it is possible to combine randomized coordinate descent with Nesterov's acceleration technique to yield improve theoretical guarantees. However, this raises implementation issues that may alleviate the practical interest of coordinate descent approaches.*

2.1.2 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
$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{x}), \quad f_i(\boldsymbol{x}) := \ell_i(\boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x}), \tag{2.1.4}$$

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 (2.1.4) through (Fenchel) duality, given by

$$\underset{\boldsymbol{v}\in\mathbb{R}^n}{\text{maximize}}\, g(\boldsymbol{v}) := -\frac{1}{n}\sum_{i=1}^{n} f_i^*(v_i) \tag{2.1.5}$$

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 (2.1.5) 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

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha_k \nabla_i g(\boldsymbol{v}_k), \tag{2.1.6}$$

leading to updating the iterate one coordinate at a time. Under appropriate assumptions on the problem, the iteration (2.1.6) 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.

2.1.3 Separability and proximal coordinate descent

Although coordinate descent methods seem cheaper to apply than gradient-type methods, their cost actually depends on that of computing partial derivatives. Indeed, if it turns out that all coordinates of the gradient depend on all coordinates of the input vector, then there will essentially be no value in using coordinate updates.

Fortunately, it is quite common for optimization problems to exhibit a structure amenable to coordinate descent. This structure is rigorously defined below.

Definition 2.1.1 *A function $f : \mathbb{R}^d \to \mathbb{R}$ is called* **separable** *if it exists d functions $f_1, \ldots, f_d : \mathbb{R} \to \mathbb{R}$ such that*

$$\forall \boldsymbol{x} \in \mathbb{R}^d, f(\boldsymbol{x}) = \sum_{j=1}^{d} f_j([\boldsymbol{x}]_j).$$

Note that if f is separable and all f_js are $\mathcal{C}^1$, then f is also $\mathcal{C}^1$ and we have

$$\forall \boldsymbol{x} \in \mathbb{R}^d,\ \forall j = 1, \ldots, d, \nabla_j f(\boldsymbol{x}) = \nabla f_j([\boldsymbol{x}]_j),$$

which means that a coordinate descent update can be computed while only accessing one coordinate.

The separability property is a strong property, that essentially states that a function can be minimized independently with respect to each of its variables. It is actually more frequent for a function to be partially separable in the following sense.

Definition 2.1.2 *A function $f : \mathbb{R}^d \to \mathbb{R}$ is called* **partially separable** *if it exists J functions $\{f_j\}_{j=1}^{J}$ and J subsets of $\{1, \ldots, d\}$ denoted by $\mathcal{B}_1, \ldots, \mathcal{B}_J$ such that*

i) For any $\boldsymbol{x} \in \mathbb{R}^d$,

$$f(\boldsymbol{x}) = \sum_{j=1}^{J} f_j\left([\boldsymbol{x}]_{\mathcal{B}_j}\right),$$

where $[\boldsymbol{x}]_{\mathcal{B}_j} \in \mathbb{R}^{|\mathcal{B}_j|}$ *denotes the vector formed by the coordinates of* $\boldsymbol{x}$ *belonging to* $\mathcal{B}_j$*;*

ii) $\cup_{j=1}^{J} \mathcal{B}_j = \{1, \ldots, d\}$*;*

The notion of partial separability is only interesting whenever the different sets $\{\mathcal{B}_j\}_{j=1}^{J}$ do not intersect too much, the ideal case being when they form a partition of $\{1, \ldots, d\}$.

Example 2.1.1 (Regularized least squares with sparse data) *Given* $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ *with sparse rows and* $\boldsymbol{y} \in \mathbb{R}^n$*, consider the problem*

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

Apply Algorithm 1 to this problem. For any iteration k*, if we move along the* j_k*th coordinate, the partial derivative under consideration is*

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

By storing the vector $\{\boldsymbol{A}\boldsymbol{x}_k\}$ *across all iterations, the calculation of* $\nabla_{j_k} f(\boldsymbol{x}_k)$ *can be greatly reduced when* $\boldsymbol{a}_{j_k}$ *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* $\boldsymbol{a}_{j_k}$*.*

More broadly, any regularized problem of the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}}\, \frac{1}{n} \sum_{i=1}^{n} \tilde{f}_i(\boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x}) + \sum_{j=1}^{d} \Omega(w_j), \tag{2.1.7}$$

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

Proximal coordinate descent Consider a proximal gradient subproblem arising from minimizing $f(\boldsymbol{x}) + \lambda\Omega(\boldsymbol{x})$ where Ω is partially separable. The subproblem at iteration k will be of the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|^2 + \lambda\Omega(\boldsymbol{x}).$$

Both the linear term and the proximal term are separable functions, since they can be written as sums of one-dimensional functions. Since Ω is partially separable by assumption, this subproblem can be tackled by coordinate descent techniques, and this is one popular way of solving numerous proximal subproblems.

Another possibility consists in replacing the proximal gradient step by a proximal coordinate descent step. For simplicity, we describe the procedure in the context of a separable regularization

function $\Omega(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$. At iteration k, the next iterate is computed by first drawing a coordinate index j_k, then performing the update

$$z_k \in \operatorname*{argmin}_{z\in\mathbb{R}} \nabla_{j_k} f(\boldsymbol{x}_k)(z - [\boldsymbol{x}_k]_{j_k}) + \frac{1}{2\alpha_k}(z - [\boldsymbol{x}_k]_{j_k})^2 + \lambda|z|.$$

This problem has a closed-form solution, and thus the next iterate can by computed by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + (z - [\boldsymbol{x}_k]_{j_k})\boldsymbol{e}_{j_k}.$$

2.2 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. This example partly explains why coordinate descent methods fell out of favor in optimization for several decades.

Example 2.2.1 *Consider the function $f : \mathbb{R}^3 \times \mathbb{R}$ defined by*

$$f(x_1, x_2, x_3) = -(x_1x_2 + x_2x_3 + x_1x_3) + \sum_{j=1}^{3} \max\{|x_i| - 1, 0\}.$$

This function is nonconvex and $\mathcal{C}^1$. If we apply cyclic coordinate descent starting from $(1, -1, 1)$ and using the best stepsize possible (i.e. the one that minimizes the function in the chosen direction) at every iteration, the method will never converge to a minimum.

Despite this negative result, 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 2.2.1 *The objective function f in (2.1.1) is $\mathcal{C}^1$ and μ-strongly convex, with $f^* = \min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$. Moreover, for every $j = 1, \dots, d$, the partial derivative $\nabla_i f$ is L_i-Lipschitz continuous, i.e.*

$$\forall \boldsymbol{x} \in \mathbb{R}^d,\ \forall h \in \mathbb{R}, \quad |\nabla_j f(\boldsymbol{x} + h\boldsymbol{e}_j) - \nabla_j f(\boldsymbol{x})| \le L_j|h|. \tag{2.2.1}$$

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

Theorem 2.2.1 *Suppose that Assumption 2.2.1 holds, and that Algorithm 1 is applied to problem (2.1.1) with $\alpha_k = \frac{1}{L_{\max}}$ for all k and j_k being drawn uniformly at random in $\{1, \dots, d\}$. Then, for any $K \in \mathbb{N}$, we have*

$$\mathbb{E}\left[f(\boldsymbol{x}_K) - f^*\right] \le \left(1 - \frac{\mu}{dL_{\max}}\right)^K \left(f(\boldsymbol{x}_0) - f^*\right). \tag{2.2.2}$$

Suppose that $f \in \mathcal{C}^{1,1}_L$. Then $L \le dL_{\max}$, and a gradient descent iteration with stepsize $\frac{1}{L}$ satisfies

$$f(\boldsymbol{x}_k) - f^* \le \left(1 - \frac{\mu}{L}\right)^K \left(f(\boldsymbol{x}_0) - f^*\right)$$

under the same assumptions than that of Theorem 2.2.1. This result is better than (2.2.2) in terms of iterations, however the cost of an iteration of gradient descent can be d times higher than that of a coordinate descent iteration. Similarly to the reasoning we did for stochastic gradient methods, convergence rates must be thought in terms of relevant cost metrics.

Remark 2.2.1 *Other results have been established in the convex and nonconvex settings, under additional assumptions. In all cases, properties on the partial derivatives are required.*

We end this section with a result for cyclic coordinate descent, that illustrates that this method has worse guarantees than its randomized counterpart in general.

Theorem 2.2.2 *Suppose that Assumption 2.2.1 holds, that $f \in \mathcal{C}_L^{1,1}$ and that Algorithm 1 is applied to problem (2.1.1) with $\alpha_k = \frac{1}{L_{\max}}$ for all k and j_k being drawn in a cyclic fashion in $\{1, \dots, d\}$. Then, for any $K \in \mathbb{N}$, we have*

$$f(\boldsymbol{x}_K) - f^* \leq \left(1 - \frac{\mu}{2L_{\max}\left(1 + d\frac{L^2}{L_{\max}^2}\right)}\right)^{K/d} (f(\boldsymbol{x}_0) - f^*)\,. \tag{2.2.3}$$

2.3 Coordinate descent in parallel and distributed environments

Coordinate descent techniques are quite prominent in parallel optimization algorithms. In this setting, several cores are cooperating to solve problem (2.1.1): 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.

Synchronous coordinate descent A typical setup for coordinate descent is that of several processors sharing a memory. In very high dimensions, the iterate $\boldsymbol{x}_k$ will be stored on this memory, and processors can only access (blocks of) coordinates to perform updates. A synchronous coordinate descent approach then consists in assigning coordinates (or blocks thereof) to various processors, so that each of them performs an update of the form

$$[\boldsymbol{x}_k]_j \leftarrow [\boldsymbol{x}_k]_j - \alpha_k \nabla_j f(\boldsymbol{x}_k).$$

A **synchronization** step guarantees that all updates are accounted for prior to moving to iteration $k+1$. This step is essential if the function is not separable, but may result in significant idle time for processors as they wait for all updates to be completed.

Asynchronous coordinate descent In an asynchronous framework, every processor actually performs a run of coordinate descent, almost independently from the other processors. As a result, a processor may reason on an iterate that was read from memory **then updated by other processors**. The resulting method is described in Algorithm 2.

In general, one cannot obtain a convergence rate for this method, however asymptotic convergence to a solution can be established under certain conditions.

Theorem 2.3.1 *Suppose that f is convex and $\mathcal{C}^1$ with a unique minimizer $\boldsymbol{x}^*$. Suppose further that Algorithm 2 is run under the following assumptions:*

i) Every coordinate of $\boldsymbol{x}_k$ is updated infinitely often;

ii) For every processor, every coordinate of $\hat{\boldsymbol{x}}_k$ is updated infinitely often;

Algorithm 2: Asynchronous coordinate descent method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$, shared iteration counter $k = 0$.
for *all processors* **do**

1. Select a coordinate index $j_k \in \{1, \dots, d\}$.
2. Compute a steplength $\alpha_k > 0$.
3. Set
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla_{j_k} f(\hat{\boldsymbol{x}}_k) \boldsymbol{e}_{j_k}, \tag{2.3.1}$$
where $\hat{\boldsymbol{x}}_k$ conactenates the last values of the coordinates available to the processor.
4. Update $k = k + 1$.

end

iii) $\alpha_k = \alpha > 0$, *where* α *satisfies:*

$$\forall \boldsymbol{x} \in \mathbb{R}^d, \|\boldsymbol{x} - \alpha \nabla f(\boldsymbol{x}) - \boldsymbol{x}^*\|_\infty \leq \eta \|\boldsymbol{x} - \boldsymbol{x}^*\|_\infty$$

for some $\eta \in (0, 1)$.

Then, the sequence of iterates of Algorithm 2 $\{\boldsymbol{x}_k\}$ *converges to* $\boldsymbol{x}^*$ *as* $k \to \infty$.

Remark 2.3.1 *The asynchronous coordinate descent paradigm was used in conjunction with stochastic gradient in the* HOGWILD! *algorithm, that won the NeurIPS test of time award in 2020.*

2.4 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 3

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.

3.1 Linear constraints and dual problem

Consider the following optimization problem with linear equality constraints:

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.1.1}$$

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 (3.1.1), which yields a much simpler analysis.

Definition 3.1.1 *The* Lagrangian function *of problem* (3.1.1) *is given by*

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) := f(\boldsymbol{x}) + \boldsymbol{y}^{\mathrm{T}} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) . \tag{3.1.2}$$

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:

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}} \max_{\boldsymbol{y}\in\mathbb{R}^m} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}). \tag{3.1.3}$$

The solutions of the primal problem are identical to that of problem (3.1.3) in our case. The difficulty of solving problem (3.1.3) lies in the definition of its objective function as the optimal value of a maximization problem.

Definition 3.1.2 *The* **dual problem** *of* (3.1.1) *is the maximization problem*

$$\underset{\boldsymbol{y}\in\mathbb{R}^m}{\text{maximize}} \min_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}), \tag{3.1.4}$$

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 3.1.1 *We suppose that* **strong duality** *holds between problem* (3.1.1) *and its dual, that is,*

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

A sufficient condition for Assumption 3.1.1 is that f be convex, but this is not necessary.

3.2 Dual algorithms

We are now concerned with solving the dual problem (3.1.4), and we will present three methods for this purpose.

3.2.1 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:

$$\begin{cases} \boldsymbol{x}_{k+1} & \in & \operatorname{argmin}_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}_k) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \alpha_k(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}), \end{cases} \tag{3.2.1}$$

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 $\boldsymbol{y}_k$.

3.2.2 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 3.2.1 *The* **augmented Lagrangian** *of problem* (3.1.1) *is the function on* $\mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}_{++}$ *by*

$$\mathcal{L}^a(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\lambda}) := f(\boldsymbol{x}) + \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \frac{\lambda}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2. \tag{3.2.2}$$

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:

$$\left\{ \begin{array}{lcl} \boldsymbol{x}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}^a(\boldsymbol{x}, \boldsymbol{y}_k; \lambda) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \lambda(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}). \end{array} \right. \tag{3.2.3}$$

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.

3.3 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.

3.3.1 Dual decomposition

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

$$\left\{ \begin{array}{ll} \mathrm{minimize}_{\boldsymbol{u}\in\mathbb{R}^{d_1}, \boldsymbol{v}\in\mathbb{R}^{d_2}} & f(\boldsymbol{u}) + g(\boldsymbol{v}) \\ \mathrm{subject\ to} & \boldsymbol{A}\boldsymbol{u} + \boldsymbol{B}\boldsymbol{v} = \boldsymbol{c}, \end{array} \right. \tag{3.3.1}$$

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 (3.3.1) computes

$$\left\{ \begin{array}{lcl} \boldsymbol{u}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{u}\in\mathbb{R}^{d_1}} \mathcal{L}(\boldsymbol{u}, \boldsymbol{v}_k, \boldsymbol{y}_k) \\ \boldsymbol{v}_{k+1} & \in & \mathrm{argmin}_{\boldsymbol{v}\in\mathbb{R}^{d_2}} \mathcal{L}(\boldsymbol{u}_k, \boldsymbol{v}, \boldsymbol{y}_k) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \alpha_k(\boldsymbol{A}\boldsymbol{u}_{k+1} + \boldsymbol{B}\boldsymbol{v}_{k+1} - \boldsymbol{c}), \end{array} \right. \tag{3.3.2}$$

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.

3.3.2 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 (3.3.1) above. For any $\lambda > 0$, the augmented Lagrangian of problem (3.3.1) has the form

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

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 (3.2.3) is

$$\left\{\begin{array}{lcl} \boldsymbol{u}_{k+1} & \in & \operatorname{argmin}_{\boldsymbol{u}\in\mathbb{R}^{d_1}} \mathcal{L}^a(\boldsymbol{u}, \boldsymbol{v}_k, \boldsymbol{y}_k; \lambda) \\ \boldsymbol{v}_{k+1} & \in & \operatorname{argmin}_{\boldsymbol{v}\in\mathbb{R}^{d_2}} \mathcal{L}^a(\boldsymbol{u}_{k+1}, \boldsymbol{v}, \boldsymbol{y}_k; \lambda) \\ \boldsymbol{y}_{k+1} & = & \boldsymbol{y}_k + \lambda(\boldsymbol{A}\boldsymbol{u}_{k+1} + \boldsymbol{B}\boldsymbol{v}_{k+1} - \boldsymbol{c}). \end{array}\right. \tag{3.3.3}$$

The first two updates of the iteration (3.3.3) 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 3.3.1 *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 [1]. 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.

3.4 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:

$$\begin{array}{ll} \text{minimize}_{\boldsymbol{x},\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(m)}\in\mathbb{R}^d} & \sum_{i=1}^m f^{(i)}(\boldsymbol{x}^{(i)}) \\ \text{subject to} & \boldsymbol{x} = \boldsymbol{x}^{(i)} \quad \forall i = 1, \ldots, m. \end{array} \tag{3.4.1}$$

This problem is a proxy for $\text{minimize}_{\boldsymbol{x}\in\mathbb{R}^d} \sum_{i=1}^m f^{(i)}(\boldsymbol{x})$, but the latter problem cannot be solved by a single agent since every agent has exclusive access to its data by design. The formulation (3.4.1) 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 (3.4.1) 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 (3.4.1) 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:

$$\begin{array}{ll} \text{minimize}_{\{\boldsymbol{x}^{(s)}\}_{s\in\mathcal{V}}\in(\mathbb{R}^d)^{|\mathcal{V}|}} & \sum_{s\in\mathcal{V}} f^{(s)}(\boldsymbol{x}^{(s)}) \\ \text{subject to} & \boldsymbol{x}^{(s)} = \boldsymbol{x}^{(s')} \quad \forall (s, s') \in \mathcal{E}. \end{array} \tag{3.4.2}$$

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

$$\boldsymbol{x}_{k+1}^{(i)} = \sum_{j=1}^{m} [\boldsymbol{W}]_{ij} \boldsymbol{x}_k^{(j)} - \alpha_k \nabla f_i(\boldsymbol{x}_k^{(i)}). \tag{3.4.3}$$

The iteration (3.4.3) 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.

3.5 Conclusion of Chapter 3

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.

Bibliography

[1] 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.

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

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