Optimizing through change for cyber-physical and social systems

Andrea Simonetto

INSTITUT POLYTECHNIQUE DE PARIS

ENSTA

IP PARIS

Mémoire d'Habilitation à Diriger des Recherches

Optimizing through change for cyber-physical and social systems

Mémoire d'Habilitation à Diriger des Recherches
de l'Institut Polytechnique de Paris

préparée à l'UMA, ENSTA Paris, Institut Polytechnique de Paris

Mémoire présentée et soutenue à Palaiseau, le 09 septembre 2022, par

ANDREA SIMONETTO

Composition du Jury :

Prof. Mikael Johansson, KTH, Sweden	Rapporteur
Prof. Jérôme Malick, Université Grenoble Alpes, CNRS, France	Rapporteur
Prof. Vianney Perchet, ENSAE Paris, IPP, France	Rapporteur
Prof. Alexandre D'Aspremont, ENS, CNRS, France	Examinateur
Prof. Antonin Chambolle, Université Paris Dauphine - PSL, CNRS, France	Examinateur
Prof. Julien Hendrickx, Université catholique Louvain, Belgium	Examinateur
Prof. Colin Jones, EPFL, Switzerland	Examinateur
Prof. Luca Schenato, Università di Padova, Italy	Examinateur

Contents

Chapter 1

Introduction

Pe' fa' 'e cose bone ce vo' tiemp' [1]

Cosimo Rummo, pasta maker masterchef

1.1 Context and Aim

Optimization is prevalent across many engineering and science domains. Recently, some of these domains – and in particular infrastructures such as power, transportation and communication networks, as well as social, and personalized health platforms – are undergoing a fundamental transformation, driven by major technological advances across various sectors, the information explosion propelled by online social media, the pervasive sensing and computing capabilities, and the key presence of humans in the loop. Effectively, these infrastructures and platforms are transforming into complex systems operating in highly dynamic environments, with high volumes of heterogeneous information, while navigating complex human-centered constraints. This calls for revisiting several facets of workhorse optimization tools and methods under different lenses:

(1) Time scales: the ability to process data streams and provide decision-making capabilities at time scales that match the dynamics of underlying physical, social, and engineered systems using conventional optimization methods can no longer be taken for granted; in addition,

(2) Humans: the presence of humans in the loop, with their preferences and constraints, which have to be learned by using their feedback, is not something that traditional optimization tools can incorporate, inasmuch as the personal cost and constraints are unknown a priori and have to be learned.

Take as an example cyber-physical and social systems [28], e.g., ride-sharing platforms, smart grids, personalized health systems. Information streams are supplied to a decision maker, which could be energy demand/supply, traffic conditions, health signals. These time-varying streams define a time-varying optimization problem that has to be solved as fast as the data arrive. In addition, parts or all of the cost and constraints depend on models for systems that need to be learned while optimizing and could depend on the satisfaction/comfort of human users to a particular set of actions.

The above leads naturally to an optimization problem of the form,

$$\mathsf{P}(t): \ \underset{\boldsymbol{x}\in X(t)\subseteq\mathbf{R}^n}{\text{minimize}}\ f(\boldsymbol{x};t), \quad \text{for all } t\geqslant 0, \tag{1.1}$$

for properly defined cost functions, decision variables, and constraints and time t. Where, the cost f and the constraint set X are both time-varying, because of external data streams. To fix the ideas, think for example about the regularized least-squares cost $f(\boldsymbol{x};t) := \|\boldsymbol{x}-\boldsymbol{b}(t)\|_2^2 + \lambda\mathcal{R}(\boldsymbol{x})$ with a data stream $\boldsymbol{b}(t)$, and regularization $\lambda\mathcal{R}(\boldsymbol{x})$ that could be used for fast video processing in

[1] To create something good/tasty, you need time.

medical contexts, or generally for streaming inverse problems; or think of a smart grid scenario where one optimizes the power generation balancing highly volatile renewables and human satisfaction (which has to be learned) in allocating time-slots for electrical vehicle charging.

This thesis presents my work in this context and in particular, in devising algorithms for Problem (1.1) in various scenarios. In general, we will be looking at online algorithms that can generate sequences of approximate optimizers for certain discrete sampling times t_k, i.e., $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$. These sequences will be proved to converge in some sense to some pertinent limit point, in general to the time-varying optimizers trajectory $\boldsymbol{x}^\star(t)$.

In the thesis, we will care about computations: how fast and in how many computations can the algorithms deliver the next approximate optimizer given a new data point? What is the error associated with such approximate solution? Furthermore, we will care about theoretical guarantees, meaning bounds on pertinent error metrics depending on the problem class and algorithmic parameters. And finally, we will care about applications: are the algorithms usable in practice in realistic application scenarios and do they improve current ones?

1.2 Organization

My aim in these pages is to collect most of the work[2] I have been doing since the end of my PhD in late 2012, and in particular, first, I will be looking at time-varying optimization problems of the form (1.1), for function $f : \mathbf{R}^n \times \mathbf{R}_+ \to \mathbf{R}$, which is a convex function in $\boldsymbol{x}$ which is parametrized in time (i.e., changes in time); and convex set $X(t)$, which also changes in time. In particular, this will be the topic of Chapters 2 and 3.

Then, in Chapter 4, I will extend the framework (1.1) to include non-convexities in function f or in set X, which are possible in cyber-physical systems. Finally, I will close with a variation of Problem (1.1) whereby the cost function is now partly unknown. This latter case models situations in which the cost represents human dissatisfaction to decision $\boldsymbol{x}$ at time t, which might be unknown and it has to be learned via feedback.

All the embodiments of Problem (1.1), together with present and future generalizations, represent stepping stones for a solid theory of optimization in cyber-physical and social systems.

1.3 Theoretical Background

I will be trying whenever possible to keep this thesis self-contained. In general, I will try to give meta-results and informal theorems that can capture the key ideas with basic assumptions and a few key theoretical tools. Then, I will have some closer looks and references to sharper and/or fine-grained analysis.

When reading this thesis, I assume a fairly good background in convex optimization. We indicate with $\mathbf{R}, \mathbf{N}$ the real numbers and natural numbers, respectively. We will deal with convex functions $f(\boldsymbol{x}) : \mathbf{R}^n \to \mathbf{R}$, and often we assume strong convexity and smoothness. A convex function is m-strongly convex if and only if $f(\boldsymbol{x}) - m/2\|\boldsymbol{x}\|^2$ is convex, and L-smooth if and only if $L/2\|\boldsymbol{x}\|^2 - f(\boldsymbol{x})$ is convex. Smoothness implies differentiability, strong convexity does not. We also indicate with $\|\cdot\|$ the Euclidean norm, or a generic induced norm, depending on the context (which we will specify). I use the standard $O(\cdot)$ as the big-O notation, and the $\textsf{PROJ}\{\cdot\}$ and $\textsf{PROX}\{\cdot\}$ are the usual projection and proximal operators, respectively.

We will deal with algorithms, and specifically ones that generate a sequence $(\boldsymbol{x}_k)_{k\in\mathbf{N}}$ that converges to some limit point $\bar{\boldsymbol{x}}$. We are then interested in linearly converging algorithms, in the sense that we can find a $\varrho \in (0,1)$ for which $\|\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}\| \leqslant \varrho\|\boldsymbol{x}_k - \bar{\boldsymbol{x}}\|$.

These algorithms are often called Q-linearly converging, to distinguish them from R-linearly ones for which one is also interested in a supporting sequence $(\boldsymbol{y}_k)_{k\in\mathbf{N}}$, which converges as $\|\boldsymbol{y}_k - \bar{\boldsymbol{y}}\| \leqslant \beta\|\boldsymbol{x}_k - \bar{\boldsymbol{x}}\|$ with $\|\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}\| \leqslant \varrho\|\boldsymbol{x}_k - \bar{\boldsymbol{x}}\|$, for any positive β and $\varrho \in (0,1)$. Since we will not deal with R-linear convergence, we will not make this distinction.

[2] I will be leaving out a few side topics, like graph signal processing and quantum computing.

When discussing algorithms, we may leverage the very powerful machinery of operator theory. In particular, solving the convex problem $\min_{\boldsymbol{x}\in X\subseteq\mathbf{R}^n} f(\boldsymbol{x})$ is equivalent to finding zeros of some carefully constructed operator. The interesting link is that finding zeros of operators can be achieved via fixed point theory, and contracting operators give rise to linearly converging algorithms. I will try to give all the necessary background when needed.

For the rest, the basic theoretical tools I will use are Taylor expansions, triangle inequality, and some linear algebra. The interested readers can find some more background in the standard references [8, 15, 63, 73].

1.4 Selected related work, contributions, applications

To stress my main contributions, I report here some selected related work. For a in-depth literature survey, the reader is suggested to look into my original papers.

1.4.1 Time-varying optimization

Chapters 2-3 deal with time-varying optimization methods in the convex domains and my contributions captured by the works [6, 24, 85, 86, 88, 90–94].

Continuously varying optimization problems represent a natural extension of time-invariant programs when the cost function and constraints may change continuously over time [26, 33, 61, 73]. Recently, time-varying optimization formalisms and the accompanying online solvers have been proposed both in continuous-time [30, 75] and in discrete-time settings [88, 100]. Their main goal is to develop algorithms that can track trajectories of the optimizers of the continuously varying optimization program (up to asymptotic error bounds). The resultant algorithmic frameworks have demonstrated reliable performance in terms of convergence rates, with error bounds that relate tracking capabilities with computational complexity; these features make time-varying algorithms an appealing candidate to tackle dynamic optimization tasks at scale, across many engineering and science domains. Two surveys on time-varying optimization have appeared recently as [24, 90], which I co-authored.

The first survey makes the case on how traditional optimization algorithms behave quite differently when used in time-varying settings, and standard hierarchies (e.g., heavy-ball being better than gradient) may be reversed. In particular, we describe so-called unstructured methods (or correction-only), which incorporate and react to new dynamic data points, but never predict how the optimizers change in time. We also look at applications, such as video streaming and power grids, and we make the link with online learning.

For the latter, we note that the time-varying optimization formalism of unstructured methods is closely aligned with existing works on online learning in dynamic environments [35, 38, 42] from a basic mathematical standpoint. However, a key conceptual difference is that the online algorithms for time-varying optimization are "*computation limited*," (one has only limited computational time before a new datum arrived and they have to re-optimize) whereas online learning is "*data limited*" or "*information limited*" (but not necessarily computation limited), see later detailed discussion.

The second survey [90] expands on structured and unstructured methods, structured being methods that do use optimality conditions as a way to predict the evolution of the optimizer trajectory, and therefore they are called prediction-correction. In particular, we show how prediction-correction methods are more performant in several metrics, even when factoring in the increased computational complexity they require.

My main contribution here is to formalize time-varying optimization in the convex domain, design online algorithms and discuss their theoretical properties. One of the key results is the proposal of prediction-correction methods and their application in cyber-physical examples.

1.4.2 Non-convex aspects

Chapter 4 deals with non-convexities stemming from cyber-physical systems, notably from the non-convex underlying physics, such as in power grid scenarios, and from a combinatorial

problem formulation as in dynamic ridesharing. Here my contributions are captured by the works [10, 22, 23, 29, 69, 95].

Non-convex power grid problems, such as optimal power flow problems, have been dealt with a series of convex relaxations and linearizations [13, 14, 48], with various degrees of success. In the time-varying setting, the approaches my co-authors and I put forward were among the first of their kind. In particular, our proposal was to correct the relaxed problems with measurement feedback coming from the underlying physical system. We have discussed this in a series of works [10, 22, 23]. Related, concurrent, and expanding on this we can cite the works of [21, 36, 37, 67]. Online feedback optimization is now a very vibrant research domain.

Ridesharing is a new mode of transportation that is attracting a lot of attention. The underlying optimization problem, especially in dynamic ridesharing when rides are assigned on-the-go, is a large-scale combinatorial optimization problem that changes over time. Several computationally efficient relaxations of the problem exist [3] with different properties. An interesting feature of the ridesharing problem, and sequential assignment problems in general, is that they can be simplified quite drastically without affecting the quality of the time-varying solution. This latter as long as one is able to adapt to changes quickly. We discuss this in our works [29, 69, 95], which opened the way for urban-scale dynamic ridesharing at scale.

My main contribution here is the formalization and empirical evidence that relaxing or modifying non-convex problems in time-varying settings, such that the modified version is close "enough" to the original problem, yet it has all the functional properties one require for fast convergence and computations, is often enough and sometimes even better than solving the original non-convex problem at optimality. The key aspect here is online corrections of the modified problems, either by feedback or by fast repeated actions.

1.4.3 Personalized optimization

Chapter 5 deals with incorporating user's satisfaction in the cost function of time-varying problems, and presents a condensed version of my works [65, 68, 87, 89].

Incorporating user's satisfaction in the decision-making process has been fairly well studied from an economic standpoint [46]; yet, it is starting now to play a major role in emerging data-driven optimization and control paradigms, especially within the context of cyber-physical and social systems. This is fueled by learning tools that can incorporate feedback from the users, learn their preferences, and then optimization and control tools that can close the loop and determine and implement an optimal decision.

From a purely machine learning perspective, one can distinguish largely two areas of research in this area. One, online bandits (and extensions) whereby functions are learned via user's feedback with pertinent algorithms to decide where to sample next (e.g., [82]); two, preference learning, whereby one learns relationships between decisions, i.e., if the user prefers option A to option B, via noisy feedback (e.g., [20, 39]). Both areas are active research field. Here, the notion of a user's preference represents an absolute value (a continuous function).

User's satisfaction and preferences have been modeled as a Gaussian process (among others); see, e.g., [20, 39]. Gaussian processes have the appeal to naturally handle noise and sparse data, which is key in human applications. For an account of Gaussian processes we refer to [77], while for their use in control we refer the reader to [9, 55]. User's satisfaction and comfort have been taken into account from a control perspective in, e.g., control systems for houses and electric vehicles [17, 66].

Often, in the cyber-physical domain, users' functions have been modeled based on synthetic models or they have been learned a priori (i.e., before the execution of the optimization or control algorithm). In a series of work [65, 68, 89], my co-authors and I incorporate learning modules in online optimization algorithms (with the learning and the optimization tasks implemented in a concurrent timescale) and we do not utilize synthetic models. Our proposed strategy enables fine-grained personalization.

The techniques and ideas that I consider here are also related to inverse reinforcement learning [49] (even though in those settings one still needs to know what is a "good" action via

demonstration), and restless bandit problems [80] (even though they are typically in discrete spaces, while we are in compact continuous spaces).

Finally, I want to mention works that use function approximators to learn dynamical systems in reinforcement learning (RL) [25, 32, 72]. I have not worked on RL on my previous projects, but it may be a good area to keep an eye on.

My main contribution here is the formulation of time-varying optimization problems with unknown costs, that one can learn with users' feedback. The key aspect is that we perform learning and optimization concurrently in an online fashion, while possibly enforcing that the learned user's function has the functional property we require for algorithmic convergence.

1.4.4 Selected applications

Cyber-physical and social systems are ubiquitous. I have surveyed a large number of application domains in [24, 90]. For the sake of this thesis, I mention the following ones.

- Control systems: especially relevant for time-varying (and/or parameter-varying) algorithms for model predictive control, which have appeared for large-scale and embedded systems, e.g., [40, 44, 71].
- Power grid: time-varying problems for power systems can capture high volatility coming from renewable energy production; it can also accommodate dynamic pricing schemes. Time-varying problem formulations (and related online algorithms) can be utilized for tasks such as demand response, optimal power flow (OPF), and state estimation. Examples of works include real-time algorithms for voltage control, optimal power flow, as well as DER management for aggregators; see for example [23, 37, 53, 54, 83]. Human preferences have been considered in [17, 66, 68, 79].
- Transportation systems: time variations may arise from different factors (and at appropriate time-scales), such as variations in the traffic, pedestrians crossing the roads, car accidents, sports events; these factors may lead to time-dependent routing and traffic light control algorithms [31], dynamic ridesharing platforms [3, 95], and routing with and without human preferences [29, 74].
- Machine learning and signal processing: time-varying problems arise when we want to extract sparse features in videos. Works that explore dynamic ℓ_1 reconstruction are, e.g., [16, 96, 98]. Other applications include contemporary approaches for sparse, kernel-based, robust, linear regression, zeroth-order methods, and dynamic classification under concept drift. Additional lines of work include dynamic beamforming [57], and other dynamic signal processing tasks, such as maximum a posteriori estimation [43].
- Robotics: Time varying optimization problems appear in navigation [30], either for single robots or teams of robots [101], as well as robotic manipulators [50, 52, 60]. Users' preferences are considered, e.g., in [56, 59].

Chapter 2

Information streams

Abstract. *In this chapter, I introduce the concept of time-varying optimization and algorithm design for information streams. I show that standard algorithmic hierarchies do not translate verbatim into the time-varying domain, advocating for new foundations. I provide some theoretical results in this direction, and a few application examples. This chapter is adapted from [24, 85, 90, 92, 93].*

2.1 A motivating example

As we have seen in the introduction, there are many application domains that require a revision of the standard static optimization framework. However, we have not really discussed if a new theoretical understanding is needed, or one can reuse standard tools (e.g., the gradient method) in a mini-batch mode. In fact, one may think that a naïve online implementation of algorithms conceived for batch computation may just work well, with algorithms that are faster for batch optimization still being faster in time-varying optimization. If this were the case, then a time-varying algorithmic theory would be quite dull.

However, this is not always the case. Surprisingly, the best algorithms in the static case may be the worst algorithms in the dynamic case, as shown in the illustrative numerical results in Figure 2.1 from [24]. The heavy ball method (which is one of the best algorithms for static quadratic problems) can even diverge for a simple time-varying least-squares problem.

2.2 Problem formulation

Comforted by the fact that some degree of new theoretical understanding is needed for the time-varying optimization framework, I can now (re-)describe the problem more formally.

Let $f : \mathbf{R}^n \times \mathbf{R}_+ \to \mathbf{R}$ be a convex function parametrized over time, i.e., $f(\boldsymbol{x}; t)$, where $\boldsymbol{x} \in \mathbf{R}^n$ is the decision variable and $t \geqslant 0$ is time. Let $X(t) \subseteq \mathbf{R}^n$ be a convex set, which may also change over time. In the first part of this thesis, we are interested in solving:

$$\mathsf{P}(t): \ \underset{\boldsymbol{x} \in X(t)}{\text{minimize}} \ f(\boldsymbol{x}; t), \quad \text{for all } t \geqslant 0. \tag{2.1}$$

To simplify exposition, I assume (for the time being) that the cost function f is m-strongly convex for all t (this is nevertheless a standard assumption in most works), and that the constraint set is never empty. With these assumptions in place, at any time t, Problem (2.1) has a unique global optimizer. This translates to finding the optimal solution trajectory

$$\boldsymbol{x}^\star(t) := \underset{\boldsymbol{x} \in X(t)}{\text{argmin}} \ f(\boldsymbol{x}; t), \quad \text{for all } t \geqslant 0. \tag{2.2}$$

Example 2.1 *As in some robotics example, we are will describe later in the thesis, $f(\boldsymbol{x}; t)$ could represent a time-varying performance metric for the tracking performance of a robot formation that is following a*

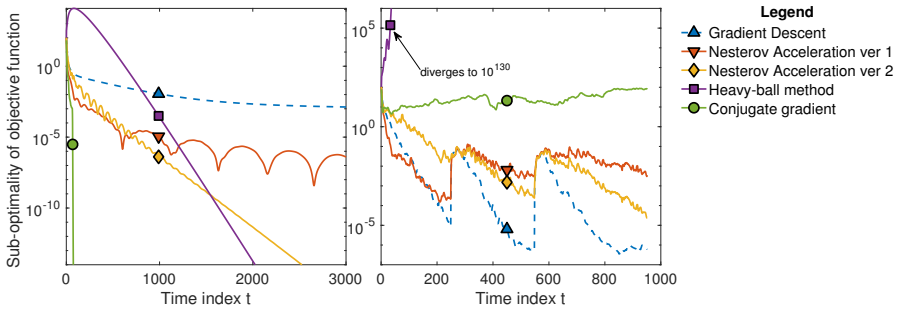

Figure 2.1. *Example of a 50 dimensional time-varying least-squares problem, defined using a sliding window of 50 data points, for 950 time points; two big jumps in the solution near time indices 250 and 550 (by design). Left: Convergence in the* static *case; Right: plot shows the performance of various algorithms on* tracking *the optimal objective value. Nesterov ver. 1 does not use knowledge of strong convexity, while ver. 2 does. The non-linear conjugate gradient exploits the quadratic objective to have an exact line-search (usually impractical), and is the variant from [64, Eq. (5.49)].*

robot leader; e.g., $f(\boldsymbol{x};t) = \|\boldsymbol{x}-\boldsymbol{b}(t)\|^2 + \lambda\mathcal{R}(\boldsymbol{x})$*, where* $\lambda\mathcal{R}(\boldsymbol{x})$ *is some pertinent regularization function and* $\boldsymbol{b}(t)$ *encodes the tracking signal. On the other hand,* $X(t)$ *represents some physical or hardware constraints for the robots. At each* t'*, the information available is* $\{f(\boldsymbol{x};t), t \leqslant t'\}$ *and* $\{X(t), t \leqslant t'\}$*; based on a possibly limited computational complexity, and without any information regarding future costs and constraints, the next decision* $\boldsymbol{x}(t')$ *has to be made; the objective is to produce a decision* $\boldsymbol{x}(t')$ *that is as close as possible to* $\boldsymbol{x}^\star(t')$*.*

If Problem (2.2) changes slowly, and sufficient computational power is available, existing batch optimization methods may identify the optimal trajectory $\boldsymbol{x}^\star(t)$; for example, if the parameter $\boldsymbol{b}(t)$ in the above example exhibits step changes every 10 seconds, and a distributed batch algorithm converges in 5 seconds, then $\boldsymbol{x}^\star(t)$ can be identified (within a given accuracy). On the other hand, in highly dynamic settings, computational and communication bottlenecks may prevent batch methods to produce solutions in a timely manner (e.g., $\boldsymbol{b}(t)$ changes every 0.5 seconds, and a distributed batch algorithm converges in 5 seconds); the problem then becomes related to the synthesis of computationally affordable algorithms that can produce an approximate optimizer trajectory $\hat{\boldsymbol{x}}(t)$ on the fly; accordingly, a key performance of these algorithms is the "distance" between the approximate solution trajectory $\hat{\boldsymbol{x}}(t)$ and the optimal one $\boldsymbol{x}^\star(t)$.

2.2.1 Time sampling

Problem (2.2) is continuous in time. Here, however, motivated by current digital technology, we proceed to sample it at discrete-time steps, tuned to match computational complexities and frequency of updates.

Consider then sampling Problem (2.2) at defined sampling times $\{t_k = kh, k \in \mathbf{N}\}$, with h the sampling period; thus, one arrives at a sequence of time-invariant problems:

$$\boldsymbol{x}^\star(t_k) := \operatorname*{argmin}_{\boldsymbol{x}\in X} f(\boldsymbol{x};t_k), \quad t_k = kh, k \in \mathbf{N}\,. \tag{2.3}$$

For simplicity of exposition, we drop the time dependency of the constraints and consider static sets. As long as one can solve each (time-invariant) Problem (2.3) within an interval h using existing algorithms, then a "batch solution mode" is sufficient to identify the optimal trajectory $(\boldsymbol{x}^\star(t_k), k \in \mathbf{N})$. As I already mentioned extensively, this batch approach is, however, hardly viable, except for low-dimensional problems that can be sampled with sufficiently long sampling periods (i.e., when the problem changes sufficiently slowly). We focus here on the case where one can afford only one or a few steps of a given algorithm within an interval h – i.e., an online approach. This setting can then be cast as **the overarching problem of synthesizing online algorithms that can *track*** $(\boldsymbol{x}^\star(t_k), k \in \mathbf{N})$**, within a given performance.** For the following, we define $(\hat{x}_k)_{k\in\mathbf{N}}$ as the sequence generated by such online algorithms.

2.2.2 Performance metrics

Different performance metrics can be considered for online algorithms that generate approximate trajectories for Problem (2.2). They all capture the fact that the computation of $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ is time-limited, computationally limited, or both, and therefore $\hat{\boldsymbol{x}}_k$ is an approximate optimizer at time t_k. Here, it is more fruitful to look at the computation of $\hat{\boldsymbol{x}}_k$ as limited by time: to compute $\hat{\boldsymbol{x}}_k$ one has at most h.

An immediate performance metric is the *asymptotical tracking error* (ATE), defined as

$$\text{ATE} := \limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\|, \tag{2.4}$$

which captures how the algorithm performs in an asymptotic sense. In general, one seeks *asymptotic consistency* of the algorithm, i.e., if $\boldsymbol{x}^\star(t_k)$ is asymptotically stationary, then the ATE should be zero. However, if $\boldsymbol{x}^\star(t_k)$ is time-varying, the ATE cannot be zero in general [24].

A more complex metric is the dynamic regret (DR), or with a better name: the objective tracking error, defined as

$$\text{DR} := \sum_{k=0}^{T} f(\hat{\boldsymbol{x}}_k; t_k) - f(\boldsymbol{x}^\star(t_k); t_k), \tag{2.5}$$

which captures how the algorithm competes with a dynamic comparator. Typically, one seeks a sub-linearly growing dynamic regret, while, in general, in time-varying settings the best one can achieve is a linearly growing DR, leading to a constant average regret. This is in par with the constant ATE.

A third metric that is relevant for time-varying optimization problems is the *time rate* (TR), defined as

$$\text{TR} := \frac{\text{time required for the computation of } \hat{\boldsymbol{x}}_k}{\text{time allowed for the computation of } \hat{\boldsymbol{x}}_k}. \tag{2.6}$$

Here we define as "time required," the time needed for the computation of an approximate $\hat{\boldsymbol{x}}_k$, which delivers a predefined ATE. The TR is a key differentiator for time-varying optimization: online algorithms need to be able to deliver an approximate $\hat{\boldsymbol{x}}_k$ in the allocated time. Data streams generate decision streams with the same frequency, and the online optimization algorithm needs to have a TR less than one to be implementable. The TR sets also an important trade-off between ATE and implementability. One typically cannot expect a very low ATE and implementable solutions.

The fourth metric is the *convergence rate* (CR), which can be informally defined as

$$\text{CR} := \text{"how fast" an algorithm converges to the ATE.} \tag{2.7}$$

For discrete-time algorithms, under current modeling assumptions, it will be possible to derive linear convergence results.

Typically, the algorithmic design will involve a trade-off between the ATE and CR; for instance, lower levels of ATE may be achievable at the expense of a higher CR. CR is then important, not only at the start, but also when abrupt changes happen (and then the CR captures how fast the algorithm responds to those changes and disturbances).

2.3 Assumptions and algorithms

A key assumption for any online approach is that the difference between solutions at two consecutive times is *bounded*:

Assumption 2.1 *The distance between optimizers at subsequent times is uniformly upper bounded as:*

$$\|\boldsymbol{x}^\star(t_k) - \boldsymbol{x}^\star(t_{k-1})\| \leqslant K, \quad \forall k > 0,\, K < \infty.$$

The constant K will play a key role in the ATE, as shown shortly. Assumption 2.1 is general, inasmuch it does not forbid the underlying trajectory $\boldsymbol{x}^\star(t)$ to have finite jumps.

A stronger assumption, often required in time-varying optimization (and we will use it more extensively in Chapter 3), is that the time derivative of the gradient of the cost function[1], i.e., $\nabla_{t\boldsymbol{x}} f(\boldsymbol{x};t)$, is bounded.

Assumption 2.2 *For all t and all $\boldsymbol{x}$: $\|\nabla_{t\boldsymbol{x}} f(\boldsymbol{x};t)\| \leqslant \Delta_0 < \infty$.*

Assumption 2.2, along with m-strong convexity of the cost function, guarantees that the trajectory $\boldsymbol{x}^\star(t)$ is globally Lipschitz in time [6, 27], and in particular

$$\|\boldsymbol{x}^\star(t') - \boldsymbol{x}^\star(t)\| \leqslant \frac{\Delta_0}{m}|t'-t|. \tag{2.8}$$

Notice that Assumption 2.2 implies Assumption 2.1 with the choice $K = \Delta_0\, h/m$.

2.3.1 Algorithms

With the problem formulation, assumptions, and nomenclature in place, we are ready now to look at algorithm design. In this chapter, we will look at correction-only algorithms, meaning algorithms that react to the changes but not predict how the optimizers may change. These algorithms are also called in different ways (among which catching up, running, unstructured) and probably firstly appeared with Moreau [61]. For example, a correction-only projected gradient to approximately solve (2.3) is given by the recursion

$$\hat{\boldsymbol{x}}_0 = \mathbf{0}, \quad \hat{\boldsymbol{x}}_k = \textsf{PROJ}_X\{\hat{\boldsymbol{x}}_{k-1} - \alpha \nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_{k-1}; t_k)\},\ k \in \mathbf{N}, \tag{2.9}$$

where $\textsf{PROJ}_X\{\cdot\}$ denotes the projection operator and α is a carefully chosen step size (that could be time-varying as well). In (2.9), the projected gradient is applied one time per time step k, but one could also apply multiple gradient steps, say C, per time step.

Notwithstanding this, in general, these correction-only algorithms achieve a high ATE. To formalize this result, we focus on a class of algorithms that exhibit a linear convergence. In particular, let $\mathcal{M}$ be an algorithm that when applied to $\hat{\boldsymbol{x}}_k$ at time t_{k+1} for function $f(\boldsymbol{x}; t_{k+1})$ produces an $\hat{\boldsymbol{x}}_{k+1}$ for which,

$$\|\hat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \varrho \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_{k+1})\|, \quad \varrho \in (0,1). \tag{2.10}$$

This class is common in time-varying optimization (e.g., projected gradient (2.9) is linear on a m-strongly convex, L-smooth cost function [63]). When the algorithm $\mathcal{M}$ is then applied C times, we obtain: $\|\hat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \varrho^C \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_{k+1})\|$. The following general result is in place.

Theorem 2.1 (Informal) *Let $\mathcal{M}$ be an optimization algorithm that converges linearly as in (2.10). Then, under Assumption 2.1, the same algorithm $\mathcal{M}$ applied C times for each time t_k, converges linearly to the optimizer trajectory of a time-varying problem up to an error bound as*

$$\|\hat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \varrho^C (\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| + K),$$

and $\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = \varrho^C O(K) = \frac{\Delta_0}{m}\varrho^C O(h)$, where the last equality is valid under Assumption 2.2.

Proof (Sketch) *At time t_k, if algorithm $\mathcal{M}$ is applied C times, starting on $\hat{\boldsymbol{x}}_k$ and ending at $\hat{\boldsymbol{x}}_{k+1}$, by linear convergence of $\mathcal{M}$, we can write*

$$\|\hat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \varrho^C (\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_{k+1})\|) \leqslant \varrho^C (\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| + \|\boldsymbol{x}^\star(t_{k+1}) - \boldsymbol{x}^\star(t_k)\|),$$

and by using Assumption 2.1 the first claim is established. The second claim is proved by recursively applying the first claim, and by geometric series summation. ⧫

The results of the theorem are general and assert that the sequence $(\hat{\boldsymbol{x}}_k)$ tracks the solution trajectory up to a ball of size $\varrho^C O(K)$. If $C \to \infty$, the time-invariant problem is solved exactly and

[1] This can be generalized for a non-smooth cost function of the form $f(\boldsymbol{x};t) + g(\boldsymbol{x})$, as long as $f(\boldsymbol{x};t)$ is differentiable, e.g., $\|x-t\|^2 + |x|$, see [6].

we are back to the batch mode (and the error is 0), i.e., the time-varying algorithm is asymptotically consistent. If Assumption 2.2 holds true, then the asymptotic error is proportional to the sampling period h (cf. Figure 3.1). In addition, *for fixed* $\varrho \in (0,1)$, $C < \infty$, *and if the path-length*

$$\sum_{k=1}^{T} \|\boldsymbol{x}^\star(t_k) - \boldsymbol{x}^\star(t_{k-1})\|$$

grows at least linearly in T, *no correction-only method of this type can reach a zero ATE* [12, 51].

We close this part with the meta-algorithm whose properties are captured by Theorem 2.1.

Algorithm 2.1 Correction-only meta-algorithm

Input: *Initial point:* $\hat{\boldsymbol{x}}_0$, *a linearly converging method* $\mathcal{M}$, *Number of correction steps* C, *sampling time* h
Output: *A sequence* $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$
1: ***for*** $k \in \mathbf{N}, k \geqslant 1$ ***do***
2: *Sample* $f(\cdot; t_k)$
3: *Correction step: apply* $\mathcal{M}$ *method for* $f(\cdot; t_k)$, C *times to* $\hat{\boldsymbol{x}}_{k-1}$ *to deliver* $\hat{\boldsymbol{x}}_k$
4: ***end for***

2.3.2 Some notable algorithms

While Theorem 2.1 and Algorithm 3.1 are very general, one can specify them to some notable linearly converging methods. My co-authors and I have presented extensive analysis of several of these methods and more in [6, 24, 85], for example:

- Correction-only forward-backward/proximal gradient method. If we consider the problem $\min_{\boldsymbol{x}\in\mathbf{R}^n} f(\boldsymbol{x}; t) + g(\boldsymbol{x})$, with f strongly convex and smooth, and g just convex, we can use the forward-backward method on the sampled problems to obtain a linear converging method. In particular, if f is uniformly m-strongly convex and L-smooth, then the forward-backward method is linearly converging with $\rho = \max\{|1-\alpha m|, |1-\alpha L|\}$, and step size $\alpha < 2/L$.
- Correction-only dual ascent method. If we consider the problem $\min_{\boldsymbol{x}\in\mathbf{R}^n} f(\boldsymbol{x}; t)$, subject to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, with f strongly convex and smooth, and $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ representing a linear equality constraint, then under standard assumptions dual ascent can deliver a Q-linearly converging method (in the dual variables, and R-linear in $\boldsymbol{x}$) for the sample problems.

2.4 Examples

Example 2.2 (Subspace tracking for video streaming) *Robust principal component analysis (PCA) can be used to separate foreground from background in video, among many other applications. Time-varying algorithms can then be used on the resulting time-varying optimization problem, see [24, Example 1] and [1].*

In time-varying optimization, regularizations (e.g., Tikhonov, smoothing) can help both *convergence and asymptotical error*, so both CR and ATE. This could seem surprising, but having fast converging algorithms, even if to a regularized problem, can reduce the ATE (especially if the regularized problem is not very far from the real one). The following example illustrates this.

Example 2.3 (Time-varying capacities in communication networks) *We use a multi-user communication network example, whose time-invariant detailed description is given in [47] and the time-varying version (where the cost is time-varying) is given in [93]. We use then a primal-dual method with double regularization (in the primal space and in the dual space) to construct linearly converging algorithms. We see in Figure 2.2 (from [93]), how for time-varying problems a regularization can help in being more reactive to changes and thereby obtaining a lower ATE than a standard non-regularized method.*

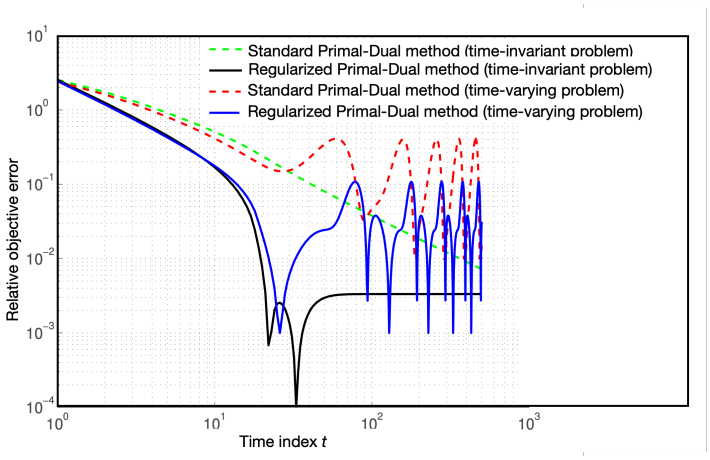

Figure 2.2. *Convergence results for online primal-dual algorithms in time-invariant and time-varying problems. As one can see, in time-invariant problems the relative objective error $|f(\hat{\boldsymbol{x}}_k; t_k) - f(\boldsymbol{x}^\star(t_k); t_k)|/|f(\boldsymbol{x}^\star(t_k); t_k)|$ is eventually higher for algorithms that solve a regularized problem, since the latter has a different optimizer. However, we see that convergence rate is also higher. This helps in time-varying scenarios, where solving a regularized problem may render the approach more agile and ultimately have a smaller ATE. Here the standard primal-dual method is a sub-linearly converging descent-ascent gradient, and we use double smoothing for the regularization, upon which we use a now linearly converging descent-ascent gradient.*

2.5 More scrambling of hierarchies: distributed computation

Figures 2.1 and 2.2 show how standard hierarchies may be scrambled in time-varying scenarios. I now present a further case for it in distributed computation settings. Distributed computation settings are especially challenging in time-varying optimization, since *(i)* to track a time-varying optimizer we need to employ constant step sizes (while many distributed optimization methods use vanishing ones); *(ii)* time sampling of the cost function has to be synchronized, otherwise one minimizes a different problem.

Consider then a prototypical consensus problem:

$$\underset{\boldsymbol{x}\in\mathbf{R}^n}{\text{minimize}}\, f(\boldsymbol{x}; t) := \sum_{i=1}^{N} f_i(\boldsymbol{x}; t)\,, \tag{2.11}$$

where N spatially distributed computing and communicating nodes labeled as $i = 1, \dots, N$, are equipped each one with a private L-smooth and strongly convex cost $f_i(\boldsymbol{x}; t)$.

In Figure 2.3, we illustrate the average tracking error $\sum_i \|\boldsymbol{y}_{i,k} - \boldsymbol{x}^\star(t_k)\|/N$ (where $\boldsymbol{y}_{i,k}$ represents local version of $\hat{\boldsymbol{x}}_k$) for a time-varying problem defined over $N = 20$ nodes (connectivity shown in upper left corner). We study the performance of decentralized gradient descent (DGD) with vanishing step size ($\alpha_k = 1/k$), DGD with constant step size, EXTRA, dual decomposition on the adjacency matrix of the communication graph, and distributed ADMM (see [24] for details on implementation and methods). Note that in this setting, the latter three methods have linear convergence in the static setting. Having better performance in static setting does not clearly predict better performance in the time-varying setting: for example, it seems that dual decomposition does much better in the time-varying scenario, while in the static setting it is worse than EXTRA and ADMM.

The lower left corner of Figure 2.3 illustrates the case where we introduce asynchronicity in the sampling of the cost function. The error is higher for all the algorithms, but it seems that DGD with constant step size is the most robust. This is striking since DGD with constant step size is the worst performing algorithm in the static setting, and shows once more that results in a static scenario cannot be easily translated into time-varying settings. This aspect has then been recently expanded upon in [99].

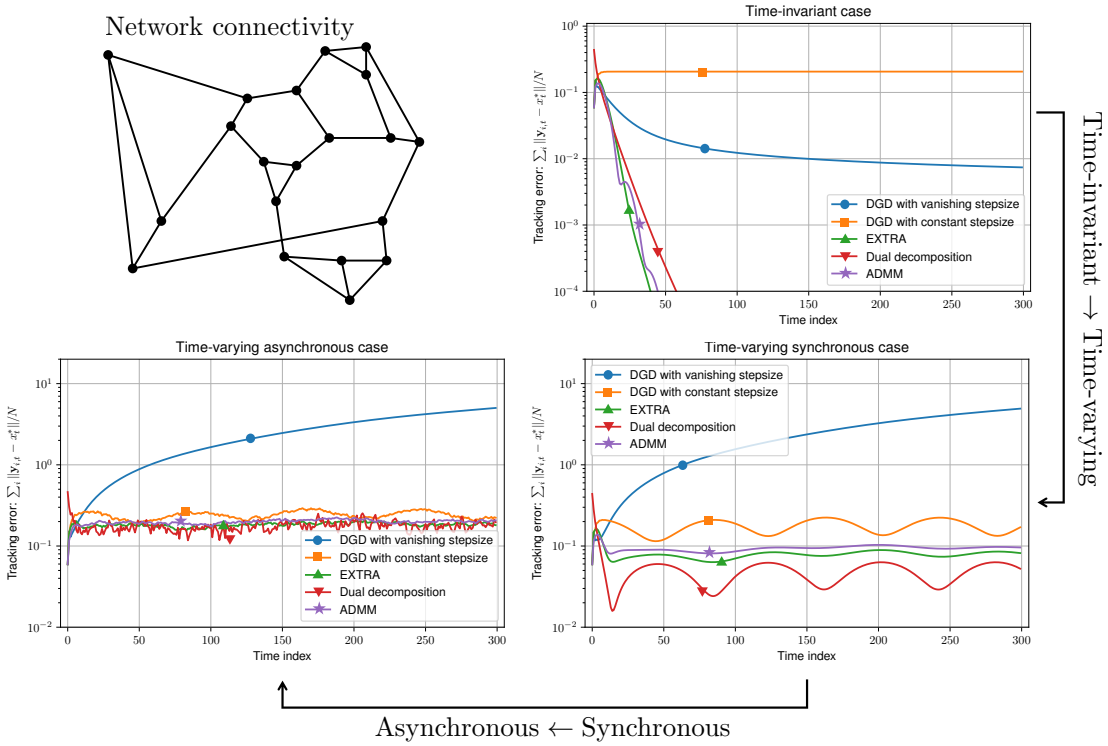

Figure 2.3. *Numerical simulations for a time-varying optimization problem solved in a distributed way.* Top left*: the communication graph consisting of 20 nodes;* Top right*: tracking error in the time-invariant case;* Bottom right*: tracking error in the time-varying synchronous case;* Bottom left*: tracking error in the time-varying asynchronous case.*

2.6 Going beyond strong convexity

So far we have assumed that function f is uniformly strongly convex, so that the solution's trajectory is unique and Lipschitz continuous. While it is possible to extend this framework to non-convex settings, with functions only locally strongly convex (see [26, 100]), it is another issue to generalize it to non-strongly convex functions (either locally or globally). And in fact, in view of the examples presented so far (e.g., Example 2.3) one may even wonder if it is needed at all. One can regularize the original problem and converge fast to a regularized solution plus an ATE, which could be better than using a slower converging algorithm to the exact solution, if in this second case the ATE is larger. Notwithstanding, we look now at this latter case.

There are many reasons why going beyond strong convexity is hard in time-varying optimization. Above all, if we lose even strict convexity, then the optimizer is in general non-unique and one can have complex situations, like sets, bifurcations, jumps, and so forth [33]. Then, convergence properties in this setting are, not just more complicated, but often quite weak to be used in "practice".

In [85], I have looked at generic convex problems in time-varying settings. The point of depart is to interpret optimization algorithms as fixed-point algorithms featuring averaged operators, and use the fixed-point iterations in a correction-only fashion. Consider $(\mathsf{T}_k)_{k\in\mathbf{N}}$ as a sequence of averaged operators from $\mathbf{R}^n \to \mathbf{R}^n$, and assume $\operatorname{fix} \mathsf{T}_k \neq \varnothing$, for all k. Let $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ be the sequence generated by the following correction-only fixed-point iteration:

$$\hat{\boldsymbol{x}}_0 = \mathbf{0}, \qquad \hat{\boldsymbol{x}}_{k+1} = \mathsf{T}_k(\hat{\boldsymbol{x}}_k). \tag{2.12}$$

Let the operators T_k be bounded, in the sense that we can define $\Omega := \max_{k\in\mathbf{N},\boldsymbol{x}_k\in\mathbf{R}} \|\mathsf{T}_k(\boldsymbol{x}_k)\|$, then, a meta-result is that the fixed-point residual would converge as,

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{k=1}^{T} \|\mathsf{T}_k(\hat{\boldsymbol{x}}_k) - \hat{\boldsymbol{x}}_k\|^2 = O(K^2\Omega), \tag{2.13}$$

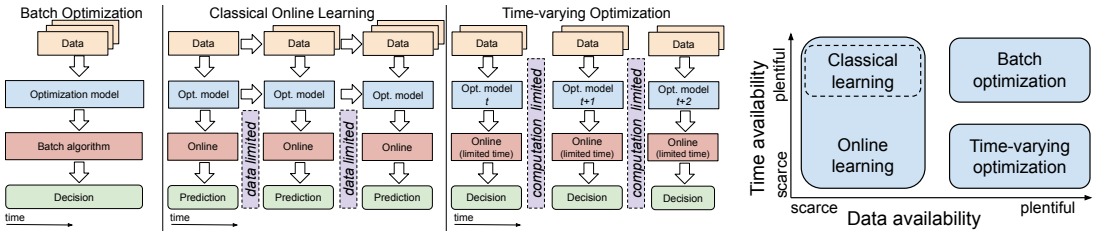

Figure 2.4. *Online learning and time-varying optimization are both sequential, but in online learning data is limited in the sense that $f(\cdot; t_{k+1})$ is not available. In time-varying optimization, there is no restriction of information, but the full problem cannot be solved within a single time step due to computational cost. The rightmost plot provides an illustrative distinction between: (i) time-varying algorithms that have knowledge of $f(\cdot; t_{k+1})$, but are computationally limited; (ii) classical learning methods (such as the follow-the-leader method) that do not necessarily have computational limitations [38, 78]; and (iii) online learning, where only one algorithmic step is performed per time interval (such as the online mirror descent method [35]). In both (ii) and (iii), no knowledge of $f(\cdot; t_{k+1})$ is available.*

where $K \geqslant 0$ represents variations on the fixed points of the operators at subsequent time steps.

One can then use the relationship between optimization and fixed-point theory to specify the above result for standard optimization algorithms in the correction-only mode. For instance, in [85], I give results for primal methods (proximal gradient, and others), as well as dual methods (e.g., ADMM). Even though, on the one hand, we are able to say something about these online methods for non-strongly convex costs, on the other hand, I feel that results like this one are a tad unsatisfactory. First, the compactness assumption and the presence of the bound Ω in the ATE bound render the asymptotic bound loose. Second, the error is in terms of average fixed-point residual, which does not capture the nature of the problem, where one is interested in how different we are from the optimizers set at time t_k, rather than as an average in the past.

More research in this direction is needed and I left it for future endeavors.

2.7 Time-varying optimization vis-à-vis Learning

We close this chapter with some discussion on the relationship between time-varying optimization and online learning.

The time-varying optimization formalism is closely aligned with existing works on "online learning in dynamic environments" [35,38,42,78] from a basic mathematical standpoint. However, a key conceptual difference is that the online algorithms for time-varying optimization described here are "*computation limited*," whereas online learning is "*data limited*" or "*information limited*" (but not necessarily computation limited). To understand this it is sufficient to look at the performance metrics. The performance of online learning is evaluated relative to the best action in hindsight (given a certain $f(\cdot; t_k)$, we compute a prediction $\hat{\boldsymbol{x}}_{k+1}$ for $f(\cdot; t_{k+1})$ and we measure how different we are to the case in which we knew $f(\cdot; t_{k+1})$). Instead, in time-varying settings we have $f(\cdot; t_{k+1})$ and the performance is measured against the solution that would have been obtained had we had the time to run an algorithm to convergence at each interval h.

In time-varying optimization, we also focus on algorithms implemented with a constant step size; this is a natural choice for cases where the optimal solution $\boldsymbol{x}^\star(t)$ remains transient and the algorithm runs indefinitely. This is another distinction relative to online learning with a time-invariant optimizer, where the step size may depend on the time horizon or a "doubling trick" [78, Sec. 2.3.1] is utilized (with the latter still involving changes in the algorithm based on how many iterations have been taken).

Chapter 3

Structured predictions

Abstract. *In this chapter, I introduce the concept of predicting the evolution of the optimizers of time-varying optimization problems and how to design algorithms accordingly. I show that this new framework delivers better results than the one I presented in Chapter 2, sometimes even when the computational effort is taken into account. I provide quite general theoretical results and cite a few application examples. This chapter is adapted from [6, 86, 88, 90, 91, 94].*

3.1 The need for prediction

In the previous chapter, I have looked at online methods that react to changes in the cost function. Since at least Polyak [73], researchers have wondered if one could not do better by predicting how the optimizers change in time, and then reacting to the change. This has led to research in prediction-correction methods.

With this in mind, this chapter overviews key modeling and algorithmic design concepts, with emphasis on prediciton-correction methodologies, here also called *time-structured* (structured for short) time-varying algorithms for convex time-varying optimization. One can use the term "structured" here to stress that we make use of the inherent temporal structure, meaning we leverage prior information (such as Lipschitz continuity or smoothness) on the evolution of the optimal trajectory to enhance convergence and tracking. In contrast, the term "unstructured" will refer to time-varying algorithms that simply rely on current information of cost and constraints (which are the ones we have looked at in the previous chapter).

To further motivate structured time-varying methods, Figure 3.1 illustrates the asymptotic tracking error (ATE) for different sampling periods (h) of discrete-time algorithms, for a robot tracking problem (see [5] and later for the setting). The value of exploiting the temporal structure of the problem can be appreciated. Even keeping computational time fixed, structured algorithms outperform unstructured ones (here by several orders of magnitude). This motivates our theoretical developments.

3.2 Predictors

We are now ready to focus on discrete-time algorithms that are endowed with a prediction. Various predictors are considered, and we will call as $\hat{\boldsymbol{x}}_{k+1|k}$ the predicted decision variable for time t_{k+1} with information up to time t_k.

- *Clairvoyant oracles and expert oracles.* Clairvoyant oracles offer an exact prediction: i.e., they provide a $\hat{\boldsymbol{x}}_{k+1|k}$, for which $\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = \|\hat{\boldsymbol{x}}_{k+1|k} - \boldsymbol{x}^\star(t_{k+1})\|$, as if they knew the function $f(\cdot; t_{k+1})$ and its gradient at time t_k. In this context, clairvoyant oracles completely remove the time effect in the optimizer and the optimizer can proceed as if the cost function were not varying in time. Clairvoyant oracles are impractical (they need to have a perfect knowledge of the future), but they offer good performance lower bounds (since, one cannot do better than them). A noteworthy example of when one can use a clairvoyant oracle is when the cost

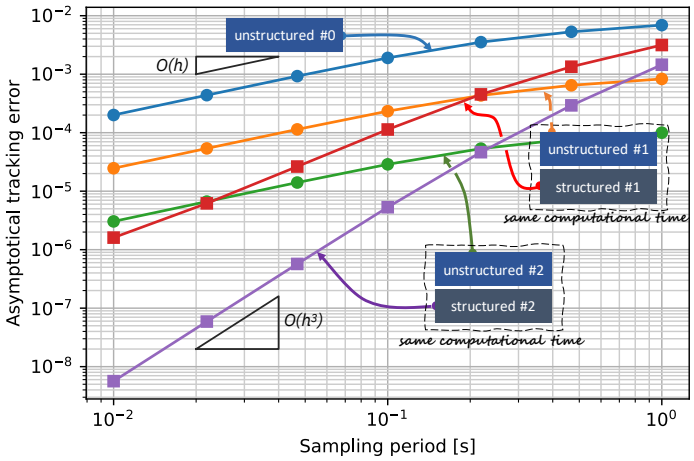

Figure 3.1. *Structured algorithms can outperform unstructured ones, even keeping the computational time fixed: here for a robot tracking problem. Unstructured algorithms 0, 1, and 2 are in this case online versions of the proximal gradient method, for which we perform 5, 7, and 9 passes of the methods, respectively. Structured algorithms here employ either a first- or a second-order Taylor model (for structured 1 and 2, respectively), and 5, and 20 passes of an online version of the proximal gradient method on a simplified quadratic problem; see later and [5] for further details.*

function has a time drift, i.e., $f(\boldsymbol{x};t) = f(\boldsymbol{x} + \boldsymbol{\alpha}t)$, and the oracle can estimate the drift vector $\boldsymbol{\alpha}$ exactly based on historical data.

Expert oracles, hints, or predictable sequences are considered, e.g., in [42, 76]. In [42], one has access to a sequence $(\boldsymbol{m}_k)_{k\in\mathbf{N}_+}$ of gradient approximators. When $\boldsymbol{m}_k = \mathbf{0}$, i.e., meaning no knowledge or prediction about the future, we recover an unstructured algorithm. When $\boldsymbol{m}_k = \nabla_{\boldsymbol{x}} f(\boldsymbol{x};t_k)$ at time t_k, then one recovers the online algorithm of [19]. Finally, when $\boldsymbol{m}_k = \nabla_{\boldsymbol{x}} f(\boldsymbol{x};t_{k+1})$, one recovers a clairvoyant oracle. Based on the error $\|\boldsymbol{m}_k - \nabla_{\boldsymbol{x}} f(\boldsymbol{x};t_{k+1})\|$, one can then derive dynamic ATE results.

- *Model-based predictors.* These predictors are built on a model of the variations of the cost function, or of its parameters.

  (i) Prediction based on first-order optimality conditions [6, 26, 86, 94, 100]. A large class of predictors comes from deriving models based on first-order optimality conditions. We could call these predictors environment-agnostic, since they are not interested in modeling how the environment changes, but only how the optimization problem is affected. To introduce these predictors, let us consider an unconstrained problem (easier than Problem (2.3)) as:

$$\boldsymbol{x}^\star(t_k) = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbf{R}^n} f(\boldsymbol{x};t_k). \tag{3.1}$$

  To derive a model for how the problem is changing from t_k to t_{k+1}, we look at the first-order optimality conditions at time t_k, which can be framed as

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x};t_k) = \mathbf{0}. \tag{3.2}$$

  To predict, how this first-order optimality condition changes in time, with information available up to t_k, we use a Taylor expansion around $(\hat{\boldsymbol{x}}_k;t_k)$ as

$$\mathbf{0} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x};t_{k+1}) \approx \varphi_k(\boldsymbol{x}) := \nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k) + \nabla_{\boldsymbol{x}\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k)(\boldsymbol{x} - \hat{\boldsymbol{x}}_k) + h\,\nabla_{t\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k), \tag{3.3}$$

  where it is assumed that the Hessian $\nabla_{\boldsymbol{x}\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k)$ exists, as well as the time derivative of the gradient $\nabla_{t\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k)$, leading to the prediction model[1]

$$\varphi_k(\boldsymbol{x}^\star_{k+1|k}) = \mathbf{0} \implies \boldsymbol{x}^\star_{k+1|k} = \hat{\boldsymbol{x}}_k - \nabla^{-1}_{\boldsymbol{x}\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k)[\nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k) + h\,\nabla_{t\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k;t_k)]. \tag{3.4}$$

[1] The time derivative $\nabla_{t\boldsymbol{x}} f(\boldsymbol{x};t_k)$ can be obtained via first-order backward finite difference if not available otherwise, see [6, 94].

The prediction (3.4) represents a nonlinear discrete-time model to compute $\boldsymbol{x}^\star_{k+1|k}$. Note that $\varphi_k(\boldsymbol{x})$ can be interpreted as a specific choice for the gradient approximator $\boldsymbol{m}_k$ in [42]– see discussion in the oracles paragraph. Let us now consider a slightly more general setting than Problem (2.3) as:

$$\boldsymbol{x}^\star(t_k) = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbf{R}^n} f(\boldsymbol{x}; t_k) + g(\boldsymbol{x}) \tag{3.5}$$

where $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is a convex closed and proper function (e.g., $g(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$). Problem (2.3) is a special case of (3.5), when $g(\boldsymbol{x})$ is the indicator function of the set X. Once again, we look at the first-order optimality conditions at time t_k, which can be framed as the generalized equation [27]

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}; t_k) + \partial g(\boldsymbol{x}) \ni \mathbf{0}. \tag{3.6}$$

To predict how this first-order optimality condition changes in time (with information up to t_k), one can use a Taylor expansion around $(\hat{\boldsymbol{x}}_k; t_k)$, leading to the prediction model

$$\varphi_k(\boldsymbol{x}^\star_{k+1|k}) + \partial g(\boldsymbol{x}^\star_{k+1|k}) \ni \mathbf{0}. \tag{3.7}$$

Thus the prediction step requires the solution of this approximated generalized equation with initial condition $\hat{\boldsymbol{x}}_k$, which can be obtained, or approximated, cheaply with e.g., a few passes of a proximal gradient method (cheaply since φ_k is a convex quadratic function with the same strong convexity and smoothness constant as f). The formulation (3.7) represents the prediction model for the presented class of optimization problems (3.5), for a first-order Taylor expansion. Other prediction models exist for other classes of optimization problems [86, 94], for higher-order Taylor expansions [26], and for more complex numerical integration methods [34, 45, 52].

(ii) Prediction based on first-order optimality conditions and extrapolation [6]. This is a variation of the Taylor model (3.3), where we use extrapolation techniques to derive the approximation of $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}; t_{k+1})$ as

$$\varphi_k(\boldsymbol{x}) = \sum_{i=1}^{I} \ell_i f(\boldsymbol{x}; t_{k+1-i}), \quad \forall \boldsymbol{x} \in \mathbf{R}^n, \qquad \ell_i := (-1)^{i-1}\binom{I}{i}. \tag{3.8}$$

The rest follows as in the Taylor expansion model discussed previously. A bit of care has only to be put in the fact that $\varphi_k(\boldsymbol{x})$ is now a more complicated function of $\boldsymbol{x}$, and a non-convex function in general. However, by assuming that the Hessian of f is time invariant, since $\sum_{i=1}^{I} \ell_i = 1$, then strong convexity is ensured.

(iii) Prediction based on parameter estimation [16]. When the time dependence hides a parameter dependence, then models obtained via filtering are a viable alternative. Let $\boldsymbol{b}(t) \in \mathbf{R}^l$ be a parameter, and let the function $f(\boldsymbol{x}; t) = f(\boldsymbol{x}; \boldsymbol{b}(t))$: e.g., the cost depends on the data stream $\boldsymbol{b}(t)$ representing for example the position of a robot to track. Then $\boldsymbol{b}(t)$ at time t_{k+1} can be estimated via, for example, a Kalman filter based on the linear time-invariant model:

$$\boldsymbol{b}(t_{k+1}) = \boldsymbol{\Gamma}\boldsymbol{b}(t_k) + \boldsymbol{w}_k, \quad \boldsymbol{y}_k = \boldsymbol{\Phi}\boldsymbol{b}(t_k) + \boldsymbol{n}_k, \tag{3.9}$$

for given matrices $\boldsymbol{\Gamma} \in \mathbf{R}^{l\times l}$, $\boldsymbol{\Phi} \in \mathbf{R}^{q\times l}$, observations $\boldsymbol{y}_k \in \mathbf{R}^q$, and noise terms $\boldsymbol{w}_k \in \mathbf{R}^l, \boldsymbol{n}_k \in \mathbf{R}^q$. Then the prediction model requires the (approximate) solution of the problem

$$\hat{\boldsymbol{x}}_{k+1|k} \approx \operatorname*{argmin}_{x\in X} f(\boldsymbol{x}; \hat{\boldsymbol{b}}_{k+1}), \tag{3.10}$$

with $\hat{\boldsymbol{b}}_{k+1}$ being the forecasted $\boldsymbol{b}(t_{k+1})$ based on the model (3.9) via e.g., a Kalman filter. Other models can be thought of based on non-linear models, more complicated forecasters, and even neural networks.

3.3 Algorithms

We have presented a few predictors for discrete-time time-varying optimization algorithms. No general result exists to encompass all the predictors. However, for a particular class of

predictors (the one that employs first-order optimality conditions as prediction model) some general results can be derived. These methods have roots in non-stationary optimization [73], parametric programming [26,27,100], and continuation methods in numerical mathematics [2].

Consider Problem (3.5) for simplicity (although arguments are generalizable). Let $\mathcal{P}$ be a predictor method that approximates $\boldsymbol{x}^\star_{k+1|k}$ based on (3.7), in a linear convergent fashion: one application of $\mathcal{P}$ acting on $\hat{\boldsymbol{x}}_k$ delivers a $\hat{\boldsymbol{x}}_{k+1|k}$ for which

$$\|\hat{\boldsymbol{x}}_{k+1|k} - \boldsymbol{x}^\star_{k+1|k}\| \leqslant \varrho_1 \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star_{k+1|k}\|, \quad \varrho_1 \in (0,1). \tag{3.11}$$

E.g., $\mathcal{P}$ could be a proximal gradient algorithm, in which case:

$$\hat{\boldsymbol{x}}_{k+1|k} = \textsf{PROX}_{\alpha g}\{\hat{\boldsymbol{x}}_k - \alpha \nabla_{\boldsymbol{x}} \varphi_k(\hat{\boldsymbol{x}}_k)\}, \tag{3.12}$$

where $\textsf{PROX}_{\alpha g}\{\cdot\}$ is the proximal operator for function g and step-size α, which could be applied one or multiple, say P, times for time step. Let now $\mathcal{M}$, belonging to the same algorithm class of (2.10), be applied to the update (correction) step after function acquisition at t_{k+1}, for which one application on $\hat{\boldsymbol{x}}_{k+1|k}$, delivers

$$\|\hat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \varrho_2 \|\hat{\boldsymbol{x}}_{k+1|k} - \boldsymbol{x}^\star(t_{k+1})\|, \quad \varrho_2 \in (0,1), \tag{3.13}$$

for example another proximal gradient step as

$$\hat{\boldsymbol{x}}_{k+1} = \textsf{PROX}_{\alpha g}\{\hat{\boldsymbol{x}}_{k+1|k} - \alpha \nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_{k+1|k}; t_{k+1})\}. \tag{3.14}$$

Then, we can present the following result is in place pertaining a predictor in the form of a Taylor expansion of the first order, like in Eq. (3.3).

Theorem 3.1 (Informal) *Consider the time-varying Problem (3.5) and two methods $\mathcal{P}$ and $\mathcal{M}$ for which (3.11)-(3.13) hold. Let the predictor $\mathcal{P}$ based on a first-order Taylor expansion of f (Cf. (3.3)) be applied P times during the prediction step, and the corrector $\mathcal{M}$ be applied C times. Consider Assumption 2.2 to hold and additionally, let $f(\boldsymbol{x};t)$ be L-smooth (in addition to being m-strongly convex), with a well-defined Hessian $\nabla_{\boldsymbol{xx}} f(\boldsymbol{x};t)$. Then, there exists a minimal number of prediction and correction steps P, C for which globally (i.e., starting from any initial condition)*

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = \frac{\Delta_0}{m} \varrho_1^C \, O(h).$$

In addition, if we consider the assumption that higher-order derivatives of the cost function are bounded[2] *as*

$$\max\{\|\nabla_{\boldsymbol{xxx}} f(\boldsymbol{x};t)\|, \, \|\nabla_{t\boldsymbol{xx}} f(\boldsymbol{x};t)\|, \, \|\nabla_{tt\boldsymbol{x}} f(\boldsymbol{x};t)\|\} \leqslant \Delta_1,$$

uniformly in time and for all $x \in \mathbf{R}^n$, then locally (and for small h), there exists a minimal number of prediction and correction steps P, C so that

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = \underbrace{O(\Delta_1 \varrho_1^C \, h^2)}_{\textit{prediction gain}} + \underbrace{O(\Delta_0 \varrho_1^C \varrho_2^P \, h)}_{\textit{approximation error}}.$$

Proof *(Sketch) The proof here proceeds as follows: we first bound the error coming from the prediction (e.g., (3.12)), next bound the one from the correction (e.g., (3.14)), and then combine them. For the error coming from the prediction, two errors must be considered, one coming from the model (due to the Taylor expansion error), the other coming from the P prediction steps. When considering exact prediction ($P \to \infty$), the leading error is the Taylor expansion error (namely the error in (3.3)), which is $O(h)$ in general, and $O(h^2)$ when higher-order derivatives are bounded.* ♦

The results of Theorem 3.1 are fairly general and apply to different problem classes [6,86,94]. For example, in [6], we look at a fairly good number of primal and dual methods that can be applied in the time-varying domain, which are based on the results of Theorem 3.1 and generalizations thereof.

[2] Where induced Euclidean norms are considered for tensors.

Theorem 3.1 indicates that tracking is not worse than correction-only methods in the worst case. If the function has some higher degree of smoothness, and we are interested in a local result, then a better ATE can be achieved, provided some (stricter) conditions on the number of prediction and correction steps are verified. The ATE is composed of two terms; one which is labeled as approximation error, which is due to the early termination of the prediction step (if $P \to \infty$ and prediction is exact, this term goes to 0). The other, named prediction gain, is the gain coming from using a prediction step, which brings the error down to a $O(h^2)$ dependence on h. This depends on the first-order Taylor expansion employed; other methods can further reduce this to $O(h^4)$ or less [26,34,45,52] (look again at Figure 3.1, where we have also employed a Taylor model up to degree 2 for (3.3) to obtain an $O(h^3)$ error bound).

The higher degree of smoothness required for the local results imposes boundedness of the tensor $\nabla_{\boldsymbol{xxx}} f(\boldsymbol{x}; t)$, which is a typical assumption for second-order algorithms (notice that the predictor requires second-order information, cf. (3.4)-(3.7) and its solution is comparable to solving a Newton step, which is *locally* quadratically converging). Moreover, it bounds the variability of the Hessian of f over time, which guarantees the possibility of performing more accurate predictions of the optimal trajectory. Theorem 3.1 depicts a key result in prediction-correction methods: the prediction value is fully exploited with higher smoothness.

We close this part with the meta-algorithm whose properties are captured by Theorem 3.1.

Algorithm 3.1 Prediction-correction meta-algorithm

Input: *Initial point: $\hat{\boldsymbol{x}}_{1|0}$, linearly converging methods $\mathcal{P}, \mathcal{M}$, Number of prediction and correction steps P, C, sampling time h*

Output: *A sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$*

1: ***for*** $k \in \mathbf{N}, k \geqslant 1$ ***do***
2: *Sample $f(\cdot; t_k)$*
3: *Correction step: apply $\mathcal{M}$ method for $f(\cdot; t_k)$, C times to $\hat{\boldsymbol{x}}_{k|k-1}$ to deliver $\hat{\boldsymbol{x}}_k$*
4: *Prediction step: apply $\mathcal{P}$ method for an approximation of $f(\cdot; t_{k+1})$, P times to $\hat{\boldsymbol{x}}_k$ to deliver $\hat{\boldsymbol{x}}_{k+1|k}$*
5: ***end for***

3.4 A closer look

After having presented the big lines of prediction-correction methods, I am going now to give a closer look on the mathematical details. This will help in understanding the challenges and the opportunities for improvement.

3.4.1 Prediction-agnostic error bound

First, let us present general convergence results which are independent of the particular prediction method that is employed. This could be used then to quickly generalize to any prediction method. Consider once again Problem (3.5) and two methods $\mathcal{P}$ and $\mathcal{M}$ for which (3.11)-(3.13) hold. Here we assume the two methods to have the same contraction parameter $\varrho_1 = \varrho_1 = \varrho \in (0, 1)$, and remind that f is m-strongly convex uniformly in time. Define the two functions,

$$\zeta(\ell) := \begin{cases} 1, & \text{for } \ell = 0, \\ \frac{1}{m}\varrho^\ell, & \text{otherwise} \end{cases} \quad \text{and} \quad \xi(\ell) := \begin{cases} 0, & \text{for } \ell = 0, \\ 1 + \frac{1}{m}\varrho^\ell, & \text{otherwise} \end{cases}. \tag{3.15}$$

The main result is as follows.

Proposition 3.1 (General error bound, [6] Proposition 5.1) *Consider the time-varying Problem (3.5) and two methods $\mathcal{P}$ and $\mathcal{M}$ for which (3.11)-(3.13) hold. Let the predictor $\mathcal{P}$ be applied P times during the prediction step, and the corrector $\mathcal{M}$ be applied C times. Let $K_k, \tau_k \in (0, +\infty)$ be such that for any $k \in \mathbf{N}$:*

$$\|\boldsymbol{x}^\star(t_{k+1}) - \boldsymbol{x}^\star(t_k)\| \leqslant K_k \quad \textit{and} \quad \|\hat{\boldsymbol{x}}^\star_{k+1|k} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \tau_k, \tag{3.16}$$

with $\hat{\boldsymbol{x}}^\star_{k+1|k}$ indicating the solution of the prediction problem (for any prediction).

Choose the number of prediction and correction steps P and C such that $\zeta(C)\zeta(P) < 1$. Then the error incurred by a prediction-correction method that uses method $\mathcal{M}$ for P prediction steps and C correction steps, is upper bounded by:

$$\|\hat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \zeta(C)\Big(\zeta(P)\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| + \zeta(P)K_k + \xi(P)\tau_k\Big), \tag{3.17}$$

with functions ζ and ξ defined in (3.15).

Now letting $K := \sup_{k\in\mathbf{N}} K_k$ and $\tau := \sup_{k\in\mathbf{N}} \tau_k$, we have,

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = \frac{\zeta(C)}{1-\zeta(C)\zeta(P)}\left(\zeta(P)K + \xi(P)\tau\right) \tag{3.18}$$

which gives an upper bound to this asymptotic tracking error. Look now back at Theorem 3.1 and see how Eq. (3.18) highlights the same fundamental result.

With Proposition 3.1 in place, one can derive bounds for different prediction strategies, which determine different τ_k's, as we have done in [6], and I report briefly next.

3.4.2 Taylor-based prediction

Let us consider now the prediction as in Equation (3.3) for $\nabla_{\boldsymbol{x}} f(\cdot; t_{k+1})$, for an arbitrary order:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}; t_{k+1}) \approx \varphi_k(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k; t_k) + \nabla_{\boldsymbol{x}\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k; t_k)(\boldsymbol{x} - \hat{\boldsymbol{x}}_k) + h\,\nabla_{t\boldsymbol{x}} f(\hat{\boldsymbol{x}}_k; t_k) + \cdots . \tag{3.19}$$

In this case, for a I-1-th order Taylor expansion, if $\varphi_k(\boldsymbol{x})$ is at least locally m-strongly convex, we encounter an error of

$$\tau_k = \frac{1}{m}\sum_{i=0}^{I} \frac{1}{i\,!\,(I-i)\,!}\|\nabla_{(t)^i(\boldsymbol{x})^{I-i}} f\|\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_{k+1})\|^{I-i}\, h^i\,. \tag{3.20}$$

We can then use the result (3.17), and the fact that $\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_{k+1})\| \leqslant \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| + O(h)$ and refine the findings of Theorem 3.1. In particular, under higher-order derivative smoothness and locally (small enough $\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\|$, small enough h), one can arrive at conditions for which

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = O(\zeta(P)h) + O(\xi(P)h^I). \tag{3.21}$$

3.4.3 Extrapolation-based prediction

Let us consider now the prediction using an extrapolation as in Equation (3.8). Here, under the additional assumption that $\nabla_{\boldsymbol{x}\boldsymbol{x}} f(\cdot; t_k)$ is time invariant (which is the case in many applications), we can obtain global results. In particular, under higher-order derivative smoothness, and the step condition $\zeta(C)\zeta(P) < 1$, one can arrive at the tracking error

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = O(\zeta(P)h) + O(\xi(P)h^I), \tag{3.22}$$

for a I-1-th order extrapolation, for any initial condition, and sampling period h.

We see that extrapolation-based predictions seem to be more flexible and global: they do not require one to compute high-order terms, but only to keep in memory function gradients at previous time steps, e.g., $\nabla_{\boldsymbol{x}} f(\cdot; t_k), \nabla_{\boldsymbol{x}} f(\cdot; t_{k-1}), \ldots$, and they converge globally up to $O(h^I)$. However, they require the time invariance of the Hessian, and, if we impose this condition, they are (at least empirically) very similar to Taylor-based prediction methods.

3.5 Examples

Many examples have been reported in my papers, such as [5, 81, 88, 90] stemming mainly from power grids and robotics. Figure 3.1 reports the robotic example from [5], pertaining a leader following problem for a group of N robots in a rigid formation.

Chapter 4

Non-convexities

Abstract. *In this chapter, I look at time-varying non-convex problems through specific examples stemming from cyber-physical systems. The aim is to show that in time-varying settings, even moderate-to-big simplifications of the problem might be reasonable and can lead to satisfactory time-varying solutions. In particular, I will present simplifications corrected by feedback in smart-grid settings, and simplifications corrected by repeated actions in ridesharing platforms. This chapter is adapted from [10,22,23,29,69,95].*

4.1 A general plea for model simplicity

So far, we stayed in the comfort zone of convexity, with its solid theoretical guarantees and well-behaving algorithms. In this chapter, we take a little detour in the non-convex domain, which is widespread in cyber-physical systems. The approach I will take hinges on the fact that we are in time-varying settings and batch solutions are not available. So, since we cannot solve for $\boldsymbol{x}^\star(t)$, and we will have always an asymptotic error, then we may as well try to modify the problem formulation in an easier problem and solve a simplified version, faster. This is the same idea of the regularizations in Chapter 2, Example 2.3, but transposed in the non-convex setting.

There are a few ways to formalize and implement this idea in the non-convex domain, and I will be looking at two of them, which really stress important aspects of cyber-physical systems: (1) an underlying physical system; (2) the possibility of integer decisions.

The reader is also invited to see the more recent [7], where with my co-authors, I formulate the problem in a bit more generic terms as trying to find the "closest" time-varying convex problem to a given time-varying non-convex one (under some weakly-convex assumptions – a function $f : \mathbf{R}^n \to \mathbf{R}$ μ-weakly convex if and only if $f(\boldsymbol{x}) + \mu/2\|\boldsymbol{x}\|^2$ is convex).

For the more theoretical-prone reader, I remark that the results in this chapter are somewhat less "sharp" than the ones presented in the other chapters (since in general the non-convex domain is more theoretically challenging) but they are well supported empirically in real-life settings.

4.2 Non-convex constraints: the value of feedback

4.2.1 Cyber-physical systems

Cyber-physical systems are associated with a collection of systems coupled through physical interdependencies, and logically connected by an information infrastructure that supports control and optimization tasks. Examples include communication systems, power grids, and robotic networks. In practice, one has a digital layer, where one takes decisions, and a physical layer, where the decisions are carried out. In the power grid setting, decisions may be power or voltages commands, that are then implemented and affect other physical aspects of the grid.

Here, we consider a physical system that maps an input $\boldsymbol{x} \in X \subseteq \mathbf{R}^n$ and an unknown distur-

bance/uncontrollable input $\boldsymbol{w} \in W \subseteq \mathbf{R}^p$ to the output $\boldsymbol{y} \in \mathbf{R}^m$ according to the map

$$\boldsymbol{y} = \pi(\boldsymbol{x}, \boldsymbol{w}), \tag{4.1}$$

where $\pi : X \times W \to \mathbf{R}^m$ is continuously differentiable and locally Lipschitz continuous in $\boldsymbol{x}$. We wish to optimally manage the physical system (4.1), and, as such, we are interested in solving an optimization problem of the form

$$\underset{\boldsymbol{x}\in X, \boldsymbol{y}\in\mathbf{R}^m}{\text{minimize}} \; F(\boldsymbol{x}, \boldsymbol{y}; t), \qquad \text{subject to } \boldsymbol{y} = \pi(\boldsymbol{x}, \boldsymbol{w}). \tag{4.2}$$

Problem (4.2) is a time-varying optimization problem in the form of (2.1). In particular, the cost can model time-varying conditions and depends on the decision variable that we apply to the system, $\boldsymbol{w} \in W$ is an unknown disturbance, the convex set X represents physical limits on the input quantities. Even when F is convex in $(\boldsymbol{x}, \boldsymbol{y})$, which will be the case for us (and in most cases, since it is we that design it), the physics map π is nonlinear, making the overall problem non-convex.

Example 4.1 *As an example of π consider the power-flow equation, for which $\boldsymbol{y}$ represents voltages and $\boldsymbol{x}$ and $\boldsymbol{w}$ controllable and uncontrollable powers. The relationship between voltage V and power P is non-linear (and from high-school physics, an upgraded version of the $V = \sqrt{P/R}$ for a resistance R).*

Besides being non-convex, Problem (4.2) is hard in general since we require a precise knowledge of π and of the disturbance $\boldsymbol{w}$, which is often unavailable. However, in many real-world applications, the system operator will have access to a linearized model of the system (4.1) of the form

$$\boldsymbol{y} \approx \Pi \boldsymbol{x} + \Pi_{\boldsymbol{w}} \boldsymbol{w} \tag{4.3}$$

and to a forecast or guess $\hat{\boldsymbol{w}}$ for the disturbance $\boldsymbol{w}$. We can then solve the *feedforward* optimization problem

$$\underset{\boldsymbol{x}\in X, \boldsymbol{y}\in\mathbf{R}^m}{\text{minimize}} \; F(\boldsymbol{x}, \boldsymbol{y}; t), \qquad \text{subject to } \boldsymbol{y} = \Pi \boldsymbol{x} + \Pi_{\boldsymbol{w}} \hat{\boldsymbol{w}}. \tag{4.4}$$

Solving (4.4) and applying its solution to the system can be seen as analogous to applying feedforward control to a dynamical system.

For the sake of simplicity, let us assume that $F(\boldsymbol{x}, \boldsymbol{y}; t) := f(\boldsymbol{x}; t) + g(\boldsymbol{y}; t)$, so we divide the dependencies (this is hardly a simplification, since we can always redefine π to include an identity map, and therefore regain the original problem). Assume also that f and g are smooth convex functions of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively.

With this, we can rewrite (4.4) as

$$\underset{\boldsymbol{x}\in X}{\text{minimize}} \; F_\ell(\boldsymbol{x}; \hat{\boldsymbol{w}}; t) := f(\boldsymbol{x}; t) + g(\Pi \boldsymbol{x} + \Pi_{\boldsymbol{w}} \hat{\boldsymbol{w}}; t), \tag{4.5}$$

we can sample it at discrete time steps t_k, and the use, e.g., a correction-only projected gradient descent to find and track $\boldsymbol{x}^\star(t)$ (since now the problem is smooth and convex, and we can also safely assume strongly so). In practice, we can write (for a step size $\alpha > 0$)

$$\hat{\boldsymbol{x}}_0 = \mathbf{0}, \qquad \hat{\boldsymbol{x}}_k = \textsc{Proj}_X[\hat{\boldsymbol{x}}_{k-1} - \alpha(\nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_{k-1}; t_k) + \Pi^{\mathsf{T}} \nabla_{\boldsymbol{y}} g(\Pi \hat{\boldsymbol{x}}_{k-1} + \Pi_{\boldsymbol{w}} \hat{\boldsymbol{w}}_{k-1}; t_k)], \quad k \in \mathbf{N}_{>0}. \tag{4.6}$$

Since the cost is convex in $\boldsymbol{x}$ and the set X is convex, the online projected gradient will converge under suitable assumptions (see Chapter 2) to the optimizer trajectory. Optimizers which will be different from the ones of the original Problem (4.2), because we have approximated π, and forecasted the disturbance. However, the most problematic issue here is that the optimal input-output couple will not be feasible for the system (since $\boldsymbol{y} \neq \pi(\boldsymbol{x}, \boldsymbol{w})$), which could be very difficult to justify to system operators, and implement in practice. In addition, we still assume the availability of a forecast for the disturbance, which is hard to obtain in practice.

To address challenges outlined above, the idea is to suitably modify the algorithmic updates of online optimization methods, such as (4.6), to accommodate measurement feedback – something that henceforth is referred to as *online optimization with feedback*. In particular, we could

implement the decision $\hat{\boldsymbol{x}}_k$ and *measure* the output $\hat{\boldsymbol{y}}_k = \pi(\hat{\boldsymbol{x}}_k, \boldsymbol{w}_k)$, thereby letting the underlying system "solve" the non-linearities for us. As such, we consider modifying (4.6) as

$$\hat{\boldsymbol{x}}_0 = \mathbf{0}, \qquad \hat{\boldsymbol{x}}_k = \textsc{Proj}_X[\hat{\boldsymbol{x}}_{k-1} - \alpha(\nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_{k-1}; t_k) + \Pi^{\mathsf{T}} \nabla_{\boldsymbol{y}} g(\hat{\boldsymbol{y}}_{k-1}; t_k)], \quad k \in \mathbf{N}_{>0}. \tag{4.7}$$

where the measurement $\hat{\boldsymbol{y}}_{k-1}$ replaces the model $\boldsymbol{y}_{k-1} = \Pi \hat{\boldsymbol{x}}_{k-1} + \Pi_{\boldsymbol{w}} \hat{\boldsymbol{w}}_{k-1}$. This simple conceptual modification leads to the following key advantages: (1) instead of measuring/estimating $\boldsymbol{w}$ we rely on m measurements of the outputs $\boldsymbol{y}$, which are easier to obtain in current cyber-physical systems; (2) the algorithm naturally accounts for the underlying physics via the measurements $\hat{\boldsymbol{y}}_k$, so the optimal input-output couple is feasible and implementable.

Empirical evidence on real systems goes even beyond point (2) suggesting that the so-called sensitivity matrix Π may be even quite far from the underlying system, and the online algorithm with feedback would still be performing well in practice. This makes updates like (4.7) practically model-free, and highlights the value of feedback in non-convex time-varying settings.

Remark 1 *Looking at updates (4.6), one could think of using $\nabla_{\boldsymbol{x}} \pi(\hat{\boldsymbol{x}}_{k-1}, \hat{w}_{k-1})$ instead of Π at every time step. This would basically mean solving the non-convex problem (4.2), via gradient descent. This is hardly feasible in practice, since $\nabla_{\boldsymbol{x}} \pi$ is very hard to obtain for large-scale systems, as power grids, transportation systems, etc., and gradient descent in non-convex problems could converge arbitrarily slowly, especially if we employ line-search for the step size. See [10, 21].*

Remark 2 *We have implicitly assumed that the underlying physical system reacts with faster time scales than the optimization algorithm. That is, we can safely assume that $\hat{\boldsymbol{y}}_k = \pi(\hat{\boldsymbol{x}}_k, \boldsymbol{w}_k)$ with no delay. One can also consider a dynamical case, but convergence is only ensured assuming time scale separation [36].*

We report in Algorithm 4.1 a meta-algorithm, where one can appreciate how we use the underlying physical system to "solve" the optimization non-linearities for us.

Algorithm 4.1 Online optimization with feedback meta-algorithm

Input: *Initial points: $\hat{\boldsymbol{x}}_0, \hat{\boldsymbol{y}}_0$, a sensitivity matrix Π, a step size α, sampling time h*
Output: *A sequence $(\hat{\boldsymbol{x}}_k)_{k \in \mathbf{N}}$*
1: ***for*** $k \in \mathbf{N}, k \geqslant 1$ ***do***
2: *Sample $f(\cdot; t_k)$, $g(\cdot; t_k)$*
3: *Correction step: apply one step of online projected gradient descent to $\hat{\boldsymbol{x}}_{k-1}, \hat{\boldsymbol{y}}_{k-1}$ to deliver $\hat{\boldsymbol{x}}_k$:*

$$\hat{\boldsymbol{x}}_k = \textsc{Proj}_X[\hat{\boldsymbol{x}}_{k-1} - \alpha(\nabla_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}_{k-1}; t_k) + \Pi^{\mathsf{T}} \nabla_{\boldsymbol{y}} g(\hat{\boldsymbol{y}}_{k-1}; t_k)]$$

4: *Feedback step: implement $\hat{\boldsymbol{x}}_k$ and measure $\hat{\boldsymbol{y}}_k = \pi(\hat{\boldsymbol{x}}_k, \boldsymbol{w}_k)$.*
5: ***end for***

We remark here that only one online projected gradient descent step is run, since feedback is obtained with the same time scale of the optimization algorithm. We could generalize this to different time scales, but the qualitative results will not change.

4.2.2 Assumptions and convergence

Let us look now at some high-level results pertaining convergence of the proposed online feedback approach. Consider the original problem (4.2), and let $\bar{\boldsymbol{x}}(t_k)$ be a KKT point of it at time t_k, meaning

$$\nabla_{\boldsymbol{x}} f(\bar{\boldsymbol{x}}(t_k); t_k) + \nabla_{\boldsymbol{x}} \pi(\bar{\boldsymbol{x}}(t_k), \boldsymbol{w}_k) \nabla_{\boldsymbol{y}} g(\bar{\boldsymbol{y}}(t_k); t_k) + \mathcal{N}_X(\bar{\boldsymbol{x}}(t_k)) \ni 0, \qquad \bar{\boldsymbol{y}}(t_k) = \pi(\bar{\boldsymbol{x}}(t_k), \boldsymbol{w}_k), \tag{4.8}$$

where $\mathcal{N}_X(\cdot)$ is the normal cone operator of convex set X.

Consider now the unique optimizer of the sampled version of the convex problem (4.5), and call it $\boldsymbol{x}^\star(t_k)$, i.e., $\boldsymbol{x}^\star(t_k) = \arg\min_{\boldsymbol{x} \in X} F_\ell(\boldsymbol{x}, \hat{\boldsymbol{w}}_k; t_k)$.

We can look at convergence of the sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ is two possible ways: either by using $\boldsymbol{x}^\star(t_k)$ or by using Algorithm 4.1 as a frame of reference, leading to similar results, under similar assumptions.

Theorem 4.1 *Assume that function $F_\ell(\boldsymbol{x}, \hat{\boldsymbol{w}}; t)$ is a L-smooth strongly convex function in $\boldsymbol{x}$ for all $t \geqslant 0$. Assume also that for all $k \in \mathbf{N}$, we can bound*

$$\|\boldsymbol{x}^\star(t_k) - \boldsymbol{x}^\star(t_{k-1})\| \leqslant K, \qquad \|\Pi^\mathsf{T}\nabla_{\boldsymbol{y}} g(\Pi\hat{\boldsymbol{x}}_{k-1} + \Pi_{\boldsymbol{w}}\hat{\boldsymbol{w}}_{k-1}; t_k) - \Pi^\mathsf{T}\nabla_{\boldsymbol{y}} g(\hat{\boldsymbol{y}}_{k-1}; t_k)\| \leqslant c_{\Delta_1},$$

Then the sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ generated by Algorithm 4.1 with step size $0 < \alpha < 2/L$ converges as

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \boldsymbol{x}^\star(t_k)\| = \underbrace{O(K)}_{\text{Time variations}} + \underbrace{O(c_{\Delta_1})}_{\text{Price of feasibility}} .$$

Proof *Immediate via triangle inequality and smooth strongly convex cost, for which projected gradient is a linearly converging algorithm.* ♦

Define now the operator $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}; t) := \nabla_{\boldsymbol{x}} f(\boldsymbol{x}; t) + \Pi^\mathsf{T}\nabla_{\boldsymbol{y}} g(\pi(\boldsymbol{x}; \boldsymbol{w}); t)$. Algorithm 4.1 can be interpreted as finding the time-varying fixed point of $\boldsymbol{x} = \textsf{PROJ}_X(\boldsymbol{x} - \alpha\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}; t))$. Consider for now a time-invariant scenario. If we have that $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}; \bar{t})$ is ϱ-strongly monotone and L-Lipschitz continuous in $\boldsymbol{x}$ for $t = \bar{t}$, then the solution $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}; \bar{t}) = \mathbf{0}$ is unique and Algorithm 4.1 with step size $0 < \alpha < 2\varrho/L^2$ converges linearly to it. Let $\boldsymbol{x}^{\boldsymbol{F}}(\bar{t})$ be said solution. Let us do another step.

Theorem 4.2 *Assume that $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}; t)$ is ϱ-strongly monotone and L-Lipschitz continuous in $\boldsymbol{x}$ for all $t \geqslant 0$ and denote with $\boldsymbol{x}^{\boldsymbol{F}}(t)$ the unique solutions $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}; t) = \mathbf{0}$. Assume also that for all $k \in \mathbf{N}$, we can bound*

$$\|\boldsymbol{x}^{\boldsymbol{F}}(t_k) - \boldsymbol{x}^{\boldsymbol{F}}(t_{k-1})\| \leqslant K', \qquad \|(\Pi - \nabla_{\boldsymbol{x}}\pi(\bar{\boldsymbol{x}}(t_k), \boldsymbol{w}))^\mathsf{T}\nabla_{\boldsymbol{y}} g(\pi(\bar{\boldsymbol{x}}(t_k), \boldsymbol{w}); t_k)\| \leqslant c_{\Delta_2},$$

Then the sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ generated by Algorithm 4.1 with step size $0 < \alpha < 2\varrho/L^2$ converges as

$$\limsup_{k\to\infty} \|\hat{\boldsymbol{x}}_k - \bar{\boldsymbol{x}}(t_k)\| = \underbrace{O(K')}_{\text{Time variations}} + \underbrace{O(c_{\Delta_2})}_{\text{Price of implementability}}, \quad \text{and } \|\bar{\boldsymbol{x}}(t_k) - \boldsymbol{x}^{\boldsymbol{F}}(t_k)\| \leqslant c_{\Delta_2}/\varrho.$$

Proof *The proof follows from monotone operator theory together with implicit function theorems, see [21], [62, Theorem 1.14].* ♦

Let us look at the results a little closer. Theorem 4.1 uses the convex problem (4.5) as a frame of reference, since its optimizer trajectory is well defined. We then assume boundedness of the time variations (as done in Theorem 2.1 of Chapter 2), and a bound on the approximation quality of the linear model and disturbance prediction. Finally, we can derive a tracking result which showcases an asymptotic tracking error determined by the bound on the time variability and the model error.

If the model was correct, then $c_{\Delta_1} = 0$ and $\boldsymbol{x}^\star(t) = \bar{\boldsymbol{x}}(t)$, and we are back in the standard convex time-varying case. Otherwise, when the model is not correct, we pay the price of feasibility (meaning that we consider a feasible update $\boldsymbol{y} = \pi(\boldsymbol{x}, \boldsymbol{w})$, instead of one coming from the model).

Theorem 4.1 is a bit unsatisfactory, since the assumption on the model error c_{Δ_1} depends on the algorithmic sequence, and even though assuming compactness of X and W one can possibly bound c_{Δ_1}, this may be quite loose. In addition, the convergence is to a point $\boldsymbol{x}^\star(t_k)$ which is practically irrelevant, and the feedback, which should bring a plus to the scheme, is treated as an error term.

To overcome these challenges, Theorem 4.2 starts with a different frame of reference: the algorithm directly, seen as a fixed point. Under strong monotonicity and Lipschitz continuity, plus assumptions on bounded time variations and model error boundedness *at a KKT point*, then a tracking result is derived. The result pertains the distance of $\hat{\boldsymbol{x}}_k$ to a KKT of the original problem, and it is bounded by time variation errors and the price we pay for implementability (i.e., the possibility to run the algorithm in real time, without having to determine $\boldsymbol{w}$ and $\nabla_{\boldsymbol{x}}\pi(\boldsymbol{x}, \boldsymbol{w})$, and with the knowledge that the algorithm is converging to a unique fixed point).

Theorem 4.2 has more reasonable assumptions than Theorem 4.1, and the only strong one is the ϱ-strongly monotone assumption. However, since we can design f (and tune Π), as long as $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x};t)$ is L-Lipschitz continuous, we can add a Tikonhov regularization to bring $\boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x};t)$ to be strongly monotone, which we know it is not very problematic in time-varying settings.

4.2.3 Extensions

So far we have looked at problems where only the cost was dependent on the output $\boldsymbol{y}$. One can extend this setting to problems with constraints that depend on $\boldsymbol{y}$ and combinations thereof. In [10,22,23], my co-authors and I have considered this setting modifying the gradient scheme with a primal-dual, and obtaining qualitatively the same guarantees.

We explore such an extension with a practical example in power grids.

Example 4.2 (Optimal power flow pursuit) *Consider the problem of optimizing in real time the operation of aggregations of distributed energy resources (DERs) located in (a portion of) a distribution system (e.g., a distribution feeder). The problem is formulated as a time-varying optimal power flow problem (OPF) and fits the proposed framework. Particularly, we consider a distribution network with one slack bus and N PQ-buses. We assume, without loss of generality, that a controllable resource is connected at every PQ bus $i = 1,\dots,N$. The controllable quantity of resource i is given by $\boldsymbol{x}_i := (P_i, Q_i)^\mathsf{T} \subseteq \mathbf{R}^2$, where P_i and Q_i are the net active and reactive power injections from resource i, respectively. The objective function for a photovoltaic (PV) system i is $f_i(P_i, Q_i; t) = c_p \left(P_i - \bar{P}_i(t)\right)^2 + c_q Q_i^2$, where $\bar{P}_i(t)$ is the maximum real power available at PV system i at time t. We set $X_i(t) = \left\{(P_i, Q_i) : P_i^2 + Q_i^2 \leqslant S_{i,\max}^2,\ 0 \leqslant P_i \leqslant \bar{P}_i(t)\right\}$ where $S_{i,\max}$ is the rated apparent power for the PV system i; similar costs and sets are considered for energy storage systems, with the additional constraints on P_i based on the current state of charge.*

We consider the mapping $\boldsymbol{y} = \pi(\boldsymbol{x}, \boldsymbol{w}) \approx \Pi\boldsymbol{x} + \Pi_{\boldsymbol{w}}\boldsymbol{w} \in \mathbf{R}^{3N}$ as the linearized *mapping from power injections to the voltage magnitudes at each node, derived using, e.g., [4]; the vector $\boldsymbol{w}$ collects the uncontrollable power injections at every node. We also consider engineering constraints involving the voltage magnitudes at each node to be within an interval $[0.95, 1.05]$ p.u., thus defining the following constraints: $\boldsymbol{c}(\boldsymbol{y}) = ([0.95\boldsymbol{1} - \boldsymbol{y}]^\mathsf{T}, [\boldsymbol{y} - 1.05\boldsymbol{1}]^\mathsf{T})^\mathsf{T} \leqslant 0$.*

We consider then the time-varying optimization problem

$$\underset{\boldsymbol{x}\in\mathbf{R}^{2N},\boldsymbol{y}\in\mathbf{R}^{3N}}{\text{minimize}} \sum_{i}^{N} f_i(\boldsymbol{x}_i;t), \text{ subject to: } \boldsymbol{x}_i \in X_i(t),\ \boldsymbol{c}(\boldsymbol{y}) \leqslant 0, \quad \boldsymbol{y} = \pi(\boldsymbol{x}, \boldsymbol{w}), \tag{4.9}$$

which we solve via a projected primal-dual scheme on the linearized $\pi(\boldsymbol{x}, \boldsymbol{w}) \approx \Pi\boldsymbol{x} + \Pi_{\boldsymbol{w}}\boldsymbol{w}$ and corrected via feedback. Illustrative numerical result for the AC OPF problem are then reported in [10,22,23], and they showcase how the discussed method enforces feasibility online and it is (by construction) implementable in real-time system, and better performance with respect to linearization alone, as well as other traditional methods.

4.3 Combinatorial problems: an example

We look now at combinatorial problems stemming from transportation and ridesharing systems. It goes without saying that combinatorial problems are much harder than convex programs, and even more so in time-varying settings, where a plethora of behaviors could occur (solutions' bifurcations, jumps, discontinuities, etc.). More than that, one could actually wonder whether the solution at time t_k has anything to do with the solution at time t_{k+1}, if the coefficients of the problem change. In particular, we cannot expect to have Lipschitz continuous optimizer trajectories in combinatorial problems.

Our methodology in continuous optimization settings was to approximate the sequence of solutions $(\boldsymbol{x}^\star(t_k))_{k\in\mathbf{N}}$ with another sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ which was computed by computationally frugal methods within each sampling period h. The errors at each time step were then limited by leveraging the knowledge of past solutions and, possibly the evolution of the optimizers

written in the optimality conditions. One of the main results we derived was that if the sampling period was sufficiently small, or the problem was changing sufficiently slow, then the solution of the time-varying problem was eventually indistinguishable from $(\boldsymbol{x}^\star(t_k))_{k\in\mathbf{N}}$.

Since the solutions at different sample times $\boldsymbol{x}^\star(t_k)$ in combinatorial settings are almost different entities, our continuous optimization methodology cannot be expected to work, in general.

4.3.1 Sequential assignment problems

One of the cornerstones of combinatorial problems is assignment problems, where one wants to assign a series of goods to a series of customers, minimizing a pertinent metric. Linear assignment problems are a particular case of assignment problems, where the cost is linear.

Let us consider n goods to be assigned to m customers, such that no customer gets more than one good, and all the goods are assigned. Let $\boldsymbol{x} \in \{0,1\}^{n\times m}$ be the assignment matrix ($x_{ij} = 1$ if goods i is assigned to customer j, and 0 otherwise). Then a time-varying assignment problem is

$$\underset{\boldsymbol{x}\in\{0,1\}^{n\times m}}{\text{minimize}} \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}(x_{ij};t), \qquad \text{subject to} \quad \forall j: \sum_{i=1}^{n} x_{ij} \leqslant 1,\ \forall i: \sum_{j=1}^{m} x_{ij} = 1, \tag{4.10}$$

where f_{ij}'s are scalar non-negative cost functions associated with the assignment x_{ij}.

Solving assignment problems is hard, and more so if f_{ij}'s change and the sampling period is small. In addition, what does it really mean to have a time-varying assignment problem? Do customers get assigned goods and then after a time we could take them away from them? Or just, we have new customers and the same goods that can be shared among different customers? Or?

As we see, it is quite complicated to talk about time-varying combinatorial problems and the meaning of their solutions.

In the following, we will put ourselves in the setting in which customers are revealed over time, to which we assign available goods (goods are then assigned to customers for a time period and then freed). In this setting, while finding an approximate sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$ has little meaning, it makes sense to find an approximate sequence of problems, easier to be solved *exactly* in the time period h, which, however, can deliver overall good solutions in the long run.

This is a change in perspective, which is, however, not so distant from the regularization ideas of Chapter 2 and the previous sections of this chapter. We solve an easier problem at each time step t_k and we look at solution quality in the long run. To make things easier to understand, let us look at a specific problem.

4.3.2 Ridesharing problem

Let us consider a concrete problem stemming from dynamic ridesharing: we have customer requests for trips that have to be assigned dynamically to available vehicles, and vehicles can be shared with customers, if their trip allows it. If we were to consider the complete problem, at each time t_k, we would need to assign vehicles to multiple customers who can share their ride. This is the approach of [3], which requires the solution of a hard combinatorial problem, which in turn requires high computational resources and time.

Our approach is different: let us simplify the problem and consider only one customer to vehicle assignment per time step. This leads to a convex problem (!) which can be solved very efficiently, so we can lower the computational requirements and sampling time. This makes us also more reactive and possibly better in the long run.

Let us look at it in more details.

Formal problem formulation

Given a set $\mathcal{M}$ of m customer trip requests at time t_k and a ridesharing fleet $\mathcal{C} = \{1,\dots,n\}$ of vehicles, each with capacity C_i, we are interested in determining ridesharing solutions in real

Table 4.1. *Results and comparison with [3]. SR stands for service rate, i.e., the percentage of accepted requests that satisfy trip constraints such as waiting time and detour times of less than 7 minutes.*

	N. vehicles	capacity	h [s]	SR [%]	waiting [min]	detour [min]	comp. time [s]
Here and [95]	3000	4	10	99.87	3.23	2.59	7.87
	3000	4	30	99.09	3.04	2.47	12.77
[3]	3000	4	30	97.91	2.70	2.28	51.55
	3000	10	30	98.58	2.56	2.74	60.39

time. Our ridesharing service aim at providing an assignment of requests (customers) to available vehicles and their correspondent routes, according to some optimization criteria. Available vehicles are those that can pick up customers, while complying with the time constraints associated with the requests, and without exceeding their seat capacity.

We express the ridesharing service as an optimization engine, which is run at specific time periods t_k ($k = 0, 1, 2, \dots$). At each such time instant t_k, the service processes the requests submitted by customers in the time window $(t_{k-1}, t_k]$, and find optimal vehicle-costumer assignment and correspondent routes.

Our algorithm leverages the following feature: at each optimization run, at most one new request (customer) is assigned to a vehicle. This design choice enables us to reduce the ridesharing optimization problem into a linear assignment problem (whose convex relaxation is exact), which can be solved efficiently and in a very fast manner.

This simplifying assumption is not restrictive: vehicles will have multiple customers assigned to them, just no more than one coming from the same time window $(t_{k-1}, t_k]$. We see already that if the sampling period h is small enough, then this simplification is a small price to pay, which unlocks fast and efficient assignment algorithms. In fact, this simplification may result even in better solutions in the long run, since the problem is highly dynamic, so optimality at time t_k does not imply optimality in the long run (practically, assigning two customers to a vehicle now, may prevent that vehicle to have better assignments later).

4.3.3 Results

While the technical details can be found in [95], we report here some interesting results considering the New York City taxi public dataset and extracting one week of data trips. The findings presented in Table 4.1 show qualitatively similar performance and lower computational time with the competing algorithm of [3] with solves an exact assignment problem at each sampling period (e.g., no one customer per vehicle assumption). In particular, we achieve a similar performance as [3] while our computational time is as much as 4 times lower than theirs. This is even more impressive, since we implement our algorithm using the Python 2.7 language on a 2.7GHz Intel i5 laptop with 8GB RAM, while in [3] their algorithm is implemented in C++ on a 24 core machine.

Table 4.1 supports our idea of simplifying combinatorial problems to solve them more efficiently, with "corrections" coming from their fast repeated implementation. Extending this to other combinatorial problems could be possible, given the often very fine line between easy and hard problems in combinatorial domains [70]. However, this is hardly a general statement and in some cases things are more nuanced. The interested reader can have a look at [29], where my co-authors and I showcase another combinatorial problem from transportation, where the convexification works a bit less good than here, but we are still able to find good problem simplifications to trade off fast solutions with long-term optimality.

Chapter 5

Human preferences

Abstract. *In this chapter, I introduce the concept of human preference and satisfaction and I move from cyber-physical systems to cyber-physical and social systems. Preferences are modeled via an unknown cost function that has to be learned via user's feedback, which resembles the feedback action in Chapter 4. I provide general theoretical results for different learning mechanisms and an example from vehicle platooning. This chapter is adapted from [65, 68, 87, 89].*

5.1 Intermezzo: Connecting the dots

In the previous chapter, we have studied cases in which the decisions affect the physics of a system and change its state via a (possibly) non-linear model $\boldsymbol{y} = \pi(\boldsymbol{x}, \boldsymbol{w})$, see Equation (4.1). We handle these nonlinearities with measurement feedback. The next reasonable step[1] is to consider a social model measuring satisfaction or dissatisfaction of users to which we feed the decision $\boldsymbol{x}$. In some sense, humans act as a physical system that given an input it provides a feedback. We look at this next.

Remark 3 *Here, I use the word preference in an online bandit-like (and extensions) setting, whereby functions are learned via user's feedback. Sometimes, preference learning in machine learning is intended as the setting where one learns relationships between decisions, i.e., if the user prefers option A to option B, via noisy feedback. I do not look at the latter here.*

5.2 Problem formulation

I extend the setting from the previous chapters to include human preferences as a cost function to be minimized. In particular, consider again Problem (1.1), but here with the twist of a composite structure:

$$\mathsf{P}(t): \quad \underset{\boldsymbol{x}\in X}{\text{minimize}}\, f(\boldsymbol{x};t) = V(\boldsymbol{x};t) + U(\boldsymbol{x}), \quad \text{for } t \geqslant 0, \tag{5.1}$$

with $V(\boldsymbol{x};t) : \mathbf{R}^n \times \mathbf{R}_+ \to \mathbf{R}$ being strong convex and known, while $U(\boldsymbol{x}) : \mathbf{R}^n \to \mathbf{R}$ unknown and to be learned. The two functions represent the engineering objective and the users (human) dissatisfaction to a particular decision $\boldsymbol{x}$ at time t. Here, I implicitly assume that the user's dissatisfaction function $U(\boldsymbol{x})$ does not change in time; however, slow changes of the function do not affect reasoning and technical approach[2]. Furthermore, $U(\boldsymbol{x})$ is not necessarily a convex function, thus rendering $\mathsf{P}(t)$ a challenging problem even in the static setting. The structure of Problem (5.1) naturally suggests that optimal decisions strike a balance between design choices

[1] At least it sounded reasonable after a nice conversation with Prof. Dough Leith at Trinity College Dublin and my former colleague Julien Monteil.

[2] Since we consider systems for which the changes in V have the time scale of seconds or minutes, it is unlikely that humans change their preferences at this scale [56, 72]. The case of time-varying preferences, extensions could be envisioned by leveraging time-varying or contextual bandits.

(which may be time-varying, e.g., in tracking problems) and user's preferences, which are often slowly varying and not known beforehand.

As done in the previous chapters, consider sampling Problem (5.1) at discrete time instances t_k, $k = 1, 2, \ldots$. This leads to a sequence of time-invariant problems as:

$$\mathsf{P}_k : \quad \underset{\boldsymbol{x} \in X}{\text{minimize}}\, f(\boldsymbol{x}; t_k) = V(\boldsymbol{x}; t_k) + U(\boldsymbol{x}), \tag{5.2}$$

which we want to solve approximately within the sampling period h to generate a sequence of approximate optimizers $\{\boldsymbol{x}_k\}_{k \in \mathbf{N}}$ (one for each problem P_k) that eventually converges to an optimal decision trajectory $\{\boldsymbol{x}^\star(t_k)\}_{k \in \mathbf{N}}$ up to a bounded error. A key difference in the setting proposed here with respect to the previous chapters is that we construct approximate optimizers *concurrently* with the learning of the (unknown) user's function $U(\boldsymbol{x})$. The main operating principles of the algorithm to be explained shortly are, qualitatively, as follows: (i) at time t_k, an approximate optimizer $\hat{\boldsymbol{x}}_k$ is computed based on a partial knowledge of $U(\boldsymbol{x})$; (ii) the optimizer is implemented, and it generates some "feedback" from the user in the form of, for example, $\hat{y}_k = U(\hat{\boldsymbol{x}}_k) + \varepsilon$, where ε is noise; and (iii) $\hat{y}_k$ is collected and utilized to "refine" the knowledge of $U(\boldsymbol{x})$. At time t_{k+1}, the process is then repeated.

5.3 Learning cost functions

As we are going to see shortly, the main difference with standard time-varying optimization is that one needs to learn the cost function concurrently to the implementation of approximate optimizers. This learning procedure requires feedback from the users in terms of noisy function evaluation and it can be done in different ways. I review four below that I have been using in my works [65, 68, 87, 89]. Others, such as deep neural networks could be in principle also used.

We want to estimate an unknown function $U(\boldsymbol{x}), \boldsymbol{x} \in \mathbf{R}^n$ from d noisy feedback points $\hat{y}_i = U(\hat{\boldsymbol{x}}_i) + \varepsilon_i$, $i \in [1, \ldots, d] =: I_d$, with ε_i a zero-mean Gaussian noise term. Let us see how to do it.

5.3.1 Gaussian processes

Standard method. Gaussian processes (GPs) can be used to model the unknown user function $U(\boldsymbol{x})$. Such semi-parametric model is advantageous in the present setting because of (1) the simplicity of the online updates of both mean and covariance; (2) the inherent ability to handle asynchronous and intermittent updates (which is an important feature in user's feedback systems); and, (3) the implicit and smooth handling of measurement (i.e., feedback) noise.

For d data points, GPs can deliver approximation of the unknown function U in terms of mean $\mu_d(\boldsymbol{x})$ and variance $\sigma_d(\boldsymbol{x})$. Different methods then would weigh the former and the latter differently to obtain the sought approximation, e.g., we could set $U(\boldsymbol{x}) \approx \mu_d(\boldsymbol{x}) + \beta_d \sigma_d(\boldsymbol{x})$, with a scalar weight β_d.

If the computation of the kernel matrix is computationally efficient, the computational complexity of GP regression is dominated by the inversion of the kernel matrix, yielding an $O(d^3)$ complexity.

Shape constraints. In many cases, $U(\boldsymbol{x})$ can be assumed to be convex or with another structural property. The convex case is common if we are interested in capturing the dissatisfaction of the users around a working solution, or we are interested only in a general trend, rather than extremely fine-grained characterizations. Therefore one needs to be able to impose such constraints in the approximation procedure to facilitate learning with scarce and noisy data.

Shape-constrained GPs [97] are regression methods that do exactly that. The key intuition is that differentiation is a linear operator, so derivatives of a GP remain a GP [77]. Imposing convexity means impose restrictions on the Hessian of a GP, whose coefficients are GPs. When the functions to be learned are mono-dimensional, then Hessians are also mono-dimensional functions, and their sample-path can be restricted to be non-negative (for convexity), or upper and lower bounded for smooth and strongly convex functions. The exact procedure is described in [97] and in my [68]. Note that one imposes constraints only on a subset of points, both

for computational and theoretical reasons, therefore shape-constrained GPs guarantee that the posterior mean function $\mu_d(\boldsymbol{x})$ has practically[3] the imposed properties [97].

Two remarks are now in order. First, note that the procedure is still computationally viable when the functions are mono-dimensional (i.e., $\boldsymbol{x} \in \mathbf{R}$). The same procedure for $\boldsymbol{x} \in \mathbf{R}^n$, would require describing $n \times n$ coefficients of the Hessian as a GP, and imposing, e.g., semi-definiteness on s points, would amount at a total cost of $O(sn^8)$, and therefore a total computational complexity[4] of $O(d^3 + sn^8)$.

Second, even if the posterior mean function $\mu_d(\boldsymbol{x})$ has the properties that we impose, the variance $\sigma_d(\boldsymbol{x})$ does not, so the approximation for $U(\boldsymbol{x})$ needs to be $U(\boldsymbol{x}) \approx \mu_d(\boldsymbol{x})$, which obliges one to avoid the use of more performing upper confidence bound algorithms, as we will see.

5.3.2 Parametric and nonparametric approaches

Another set of methods that one can use to learn a function from noisy evaluation are parametric and nonparametric methods. In the former, one assumes a specific function form parametrized via a small set of free variables that have to be learned. In the latter, we only assume the functional class. In both cases, in contrast with GP regression, we can impose shape constraints directly in the learning method by our choice of class.

Parametric methods. Parametric methods are very useful when one has prior knowledge of the functional form. As I mentioned before, this is not such a weird thing to assume, especially if users have sharp preferences and we only want to capture the general trend, or behavior around the working point. In this scenario, one can as well assume a convex quadratic function of the form

$$U(\boldsymbol{x}) := \frac{1}{2}\boldsymbol{x}^\mathsf{T} P \boldsymbol{x} + q^\mathsf{T}\boldsymbol{x} + r, \tag{5.3}$$

with parameters (P, q, r) to be learned. For any d feedback points: $\hat{y}_i = U(\hat{\boldsymbol{x}}_i) + \varepsilon_i$, $i \in I_d$, we can set up a constrained least-squares (LS) method as

$$\underset{P,q,r}{\text{minimize}} \sum_{i=1}^{n} \left\| \frac{1}{2}\hat{\boldsymbol{x}}_i^\mathsf{T} P \hat{\boldsymbol{x}}_i + q^\mathsf{T}\hat{\boldsymbol{x}}_i + r - \hat{y}_i \right\|^2 \quad \text{subject to } P \in \mathcal{P}, \tag{5.4}$$

where the set $\mathcal{P}$ captures smooth and strong convexity assumptions, or combinations thereof (e.g., $\mathcal{P}$ is the positive semidefinite cone). If then $\mathcal{P}$ is convex, then the whole problem is convex.

In [65], we explore a slightly different LS method, removing the constraint and using recursive least-squares (RLS) instead of least-squares, which has a better computational complexity, even though the qualitative idea is the same. For RLS the computational complexity is $O(n^4)$ (independent of the number of data points, since recursive).

And finally, note that *any* linearly parametrized convex functions could lead to convex estimation problems.

Nonparametric methods. If imposing a structure is too much to ask, one can also just impose the functional class, like convex or smooth strongly convex. Doing that would require solving the infinite dimensional problem

$$\underset{\varphi_d}{\text{minimize}} \sum_{i=1}^{d} \|\varphi_d(\hat{\boldsymbol{x}}_i) - \hat{y}_i\|^2 \quad \text{subject to } \varphi \in \mathcal{F}, \tag{5.5}$$

for a specific functional class $\mathcal{F}$ (note here that φ_d is a function). While this sounds hard to do, it is possible to show that this is in fact a finite-dimensional convex problem at least if $\mathcal{F}$ is the convex class [15]. While the case in which $\mathcal{F}$ convex is rather acquired, the case in which we want to impose strong convexity and smoothness is less direct, and I have explored it in [87]. The results there are based on Adrien Taylor's work [84].

The resulting algorithm has a computational complexity that grows at least as $O(d^3 n^3)$.

[3] With a limited number of enforcing points s, the mean has the functional properties we require up to a very (practically negligible) small error.

[4] This can be derived by considering that in the constraint-enforcing procedure one needs to solve n^2 convex problems featuring semi-definite constraints of dimension n^2.

Table 5.1. *Comparison of estimation methods and their asymptotic learning bounds. Cvx-R stands for convex regression. The parameter r is $\frac{2}{n+4}$ for $n=1,2,3$, $\frac{1}{4}\sqrt{\log(d)}$ for $n=4$, and $\frac{1}{n}$ for $n \geqslant 5$.*

Method	Shape const.	Comp. compl.	$U(\boldsymbol{x})$	Asympt. learning bound under extra assumptions
Standard GP	No	$O(d^3)$	$\mu_d(\boldsymbol{x}) + \beta_d\sigma_d(\boldsymbol{x})$	$O(1/\sqrt{d})$, $O(1/d)$ [77]
Shape-const. GP	Yes	$O(d^3 + sn^8)$	$\mu_d(\boldsymbol{x})$	$O(1/\sqrt{d})$, $O(1/d)$ [68]
RLS	Yes	$O(n^4)$	Eq. (5.3)	$O(1/\sqrt{d})$ [65]
Cvx-R	Yes	$O(n^3d^3)$	$\hat{\varphi}_d(\boldsymbol{x})$, see [87]	$O(1/d^r)$ [58]

Comparison and asymptotic learning bounds

In Table 5.1, I report a comparison of the estimation methods we have presented, together with their asymptotic learning bounds. The latter ones describe how the estimated function converges to true one as the amount of data point increases. The reported "typical" asymptotic learning bounds are subject to more restrictive assumptions on functional classes and data richness and they are shown indicatively to gauge the relative learning "speed" among the different methods (sharper ones can also be found in the literature depending on the set of assumptions and functional classes). We can see how standard GP and recursive least-squares provide a good trade-off between computational complexity and asymptotic learning bounds. However, the other methods may be competitive, when one has access to scarce data and the "asymptotic regime" never kicks in.

5.4 Optimism in the face of uncertainty

Having described methods to learn the unknown users' dissatisfaction, we are now ready to derive an algorithm that finds and track the optimizers of Problem (5.1).

Let $\mathcal{L}$ be a method that learns a cost function based on d feedback points, returning $\hat{U}_d(\boldsymbol{x})$. Consider then function $\hat{f}_{kd}(\boldsymbol{x}) := V(\boldsymbol{x};t_k) + \hat{U}_d(\boldsymbol{x})$ which approximate $f(\boldsymbol{x};t_k)$ at time t_k for d feedback points. The optimizer of $f(\boldsymbol{x};t_k)$ is still $\boldsymbol{x}^\star(t_k)$, and we let $\hat{\boldsymbol{x}}^\star_{kd}$ be the optimizer of $\hat{f}_{kd}(\boldsymbol{x})$. Here I assume both of them to be unique (but we could generalize this).

Let now $\mathcal{M}'$ a method that when applied to $\hat{\boldsymbol{x}}_{k-1}$ for function $\hat{f}_{kd}(\boldsymbol{x})$ is functionally linearly converging, in the sense that it delivers a $\hat{\boldsymbol{x}}_k$ such that

$$\hat{f}_{kd}(\hat{\boldsymbol{x}}_k) - \hat{f}_{kd}(\boldsymbol{x}^\star_{kd}) \leqslant \eta(\hat{f}_{kd}(\hat{\boldsymbol{x}}_{k-1}) - \hat{f}_{kd}(\boldsymbol{x}^\star_{kd})), \qquad \eta \in (0,1). \tag{5.6}$$

Method $\mathcal{M}'$ is a twist with respect to Chapters 2-3 to accommodate more functional classes, e.g., invex functions (note that, smooth strongly convex f's enjoy both methods $\mathcal{M}$ and $\mathcal{M}'$).

We are now ready for our Learning and Optimizing meta-algorithm, presented below, in Algorithm 5.1. As we see, the algorithm proceeds in rounds, where we estimate the user's function based on d feedback points, and we sample the cost at time t_k. When the decision at time t_k is computed, it is applied and feedback is sought. If the latter is given, then the counter d is incremented and at the next step the estimated $\hat{U}_d$ will be different. Otherwise, d is kept the same, as well as $\hat{U}_d$. This is also captured in Figure 5.1. From the figure and the algorithm, one can really see the existence and interplay of three time scales: the learning scale (how fast we are learning U), the time-varying scale (how fast V is changing), and the optimization scale (how fast we are converging to the optimizer's trajectory). These scales will play a major role in the performance analysis next.

5.4.1 Assumptions and convergence

It is remarkable that one can say anything at all for a meta-algorithm like Algorithm 5.1. I show here that under some blanket assumptions, we can describe the dynamic regret (see definition in Chapter 2). We will then have a closer look for a specific learning method. We start by assuming

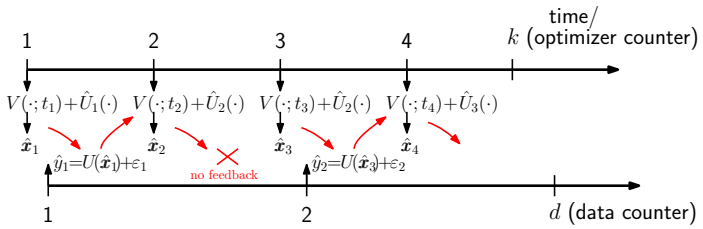

Figure 5.1. *Graphical explanation of Algorithm 5.1. At each t_k, we compute an approximate $\hat{\boldsymbol{x}}_k$ based on the engineering function $V(\cdot;t_k)$ and the user's function $\hat{U}_d$. Based on $\hat{\boldsymbol{x}}_k$, we ask feedback to the user, if given: we update $\hat{U}_d$ and increase the data counter, otherwise we keep $\hat{U}_d$ as is.*

that $d = k$, meaning that we receive feedback at every time step. This is the worst-case scenario, since $\hat{U}_d$ changes at every time step (see [89]). Consistently, we call $\hat{f}_k(\boldsymbol{x}) = f(\boldsymbol{x};t_k) + U_k(\boldsymbol{x})$.

Algorithm 5.1 Learning and Optimizing meta-algorithm

Input: *Initial point: $\hat{\boldsymbol{x}}_0$, a linearly converging method $\mathcal{M}'$, Number of correction steps C, sampling time h, learning method $\mathcal{L}$, $d \geqslant 0$ feedback points*
Output: *A sequence $(\hat{\boldsymbol{x}}_k)_{k\in\mathbf{N}}$, an estimation for $U(\boldsymbol{x})$*
1: ***for*** $k \in \mathbf{N}, k \geqslant 1$ ***do***
2: *Sample $V(\cdot;t_k)$*
3: *Estimate $\hat{U}_d(\boldsymbol{x})$ via learning method $\mathcal{L}$*
4: *Correction step: apply $\mathcal{M}'$ method for $\hat{f}_{kd}$, C times to $\hat{\boldsymbol{x}}_{k-1}$ to deliver $\hat{\boldsymbol{x}}_k$*
5: *Feedback step: ask feedback to users on $\hat{\boldsymbol{x}}_k$. If given, then $d \leftarrow d+1$.*
6: ***end for***

Assumption 5.1 *We have a method $\mathcal{M}'$ that is functionally linearly convergent (see Equation (5.6)) if $\hat{f}_k$ is in some functional class $\mathcal{F}$ (e.g., convex, invex), and it delivers bounded $\hat{\boldsymbol{x}}_k$ and $\hat{f}_k(\hat{\boldsymbol{x}}_k)$, otherwise.*

Assumption 5.2 *We have a learning method $\mathcal{L}$ for which there exists a finite number of points $\bar{k}$, such that $\hat{U}_k(\boldsymbol{x})$, $k \geqslant \bar{k}$, belongs to the functional class of $U(\boldsymbol{x})$ with high probability.*

For method $\mathcal{L}$, we define the learning length as

$$c_U := \sum_{k=1}^{T} \sup_{\boldsymbol{x}} |\hat{U}_k(\boldsymbol{x}) - U(\boldsymbol{x})| \leqslant \sum_{k=1}^{T} e_k. \tag{5.7}$$

Note that for the learning methods presented in the previous section, one can carefully leverage their asymptotic learning bounds under extra technical assumptions to derive bounds on c_U. In particular, all the presented methods enjoy a sub-linear c_U, under (un)-said assumptions.

Assumption 5.3 *Time variations in the costs are bounded for all k and $\boldsymbol{x}$, meaning*

$$|V(\boldsymbol{x},t_k) - V(\boldsymbol{x};t_{k-1})| \leqslant \Delta_k < \infty. \tag{5.8}$$

We define accordingly the functional path length as

$$c_V := \sum_{k=1}^{T} \sup_{\boldsymbol{x}} |V(\boldsymbol{x},t_k) - V(\boldsymbol{x};t_{k-1})| \leqslant \sum_{k=1}^{T} \Delta_k, \tag{5.9}$$

and we can also derive, $\sup_{\boldsymbol{x}} |V(\boldsymbol{x},t_k) + U_k(\boldsymbol{x}) - V(\boldsymbol{x};t_{k-1}) - U_{k-1}(\boldsymbol{x})| \leqslant \Delta_k + e_k + e_{k-1}$.

Assumptions are quite reasonable and they can be tightened in some special cases. Then the following general meta-result is in place

Theorem 5.1 (Informal) *Consider Problem (5.1), for which cost f belongs to functional class $\mathcal{F}$, uniformly in time. Consider Algorithm 5.1 and Assumptions 5.1 through 5.3. Then, with high probability, the dynamic regret goes as follows*

$$R_T((\hat{\boldsymbol{x}}_k)_{k=0}^T) = \sum_{k=0}^{T} f(\hat{\boldsymbol{x}}_k,t_k) - f(\boldsymbol{x}^\star(t_k);t_k) \leqslant O(1) + O(c_V) + O(c_U). \tag{5.10}$$

The result of Theorem 5.1 unpacks the three time scales we discussed before thanks to the linear converging method $\mathcal{M}'$. In particular, $O(1)$ is the initialization error, which is constant for the convergence rate of the method $\mathcal{M}'$. The term $O(c_V)$ is due to the time-varying time scale, while the term $O(c_U)$ is due to the learning time scale. If any of the latter two is sub-linear in T, then it will vanish in the average regret. If both are sub-linear in T, we obtain a no-regret result. In a time-varying scenario, however, we expect at least $O(c_V)$ to grow linearly in time.

We look now at the proof of Theorem 5.1 since quite informative.

Proof (Sketch) *We work with high probability. Write the regret dividing the part in which we do not have the functional properties we require for convergence of method $\mathcal{M}'$, and the part we do. For Assumption 5.1, the first part is finite and therefore $O(1)$. For Assumption 5.2, the second part can be bounded extracting $\hat{f}_k$. Therefore*

$$\sum_{k=0}^{T} f(\hat{\boldsymbol{x}}_k, t_k) - f(\boldsymbol{x}^\star(t_k); t_k) \leqslant \sum_{k=0}^{\bar{k}-1} f(\hat{\boldsymbol{x}}_k, t_k) - f(\boldsymbol{x}^\star(t_k); t_k) + \sum_{k=\bar{k}}^{T} f(\hat{\boldsymbol{x}}_k, t_k) - f(\boldsymbol{x}^\star(t_k); t_k) =$$
$$O(1) + \sum_{k=\bar{k}}^{T} \left[\hat{f}_k(\hat{\boldsymbol{x}}_k) - \hat{f}_k(\boldsymbol{x}^\star(t_k)) + 2e_k \right] = O(1) + O(c_U) + \sum_{k=\bar{k}}^{T} \hat{f}_k(\hat{\boldsymbol{x}}_k) - \hat{f}_k(\boldsymbol{x}^\star(t_k)).$$

Now use optimality of $\hat{\boldsymbol{x}}_k^\star$, and therefore $\hat{f}_k(\boldsymbol{x}^\star(t_k)) \geqslant \hat{f}_k(\hat{\boldsymbol{x}}_k^\star)$ and Assumption 5.2

$$\hat{f}_k(\hat{\boldsymbol{x}}_k) - \hat{f}_k(\boldsymbol{x}^\star(t_k)) \leqslant \hat{f}_k(\hat{\boldsymbol{x}}_k) - \hat{f}_k(\hat{\boldsymbol{x}}_k^\star) \leqslant \eta(\hat{f}_k(\hat{\boldsymbol{x}}_{k-1}) - \hat{f}_k(\hat{\boldsymbol{x}}_k^\star)).$$

By using Equations (5.7)-(5.9), we then have

$$\begin{aligned} \eta(\hat{f}_k(\hat{\boldsymbol{x}}_{k-1}) - \hat{f}_k(\hat{\boldsymbol{x}}_k^\star)) &\leqslant \eta[\hat{f}_{k-1}(\hat{\boldsymbol{x}}_{k-1}) - \hat{f}_{k-1}(\hat{\boldsymbol{x}}_{k-1}^\star)] + 2\eta(e_k + e_{k-1} + \Delta_k) \\ &\leqslant \eta^{k-\bar{k}+1}[\hat{f}_{\bar{k}-1}(\hat{\boldsymbol{x}}_{\bar{k}-1}) - \hat{f}_{\bar{k}-1}(\hat{\boldsymbol{x}}_{\bar{k}-1}^\star)] + 2\sum_{i=\bar{k}}^{k} \eta^{k-i+1}(e_i + e_{i-1} + \Delta_i). \end{aligned}$$

And finally, since

$$\sum_{k=\bar{k}}^{T} \eta^{k-\bar{k}+1}[\hat{f}_{\bar{k}-1}(\hat{\boldsymbol{x}}_{\bar{k}-1}) - \hat{f}_{\bar{k}-1}(\hat{\boldsymbol{x}}_{\bar{k}-1}^\star)] = O(1), \qquad \sum_{k=\bar{k}}^{T}\sum_{i=\bar{k}}^{k} \eta^{k-i+1}(e_i + e_{i-1} + \Delta_i) = O(c_U) + O(c_V).$$

the thesis follows[5]. ♦

Theorem 5.1 and its assumptions are quite general, and any of the learning methods that we have presented above (e.g., convex regression) **could** fit the bill and be applied with their own sub-linear c_U. **However**, the extra technical assumptions needed for the asymptotic learning bounds can be tricky to enforce. Typically one needs data richness and a good trade-off between exploration and exploitation for those asymptotic learning bounds to hold. In addition (and because of it), using the mean for the estimation of $U(\boldsymbol{x})$ is not a good strategy at all in static settings (see [82]). The latter fact is not so critical in time-varying settings, where variations in function V play the role of exploration (even if this is more empirical evidence than hard truth).

I look next at a specific technique (the upper confidence bound), that can *enforce* a sub-linear c_U by sampling $U(\boldsymbol{x})$ in a special way (instead of assuming, e.g., data richness), which makes $O(c_V)$ the leading term of the regret.

5.4.2 An upper confidence bound algorithm

We look now at a special case of learning, GPs combined with an upper confidence bound algorithm[6] to determine $\hat{U}_k(\boldsymbol{x}) = \mu_k(\boldsymbol{x}) - \beta_k(\boldsymbol{x})\sigma_k(\boldsymbol{x})$ for growing positive β_k's, so to trade off low mean and high variance, in an exploration-exploitation trade-off.

The assumptions are the same as before, with $\mathcal{L}$ being a GP regression with square exponential kernel, the class $\mathcal{F}$ being the set of invex functions, for which a projected gradient method

[5] The last inequality follows since: $\sum_{k=\bar{k}}^{T}\sum_{i=\bar{k}}^{k} \eta^{k-i+1} p_i = \sum_{k=\bar{k}}^{T}\sum_{\ell=1}^{k-\bar{k}+1} \eta^{\ell} p_{k-\ell+1} = \sum_{k=\bar{k}}^{T} \eta^k \sum_{\ell=\bar{k}-1}^{T-k} p_{\ell+1} \leqslant \sum_{k=1}^{T} \eta^k O(\sum_{k=1}^{T} p_k) = O(\sum_{k=1}^{T} p_k)$.

[6] Technically since we are dealing with minimization, I should call it a lower confidence bound algorithm.

is linearly converging, and X being a compact set (for the upper confidence bound to work). Invex functions extend the convexity for sharp users' preferences, but one has to pay a little attention to them, since the invex class is not closed under addition. Then, for Algorithm 5.1, with the above setting, we have the following result ($O^*(\cdot)$ indicates $O(\cdot)$ up to poly-logarithmic terms).

Theorem 5.2 (Regret bound) *Let Assumptions 5.1–5.3 hold. Set the parameter β_k as*

$$\beta_k = 2\log(k^2 C_1) + 2k\log(nk^2 C_2\sqrt{\log(nC_3)}),$$

for $k \geqslant 1$, and where C_1, C_2, C_3 are tunable parameters (see [89]). Running Algorithm 5.1 with β_k for a sample U of a GP with mean function zero and square exponential kernel, and projected gradient method for $\mathcal{M}'$, we obtain a regret bound of $O(1) + O^*(\sqrt{nT(\log T)^{n+1}}) + O(c_V)$ with high probability.

Theorem 5.2 yields the promised result, where $O(c_U) = O^*(\sqrt{nT(\log T)^{n+1}})$, which is sub-linear in T as desired. Similar results are presented in [65,68] with other technical assumptions.

5.5 An example

Example 5.1 (Vehicle Platooning) *In [89], we consider a vehicle platooning problem, whereby vehicles are controlled to maintain a given reference distance between each other and follow a leader vehicle. We look at the problem of deciding which are the best inter-vehicle distances such that they are as close as possible to some desired values that are dictated by road, aerodynamics considerations, and platoon's speed, while being* comfortable *to the car riders. This problem formulation fits Problem (5.2) and we solve it accordingly by modeling the user satisfaction as a standard GP and applying an approximate upper confidence bound algorithm (AGP-UCB). What it is interesting here is how the approach compares to other possible strategies. In particular, we discuss* (i) *a projected gradient method based on a synthetic one-fits-all model of the user's satisfaction function;* (ii) *a projected zero-order method based on the user's feedback at the current and previous point(s) to estimate the gradient of the user's satisfaction function.*

The synthetic model for the user's satisfaction functions is taken as equal for both vehicles (to exactly match the parametric model which the GP samples mimic, with a small parameter inaccuracy). We remark that it is in general complicate to obtain good one-fits-all models for human preferences, and even small inaccuracies can cause bias and loss of perceived fairness. The zero-order method uses the user's feedback to obtain an estimate of the gradient of the user's satisfaction function. We implement both a two-point estimate, where we use the current feedback and the previous one, as well as a four-point estimate, where we use the current feedback and three previous ones. We remark that the implementation of zero-order methods is in general hard with user's feedback, since the feedback may be very noisy and intermittent. While our GP-based method does work seamlessly with intermittent noisy feedback, a zero-order method would not. In Figure 5.2, we see how our algorithm outperforms the others in terms of average regret. This is reasonable, since the synthetic model has no feedback to learn the true user's satisfaction functions, and the zero-order model estimates a noisy gradient (and, likewise, it does not learn the user's functions). More results and discussions are then presented in [89].

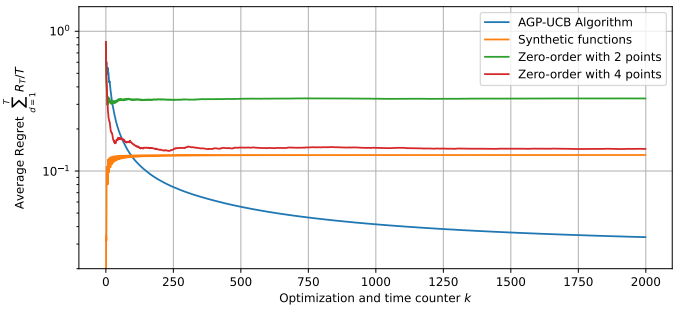

Figure 5.2. *Comparison of the different algorithms in terms of average regret.*

Chapter 6

Perspectives

We will never again understand nature as well as Greek philosophers did.

John R. Pierce

... the joke is that
whenever you find a missing link you've created two more missing links.

Sean Carroll talking with Niel Shubin about evolution

6.1 A look ahead

The thesis has depicted the vibrant research area of optimization and learning for cyber-physical and social systems. If I look ahead to my own research interests, my general goal is to shape this area, where humans are an integral part of a complex, ever-changing, environment, which nevertheless requires optimization algorithms to be able to give hard guarantees on safety, reliability, and performance. The ultimate aim is to change how humans experience technology and interact with it.

In the short to medium term, my specific scientific objectives can be listed as follows.

Data-driven optimization

As we have discussed in Chapter 3, there are different ways for one to do prediction and correction of time-varying problems. If we look the optimization problem $\min_{\boldsymbol{x}\in X} f(\boldsymbol{x};t)$, with a specific dependence on a parameter $b(t)$, i.e., $\mathsf{P}(\boldsymbol{b}(t)) : \min_{\boldsymbol{x}\in X} f(\boldsymbol{x};\boldsymbol{b}(t))$, we have offered methods that require the knowledge of $f(\cdot;\boldsymbol{b}(t))$ at some time steps and therefore the knowledge of $\boldsymbol{b}(t)$. In the case $\boldsymbol{b}(t)$ is noisy, we converge to the noisy optimizers of $\mathsf{P}(\boldsymbol{b}(t))$.

Prediction is then typically based on the dynamical system that captures the evolving optimality conditions, while correction is based on sub-optimality measures, such as the gradient.

With the growing literature in learning, it is quite natural to ask oneself how to upgrade current prediction-correction methods by learning the underlying dynamical system from data, having the access to some prior on $\boldsymbol{b}(t)$, and modeling choices (such as a spatiotemporal Gaussian process for f). Priors on $\boldsymbol{b}(t)$ do not have to be very specific: it is often the case that $\boldsymbol{b}(t)$ is a quasi-periodic signal, such as requests in ridesharing systems, or irradiance on photovoltaic panels, so modeling $\boldsymbol{b}(t)$ with a quasi-periodic kernel may be already enough.

This could also lead to solving the smoothed problem

$$\underset{\boldsymbol{x}\in X}{\text{minimize}}\ \mathbf{E}_{\boldsymbol{b}(t)\sim\mathcal{B}(t)}[f(\boldsymbol{x};\boldsymbol{b}(t))], \tag{6.1}$$

where $\mathbf{E}_{\boldsymbol{b}(t)\sim\mathcal{B}(t)}[\cdot]$ represents the expectation with respect to the noisy variable $\boldsymbol{b}$ which is drawn from a time-varying distribution $\mathcal{B}(t)$.

Research in this direction would entangle once more dynamical systems, optimization, and estimation theory.

Marrying traditional statistical learning with the need of optimization

When introducing a learning component for a part of the optimization problem, for example the cost or the constraints, one has to make sure the learned version has all the functional properties of the original one. We have discussed this in Chapter 5. Within data-driven optimization approaches as above and for learning users' preferences, this is key.

An interesting research path is then to further expand shape-constrained learning for the need of optimization: how to impose structural constraints (such as convexity, sharpness, monotonicity, etc.) with minimal computational overhead and good asymptotic consistency bounds? How to design good spatiotemporal kernels? How to learn the best regularization of a given problem, or the best algorithm for it?

This research area is growing and (part of it) goes under the name of learning to optimize, see for example [18] and my recent [7].

Time-varying in combinatorial domains

In Chapter 4, we have looked at combinatorial problems. These problems are tricky in time-varying domains since Lipschitz continuity of the solution trajectory cannot be expected. Focusing on the computational aspects alone, one can, however, still derive fast algorithms that optimize appropriate simplifications of the original problem and still retain good streaming solutions.

The next big step is to generalize what we have seen for the ridesharing example to a larger class of combinatorial problems. Here, one could plan to look at recent work in using neural networks to learn solutions of combinatorial problems and their sensitivity as parameters change (see for instance [11]). The step from that to time-varying combinatorial problems could be very useful in practice.

Given users a choice

In Chapter 5, we have talked about users' feedback and learning from their preferences to steer the decision-making process. There, we have discussed the idea to impose a solution to the users and measure their reaction to it.

With the growing interest in weak control [41], one can better ask whether we could propose a set of possible solutions and learn from it. This would give users a choice and make decisions more democratic.

Let us look at an example. If we were to decide when to plug an electrical vehicle under various constraints and costs, we may ask the user to plug it at exactly 08h37m36s PM, while giving a set, we could offer between 08h and 09h PM. The latter is more natural, and it does also make sense: at the end the actual optimizer is the child of a cost function that in more or less arbitrarily constructed by system engineers, and we could be slightly flexible about (as long as the constraints are satisfied).

This research direction would upgrade traditional optimization to set-based solutions and their convergence; as a matter of fact, with the users, we are exploring an unknown (possibly time-varying) Pareto front and eventually converging to a point on it.

6.2 Epilogue: a personal retrospective

This thesis has looked at my work in the past ten or so years, working in four different countries, both in academia and in industry. If I even look back a little further, when I started my PhD at Delft University of Technology in 2008, what the scientific community has achieved in the digital space is mind-blowing. In 2008, we did not really have smart phones, and we were still using mp3 players to listen to our favorite music. Convex optimization was starting to conquer

almost all application endeavors, and its distributed version, together with distributed control, were at their infancy. AI and machine learning were there already, but much more in the fringes of the scientific arena, and if a colleague had told you they were using artificial neural networks for control or signal processing purposes, you would have thought they were out of their minds. But, on the other hand, there was a lack of usable data and plenty to do in the convex realm and no bother looking at something more complicated.

In 14 years, things have changed in an almost unrecognizable way. For starters, I am eating *pains aux raisins* instead of *frikandels*. But also many areas have reached their peak and left the spotlight to others. Convex optimization is now quite an established technology. With the explosion of data available paired with ever increasing computational power, machine learning and AI have taken the central stage in many application domains. More strikingly, with the advent of smart phones and widespread use of open-source coding, the AI community is making learning tools more democratic and accessible. This blurs the line between scientific community, developers, and general public, ultimately making the adoption of new technology easier and faster than ever before. Companies from many different domains, ranging from health to agriculture, are also very agile in adopting and extremely keen in trying these new AI tools.

Distributed algorithms have yielded to federated learning; this primarily pushed by the current widespread adoption of cloud-based technologies and current trends in the internet of things. This also makes training of personalized models easier and more transferrable, almost as a service.

All these developments bring the general public more on the spotlight, them being able to use quite the latest algorithms (hidden in their phones or on the cloud), them being "used" as a source of data, or them being "recommended" and profiled. This makes me think that as never before, we have, as scientists, the responsibility and opportunity to incorporate the social part in our algorithm design. The algorithms that we design today, could be adopted in a month in a company or start-up and in two they could be on somebody's phone. This should also change how we educate the future generation of scientists, where ethics must play a major role.

I think as "digital" scientists, we can now make a real difference in the welfare of people, and we should take the lead.

What's going to happen in 10-14 years? It is obviously hard to say. I think humanity has a lot of challenges ahead of them, ranging from protecting democracy, fighting diseases, to climate change. "Digital" scientists have the right mindset to propose sound and mathematically-grounded solutions. Learning and optimization will be key players. Scientifically, other theories will become technology, other will take their place in the sun for a moment. As for me, even just being a part of it will suffice.

About the author. Andrea Simonetto earned his PhD in systems and control at Delft University of Technology, the Netherlands in 2012. He then spent $3 + 1$ years as a postdoctoral researcher, first in the signal processing group at Delft University of Technology, and then at the applied mathematics group at Université catholique Louvain, Louvain-la-Neuve, Belgium. From February 2017 to August 2021, he was a research staff member at IBM Research Europe, in Dublin, Ireland. Since September 2021, he is a professor of optimization at ENSTA Paris, Institute Polytechnique de Paris, France.

Bibliography

[1] A. Akhriev, J. Marececk, and **A. Simonetto**. Pursuit of Low-Rank Models of Time-Varying Matrices Robust to Sparse and Measurement Noise. In *Proceedings of AAAI*, 2020.

[2] E. L. Allgower and K. Georg. *Numerical Continuation Methods: An Introduction*. Springer-Verlag, 1990.

[3] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand High-capacity Ride-sharing via Dynamic Trip-Vehicle Assignment. *PNAS*, 114(3):462 – 467, 2017.

[4] M. E. Baran and F. F. Wu. Network reconfiguration in distribution systems for loss reduction and load balancing. *IEEE Transactions on Power Delivery*, 4(2):1401–1407, Apr. 1989.

[5] N. Bastianello, **A. Simonetto**, and R. Carli. Prediction-Correction Splittings for Nonsmooth Time-Varying Optimization. In *Proceedings of the European Control Conference*, Napoli, Italia, June 2019.

[6] N. Bastianello, **A. Simonetto**, and R. Carli. Primal and dual prediction-correction methods for time-varying convex optimization. *arXiv:2004.11709*, 2020.

[7] N. Bastianello, **A. Simonetto**, and E. Dall'Anese. OpReg-Boost: Learning to accelerate online algorithms with operator regression. In *Proceedings of Learning for Dynamics & Control Conference (to appear)*, 2022.

[8] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. CMS Books in Mathematics. Springer-Verlag, 2011.

[9] F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause. Safe learning of regions of attraction for uncertain, nonlinear systems with Gaussian processes. In *Proceedings of the 55th Conference on Decision and Control*, pages 4661 – 4666, December 2016.

[10] A. Bernstein, E. Dall'Anese, and **A. Simonetto**. Online primal-dual methods with measurement feedback for time-varying convex optimization. *IEEE Transactions on Signal Processing*, 67(8):1978–1991, April 2019.

[11] D. Bertsimas and B. Stellato. Online Mixed-Integer Optimization in Milliseconds. *arxiv: 1907.02206*, 2019.

[12] O. Besbes, Y. Gur, and A. Zeevi. Non-stationary Stochastic Optimization. *Operations research*, 63(5):1227 – 1244, 2015.

[13] S. Bolognani and F. Dörfler. Fast power system analysis via implicit linearization of the power flow manifold. In *2015 53rd Annual Allerton Conf. on Communication, Control, and Computing*, pages 402–409, 2015.

[14] S. Bolognani and S. Zampieri. On the existence and linear approximation of the power flow solution in power distribution networks. *IEEE Transactions on Power Systems*, 31(1):163–172, 2016.

[15] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[16] A. S. Charles, A. Balavoine, and C. J. Rozell. Dynamic filtering of time-varying sparse signals via ℓ_1 minimization. *IEEE Transactions on Signal Processing*, 64, 2016.

[17] P. Chatupromwong and A. Yokoyama. Optimization of charging sequence of plug-in electric vehicles in smart grid considering user's satisfaction. In *Proceedings of the IEEE International Conference on Power System Technology*, pages 1 – 6, October 2012.

[18] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin. Learning to optimize: A primer and a benchmark. *arXiv:2103.12828*, 2021.

[19] C.-K. Chiang, T. Yang, C.-J. Lee, M. Mahdavi, C.-J. Lu, R Jin, and S. Zhu. Online Optimization with Gradual Variations. In *Conference on Learning Theory*, 2012.

[20] W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 137 – 144, Bonn, Germany, August 2005.

[21] M. Colombino, J. W. Simpson-Porco, and A. Bernstein. Towards robustness guarantees for feedback-based optimization. In *IEEE 58th Conference on Decision and Control (CDC)*, pages 6207–6214, 2019.

[22] E. Dall'Anese, S. Guggilam, **A. Simonetto**, Yu C. Chen, and S. V. Dhople. Optimal Regulation of Virtual Power Plants. *IEEE Transactions on Power Systems*, 33(2):1868 – 1881, 2018.

[23] E. Dall'Anese and **A. Simonetto**. Optimal Power Flow Pursuit. *IEEE Transactions on Smart Grid*, 9(2):942 – 952, 2018.

[24] E. Dall'Anese, **A. Simonetto**, S. Becker, and L. Madden. Optimization and Learning with Information Streams: Time-varying Algorithms and Applications. *Signal Processing Magazine*, 37(3):71 – 83, May 2020.

[25] M.P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408 – 423, 2015.

[26] A. L. Dontchev, M. I. Krastanov, R. T. Rockafellar, and V. M. Veliov. An Euler-Newton Continuation method for Tracking Solution Trajectories of Parametric Variational Inequalities. *SIAM Journal of Control and Optimization*, 51(51):1823 – 1840, 2013.

[27] A. L. Dontchev and R. T. Rockafellar. *Implicit Functions and Solution Mappings*. Springer, 2009.

[28] F. Dressler. Cyber physical social systems: Towards deeply integrated hybridized systems. In *Intern. Conf. on Computing, Networking and Communications*, 2018.

[29] E. Eser, J. Monteil, and **A. Simonetto**. On the Tracking of Dynamical Optimal Meeting Points. In *Proceedings of the 15th IFAC Symposium on Control in Trasportation Systems*, Savona, Italy, June 2018.

[30] M. Fazlyab, S. Paternain, V.M. Preciado, and A. Ribeiro. Prediction-Correction Interior-Point Method for Time-Varying Convex Optimization. *IEEE Transactions on Automatic Control*, 63(7), 2018.

[31] M. Gendreau, G. Ghiani, and E. Guerriero. Time-dependent routing problems: A review. *Computers & Op.s Research*, 64, 2015.

[32] M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar. Bayesian Reinforcement Learning: A Survey. *Foundations and Trends(R) in Machine Learning*, 8(5-6):359 – 492, 2015.

[33] J. Guddat and F. Guerra Vazquez and H. T. Jongen. *Parametric Optimization: Singularities, Pathfollowing and Jumps*. John Wiley & Sons, Chichester, UK, 1990.

[34] D. Guo, X. Lin, Z. Su, S. Sun, and Z. Huang. Design and analysis of two discrete-time ZD algorithms for time-varying nonlinear minimization. *Numerical Algorithms*, 77(1):23 – 36, 2018.

[35] E. C. Hall and R. M. Willett. Online convex optimization in dynamic environments. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):647–662, 2015.

[36] A. Hauswirth, S. Bolognani, G. Hug, and F. Dörfler. Timescale separation in autonomous optimization. *IEEE Transactions on Automatic Control*, 66(2):611–624, 2021.

[37] A. Hauswirth, A. Zanardi, S. Bolognani, Florian Dörfler, and G. Hug. Online optimization in closed loop on the power flow manifold. In *Proceedings of the IEEE PowerTech conference*, Manchester, UK, June 2017.

[38] E. Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[39] N. Houlsby, J.M. Hernández-Lobato, F. Huszár, and Z. Ghahramani. Collaborative Gaussian processes for preference learning. *Advances in Neural Information Processing Systems*, 3:2096 – 2104, January 2012.

[40] J.-H. Hours and C. N. Jones. A Parametric Non-Convex Decomposition Algorithm for Real-Time and Distributed NMPC. *IEEE Transactions on Automatic Control*, 61(2):287 – 302, 2016.

[41] M. Inoue and V. Gupta. "Weak" Control for Human-in-the-Loop Systems. *IEEE Control Systems Letters*, 3(2):440–445, 2019.

[42] A. Jadbabaie, A. Rakhlin, S. Shahrampour, and K. Sridharan. Online Optimization: Competing with Dynamic Comparators. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, PMLR*, number 38, pages 398 – 406, 2015.

[43] F. Y. Jakubiec and A. Ribeiro. D-MAP: Distributed Maximum a Posteriori Probability Estimation of Dynamic Systems. *IEEE Transactions on Signal Processing*, 61(2):450 – 466, 2013.

[44] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded Online Optimization for Model Predictive Control at Megahertz Rates. *IEEE Transactions on Automatic Control*, 59(12):3238 – 3251, 2014.

[45] L. Jin and Y. Zhang. Continuous and discrete Zhang dynamics for real-time varying nonlinear optimization. *Numerical Algorithms*, 73(1):115 – 140, 2016.

[46] D. Kahneman and A. Tversky. Prospect Theory: An Analysis of Decision under Risk. *Econometrica*, 47(2):263 – 291, 1979.

[47] J. Koshal, A. Nedić, and U. Y. Shanbhag. Multiuser Optimization: Distributed Algorithms and Error Analysis. *SIAM Journal on Optimization*, 21(3):1046 – 1081, 2011.

[48] J. Lavaei and S.H. Low. Zero duality gap in optimal power flow problem. *IEEE Transactions on Power Systems*, 27(1):92–107, 2012.

[49] S. Levine, Z. Popovic, and V. Koltun. Nonlinear Inverse Reinforcement Learning with Gaussian Processes. In *Advances in Neural Information Processing Systems 24*, pages 19 – 27, 2011.

[50] J. Li, M. Mao, F. Uhlig, and Y. Zhang. Z-type neural-dynamics for time-varying nonlinear optimization under a linear equality constraint with robot application. *Journal of Computational and Applied Mathematics*, 327(1):155 – 166, 2018.

[51] Y. Li, G. Qu, and N. Li. Using Predictions in Online Optimization with Switching Costs: A Fast Algorithm and A Fundamental Limit. In *Proceedings of the American Control Conference*, 2018.

[52] B. Liao, Y. Zhang, and L. Jin. Taylor $O(h^3)$ Discretization of ZNN Models for Dynamic Equality-Constrained Quadratic Programming With Application to Manipulators. *IEEE Transactions on Neural Networks and Learning Systems*, 27(2):225 – 237, 2016.

[53] H. J. Liu, W. Shi, and H. Zhu. Decentralized Dynamic Optimization for Power Network Voltage Control. *IEEE Transactions on Signal and Information Processing over Networks*, 3(3):568 – 579, 2017.

[54] J. Liu, J. Marecek, **A. Simonetto**, and M. Takac. A Coordinate-Descent Algorithm for Tracking Solutions in Time-Varying Optimal Power Flows. In *Proceedings of the XX Power Systems Computation Conference*, Dublin, Ireland, June 2018.

[55] M. Liu, G. Chowdhary, B. Castra da Silva, S. Liu, and J. P. How. Gaussian Processes for Learning and Control: A Tutorial with Examples. *IEEE Control Systems Magazine*, 38(5):53 – 86, 2018.

[56] X. Luo, Y. Zhang, and M. M. Zavlanos. Socially-Aware Robot Planning via Bandit Human Feedback. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 216–225, 2020.

[57] M. Maros and J. Jalden. ADMM for Distributed Dynamic Beam-forming. *IEEE Transactions on Signal and Information Processing over Networks*, 4(2):220 – 235, 2018.

[58] R. Mazumder, A. Choudhury, G. Iyengar, and B. Sen. A Computational Framework for Multivariate Convex Regression and Its Variants. *Journal of the American Statistical Association*, 114(525):318–331, 2019.

[59] M. Menner, L. Neuner, L. Lünenburger, and M. N. Zeilinger. Using human ratings for feedback control: A supervised learning approach with application to rehabilitation. *IEEE Transactions on Robotics*, 36, 2020.

[60] P. Miao, Y. Shen, Y. Huang, and Y.-W. Wang. Solving time-varying quadratic programs based on finite-time Zhang neural networks and their application to robot tracking. *Neural Computing and Applications*, 26(3):693 – 703, 2015.

[61] J. J. Moreau. Evolution Problem Associated with a Moving Convex Set in a Hilbert Space. *Journal of Differential Equations*, 26:347 – 374, 1977.

[62] A. Nagurney. *Network economics: A variational inequality approach*. Springer Science & Business Media, 2013.

[63] Y. Nesterov. *Introductory Lectures on Convex Optimization*. Applied Optimization. Springer, 2004.

[64] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[65] I. Notarnicola, **A. Simonetto**, F. Farina, and G. Notarstefano. Distributed personalized gradient tracking with convex parametric models. *IEEE Transactions on Automatic Control (in press)*, 2022.

[66] F. Oldewurtel, A. Parisio, C. N. Jones, D. Gyalistras, M. Gwerder, Stauch, B. Lehmann, and M. Morari. Use of model predictive control and weather forecasts for energy efficient building climate control. *Energy and Buildings*, 45:15 – 27, 2012.

[67] L. Ortmann, A. Hauswirth, I. Caduff, F. Dörfler, and S. Bolognani. Experimental validation of feedback optimization in power distribution grids. *Electric Power Systems Research*, 189:106782, 2020.

[68] A. Ospina, **A. Simonetto**, and E. Dall'Anese. Time-varying optimization of networked systems with human preferences. *arXiv:2103.13470*, 2021.

[69] V. Pandey, J. Monteil, C. Gambella, and **A. Simonetto**. On the needs for MaaS platforms to handle competition in ridesharing mobility. *Transportation Research Part C: Emerging Technologies*, 108:269–288, 2019.

[70] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[71] S. Paternain, M. Morari, and A. Ribeiro. A prediction-correction algorithm for real-time model predictive control. *arXiv preprint arXiv:1911.10051*, 2019.

[72] R. Pinsler, R. Akrour, T. Osa, J. Peters, and G. Neumann. Sample and Feedback Efficient Hierarchical Reinforcement Learning from Human Preferences. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 596–601, 2018.

[73] B. T. Polyak. *Introduction to Optimization*. Optimization Software, Inc., 1987.

[74] D. Quercia, R. Schifanella, and L. M. Aiello. The Shortest Path to Happiness: Recommending Beautiful, Quiet, and Happy Routes in the City. In *Proceedings of Conference on Hypertext and Social Media*, pages 116 – 125, Santiago, Chile, September 2014.

[75] S. Rahili and W. Ren. Distributed Convex Optimization for Continuous-Time Dynamics with Time-Varying Cost Functions. *IEEE Transactions on Automatic Control*, 62(4):1590 – 1605, 2017.

[76] A. Rakhlin and K. Sridharan. Online learning with predictable sequences. In *COLT, PMLR*, 2013.

[77] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, MA, US, 2006.

[78] S. Shalev-Shwartz. Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning*, 4(2):107 – 194, 2012.

[79] S. Shibasaki, M. Inoue, M. Arahata, and V. Gupta. Weak control approach to consumer-preferred energy management. *IFAC-Papers online*, 53, 2020.

[80] A. Slivkins and E. Upfal. Adapting to a Changing Environment: the Brownian Restless Bandits. In *Proceedings of the Conference on Learning Theory*, pages 343 – 354, Helsinki, Finland, July 2008.

[81] J. Song, E. Dall'Anese, **A. Simonetto**, and H. Zhu. Dynamic distribution state estimation using synchrophasor data. *IEEE Transactions on Smart Grid*, 2019.

[82] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Transactions on Information Theory*, 58(5):3250 – 3265, 2012.

[83] Y. Tang, K. Dvijotham, and S. Low. Real-Time Optimal Power Flow. *IEEE Transactions on Smart Grid*, 8(6):2963 – 2973, 2017.

[84] A.B. Taylor, J.M. Hendrickx, and F. Glineur. Smooth Strongly Convex Interpolation and Exact Worst-case Performance of First-order Methods. *Mathematical Programming*, 2016.

[85] **A. Simonetto**. Time-Varying Convex Optimization via Time-Varying Averaged Operators . *arXiv: 1704.07338v1*, 2017.

[86] **A. Simonetto**. Dual Prediction-Correction Methods for Linearly Constrained Time-Varying Convex Programs. *IEEE Transactions on Automatic Control*, 64(8):3355 – 3361, 2019.

[87] **A. Simonetto**. Smooth Strongly Convex Regression. In *European Signal Processing Conference*, pages 2130–2134. IEEE, 2021.

[88] **A. Simonetto** and E. Dall'Anese. Prediction-Correction Algorithms for Time-Varying Constrained Optimization. *IEEE Transactions on Signal Processing*, 65(20):5481 – 5494, 2017.

[89] **A. Simonetto**, E. Dall'Anese, J. Monteil, and A. Bernstein. Personalized optimization with user's feedback. *Automatica*, 131, 2021.

[90] **A. Simonetto**, E. Dall'Anese, S. Paternain, G. Leus, and G. B. Giannakis. Time-Varying Convex Optimization: Time-Structured Algorithms and Applications. *Proceedings of the IEEE*, 108(11), 2020.

[91] **A. Simonetto**, A. Koppel, A. Mokhtari, G. Leus, and A. Ribeiro. Decentralized Prediction-Correction Methods for Networked Time-Varying Convex Optimization. *IEEE Transactions on Automatic Control*, 62(11):5724 – 5738, 2017.

[92] **A. Simonetto** and G. Leus. Distributed Asynchronous Time-Varying Constrained Optimization. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, USA, November 2014.

[93] **A. Simonetto** and G. Leus. Double Smoothing for Time-Varying Distributed Multi-user Optimization. In *Proceedings of the IEEE Global Conference on Signal and Information Processing*, Atlanta, US, December 2014.

[94] **A. Simonetto**, A. Mokhtari, A. Koppel, G. Leus, and A. Ribeiro. A Class of Prediction-Correction Methods for Time-Varying Convex Optimization. *IEEE Transactions on Signal Processing*, 64(17):4576 – 4591, 2016.

[95] **A. Simonetto**, J. Monteil, and C. Gambella. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101, 2019.

[96] N. Vaswani and J. Zhan. Recursive Recovery of Sparse Signal Sequences from Compressive Measurements: A Review. *IEEE Transactions on Signal Processing*, 64(13):3523 – 3549, 2016.

[97] X. Wang and J. O. Berger. Estimating shape constrained functions using Gaussian Processes. *SIAM/ASA Journal on Uncertainty Quantification*, 4(1):1–25, 2016.

[98] Y. Yang, M. Zhang, M. Pesavento, and D. P. Palomar. An Online Parallel and Distributed Algorithm for Recursive Estimation of Sparse Signals. *IEEE Transactions on Signal and Information Processing over Networks*, 2(3):290 – 305, 2016.

[99] K. Yuan, W. Xu, and Q. Ling. Can primal methods outperform primal-dual methods in decentralized dynamic optimization? *arXiv preprint arXiv:2003.00816*, 2020.

[100] V. M. Zavala and M. Anitescu. Real-Time Nonlinear Optimization as a Generalized Equation. *SIAM Journal of Control and Optimization*, 48(8):5444 – 5467, 2010.

[101] M. M. Zavlanos, A. Ribeiro, and G. J. Pappas. Network Integrity in Mobile Robotic Networks. *IEEE Transactions on Automatic Control*, 58(1):3 – 18, 2013.